

# Table of Contents

## [Service Fabric Documentation](#)

### [Overview](#)

#### [What is Service Fabric?](#)

### [Quickstarts](#)

#### [Create .NET application](#)

#### [Deploy a Linux container application](#)

#### [Deploy a Windows container application](#)

### [Tutorials](#)

#### [Deploy a .NET app](#)

##### [1- Build a .NET application](#)

##### [2- Deploy the application](#)

##### [3- Configure CI/CD](#)

##### [4- Monitor and Diagnose](#)

#### [Containerize an existing .NET app](#)

##### [1- Create a secure cluster on Azure](#)

##### [2- Deploy a .NET app using Docker Compose](#)

##### [3- Monitor your container](#)

#### [Create a Linux container app](#)

##### [1- Create container images](#)

##### [2- Package and deploy containers](#)

##### [3- Fail over and scale](#)

#### [Create and manage a cluster](#)

##### [1- Create a cluster on Azure](#)

##### [2- Scale the cluster](#)

##### [3- Deploy API Management with Service Fabric](#)

### [Samples](#)

#### [Code samples](#)

#### [Azure PowerShell](#)

#### [Service Fabric CLI](#)

## Concepts

Understand microservices

Big picture

Application scenarios

Patterns and scenarios

Architecture

Terminology

Build applications and services

Supported programming models

Application model

Hosting model

Services

Scalability of applications

ASP.NET Core

Plan application capacity

Manage applications

Overview

The ImageStoreConnectionString setting

Application upgrade

Fault analysis overview

Create and manage clusters

Overview

Plan and prepare

Describing a cluster

Cluster security

Feature differences between Linux and Windows

Clusters on Azure

Cluster resource manager

Integrate with API Management

Monitor and diagnose

Overview

Health model

[Diagnostics in stateful Reliable Services](#)  
[Diagnostics in Reliable Actors](#)  
[Performance counters for Reliable Service Remoting](#)

## How-to guides

[Set up your development environment](#)  
    [Windows](#)  
    [Linux](#)  
    [Mac OS](#)  
[Set up the Service Fabric CLI](#)  
[Build an application](#)  
    [Create your first C# app in Visual Studio](#)  
    [Build a guest executable service](#)  
    [Build a container service](#)  
    [Build a Reliable Services service](#)  
    [Build a Reliable Actors service](#)  
    [Migrate old Java Application to support Maven](#)  
    [Configure reverse proxy for secure communication](#)  
    [Build an ASP.NET Core service](#)  
    [Configure security](#)  
[Work in a Windows dev environment](#)  
    [Manage applications in Visual Studio](#)  
    [Configure secure connections in Visual Studio](#)  
    [Configure your application for multiple environments](#)  
    [Debug a .NET service in VS](#)  
    [Common errors and exceptions](#)  
    [Monitor and diagnose locally](#)  
[Work in a Linux dev environment](#)  
    [Get started with Eclipse plugin for Java development](#)  
    [Debug a Java service in Eclipse](#)  
    [Monitor and diagnose locally](#)  
[Deploy API Management and Service Fabric to Azure](#)  
[Migrate from Cloud Services](#)

[Compare Cloud Services with Service Fabric](#)

[Migrate to Service Fabric](#)

[Recommended practices](#)

[Manage application lifecycle](#)

[Package an application](#)

[Deploy or remove applications](#)

[Upgrade applications](#)

[Test applications and services](#)

[Set up continuous integration](#)

[Create and manage clusters](#)

[Clusters on Azure](#)

[Standalone clusters](#)

[Visualize a cluster](#)

[Connect to a secure cluster](#)

[Patch cluster nodes](#)

[Manage and orchestrate cluster resources](#)

[Monitor and diagnose](#)

[Monitor and diagnose applications](#)

[Generate events](#)

[Inspect application and cluster health](#)

[Aggregate events](#)

[Analyze events](#)

[Troubleshoot your local cluster](#)

[Reference](#)

[Azure PowerShell](#)

[PowerShell](#)

[Azure CLI \(az sf\)](#)

[Service Fabric CLI \(sfctl\)](#)

[sfctl application](#)

[sfctl chaos](#)

[sfctl cluster](#)

[sfctl compose](#)

[sfctl is](#)  
[sfctl node](#)  
[sfctl partition](#)  
[sfctl replica](#)  
[sfctl rpm](#)  
[sfctl service](#)  
[sfctl store](#)

[Java API](#)

[.NET](#)

[REST](#)

[Service model XML schema](#)

## Resources

[Azure Roadmap](#)

[Common questions](#)

[Learning path](#)

[MSDN Forum](#)

[Pricing](#)

[Pricing calculator](#)

[Sample code](#)

[Support options](#)

[Service Updates](#)

## Videos

# Overview of Azure Service Fabric

9/25/2017 • 6 min to read • [Edit Online](#)

Azure Service Fabric is a distributed systems platform that makes it easy to package, deploy, and manage scalable and reliable microservices and containers. Service Fabric also addresses the significant challenges in developing and managing cloud native applications. Developers and administrators can avoid complex infrastructure problems and focus on implementing mission-critical, demanding workloads that are scalable, reliable, and manageable. Service Fabric represents the next-generation platform for building and managing these enterprise-class, tier-1, cloud-scale applications running in containers.

This short video introduces Service Fabric and microservices:



## Applications composed of microservices

Service Fabric enables you to build and manage scalable and reliable applications composed of microservices that run at high density on a shared pool of machines, which is referred to as a cluster. It provides a sophisticated, lightweight runtime to build distributed, scalable, stateless, and stateful microservices running in containers. It also provides comprehensive application management capabilities to provision, deploy, monitor, upgrade/patch, and delete deployed applications including containerized services.

Service Fabric powers many Microsoft services today, including Azure SQL Database, Azure Cosmos DB, Cortana, Microsoft Power BI, Microsoft Intune, Azure Event Hubs, Azure IoT Hub, Dynamics 365, Skype for Business, and many core Azure services.

Service Fabric is tailored to create cloud native services that can start small, as needed, and grow to massive scale with hundreds or thousands of machines.

Today's Internet-scale services are built of microservices. Examples of microservices include protocol gateways, user profiles, shopping carts, inventory processing, queues, and caches. Service Fabric is a microservices platform that gives every microservice (or container) a unique name that can be either stateless or stateful.

Service Fabric provides comprehensive runtime and lifecycle management capabilities to applications that are composed of these microservices. It hosts microservices inside containers that are deployed and activated across the Service Fabric cluster. A move from virtual machines to containers makes possible an order-of-magnitude increase in density. Similarly, another order of magnitude in density becomes possible when you move from containers to microservices in these containers. For example, a single cluster for Azure SQL Database comprises hundreds of machines running tens of thousands of containers that host a total of hundreds of thousands of databases. Each database is a Service Fabric stateful microservice.

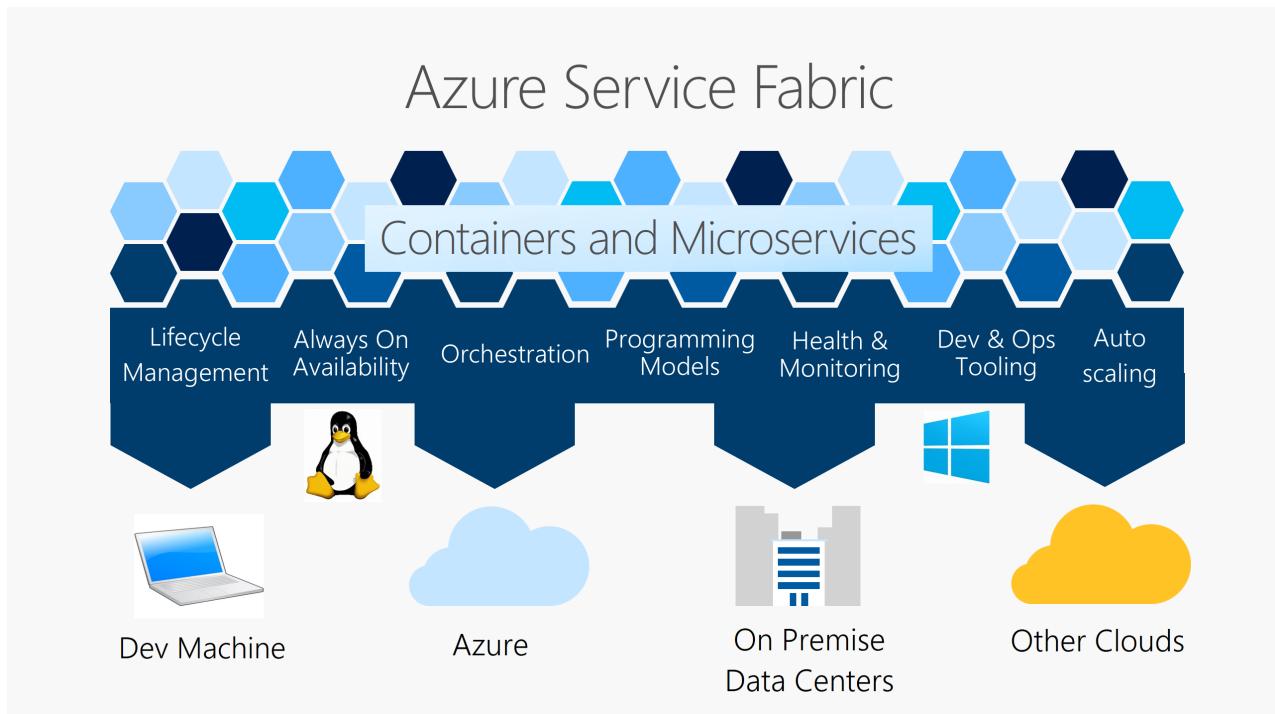
For more on the microservices approach, read [Why a microservices approach to building applications?](#)

## Container deployment and orchestration

Service Fabric is Microsoft's [container orchestrator](#) deploying microservices across a cluster of machines. Microservices can be developed in many ways from using the [Service Fabric programming models](#), [ASP.NET Core](#), to deploying [any code of your choice](#). Importantly, you can mix both services in processes and services in containers in the same application. If you just want to [deploy and manage containers](#), Service Fabric is a perfect choice as a container orchestrator.

## Any OS, any cloud

Service Fabric runs everywhere. You can create clusters for Service Fabric in many environments, including Azure or on premises, on Windows Server, or on Linux. You can even create clusters on other public clouds. In addition, the development environment in the SDK is **identical** to the production environment, with no emulators involved. In other words, what runs on your local development cluster deploys to the clusters in other environments.



For Windows development, the Service Fabric .NET SDK is integrated with Visual Studio and Powershell. See [Prepare your development environment on Windows](#). For Linux development, the Service Fabric Java SDK is integrated with Eclipse, and Yeoman is used to generate templates for Java, .NET Core, and container applications. See [Prepare your development environment on Linux](#).

For more information on creating clusters, read [creating a cluster on Windows Server or Linux](#) or for Azure creating a cluster [via the Azure portal](#).

## Stateless and stateful microservices for Service Fabric

Service Fabric enables you to build applications that consist of microservices or containers. Stateless microservices (such as protocol gateways and web proxies) do not maintain a mutable state outside a request and its response from the service. Azure Cloud Services worker roles are an example of a stateless service. Stateful microservices (such as user accounts, databases, devices, shopping carts, and queues) maintain a mutable, authoritative state beyond the request and its response. Today's Internet-scale applications consist of a combination of stateless and stateful microservices.

A key differentiation with Service Fabric is its strong focus on building stateful services, either with the [built-in programming models](#) or with containerized stateful services. The [application scenarios](#) describe the scenarios where stateful services are used.

# Application lifecycle management

Service Fabric provides support for the full application lifecycle and CI/CD of cloud applications including containers. This lifecycle includes development through deployment, daily management, and maintenance to eventual decommissioning.

Service Fabric application lifecycle management capabilities enable application administrators and IT operators to use simple, low-touch workflows to provision, deploy, patch, and monitor applications. These built-in workflows greatly reduce the burden on IT operators to keep applications continuously available.

Most applications consist of a combination of stateless and stateful microservices, containers, and other executables that are deployed together. By having strong types on the applications, Service Fabric enables the deployment of multiple application instances. Each instance is managed and upgraded independently. Importantly, Service Fabric can deploy containers or any executables and make them reliable. For example, Service Fabric can deploy .NET, ASP.NET Core, node.js, Windows containers, Linux containers, Java virtual machines, scripts, Angular, or literally anything that makes up your application.

Service Fabric is integrated with CI/CD tools such as [Visual Studio Team Services](#), [Jenkins](#), and [Octopus Deploy](#) and can be used with any other popular CI/CD tool.

For more information about application lifecycle management, read [Application lifecycle](#). For more about how to deploy any code, see [deploy a guest executable](#).

## Key capabilities

By using Service Fabric, you can:

- Deploy to Azure or to on-premises datacenters that run Windows or Linux with zero code changes. Write once, and then deploy anywhere to any Service Fabric cluster.
- Develop scalable applications that are composed of microservices by using the Service Fabric programming models, containers, or any code.
- Develop highly reliable stateless and stateful microservices. Simplify the design of your application by using stateful microservices.
- Use the novel Reliable Actors programming model to create cloud objects with self contained code and state.
- Deploy and orchestrate containers that include Windows containers and Linux containers. Service Fabric is a data aware, stateful, container orchestrator.
- Deploy applications in seconds, at high density with hundreds or thousands of applications or containers per machine.
- Deploy different versions of the same application side by side, and upgrade each application independently.
- Manage the lifecycle of your applications without any downtime, including breaking and nonbreaking upgrades.
- Scale out or scale in the number of nodes in a cluster. As you scale nodes, your applications automatically scale.
- Monitor and diagnose the health of your applications and set policies for performing automatic repairs.
- Watch the resource balancer orchestrate the redistribution of applications across the cluster. Service Fabric recovers from failures and optimizes the distribution of load based on available resources.

## Next steps

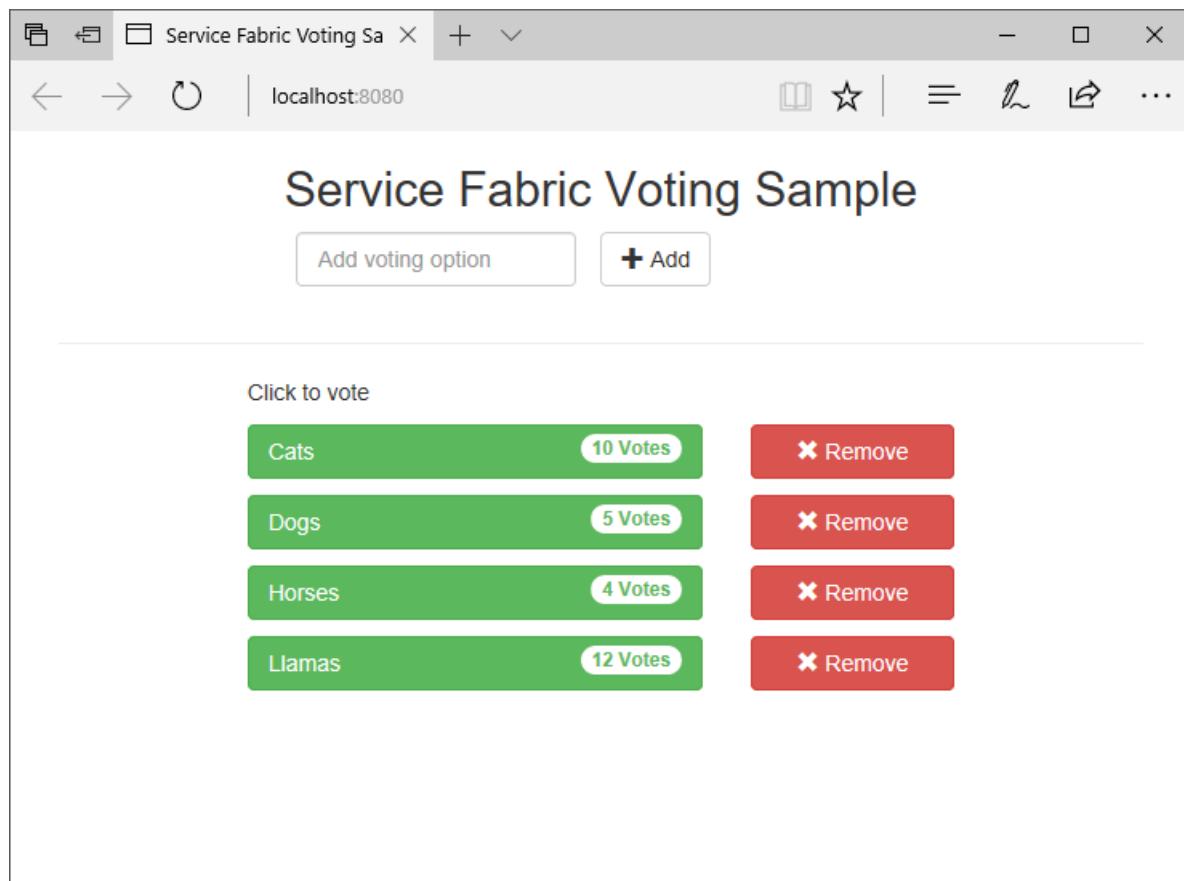
- For more information:
  - [Why a microservices approach to building applications?](#)
  - [Terminology overview](#)
- Setting up your [Windows development environment](#)
- Setting up your [Linux development environment](#)
- Learn about [Service Fabric support options](#)

# Create a .NET Service Fabric application in Azure

10/3/2017 • 8 min to read • [Edit Online](#)

Azure Service Fabric is a distributed systems platform for deploying and managing scalable and reliable microservices and containers.

This quickstart shows how to deploy your first .NET application to Service Fabric. When you're finished, you have a voting application with an ASP.NET Core web front-end that saves voting results in a stateful back-end service in the cluster.



Using this application you learn how to:

- Create an application using .NET and Service Fabric
- Use ASP.NET core as a web front-end
- Store application data in a stateful service
- Debug your application locally
- Deploy the application to a cluster in Azure
- Scale-out the application across multiple nodes
- Perform a rolling application upgrade

## Prerequisites

To complete this quickstart:

1. [Install Visual Studio 2017](#) with the **Azure development** and **ASP.NET and web development** workloads.
2. [Install Git](#)
3. [Install the Microsoft Azure Service Fabric SDK](#)

4. Run the following command to enable Visual Studio to deploy to the local Service Fabric cluster:

```
powershell Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Force -Scope CurrentUser
```

## Download the sample

In a command window, run the following command to clone the sample app repository to your local machine.

```
git clone https://github.com/Azure-Samples/service-fabric-dotnet-quickstart
```

## Run the application locally

Right-click the Visual Studio icon in the Start Menu and choose **Run as administrator**. In order to attach the debugger to your services, you need to run Visual Studio as administrator.

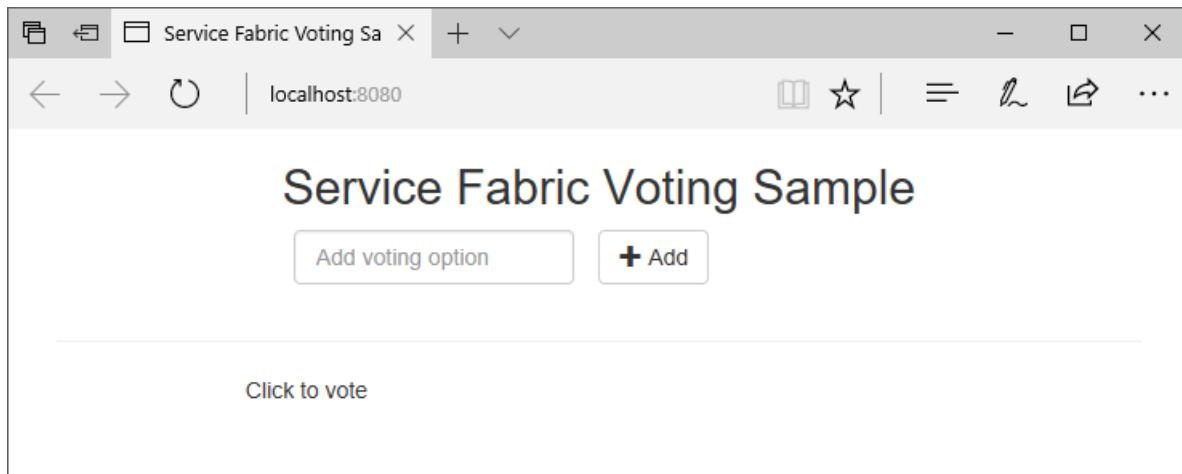
Open the **Voting.sln** Visual Studio solution from the repository you cloned.

To deploy the application, press **F5**.

### NOTE

The first time you run and deploy the application, Visual Studio creates a local cluster for debugging. This operation may take some time. The cluster creation status is displayed in the Visual Studio output window.

When the deployment is complete, launch a browser and open this page: <http://localhost:8080> - the web front-end of the application.

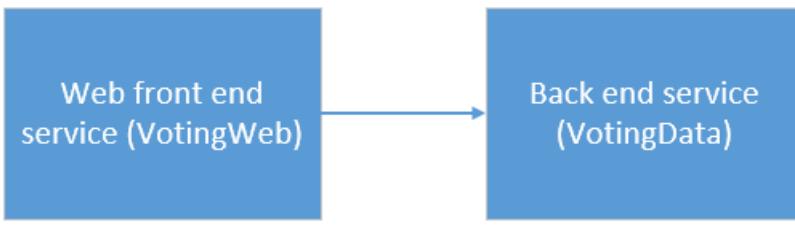


You can now add a set of voting options, and start taking votes. The application runs and stores all data in your Service Fabric cluster, without the need for a separate database.

## Walk through the voting sample application

The voting application consists of two services:

- Web front-end service (VotingWeb)- An ASP.NET Core web front-end service, which serves the web page and exposes web APIs to communicate with the backend service.
- Back-end service (VotingData)- An ASP.NET Core web service, which exposes an API to store the vote results in a reliable dictionary persisted on disk.



When you vote in the application the following events occur:

1. A JavaScript sends the vote request to the web API in the web front-end service as an HTTP PUT request.
2. The web front-end service uses a proxy to locate and forward an HTTP PUT request to the back-end service.
3. The back-end service takes the incoming request, and stores the updated result in a reliable dictionary, which gets replicated to multiple nodes within the cluster and persisted on disk. All the application's data is stored in the cluster, so no database is needed.

## Debug in Visual Studio

When debugging application in Visual Studio, you are using a local Service Fabric development cluster. You have the option to adjust your debugging experience to your scenario. In this application, we store data in our back-end service, using a reliable dictionary. Visual Studio removes the application per default when you stop the debugger. Removing the application causes the data in the back-end service to also be removed. To persist the data between debugging sessions, you can change the **Application Debug Mode** as a property on the **Voting** project in Visual Studio.

To look at what happens in the code, complete the following steps:

1. Open the **VotesController.cs** file and set a breakpoint in the web API's **Put** method (line 47) - You can search for the file in the Solution Explorer in Visual Studio.
  2. Open the **VoteDataController.cs** file and set a breakpoint in this web API's **Put** method (line 50).
  3. Go back to the browser and click a voting option or add a new voting option. You hit the first breakpoint in the web front-end's api controller.
- This is where the JavaScript in the browser sends a request to the web API controller in the front-end service.

```

public async Task<IActionResult> Put(string name)
{
    string payload = $"{{ 'name' : '{name}' }}";
    StringContent putContent = new StringContent(payload, Encoding.UTF8, "application/json");
    putContent.Headers.ContentType = new MediaTypeHeaderValue("application/json");

    ① string proxyUrl = $"{serviceProxyUrl}/{name}?PartitionKind={partitionKind}&PartitionKey={partitionKey}";
    ② HttpResponseMessage response = await this.httpClient.PutAsync(proxyUrl, putContent);

    return new ContentResult()
    ③ {
        StatusCode = (int)response.StatusCode,
        Content = await response.Content.ReadAsStringAsync()
    };
}

```

- First we construct the URL to the ReverseProxy for our back-end service **(1)**.
  - Then we send the HTTP PUT Request to the ReverseProxy **(2)**.
  - Finally the we return the response from the back-end service to the client **(3)**.
4. Press **F5** to continue
- You are now at the break point in the back-end service.

```

public async Task<IActionResult> Put(string name)
{
    1 var votesDictionary = await this.stateManager.GetOrAddAsync<IReliableDictionary<string, int>>("counts");
    2 using (ITransaction tx = this.stateManager.CreateTransaction())
    {
        3 await votesDictionary.AddOrUpdateAsync(tx, name, 1, (key, oldvalue) => oldvalue + 1);
    }

    return new OkResult();
}

```

- In the first line in the method **(1)** we are using the `stateManager` to get or add a reliable dictionary called `counts`.
- All interactions with values in a reliable dictionary require a transaction, this using statement **(2)** creates that transaction.
- In the transaction, we then update the value of the relevant key for the voting option and commits the operation **(3)**. Once the commit method returns, the data is updated in the dictionary and replicated to other nodes in the cluster. The data is now safely stored in the cluster, and the back-end service can fail over to other nodes, still having the data available.

5. Press **F5** to continue

To stop the debugging session, press **Shift+F5**.

## Deploy the application to Azure

To deploy the application to a cluster in Azure, you can either choose to create your own cluster, or use a Party Cluster.

Party clusters are free, limited-time Service Fabric clusters hosted on Azure and run by the Service Fabric team where anyone can deploy applications and learn about the platform. To get access to a Party Cluster, [follow the instructions](#).

For information about creating your own cluster, see [Create your first Service Fabric cluster on Azure](#).

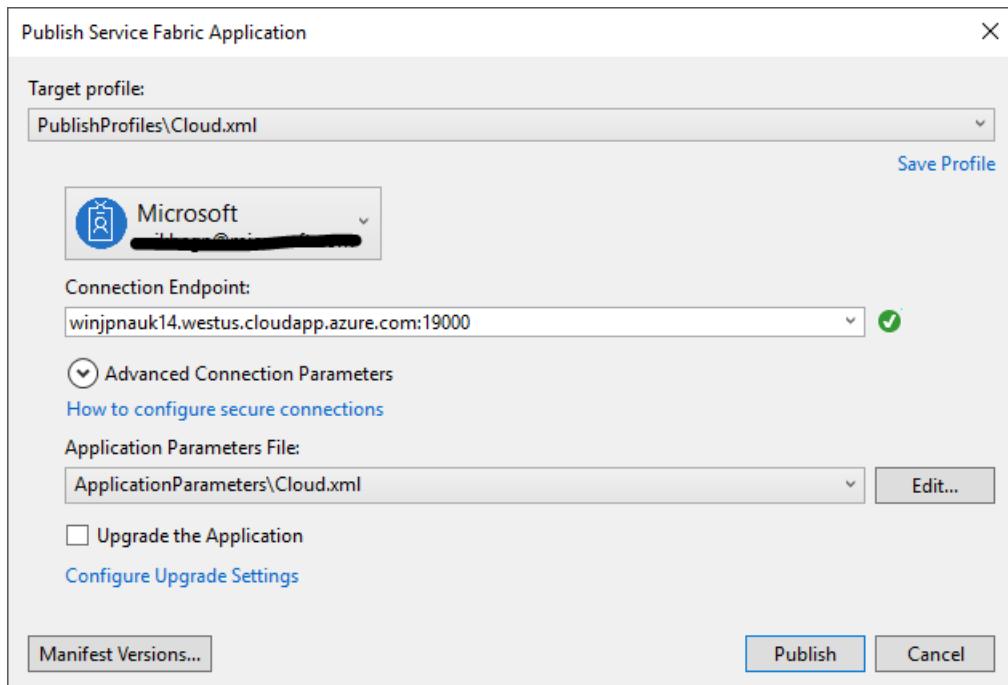
### NOTE

The web front-end service is configured to listen on port 8080 for incoming traffic. Make sure that port is open in your cluster. If you are using the Party Cluster, this port is open.

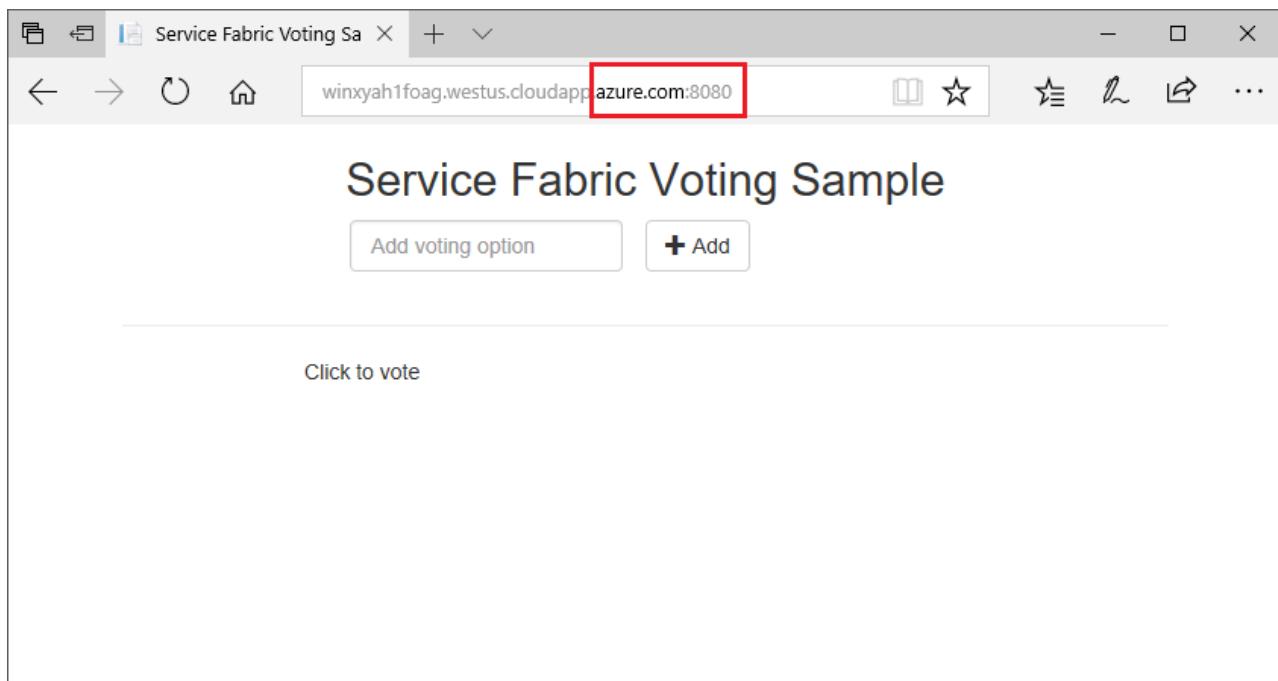
## Deploy the application using Visual Studio

Now that the application is ready, you can deploy it to a cluster directly from Visual Studio.

- Right-click **Voting** in the Solution Explorer and choose **Publish**. The Publish dialog appears.



2. Type in the Connection Endpoint of the cluster in the **Connection Endpoint** field and click **Publish**. When signing up for the Party Cluster, the Connection Endpoint is provided in the browser. - for example, `winh1x87d1d.westus.cloudapp.azure.com:19000`.
3. Open a browser and type in the cluster address followed by ':8080' to get to the application in the cluster - for example, `http://winh1x87d1d.westus.cloudapp.azure.com:8080`. You should now see the application running in the cluster in Azure.



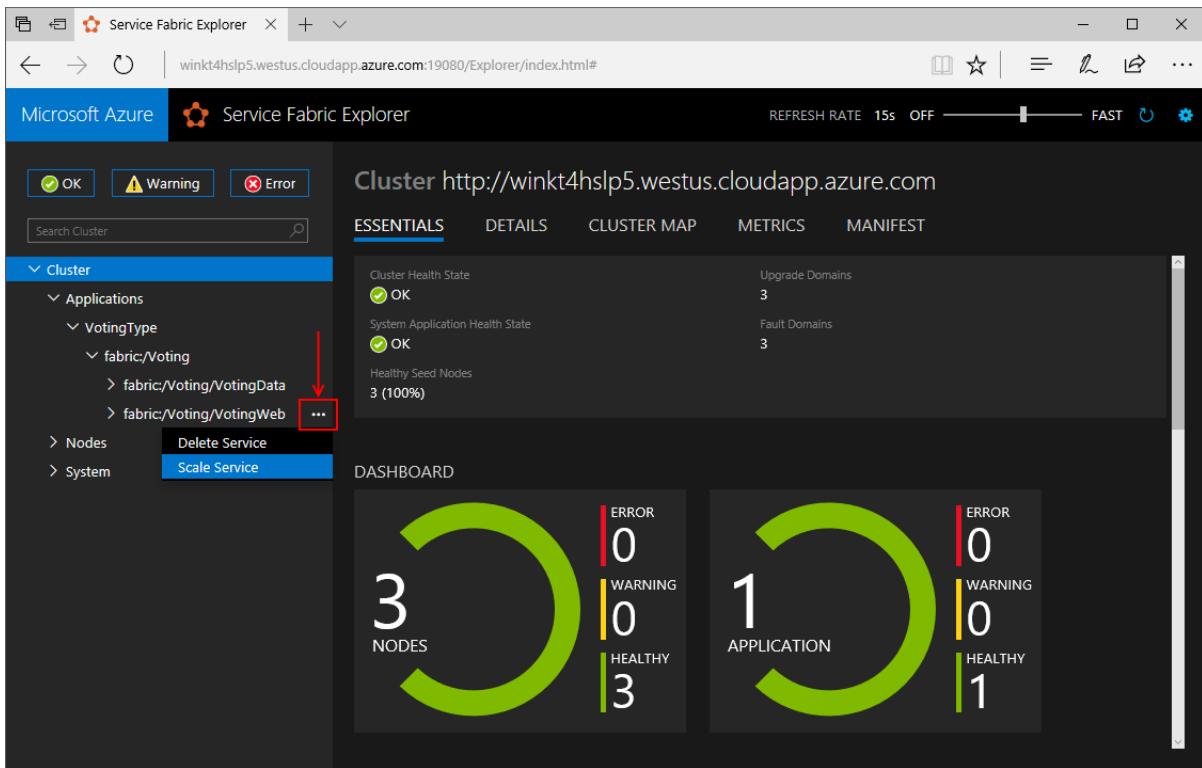
## Scale applications and services in a cluster

Service Fabric services can easily be scaled across a cluster to accommodate for a change in the load on the services. You scale a service by changing the number of instances running in the cluster. You have multiple ways of scaling your services, you can use scripts or commands from PowerShell or Service Fabric CLI (sfctl). In this example, we are using Service Fabric Explorer.

Service Fabric Explorer runs in all Service Fabric clusters and can be accessed from a browser, by browsing to the clusters HTTP management port (19080), for example, `http://winh1x87d1d.westus.cloudapp.azure.com:19080`.

To scale the web front-end service, do the following steps:

1. Open Service Fabric Explorer in your cluster - for example, <http://winkt4hsdp5.westus.cloudapp.azure.com:19080>.
2. Click on the ellipsis (three dots) next to the **fabric:/Voting/VotingWeb** node in the treeview and choose **Scale Service**.



You can now choose to scale the number of instances of the web front-end service.

3. Change the number to **2** and click **Scale Service**.
4. Click on the **fabric:/Voting/VotingWeb** node in the tree-view and expand the partition node (represented by a GUID).

The screenshot shows the Microsoft Azure Service Fabric Explorer interface. On the left, a tree view displays the cluster structure under 'Cluster' > 'Applications' > 'VotingType' > 'fabric:/Voting'. A red box highlights the 'fabric:/Voting/VotingWeb' node. Underneath it, two instances are listed: '\_nt1party\_0' and '\_nt1party\_1', also highlighted with a red box. The main pane is titled 'Service fabric:/Voting/VotingWeb' and contains three tabs: 'ESSENTIALS' (selected), 'DETAILS', and 'MANIFEST'. The 'ESSENTIALS' tab shows the service's name as 'fabric:/Voting/VotingWeb', service type as 'VotingWebType', health state as 'OK', status as 'Active', service kind as 'Stateless', and instance count as '2'. The 'PARTITIONS' section shows one partition with ID '0ed687d3-ba25-4ed6-b228-31e269a979a3', kind as 'Singleton', health state as 'OK', and status as 'Ready'. The 'UNHEALTHY EVALUATIONS' section is empty.

You can now see that the service has two instances, and in the tree view you see which nodes the instances run on.

By this simple management task, we doubled the resources available for our front-end service to process user load. It's important to understand that you do not need multiple instances of a service to have it run reliably. If a service fails, Service Fabric makes sure a new service instance runs in the cluster.

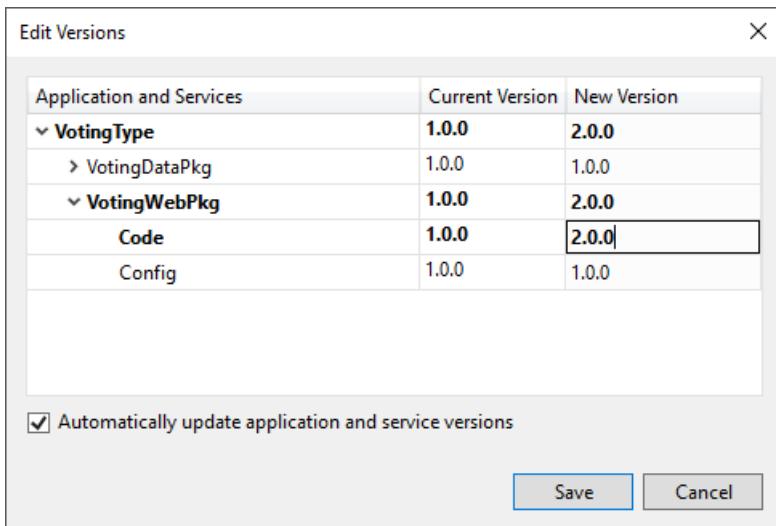
## Perform a rolling application upgrade

When deploying new updates to your application, Service Fabric rolls out the update in a safe way. Rolling upgrades gives you no downtime while upgrading as well as automated rollback should errors occur.

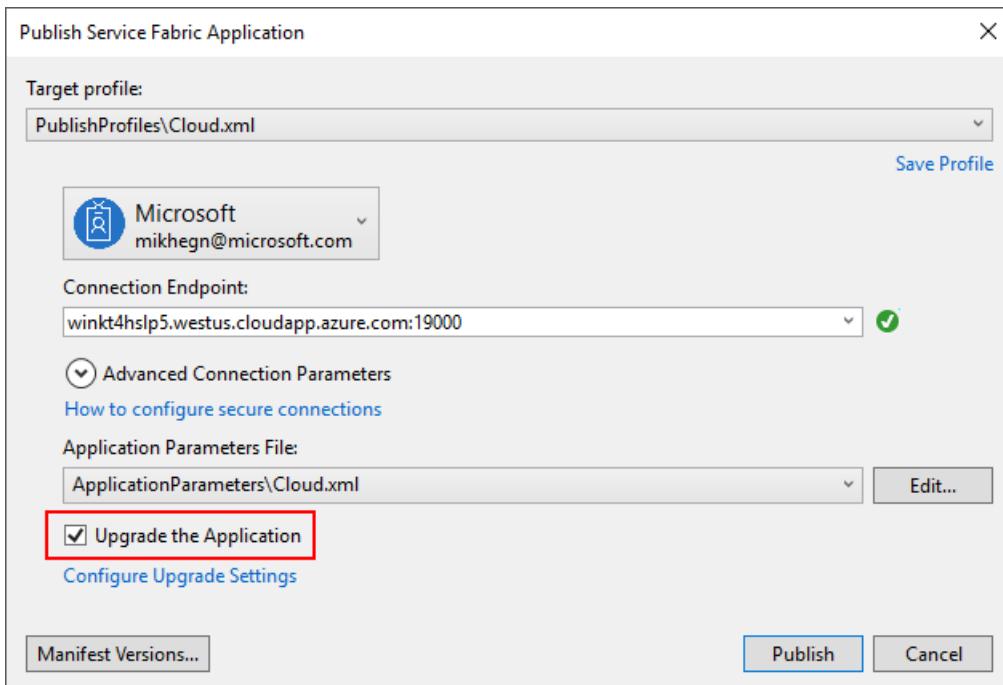
To upgrade the application, do the following:

1. Open the **Index.cshtml** file in Visual Studio - You can search for the file in the Solution Explorer in Visual Studio.
2. Change the heading on the page by adding some text - for example.  

```
html <div class="col-xs-8 col-xs-offset-2 text-center"> <h2>Service Fabric Voting Sample v2</h2> </div>
```
3. Save the file.
4. Right-click **Voting** in the Solution Explorer and choose **Publish**. The Publish dialog appears.
5. Click the **Manifest Version** button to change the version of the service and application.
6. Change the version of the **Code** element under **VotingWebPkg** to "2.0.0", for example, and click **Save**.



7. In the **Publish Service Fabric Application** dialog, check the Upgrade the Application checkbox, and click **Publish**.



8. Open your browser and browse to the cluster address on port 19080 - for example, <http://winkt4hslp5.westus.cloudapp.azure.com:19080>.
9. Click on the **Applications** node in the tree view, and then **Upgrades in Progress** in the right-hand pane. You see how the upgrade rolls through the upgrade domains in your cluster, making sure each domain is healthy before proceeding to the next.

The screenshot shows the Microsoft Azure Service Fabric Explorer interface. The left sidebar is titled 'Cluster' and contains 'Applications' (which is selected), 'VotingType' (under Applications), 'Nodes', and 'System'. The main area is titled 'Applications' and shows 'UPGRADES IN PROGRESS 1'. A table lists one application: 'fabric/Voting' (VotingType) with 'Current Version' 1.0.0 and 'Target Version' 2.0.0. The 'Progress by Upgrade Domain' bar shows 0, 1, and 2, and the 'Upgrade State' is 'RollingForwardPending'. There are also 'OK', 'Warning', and 'X' status indicators at the top.

Service Fabric makes upgrades safe by waiting two minutes after upgrading the service on each node in the cluster. Expect the entire update to take approximately eight minutes.

10. While the upgrade is running, you can still use the application. Because you have two instances of the service running in the cluster, some of your requests may get an upgraded version of the application, while others may still get the old version.

## Next steps

In this quickstart, you learned how to:

- Create an application using .NET and Service Fabric
- Use ASP.NET core as a web front-end
- Store application data in a stateful service
- Debug your application locally
- Deploy the application to a cluster in Azure
- Scale-out the application across multiple nodes
- Perform a rolling application upgrade

To learn more about Service Fabric and .NET, take a look at this tutorial:

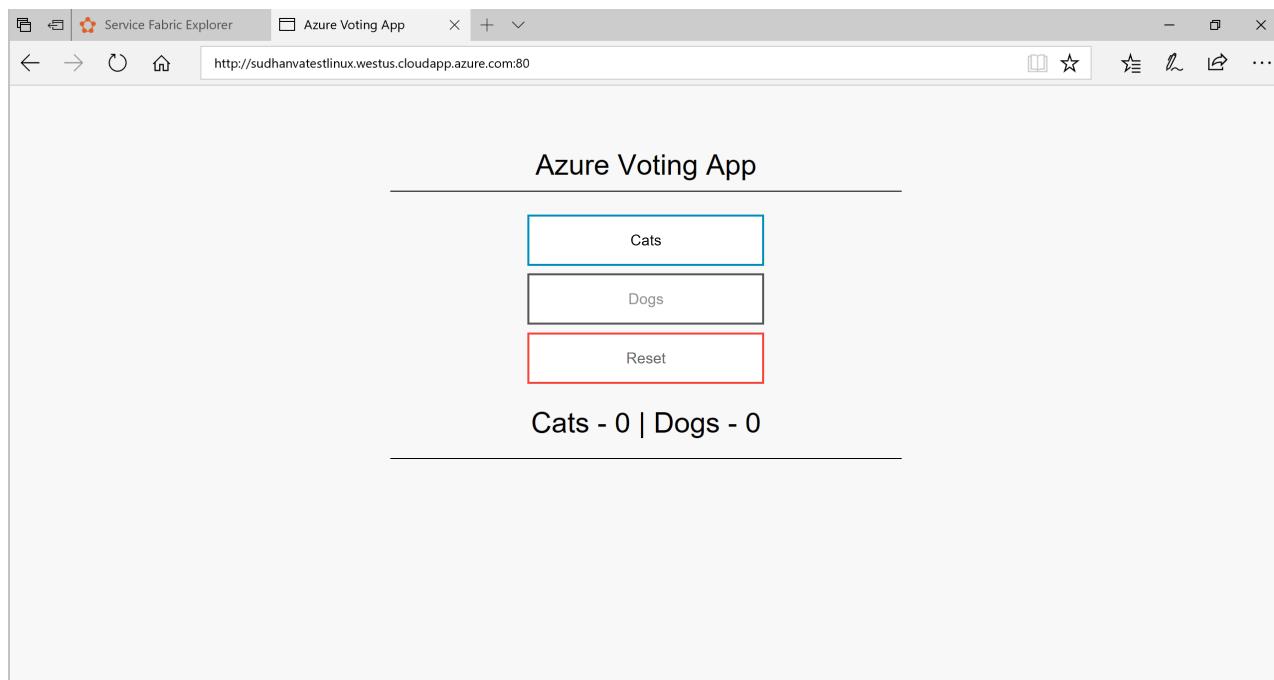
[.NET application on Service Fabric](#)

# Deploy an Azure Service Fabric Linux container application on Azure

10/23/2017 • 4 min to read • [Edit Online](#)

Azure Service Fabric is a distributed systems platform for deploying and managing scalable and reliable microservices and containers.

This quickstart shows how to deploy Linux containers to a Service Fabric cluster. Once complete, you have a voting application consisting of a Python web front-end and a Redis back-end running in a Service Fabric cluster.



In this quickstart, you learn how to:

- Deploy containers to an Azure Linux Service Fabric cluster
- Scale and failover containers in Service Fabric

## Prerequisite

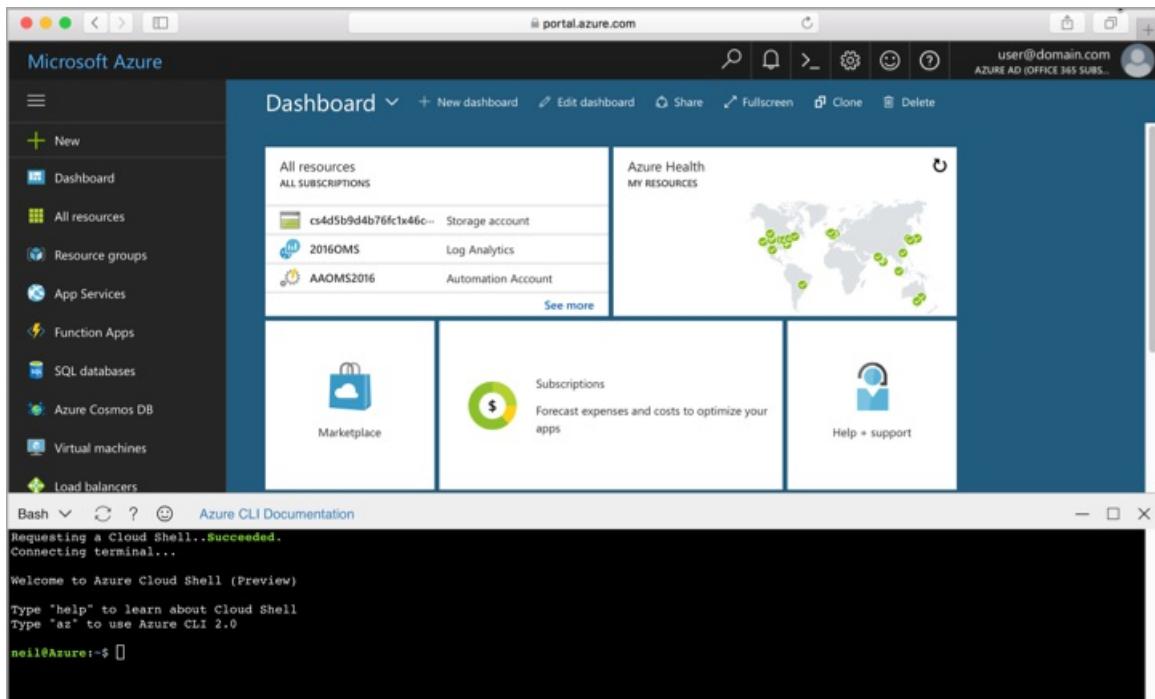
If you don't have an Azure subscription, create a [free account](#) before you begin.

## Launch Azure Cloud Shell

The Azure Cloud Shell is a free Bash shell that you can run directly within the Azure portal. It has the Azure CLI preinstalled and configured to use with your account. Click the **Cloud Shell** button on the menu in the upper-right of the [Azure portal](#).



The button launches an interactive shell that you can use to run the steps in this topic:



If you choose to install and use the command-line interface (CLI) locally, ensure you are running the Azure CLI version 2.0.4 or later. To find the version, run az --version. If you need to install or upgrade, see [Install Azure CLI 2.0](#).

## Get application package

To deploy containers to Service Fabric, you need a set of manifest files (the application definition), which describe the individual containers and the application.

In the cloud shell, use git to clone a copy of the application definition.

```
git clone https://github.com/Azure-Samples/service-fabric-dotnet-containers.git  
cd service-fabric-dotnet-containers/Linux/container-tutorial/Voting
```

## Deploy the containers to a Service Fabric cluster in Azure

To deploy the application to a cluster in Azure, use your own cluster, or use a Party cluster.

### NOTE

The application must be deployed to a cluster in Azure and not to a Service Fabric cluster on your local development machine.

Party clusters are free, limited-time Service Fabric clusters hosted on Azure. They are maintained by the Service Fabric team where anyone can deploy applications and learn about the platform. To get access to a Party cluster, [follow the instructions](#).

For information about creating your own cluster, see [Create your first Service Fabric cluster on Azure](#).

### NOTE

The web front-end service is configured to listen on port 80 for incoming traffic. Make sure that port is open in your cluster. If you are using a Party cluster, this port is open.

## Deploy the application manifests

Install the [Service Fabric CLI \(sfctl\)](#) in your CLI environment

```
pip3 install --user sfctl
export PATH=$PATH:~/local/bin
```

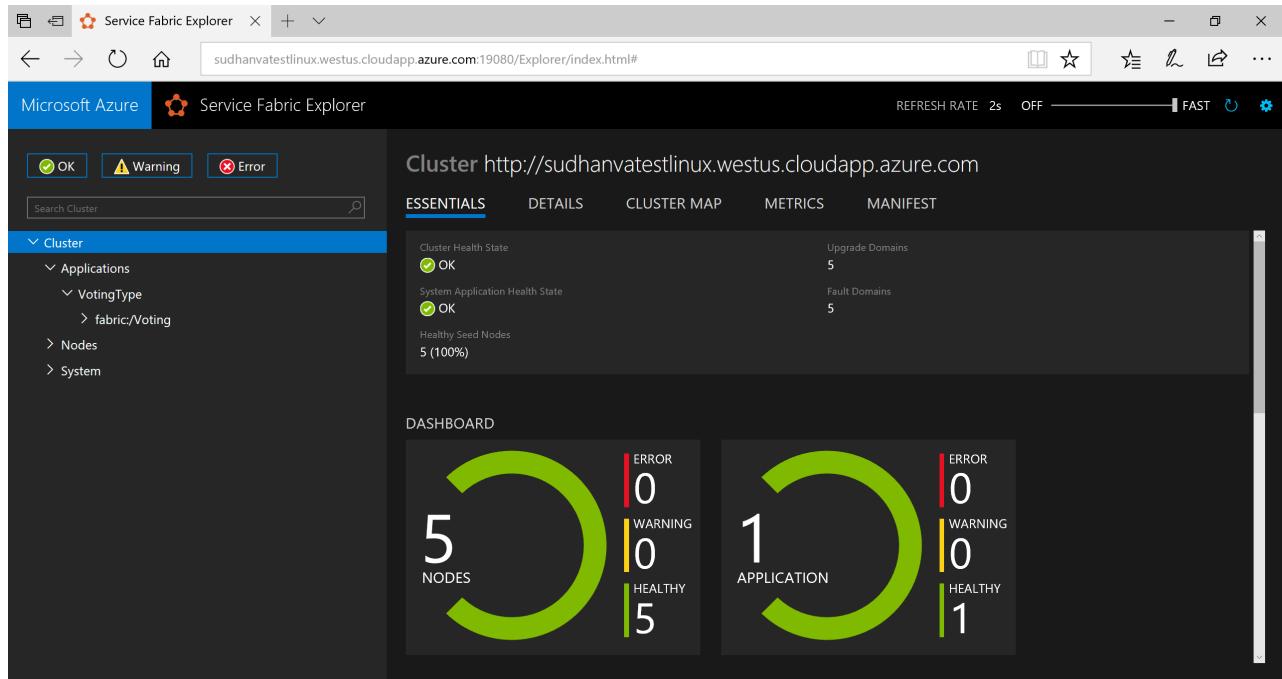
Connect to the Service Fabric cluster in Azure using the Azure CLI. The endpoint is the management endpoint of your cluster - for example, <http://linh1x87d1d.westus.cloudapp.azure.com:19080>.

```
sfctl cluster select --endpoint http://linh1x87d1d.westus.cloudapp.azure.com:19080
```

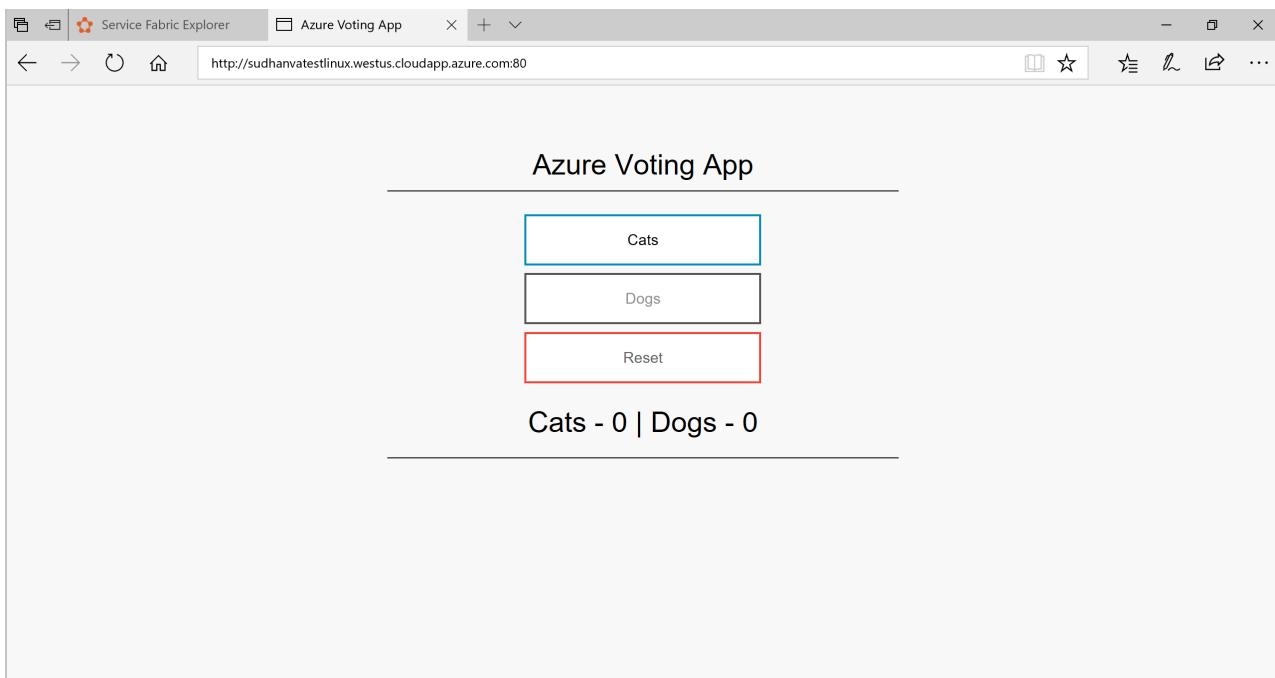
Use the install script provided to copy the Voting application definition to the cluster, register the application type, and create an instance of the application.

```
./install.sh
```

Open a browser and navigate to Service Fabric Explorer at <http://<my-azure-service-fabric-cluster-url>:19080/Explorer> - for example, <http://linh1x87d1d.westus.cloudapp.azure.com:19080/Explorer>. Expand the Applications node to see that there is now an entry for the Voting application type and the instance you created.



Connect to the running container. Open a web browser pointing to the URL of your cluster - for example, <http://linh1x87d1d.westus.cloudapp.azure.com:80>. You should see the Voting application in the browser.

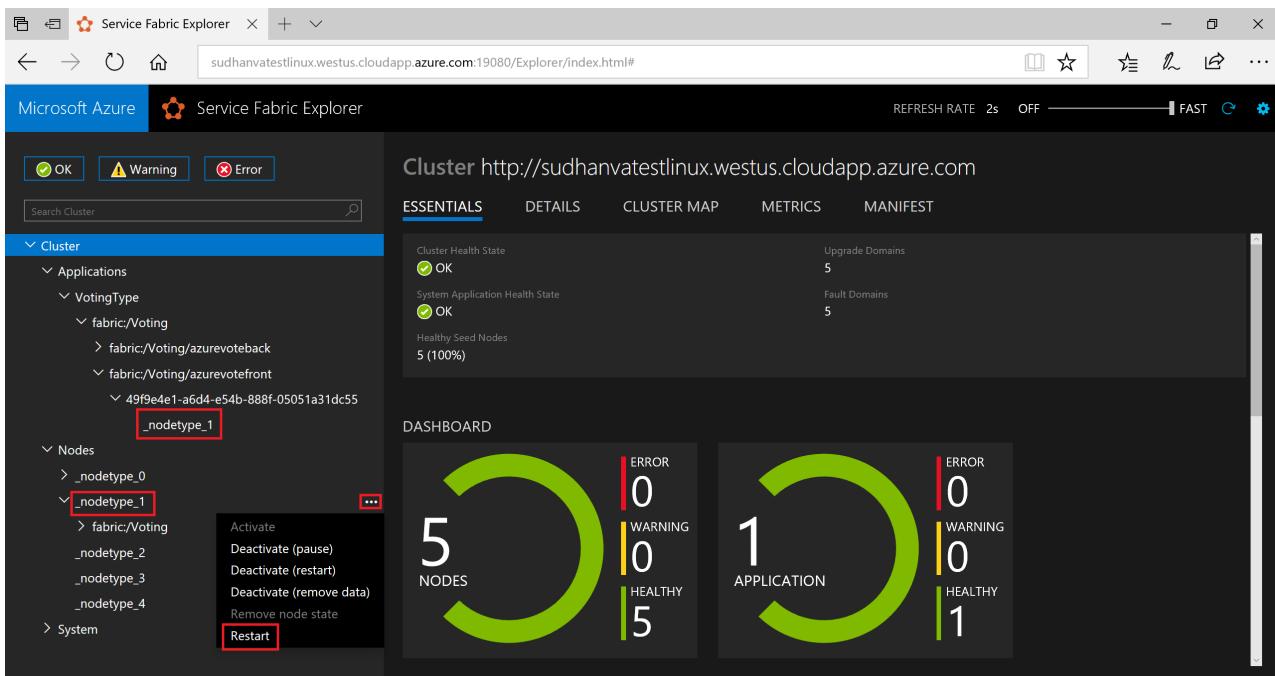


## Fail over a container in a cluster

Service Fabric makes sure your container instances automatically move to other nodes in the cluster, should a failure occur. You can also manually drain a node for containers and move them gracefully to other nodes in the cluster. You have multiple ways of scaling your services, in this example, we are using Service Fabric Explorer.

To fail over the front-end container, do the following steps:

1. Open Service Fabric Explorer in your cluster, for example,  
<http://linh1x87d1d.westus.cloudapp.azure.com:19080/Explorer>.
2. Click on the **fabric:/Voting/azurevotefront** node in the tree view and expand the partition node (represented by a GUID). Notice the node name in the treeview, which shows you the nodes that the container is currently running on - for example `_nodetype_4`
3. Expand the **Nodes** node in the tree view. Click on the ellipsis (three dots) next to the node that is running the container.
4. Choose **Restart** to restart that node and confirm the restart action. The restart causes the container to fail over to another node in the cluster.

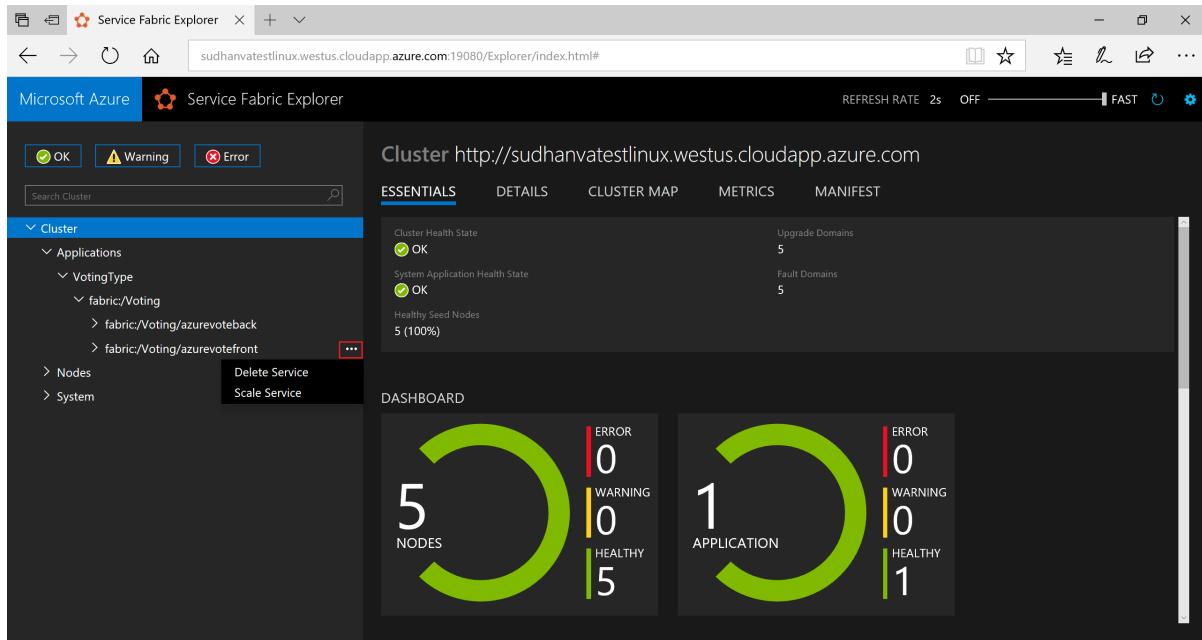


# Scale applications and services in a cluster

Service Fabric services can easily be scaled across a cluster to accommodate for the load on the services. You scale a service by changing the number of instances running in the cluster.

To scale the web front-end service, do the following steps:

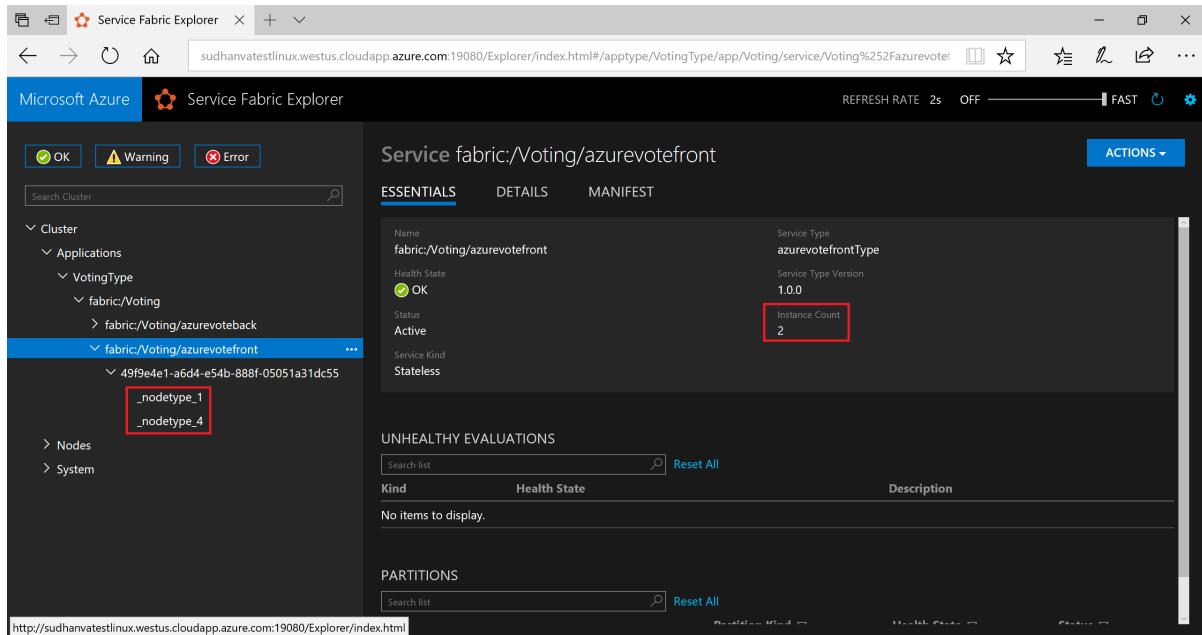
1. Open Service Fabric Explorer in your cluster - for example, <http://linh1x87d1d.westus.cloudapp.azure.com:19080>.
2. Click on the ellipsis (three dots) next to the **fabric:/Voting/azurevotefront** node in the treeview and choose **Scale Service**.



The screenshot shows the Service Fabric Explorer interface. The left sidebar has a tree view under 'Cluster' with 'Applications' expanded, showing 'VotingType' with its sub-nodes: 'fabric:/Voting/azurevoteback' and 'fabric:/Voting/azurevotefront'. A red box highlights the three-dot menu icon next to 'fabric:/Voting/azurevotefront'. The main area is titled 'Cluster http://sudhanvatestlinux.westus.cloudapp.azure.com' and shows the 'ESSENTIALS' tab. It displays cluster health state as 'OK', system application health state as 'OK', and healthy seed nodes at 5 (100%). Below this is a 'DASHBOARD' section with two circular metrics: one for 'NODES' (5 total, 5 healthy) and one for 'APPLICATION' (1 total, 1 healthy). The status bar at the bottom shows the URL as 'http://sudhanvatestlinux.westus.cloudapp.azure.com:19080/Explorer/index.html#'

You can now choose to scale the number of instances of the web front-end service.

3. Change the number to **2** and click **Scale Service**.
4. Click on the **fabric:/Voting/azurevotefront** node in the tree-view and expand the partition node (represented by a GUID).



The screenshot shows the Service Fabric Explorer interface, specifically the details for the 'Service fabric:/Voting/azurevotefront' service. The 'ESSENTIALS' tab is selected. On the right, it shows the service type as 'azrevotefrontType' and version '1.0.0'. The 'Instance Count' field is highlighted with a red box and contains the value '2'. In the 'PARTITIONS' section, there are two partitions listed: '\_nodeType\_1' and '\_nodeType\_4', both of which are highlighted with red boxes. The status bar at the bottom shows the URL as 'http://sudhanvatestlinux.westus.cloudapp.azure.com:19080/Explorer/index.html#'

You can now see that the service has two instances. In the tree view, you can see which nodes the instances run on.

By this simple management task, we doubled the resources available for our front-end service to process user load.

It's important to understand that you do not need multiple instances of a service to have it run reliably. If a service fails, Service Fabric makes sure a new service instance runs in the cluster.

## Clean up

Use the uninstall script provided in the template to delete the application instance from the cluster and unregister the application type. This command takes some time to clean up the instance and the 'install'sh' command should not be run immediately after this script.

```
./uninstall.sh
```

## Next steps

In this quickstart, you learned how to:

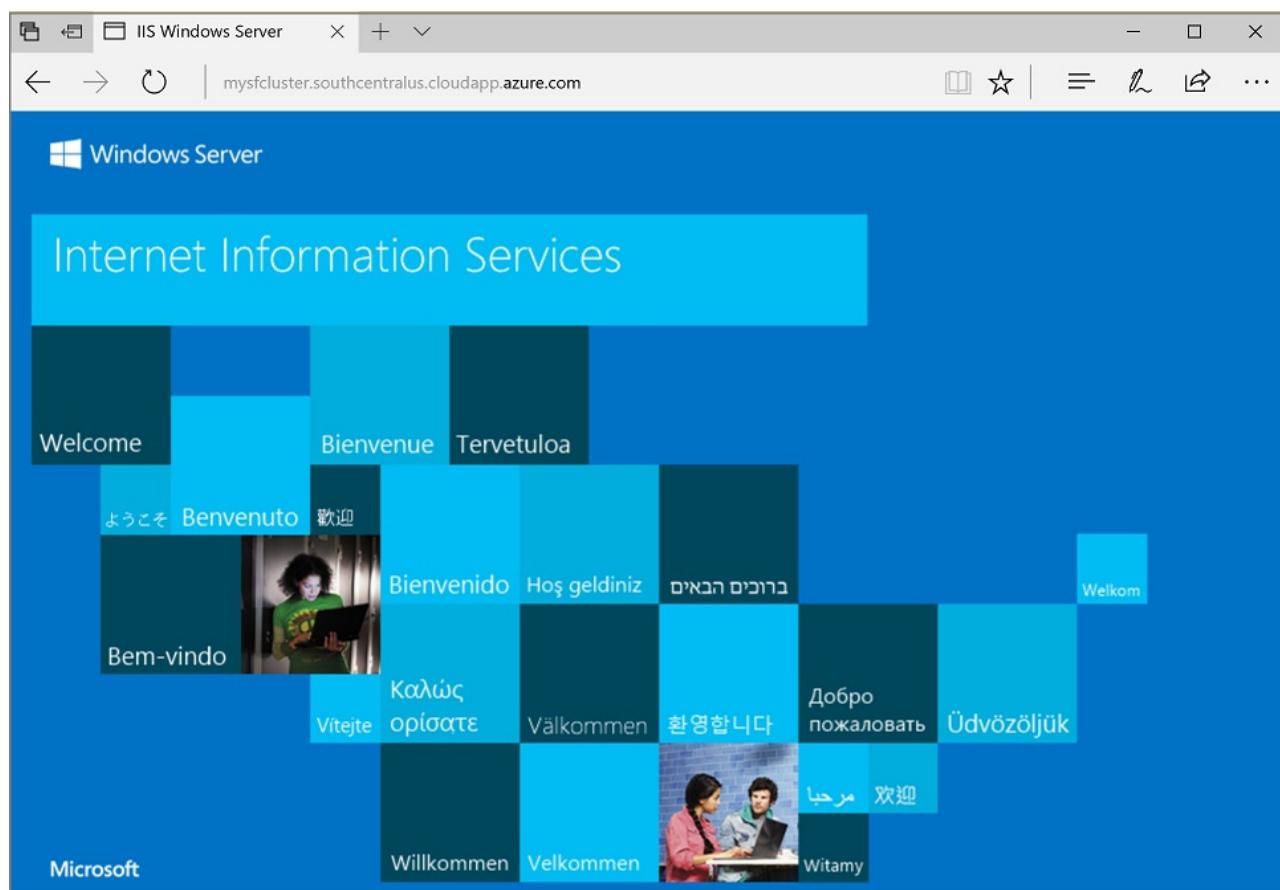
- Deploy a Linux container application to Azure
- Failover a container in a Service Fabric cluster
- Scale a container in a Service Fabric cluster
- Learn more about running [containers on Service Fabric](#).
- Learn about the Service Fabric [application life-cycle](#).
- Check out the [Service Fabric container code samples](#) on GitHub.

# Deploy a Service Fabric Windows container application on Azure

10/3/2017 • 4 min to read • [Edit Online](#)

Azure Service Fabric is a distributed systems platform for deploying and managing scalable and reliable microservices and containers.

Running an existing application in a Windows container on a Service Fabric cluster doesn't require any changes to your application. This quickstart shows you how to deploy a pre-built Docker container image in a Service Fabric application. When you're finished, you'll have a running Windows Server 2016 Nano Server and IIS container. This quickstart describes deploying a Windows container, read [this quickstart](#) to deploy a Linux container.



Using this quickstart you learn how to:

- Package a Docker image container
- Configure communication
- Build and package the Service Fabric application
- Deploy the container application to Azure

## Prerequisites

- An Azure subscription (you can create a [free account](#)).
- A development computer running:
  - Visual Studio 2015 or Visual Studio 2017.
  - [Service Fabric SDK and tools](#).

# Package a Docker image container with Visual Studio

The Service Fabric SDK and tools provide a service template to help you deploy a container to a Service Fabric cluster.

Start Visual Studio as "Administrator". Select **File > New > Project**.

Select **Service Fabric application**, name it "MyFirstContainer", and click **OK**.

Select **Container** from the list of **service templates**.

In **Image Name**, enter "microsoft/iis:nanoserver", the [Windows Server Nano Server and IIS base image](#).

Name your service "MyContainerService", and click **OK**.

## Configure communication and container port-to-host port mapping

The service needs an endpoint for communication. You can now add the protocol, port, and type to an **Endpoint** in the ServiceManifest.xml file. For this quickstart, the containerized service listens on port 80:

```
<Endpoint Name="MyContainerServiceTypeEndpoint" UriScheme="http" Port="80" Protocol="http"/>
```

Providing the **UriScheme** automatically registers the container endpoint with the Service Fabric Naming service for discoverability. A full ServiceManifest.xml example file is provided at the end of this article.

Configure the container port-to-host port mapping by specifying a **PortBinding** policy in **ContainerHostPolicies** of the ApplicationManifest.xml file. For this quickstart, **ContainerPort** is 80 and **EndpointRef** is

"MyContainerServiceTypeEndpoint" (the endpoint defined in the service manifest). Incoming requests to the service on port 80 are mapped to port 80 on the container.

```
<ServiceManifestImport>
...
<ConfigOverrides />
<Policies>
    <ContainerHostPolicies CodePackageRef="Code">
        <PortBinding ContainerPort="80" EndpointRef="MyContainerServiceTypeEndpoint"/>
    </ContainerHostPolicies>
</Policies>
</ServiceManifestImport>
```

A full ApplicationManifest.xml example file is provided at the end of this article.

## Create a cluster

To deploy the application to a cluster in Azure, you can either choose to create your own cluster, or use a party cluster.

Party clusters are free, limited-time Service Fabric clusters hosted on Azure and run by the Service Fabric team where anyone can deploy applications and learn about the platform. To get access to a party cluster, [follow the instructions](#).

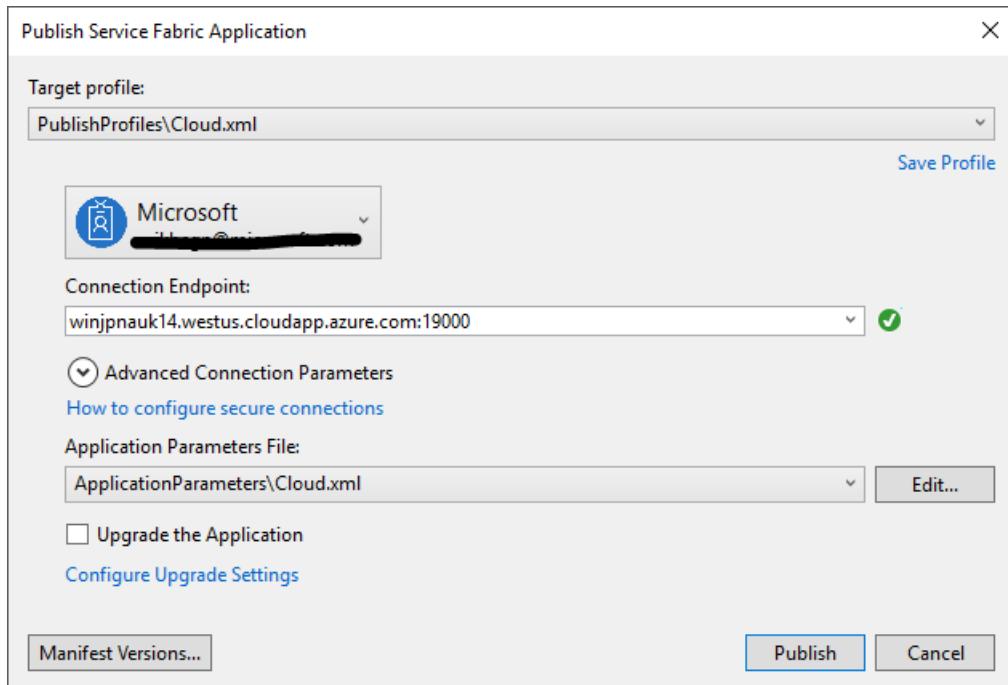
For information about creating your own cluster, see [Create your first Service Fabric cluster on Azure](#).

Take note of the connection endpoint, which you use in the following step.

## Deploy the application to Azure using Visual Studio

Now that the application is ready, you can deploy it to a cluster directly from Visual Studio.

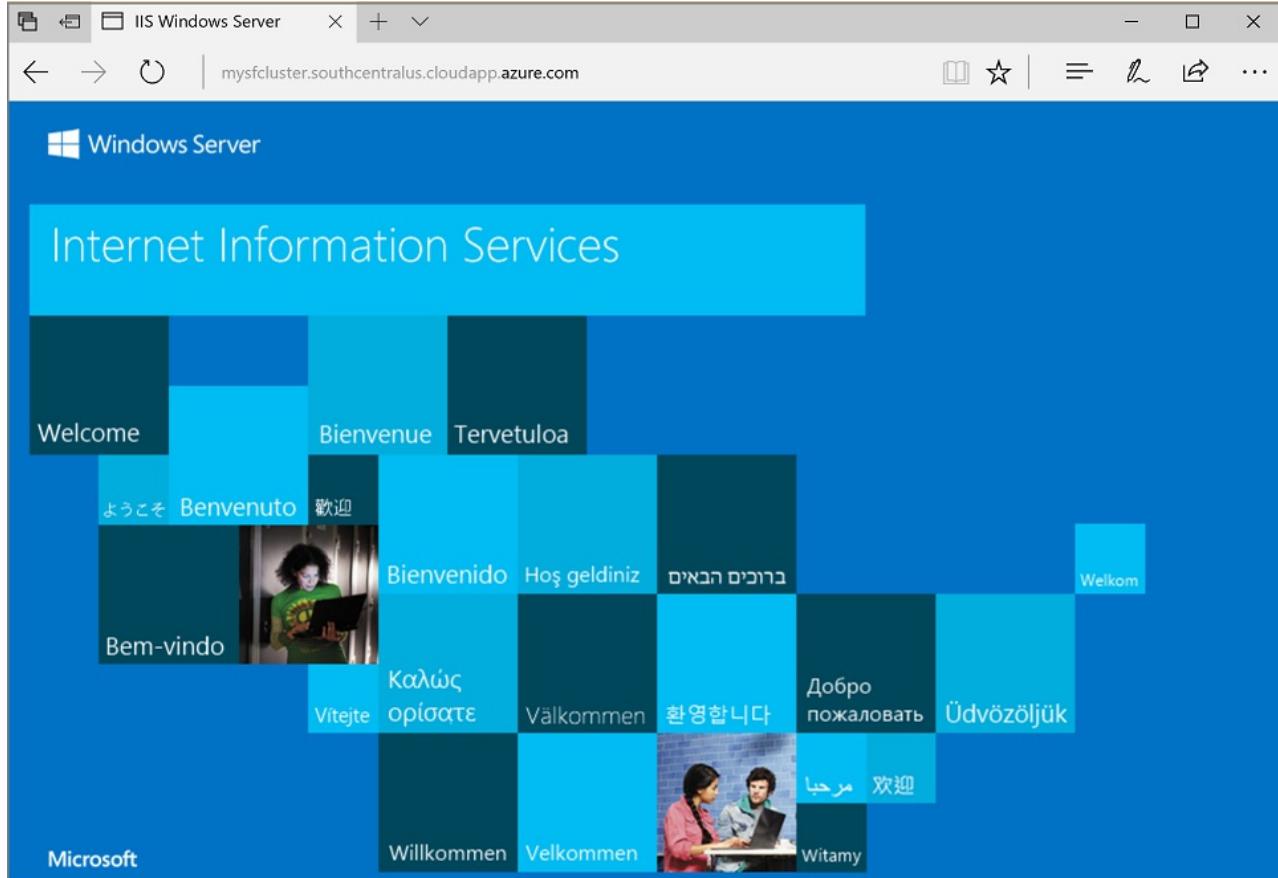
Right-click **MyFirstContainer** in the Solution Explorer and choose **Publish**. The Publish dialog appears.



Type in the connection endpoint of the cluster in the **Connection Endpoint** field. When signing up for the party cluster, the connection endpoint is provided in the browser - for example,

`winh1x87d1d.westus.cloudapp.azure.com:19000`. Click **Publish** and the application deploys.

Open a browser and navigate to <http://winh1x87d1d.westus.cloudapp.azure.com:80>. You should see the IIS default web page:



## Complete example Service Fabric application and service manifests

Here are the complete service and application manifests used in this quickstart.

## ServiceManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceManifest Name="MyContainerServicePkg"
    Version="1.0.0"
    xmlns="http://schemas.microsoft.com/2011/01/fabric"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <ServiceTypes>
        <!-- This is the name of your ServiceType.
            The UseImplicitHost attribute indicates this is a guest service. -->
        <StatelessServiceType ServiceTypeName="MyContainerServiceType" UseImplicitHost="true" />
    </ServiceTypes>

        <!-- Code package is your service executable. -->
    <CodePackage Name="Code" Version="1.0.0">
        <EntryPoint>
            <!-- Follow this link for more information about deploying Windows containers to Service Fabric:
            https://aka.ms/sfguestcontainers -->
            <ContainerHost>
                <ImageName>microsoft/iis:nanoserver</ImageName>
            </ContainerHost>
        </EntryPoint>
        <!-- Pass environment variables to your container: -->
        <!--
        <EnvironmentVariables>
            <EnvironmentVariable Name="VariableName" Value="VariableValue"/>
        </EnvironmentVariables>
        -->
    </CodePackage>

        <!-- Config package is the contents of the Config directory under PackageRoot that contains an
            independently-updateable and versioned set of custom configuration settings for your service. -->
    <ConfigPackage Name="Config" Version="1.0.0" />

    <Resources>
        <Endpoints>
            <!-- This endpoint is used by the communication listener to obtain the port on which to
                listen. Please note that if your service is partitioned, this port is shared with
                replicas of different partitions that are placed in your code. -->
            <Endpoint Name="MyContainerServiceTypeEndpoint" UriScheme="http" Port="80" Protocol="http"/>
        </Endpoints>
    </Resources>
</ServiceManifest>
```

## ApplicationManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<ApplicationManifest ApplicationTypeName="MyFirstContainerType"
    ApplicationTypeVersion="1.0.0"
    xmlns="http://schemas.microsoft.com/2011/01/fabric"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <Parameters>
        <Parameter Name="MyContainerService_InstanceCount" DefaultValue="-1" />
    </Parameters>
    <!-- Import the ServiceManifest from the ServicePackage. The ServiceManifestName and ServiceManifestVersion
        should match the Name and Version attributes of the ServiceManifest element defined in the
        ServiceManifest.xml file. -->
    <ServiceManifestImport>
        <ServiceManifestRef ServiceManifestName="MyContainerServicePkg" ServiceManifestVersion="1.0.0" />
        <ConfigOverrides />
        <Policies>
            <ContainerHostPolicies CodePackageRef="Code">
                <PortBinding ContainerPort="80" EndpointRef="MyContainerServiceTypeEndpoint"/>
            </ContainerHostPolicies>
        </Policies>
    </ServiceManifestImport>
    <DefaultServices>
        <!-- The section below creates instances of service types, when an instance of this
            application type is created. You can also create one or more instances of service type using the
            ServiceFabric PowerShell module.

            The attribute ServiceTypeName below must match the name defined in the imported ServiceManifest.xml
            file. -->
        <Service Name="MyContainerService" ServicePackageActivationMode="ExclusiveProcess">
            <StatelessService ServiceTypeName="MyContainerServiceType" InstanceCount="
[MyContainerService_InstanceCount]">
                <SingletonPartition />
            </StatelessService>
        </Service>
    </DefaultServices>
</ApplicationManifest>

```

## Next steps

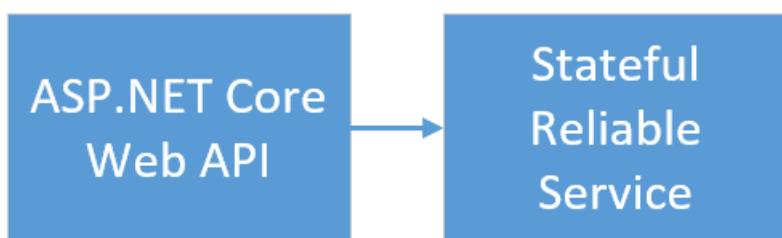
In this quickstart, you learned how to:

- Package a Docker image container
- Configure communication
- Build and package the Service Fabric application
- Deploy the container application to Azure
  
- Learn more about running [containers on Service Fabric](#).
- Read the [Deploy a .NET application in a container](#) tutorial.
- Learn about the Service Fabric [application life-cycle](#).
- Checkout the [Service Fabric container code samples](#) on GitHub.

# Create and deploy an application with an ASP.NET Core Web API front-end service and a stateful back-end service

10/30/2017 • 11 min to read • [Edit Online](#)

This tutorial is part one of a series. You will learn how to create an Azure Service Fabric application with an ASP.NET Core Web API front end and a stateful back-end service to store your data. When you're finished, you have a voting application with an ASP.NET Core web front-end that saves voting results in a stateful back-end service in the cluster. If you don't want to manually create the voting application, you can [download the source code](#) for the completed application and skip ahead to [Walk through the voting sample application](#).



In part one of the series, you learn how to:

- Create an ASP.NET Core Web API service as a stateful reliable service
- Create an ASP.NET Core Web Application service as a stateless web service
- Use the reverse proxy to communicate with the stateful service

In this tutorial series you learn how to:

- Build a .NET Service Fabric application
- [Deploy the application to a remote cluster](#)
- [Configure CI/CD using Visual Studio Team Services](#)
- [Set up monitoring and diagnostics for the application](#)

## Prerequisites

Before you begin this tutorial:

- If you don't have an Azure subscription, create a [free account](#)
- [Install Visual Studio 2017](#) and install the **Azure development** and **ASP.NET and web development** workloads.
- [Install the Service Fabric SDK](#)

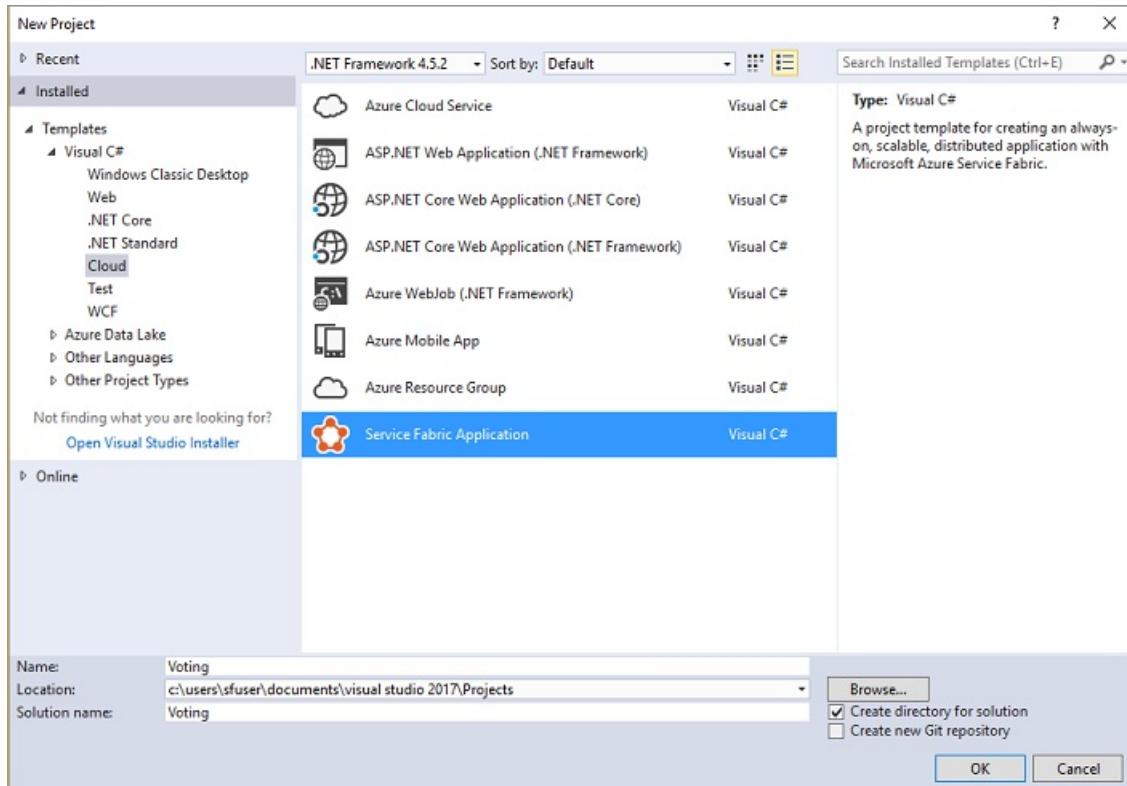
## Create an ASP.NET Web API service as a reliable service

First, create the web front-end of the voting application using ASP.NET Core. ASP.NET Core is a lightweight, cross-platform web development framework that you can use to create modern web UI and web APIs. To get a complete understanding of how ASP.NET Core integrates with Service Fabric, we strongly recommend reading through the [ASP.NET Core in Service Fabric Reliable Services](#) article. For now, you can follow this tutorial to get started quickly. To learn more about ASP.NET Core, see the [ASP.NET Core Documentation](#).

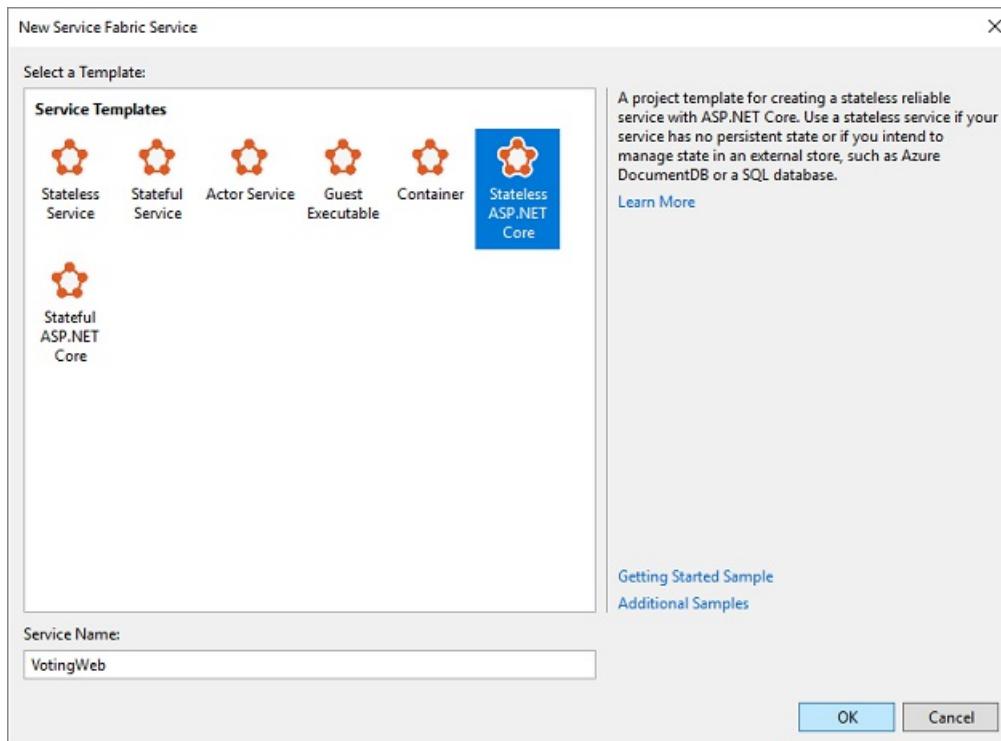
#### NOTE

This tutorial is based on the [ASP.NET Core tools for Visual Studio 2017](#). The .NET Core tools for Visual Studio 2015 are no longer being updated.

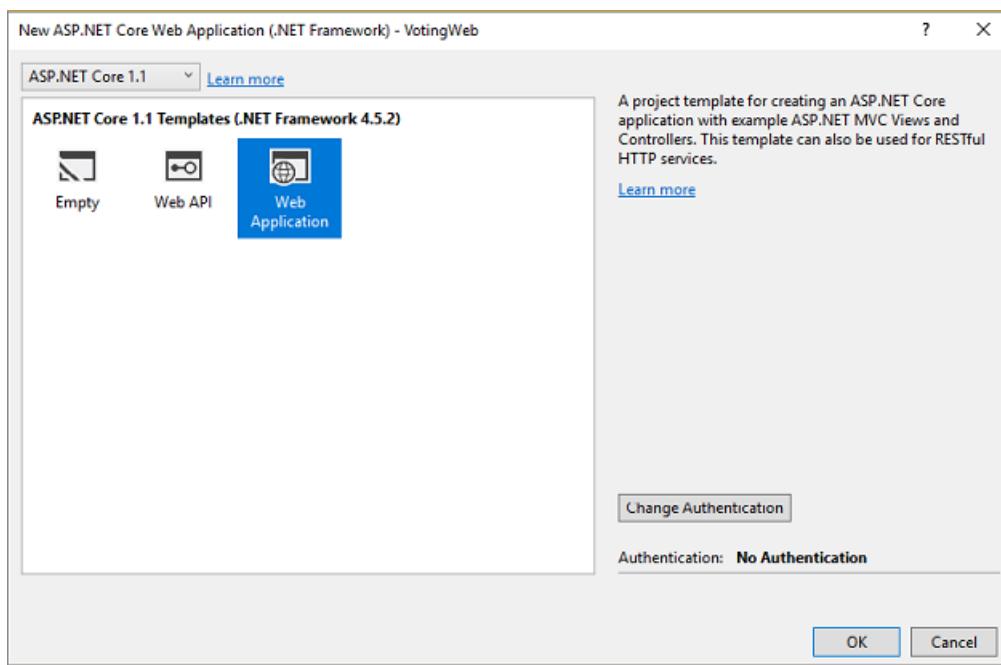
1. Launch Visual Studio as an **administrator**.
2. Create a project with **File->New->Project**
3. In the **New Project** dialog, choose **Cloud > Service Fabric Application**.
4. Name the application **Voting** and press **OK**.



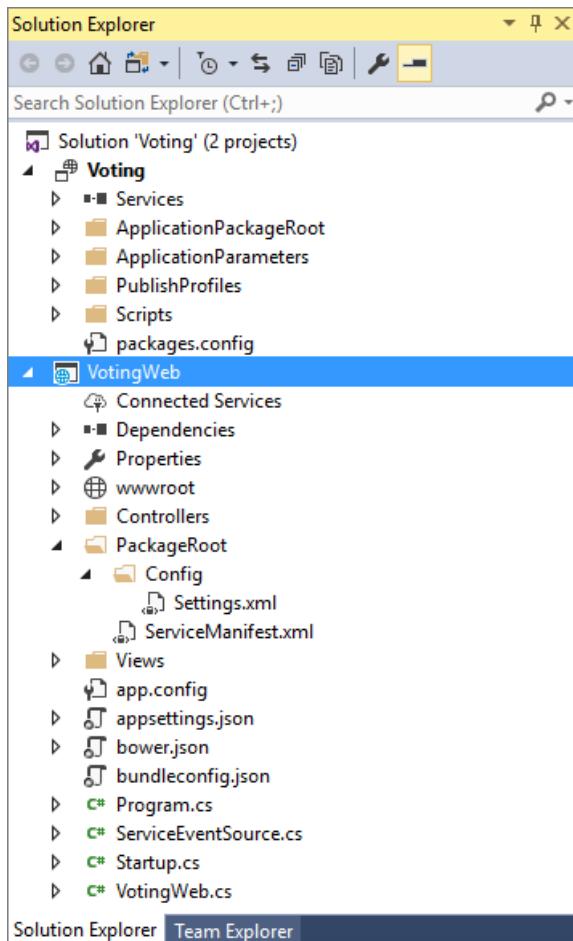
5. On the **New Service Fabric Service** page, choose **Stateless ASP.NET Core**, and name your service **VotingWeb**.



6. The next page provides a set of ASP.NET Core project templates. For this tutorial, choose **Web Application**.



Visual Studio creates an application and a service project and displays them in Solution Explorer.



## Add AngularJS to the VotingWeb service

Add [AngularJS](#) to your service using the built-in [Bower support](#). Open `bower.json` and add entries for angular and angular-bootstrap, then save your changes.

```
{
  "name": "asp.net",
  "private": true,
  "dependencies": {
    "bootstrap": "3.3.7",
    "jquery": "2.2.0",
    "jquery-validation": "1.14.0",
    "jquery-validation-unobtrusive": "3.2.6",
    "angular": "v1.6.5",
    "angular-bootstrap": "v1.1.0"
  }
}
```

Upon saving the `bower.json` file, Angular is installed in your project's `wwwroot/lib` folder. Additionally, it is listed within the `Dependencies/Bower` folder.

## Update the site.js file

Open the `wwwroot/js/site.js` file. Replace its contents with the JavaScript used by the Home views:

```
var app = angular.module('VotingApp', ['ui.bootstrap']);
app.run(function () { });

app.controller('VotingAppController', ['$rootScope', '$scope', '$http', '$timeout', function ($rootScope,
$scope, $http, $timeout) {

    $scope.refresh = function () {
        $http.get('api/Votes?c=' + new Date().getTime())
            .then(function (data, status) {
                $scope.votes = data;
            }, function (data, status) {
                $scope.votes = undefined;
            });
    };

    $scope.remove = function (item) {
        $http.delete('api/Votes/' + item)
            .then(function (data, status) {
                $scope.refresh();
            })
    };

    $scope.add = function (item) {
        var fd = new FormData();
        fd.append('item', item);
        $http.put('api/Votes/' + item, fd, {
            transformRequest: angular.identity,
            headers: { 'Content-Type': undefined }
        })
            .then(function (data, status) {
                $scope.refresh();
                $scope.item = undefined;
            })
    };
}]);
```

### Update the Index.cshtml file

Open the *Views/Home/Index.cshtml* file, the view specific to the Home controller. Replace its contents with the following, then save your changes.

```

@{
    ViewData["Title"] = "Service Fabric Voting Sample";
}

<div ng-controller="VotingAppController" ng-init="refresh()">
    <div class="container-fluid">
        <div class="row">
            <div class="col-xs-8 col-xs-offset-2 text-center">
                <h2>Service Fabric Voting Sample</h2>
            </div>
        </div>

        <div class="row">
            <div class="col-xs-8 col-xs-offset-2">
                <form class="col-xs-12 center-block">
                    <div class="col-xs-6 form-group">
                        <input id="txtAdd" type="text" class="form-control" placeholder="Add voting option"
ng-model="item" />
                    </div>
                    <button id="btnAdd" class="btn btn-default" ng-click="add(item)">
                        <span class="glyphicon glyphicon-plus" aria-hidden="true"></span>
                        Add
                    </button>
                </form>
            </div>
        </div>
    </div>

    <hr />

    <div class="row">
        <div class="col-xs-8 col-xs-offset-2">
            <div class="row">
                <div class="col-xs-4">
                    Click to vote
                </div>
            </div>
            <div class="row top-buffer" ng-repeat="vote in votes.data">
                <div class="col-xs-8">
                    <button class="btn btn-success text-left btn-block" ng-click="add(vote.key)">
                        <span class="pull-left">
                            {{vote.key}}
                        </span>
                        <span class="badge pull-right">
                            {{vote.value}} Votes
                        </span>
                    </button>
                </div>
                <div class="col-xs-4">
                    <button class="btn btn-danger pull-right btn-block" ng-click="remove(vote.key)">
                        <span class="glyphicon glyphicon-remove" aria-hidden="true"></span>
                        Remove
                    </button>
                </div>
            </div>
        </div>
    </div>
</div>

```

## Update the \_Layout.cshtml file

Open the *Views/Shared/\_Layout.cshtml* file, the default layout for the ASP.NET app. Replace its contents with the following, then save your changes.

```

<!DOCTYPE html>
<html ng-app="VotingApp" xmlns:ng="http://angularjs.org">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"]</title>

    <link href="~/lib/bootstrap/dist/css/bootstrap.min.css" rel="stylesheet" />
    <link href="~/css/site.css" rel="stylesheet" />

</head>
<body>
    <div class="container body-content">
        @RenderBody()
    </div>

    <script src="~/lib/jquery/dist/jquery.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>
    <script src="~/lib/angular/angular.js"></script>
    <script src="~/lib/angular-bootstrap/ui-bootstrap-tpls.js"></script>
    <script src="~/js/site.js"></script>

    @RenderSection("Scripts", required: false)
</body>
</html>

```

## Update the VotingWeb.cs file

Open the *VotingWeb.cs* file, which creates the ASP.NET Core WebHost inside the stateless service using the WebListener web server. Add the `using System.Net.Http;` directive to the top of the file. Replace the `CreateServiceInstanceListeners()` function with the following, then save your changes.

```

protected override IEnumerable<ServiceInstanceListener> CreateServiceInstanceListeners()
{
    return new ServiceInstanceListener[]
    {
        new ServiceInstanceListener(serviceContext =>
            new WebListenerCommunicationListener(serviceContext, "ServiceEndpoint", (url, listener) =>
            {
                ServiceEventSource.Current.ServiceMessage(serviceContext, $"Starting WebListener on {url}");

                return new WebHostBuilder().UseWebListener()
                    .ConfigureServices(
                        services => services
                            .AddSingleton<StatelessServiceContext>(serviceContext)
                            .AddSingleton<HttpClient>()
                            .UseContentRoot(Directory.GetCurrentDirectory())
                            .UseStartup<Startup>()
                            .UseApplicationInsights()
                            .UseServiceFabricIntegration(listener, ServiceFabricIntegrationOptions.None)
                            .UseUrls(url)
                            .Build();
            })
    };
}

```

## Add the VotesController.cs file

Add a controller which defines voting actions. Right-click on the **Controllers** folder, then select **Add->New item->Class**. Name the file "VotesController.cs" and click **Add**. Replace the file contents with the following, then save your changes. Later, in [Update the VotesController.cs file](#), this file will be modified to read and write voting data from the back-end service. For now, the controller returns static string data to the view.

```

using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using Newtonsoft.Json;
using System.Text;
using System.Net.Http;
using System.Net.Http.Headers;

namespace VotingWeb.Controllers
{
    [Produces("application/json")]
    [Route("api/Votes")]
    public class VotesController : Controller
    {
        private readonly HttpClient httpClient;

        public VotesController(HttpClient httpClient)
        {
            this.httpClient = httpClient;
        }

        // GET: api/Votes
        [HttpGet]
        public async Task<IActionResult> Get()
        {
            List<KeyValuePair<string, int>> votes= new List<KeyValuePair<string, int>>();
            votes.Add(new KeyValuePair<string, int>("Pizza", 3));
            votes.Add(new KeyValuePair<string, int>("Ice cream", 4));

            return Json(votes);
        }
    }
}

```

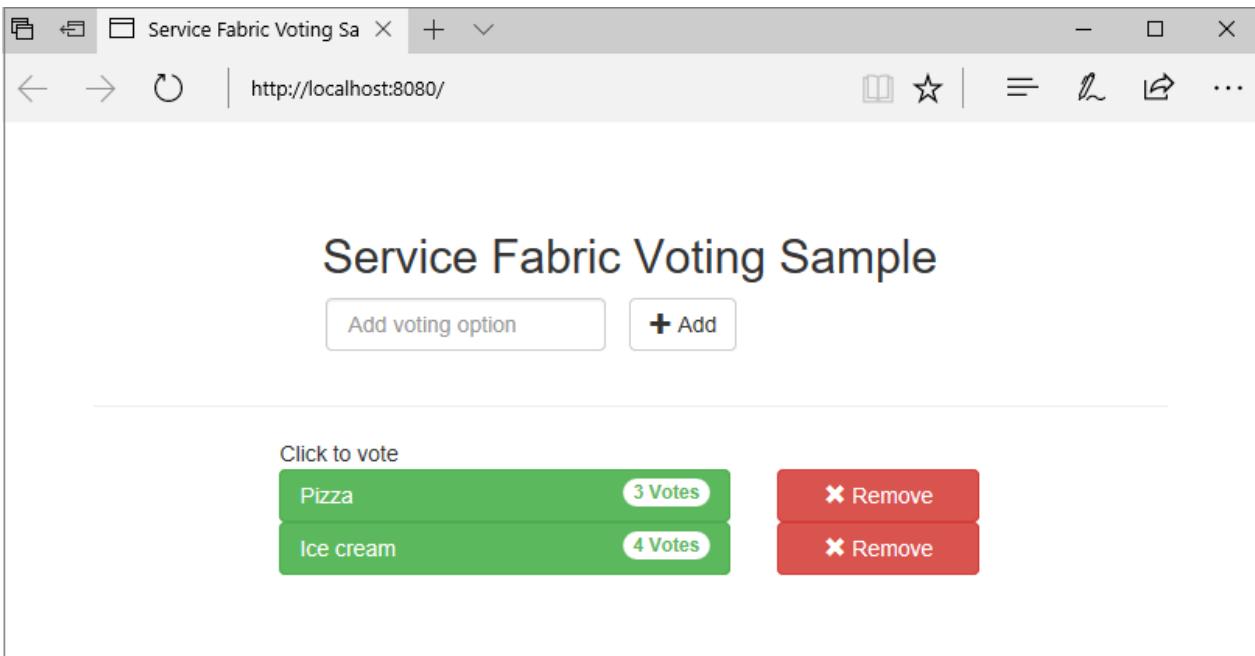
## Deploy and run the application locally

You can now go ahead and run the application. In Visual Studio, press **F5** to deploy the application for debugging. **F5** fails if you didn't previously open Visual Studio as **administrator**.

### NOTE

The first time you run and deploy the application locally, Visual Studio creates a local cluster for debugging. Cluster creation may take some time. The cluster creation status is displayed in the Visual Studio output window.

At this point, your web app should look like this:



To stop debugging the application, go back to Visual Studio and press **Shift+F5**.

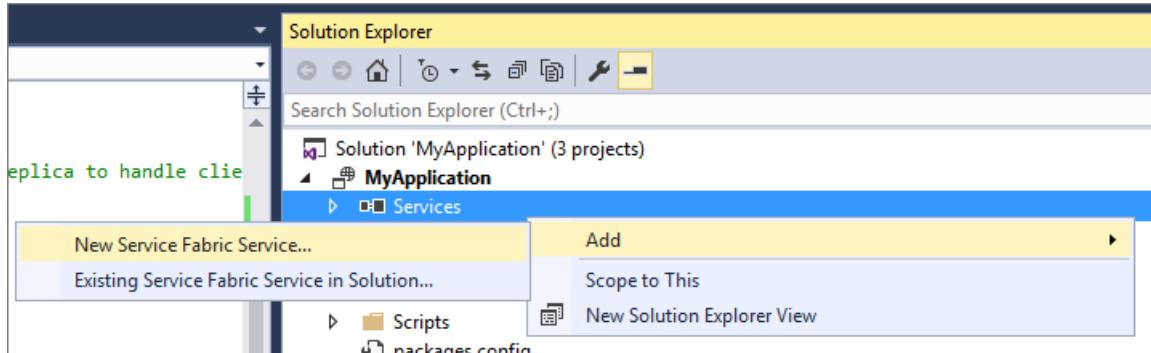
## Add a stateful back-end service to your application

Now that we have an ASP.NET Web API service running in our application, let's go ahead and add a stateful reliable service to store some data in our application.

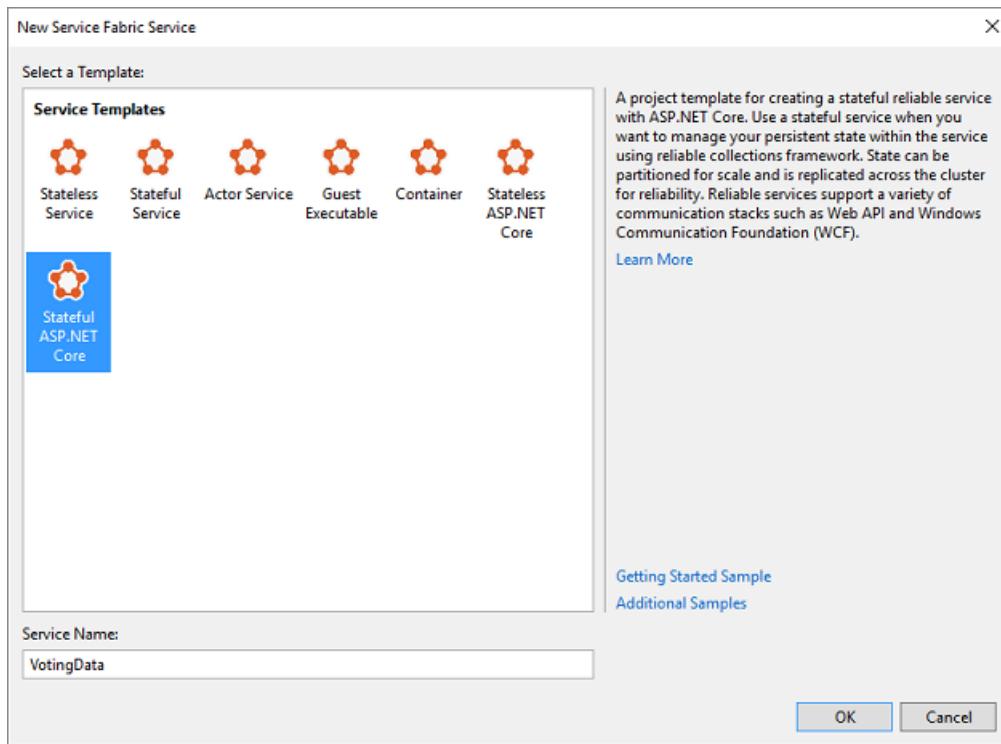
Service Fabric allows you to consistently and reliably store your data right inside your service by using reliable collections. Reliable collections are a set of highly available and reliable collection classes that are familiar to anyone who has used C# collections.

In this tutorial, you create a service which stores a counter value in a reliable collection.

1. In Solution Explorer, right-click **Services** within the application project and choose **Add > New Service Fabric Service**.

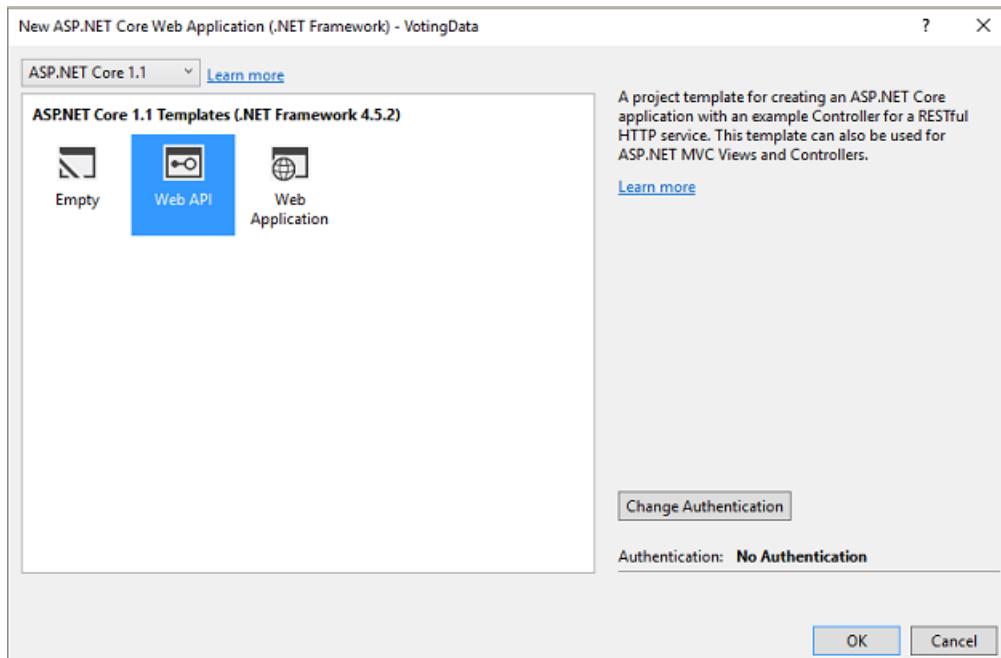


2. In the **New Service Fabric Service** dialog, choose **Stateful ASP.NET Core**, and name the service **VotingData** and press **OK**.



Once your service project is created, you have two services in your application. As you continue to build your application, you can add more services in the same way. Each can be independently versioned and upgraded.

3. The next page provides a set of ASP.NET Core project templates. For this tutorial, choose **Web API**.



Visual Studio creates a service project and displays them in Solution Explorer.

Solution Explorer

The Solution Explorer window displays three projects:

- Voting**: Contains Services, ApplicationPackageRoot, ApplicationParameters, PublishProfiles, Scripts, and packages.config.
- VotingData**: Selected project. Contains Connected Services, Dependencies, Properties, wwwroot, Controllers, and PackageRoot.
  - PackageRoot**: Contains Config, Settings.xml, ServiceManifest.xml, appsettings.json, Program.cs, ServiceEventSource.cs, Startup.cs, and VotingData.cs.
- VotingWeb**: Contains files such as index.cshtml, Home.cshtml, and a JavaScript file.

#### Add the VoteDataController.cs file

In the **VotingData** project right-click on the **Controllers** folder, then select **Add->New item->Class**. Name the file "VoteDataController.cs" and click **Add**. Replace the file contents with the following, then save your changes.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.ServiceFabric.Data;
using System.Threading;
using Microsoft.ServiceFabric.Data.Collections;
```

```

namespace VotingData.Controllers
{
    [Route("api/[controller]")]
    public class VoteDataController : Controller
    {
        private readonly IReliableStateManager stateManager;

        public VoteDataController(IReliableStateManager stateManager)
        {
            this.stateManager = stateManager;
        }

        // GET api/VoteData
        [HttpGet]
        public async Task<IActionResult> Get()
        {
            var ct = new CancellationToken();

            var votesDictionary = await this.stateManager.GetOrAddAsync<IReliableDictionary<string, int>>
("counts");

            using (ITransaction tx = this.stateManager.CreateTransaction())
            {
                var list = await votesDictionary.CreateEnumerableAsync(tx);

                var enumerator = list.GetAsyncEnumerator();

                var result = new List<KeyValuePair<string, int>>();

                while (await enumerator.MoveNextAsync(ct))
                {
                    result.Add(enumerator.Current);
                }

                return Json(result);
            }
        }

        // PUT api/VoteData/name
        [HttpPut("{name}")]
        public async Task<IActionResult> Put(string name)
        {
            var votesDictionary = await this.stateManager.GetOrAddAsync<IReliableDictionary<string, int>>
("counts");

            using (ITransaction tx = this.stateManager.CreateTransaction())
            {
                await votesDictionary.AddOrUpdateAsync(tx, name, 1, (key, oldvalue) => oldvalue + 1);
                await tx.CommitAsync();
            }

            return new OkResult();
        }

        // DELETE api/VoteData/name
        [HttpDelete("{name}")]
        public async Task<IActionResult> Delete(string name)
        {
            var votesDictionary = await this.stateManager.GetOrAddAsync<IReliableDictionary<string, int>>
("counts");

            using (ITransaction tx = this.stateManager.CreateTransaction())
            {
                if (await votesDictionary.ContainsKeyAsync(tx, name))
                {
                    await votesDictionary.TryRemoveAsync(tx, name);
                    await tx.CommitAsync();
                    return new OkResult();
                }
            }
        }
    }
}

```

```
        return new NotFoundResult();
    }
}
}
```

## Connect the services

In this next step, we will connect the two services and make the front-end Web application get and set voting information from the back-end service.

Service Fabric provides complete flexibility in how you communicate with reliable services. Within a single application, you might have services that are accessible via TCP. Other services that might be accessible via an HTTP REST API and still other services could be accessible via web sockets. For background on the options available and the tradeoffs involved, see [Communicating with services](#).

In this tutorial, we are using [ASP.NET Core Web API](#).

## Update the VotesController.cs file

In the **VotingWeb** project, open the *Controllers/VotesController.cs* file. Replace the `VotesController` class definition contents with the following, then save your changes.

```
public class VotesController : Controller
{
    private readonly HttpClient httpClient;
    string serviceProxyUrl = "http://localhost:19081/Voting/VotingData/api/VoteData";
    string partitionKind = "Int64Range";
    string partitionKey = "0";

    public VotesController(HttpClient httpClient)
    {
        this.httpClient = httpClient;
    }

    // GET: api/Votes
    [HttpGet]
    public async Task<IActionResult> Get()
    {
        IEnumerable<KeyValuePair<string, int>> votes;

        HttpResponseMessage response = await this.httpClient.GetAsync($"{serviceProxyUrl}?PartitionKind={partitionKind}&PartitionKey={partitionKey}");

        if (response.StatusCode != System.Net.HttpStatusCode.OK)
        {
            return StatusCode((int)response.StatusCode);
        }

        votes = JsonConvert.DeserializeObject<List<KeyValuePair<string, int>>>(await
response.Content.ReadAsStringAsync());

        return Json(votes);
    }

    // PUT: api/Votes/name
    [HttpPut("{name}")]
    public async Task<IActionResult> Put(string name)
    {
        string payload = $"{{ 'name' : '{name}' }}";
    }
}
```

```

        StringContent putContent = new StringContent(payload, Encoding.UTF8, "application/json");
        putContent.Headers.ContentType = new MediaTypeHeaderValue("application/json");

        string proxyUrl = $"{serviceProxyUrl}/{name}?PartitionKind={partitionKind}&PartitionKey={partitionKey}";

        HttpResponseMessage response = await this.httpClient.PutAsync(proxyUrl, putContent);

        return new ContentResult()
        {
            StatusCode = (int)response.StatusCode,
            Content = await response.Content.ReadAsStringAsync()
        };
    }

    // DELETE: api/Votes/name
    [HttpDelete("{name}")]
    public async Task<IActionResult> Delete(string name)
    {
        HttpResponseMessage response = await this.httpClient.DeleteAsync($"{serviceProxyUrl}/{name}?PartitionKind={partitionKind}&PartitionKey={partitionKey}");

        if (response.StatusCode != System.Net.HttpStatusCode.OK)
        {
            return this.StatusCode((int)response.StatusCode);
        }

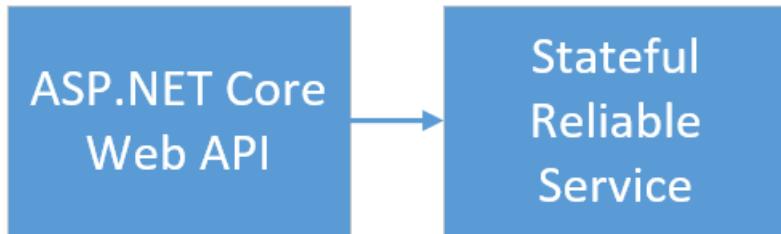
        return new OkResult();
    }
}

```

## Walk through the voting sample application

The voting application consists of two services:

- Web front-end service (VotingWeb)- An ASP.NET Core web front-end service, which serves the web page and exposes web APIs to communicate with the backend service.
- Back-end service (VotingData)- An ASP.NET Core web service, which exposes an API to store the vote results in a reliable dictionary persisted on disk.



When you vote in the application the following events occur:

1. A JavaScript sends the vote request to the web API in the web front-end service as an HTTP PUT request.
2. The web front-end service uses a proxy to locate and forward an HTTP PUT request to the back-end service.
3. The back-end service takes the incoming request, and stores the updated result in a reliable dictionary, which gets replicated to multiple nodes within the cluster and persisted on disk. All the application's data is stored in the cluster, so no database is needed.

## Debug in Visual Studio

When debugging application in Visual Studio, you are using a local Service Fabric development cluster. You have the option to adjust your debugging experience to your scenario. In this application, we store data in our back-end service, using a reliable dictionary. Visual Studio removes the application per default when you stop the debugger. Removing the application causes the data in the back-end service to also be removed. To persist the data between debugging sessions, you can change the **Application Debug Mode** as a property on the **Voting** project in Visual Studio.

To look at what happens in the code, complete the following steps:

1. Open the **VotesController.cs** file and set a breakpoint in the web API's **Put** method (line 47) - You can search for the file in the Solution Explorer in Visual Studio.
2. Open the **VoteDataController.cs** file and set a breakpoint in this web API's **Put** method (line 50).
3. Go back to the browser and click a voting option or add a new voting option. You hit the first breakpoint in the web front-end's api controller.
- a. This is where the JavaScript in the browser sends a request to the web API controller in the front-end service.

```
public async Task<IActionResult> Put(string name)
{
    string payload = $"{{ 'name' : '{name}' }}";
    StringContent putContent = new StringContent(payload, Encoding.UTF8, "application/json");
    putContent.Headers.ContentType = new MediaTypeHeaderValue("application/json");

    1 string proxyUrl = $"{serviceProxyUrl}/{name}?PartitionKind={partitionKind}&PartitionKey={partitionKey}";
    2 HttpResponseMessage response = await this.httpClient.PutAsync(proxyUrl, putContent);

    return new ContentResult()
    3 {
        StatusCode = (int)response.StatusCode,
        Content = await response.Content.ReadAsStringAsync()
    };
}
```

- b. First we construct the URL to the ReverseProxy for our back-end service **(1)**.
- c. Then we send the HTTP PUT Request to the ReverseProxy **(2)**.
- d. Finally the we return the response from the back-end service to the client **(3)**.

4. Press **F5** to continue
- a. You are now at the break point in the back-end service.

```
public async Task<IActionResult> Put(string name)
{
    1 var votesDictionary = await this.stateManager.GetOrAddAsync<IReliableDictionary<string, int>>("counts");
    2 using (ITransaction tx = this.stateManager.CreateTransaction())
    {
        3 await votesDictionary.AddOrUpdateAsync(tx, name, 1, (key, oldvalue) => oldvalue + 1);
        await tx.CommitAsync();
    }

    return new OkResult();
}
```

- b. In the first line in the method **(1)** we are using the **StateManager** to get or add a reliable dictionary called **counts**.
- c. All interactions with values in a reliable dictionary require a transaction, this using statement **(2)** creates that transaction.
- d. In the transaction, we then update the value of the relevant key for the voting option and commits the operation **(3)**. Once the commit method returns, the data is updated in the dictionary and replicated to other nodes in the cluster. The data is now safely stored in the cluster, and the back-end service can fail over to other nodes, still having the data available.

5. Press **F5** to continue

To stop the debugging session, press **Shift+F5**.

## Next steps

In this part of the tutorial, you learned how to:

- Create an ASP.NET Core Web API service as a stateful reliable service
- Create an ASP.NET Core Web Application service as a stateless web service
- Use the reverse proxy to communicate with the stateful service

Advance to the next tutorial:

[Deploy the application to Azure](#)

# Deploy an application to a Party Cluster in Azure

9/20/2017 • 4 min to read • [Edit Online](#)

This tutorial is part two of a series and shows you how to deploy an Azure Service Fabric application to a Party Cluster in Azure.

In part two of the tutorial series, you learn how to:

- Deploy an application to a remote cluster using Visual Studio
- Remove an application from a cluster using Service Fabric Explorer

In this tutorial series you learn how to:

- [Build a .NET Service Fabric application](#)
- Deploy the application to a remote cluster
- [Configure CI/CD using Visual Studio Team Services](#)
- [Set up monitoring and diagnostics for the application](#)

## Prerequisites

Before you begin this tutorial:

- If you don't have an Azure subscription, create a [free account](#)
- [Install Visual Studio 2017](#) and install the **Azure development** and **ASP.NET and web development** workloads.
- [Install the Service Fabric SDK](#)

## Download the Voting sample application

If you did not build the Voting sample application in [part one of this tutorial series](#), you can download it. In a command window, run the following command to clone the sample app repository to your local machine.

```
git clone https://github.com/Azure-Samples/service-fabric-dotnet-quickstart
```

## Set up a Party Cluster

Party clusters are free, limited-time Service Fabric clusters hosted on Azure and run by the Service Fabric team where anyone can deploy applications and learn about the platform. For free!

To get access to a Party Cluster, browse to this site: <http://aka.ms/tryservicefabric> and follow the instructions to get access to a cluster. You need a Facebook or GitHub account to get access to a Party Cluster.

You can use your own cluster instead of the Party Cluster, if you want. The ASP.NET core web front-end uses the reverse proxy to communicate with the stateful service back-end. Party Clusters and the local development cluster have reverse proxy enabled by default. If you deploy the Voting sample application to your own cluster, you must [enable the reverse proxy in the cluster](#).

#### NOTE

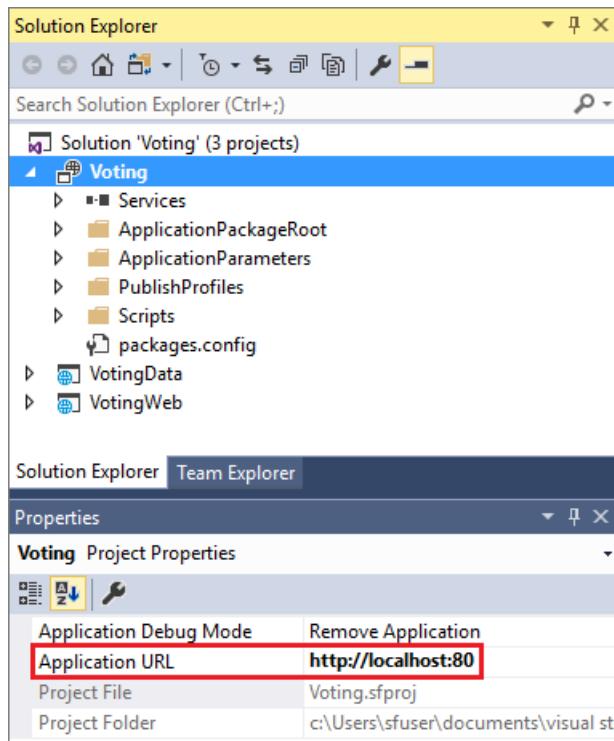
Party clusters are not secured, so your applications and any data you put in them may be visible to others. Don't deploy anything you don't want others to see. Be sure to read over our Terms of Use for all the details.

## Configure the listening port

When the VotingWeb front-end service is created, Visual Studio randomly selects a port for the service to listen on. The VotingWeb service acts as the front-end for this application and accepts external traffic, so let's bind that service to a fixed and well-known port. In Solution Explorer, open *VotingWeb/PackageRoot/ServiceManifest.xml*. Find the **Endpoint** resource in the **Resources** section and change the **Port** value to 80.

```
<Resources>
  <Endpoints>
    <!-- This endpoint is used by the communication listener to obtain the port on which to
        listen. Please note that if your service is partitioned, this port is shared with
        replicas of different partitions that are placed in your code. -->
    <Endpoint Protocol="http" Name="ServiceEndpoint" Type="Input" Port="80" />
  </Endpoints>
</Resources>
```

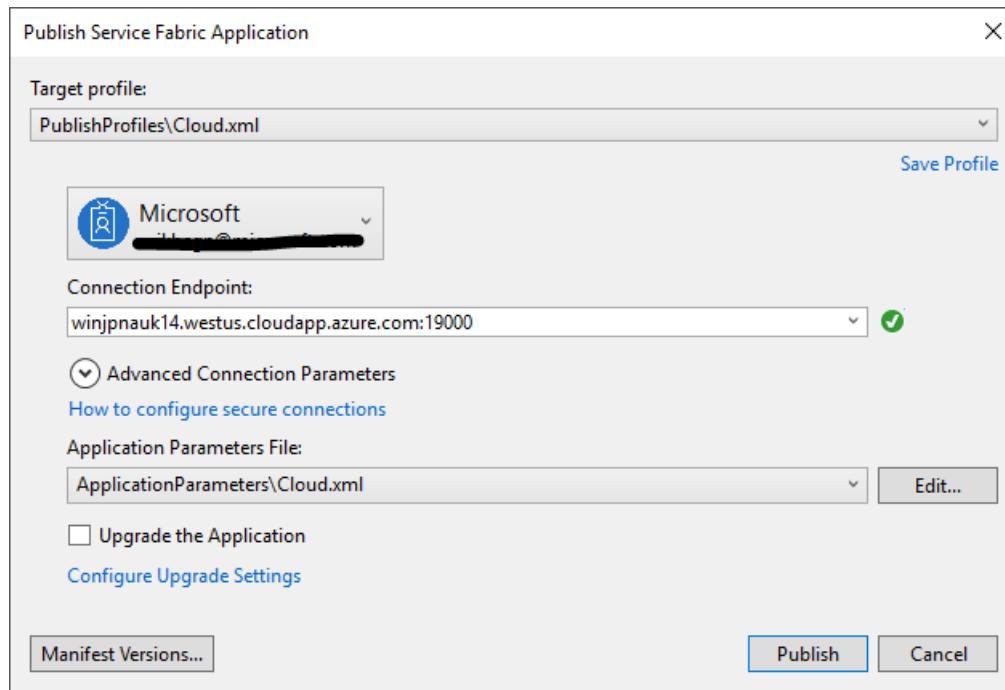
Also update the Application URL property value in the Voting project so a web browser opens to the correct port when you debug using 'F5'. In Solution Explorer, select the **Voting** project and update the **Application URL** property.



## Deploy the app to the Azure

Now that the application is ready, you can deploy it to the Party Cluster direct from Visual Studio.

1. Right-click **Voting** in the Solution Explorer and choose **Publish**.

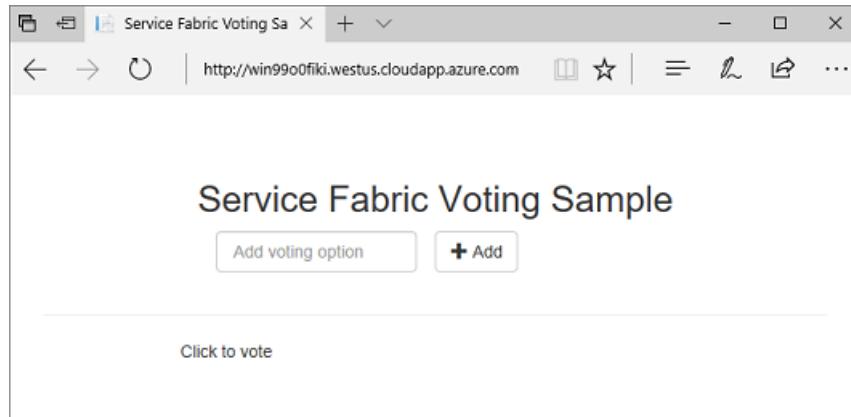


2. Type in the Connection Endpoint of the Party Cluster in the **Connection Endpoint** field and click **Publish**.

Once the publish has finished, you should be able to send a request to the application via a browser.

3. Open your preferred browser and type in the cluster address (the connection endpoint without the port information - for example, `win1kw5649s.westus.cloudapp.azure.com`).

You should now see the same result as you saw when running the application locally.



## Remove the application from a cluster using Service Fabric Explorer

Service Fabric Explorer is a graphical user interface to explore and manage applications in a Service Fabric cluster.

To remove the application from the Party Cluster:

1. Browse to the Service Fabric Explorer, using the link provided by the Party Cluster sign-up page. For example, <http://win1kw5649s.westus.cloudapp.azure.com:19080/Explorer/index.html>.
2. In Service Fabric Explorer, navigate to the **fabric://Voting** node in the treeview on the left-hand side.
3. Click the **Action** button in the right-hand **Essentials** pane, and choose **Delete Application**. Confirm deleting the application instance, which removes the instance of our application running in the cluster.

The screenshot shows the Service Fabric Explorer interface. The left sidebar has a tree view with 'Cluster' expanded, showing 'Applications' > 'VotingType' > 'fabric:/Voting'. The main area is titled 'Application fabric:/Voting' and contains tabs for 'ESSENTIALS', 'DETAILS', 'DEPLOYMENTS', and 'MANIFEST'. Under 'ESSENTIALS', it shows the application's name as 'fabric:/Voting', type as 'VotingType', health state as 'OK', and version as '1.0.0'. Below this is a section for 'UNHEALTHY EVALUATIONS' which says 'No items to display.' On the right, there is an 'ACTIONS' dropdown with a 'Delete Application' option, which is highlighted with a red box.

## Remove the application type from a cluster using Service Fabric Explorer

Applications are deployed as application types in a Service Fabric cluster, which enables you to have multiple instances and versions of the application running within the cluster. After having removed the running instance of our application, we can also remove the type, to complete the cleanup of the deployment.

For more information about the application model in Service Fabric, see [Model an application in Service Fabric](#).

1. Navigate to the **VotingType** node in the treeview.
2. Click the **Action** button in the right-hand **Essentials** pane, and choose **Unprovision Type**. Confirm unprovisioning the application type.

The screenshot shows the Service Fabric Explorer interface. The left sidebar has a tree view with 'Cluster' expanded, showing 'Applications' > 'VotingType'. The main area is titled 'Application Type VotingType' and contains tabs for 'ESSENTIALS' and 'DETAILS'. Under 'ESSENTIALS', it lists the application type's name as 'VotingType', version as '1.0.0', status as 'Available', and actions as 'Unprovision' and 'Create app instance'. Below this is a section for 'APPLICATIONS' which says 'No items to display.' On the right, there is an 'ACTIONS' dropdown with a 'Unprovision Type' option, which is highlighted with a red box.

This concludes the tutorial.

## Next steps

In this tutorial, you learned how to:

- Deploy an application to a remote cluster using Visual Studio
- Remove an application from a cluster using Service Fabric Explorer

Advance to the next tutorial:

[Set up continuous integration using Visual Studio Team Services](#)

# Deploy an application with CI/CD to a Service Fabric cluster

9/20/2017 • 6 min to read • [Edit Online](#)

This tutorial is part three of a series and describes how to set up continuous integration and deployment for an Azure Service Fabric application using Visual Studio Team Services. An existing Service Fabric application is needed, the application created in [Build a .NET application](#) is used as an example.

In part three of the series, you learn how to:

- Add source control to your project
- Create a build definition in Team Services
- Create a release definition in Team Services
- Automatically deploy and upgrade an application

In this tutorial series you learn how to:

- [Build a .NET Service Fabric application](#)
- [Deploy the application to a remote cluster](#)
- Configure CI/CD using Visual Studio Team Services
- [Set up monitoring and diagnostics for the application](#)

## Prerequisites

Before you begin this tutorial:

- If you don't have an Azure subscription, create a [free account](#)
- [Install Visual Studio 2017](#) and install the **Azure development** and **ASP.NET and web development** workloads.
- [Install the Service Fabric SDK](#)
- Create a Service Fabric application, for example by [following this tutorial](#).
- Create a Windows Service Fabric cluster on Azure, for example by [following this tutorial](#)
- Create a [Team Services account](#).

## Download the Voting sample application

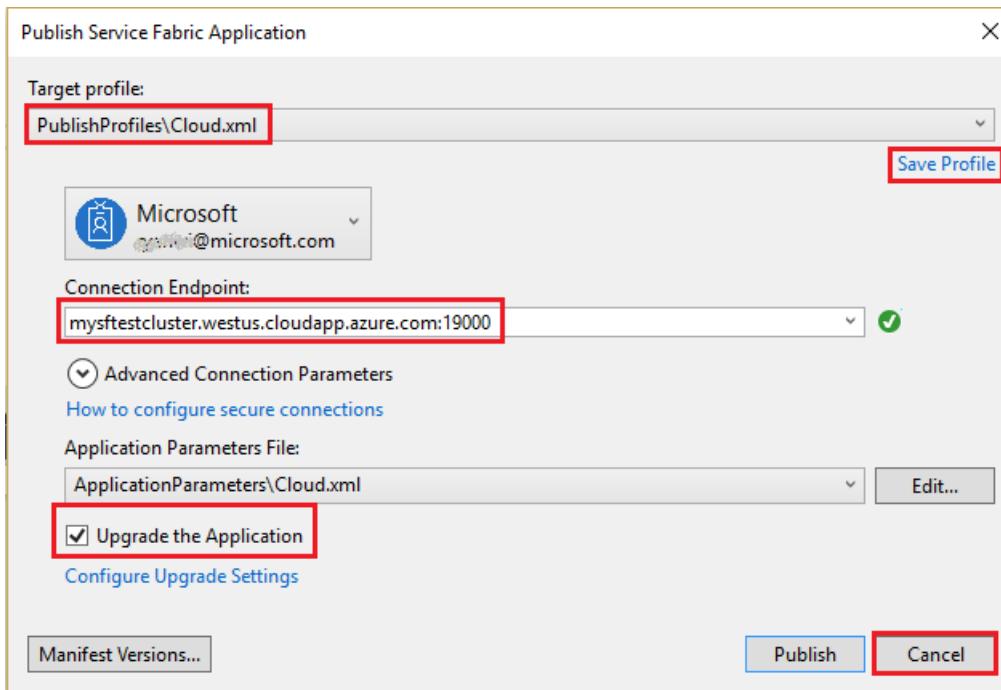
If you did not build the Voting sample application in [part one of this tutorial series](#), you can download it. In a command window, run the following command to clone the sample app repository to your local machine.

```
git clone https://github.com/Azure-Samples/service-fabric-dotnet-quickstart
```

## Prepare a publish profile

Now that you've [created an application](#) and have [deployed the application to Azure](#), you're ready to set up continuous integration. First, prepare a publish profile within your application for use by the deployment process that executes within Team Services. The publish profile should be configured to target the cluster that you've previously created. Start Visual Studio and open an existing Service Fabric application project. In **Solution Explorer**, right-click the application and select **Publish...**

Choose a target profile within your application project to use for your continuous integration workflow, for example Cloud. Specify the cluster connection endpoint. Check the **Upgrade the Application** checkbox so that your application upgrades for each deployment in Team Services. Click the **Save** hyperlink to save the settings to the publish profile and then click **Cancel** to close the dialog box.

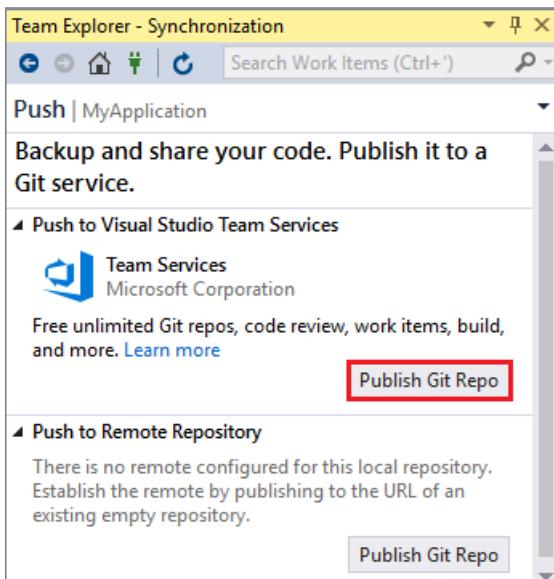


## Share your Visual Studio solution to a new Team Services Git repo

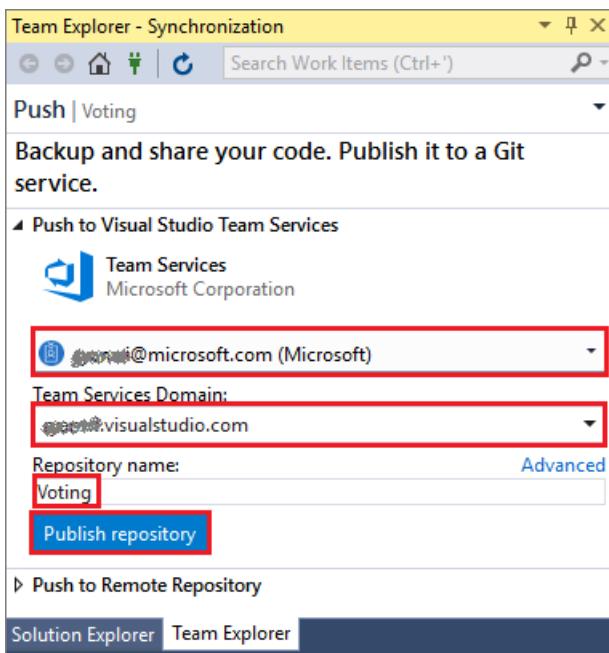
Share your application source files to a team project in Team Services so you can generate builds.

Create a new local Git repo for your project by selecting **Add to Source Control -> Git** on the status bar in the lower right-hand corner of Visual Studio.

In the **Push** view in **Team Explorer**, select the **Publish Git Repo** button under **Push to Visual Studio Team Services**.



Verify your email and select your account in the **Team Services Domain** drop-down. Enter your repository name and select **Publish repository**.



Publishing the repo creates a new team project in your account with the same name as the local repo. To create the repo in an existing team project, click **Advanced** next to **Repository** name and select a team project. You can view your code on the web by selecting **See it on the web**.

## Configure Continuous Delivery with VSTS

A Team Services build definition describes a workflow that is composed of a set of build steps that are executed sequentially. Create a build definition that that produces a Service Fabric application package, and other artifacts, to deploy to a Service Fabric cluster. Learn more about [Team Services build definitions](#).

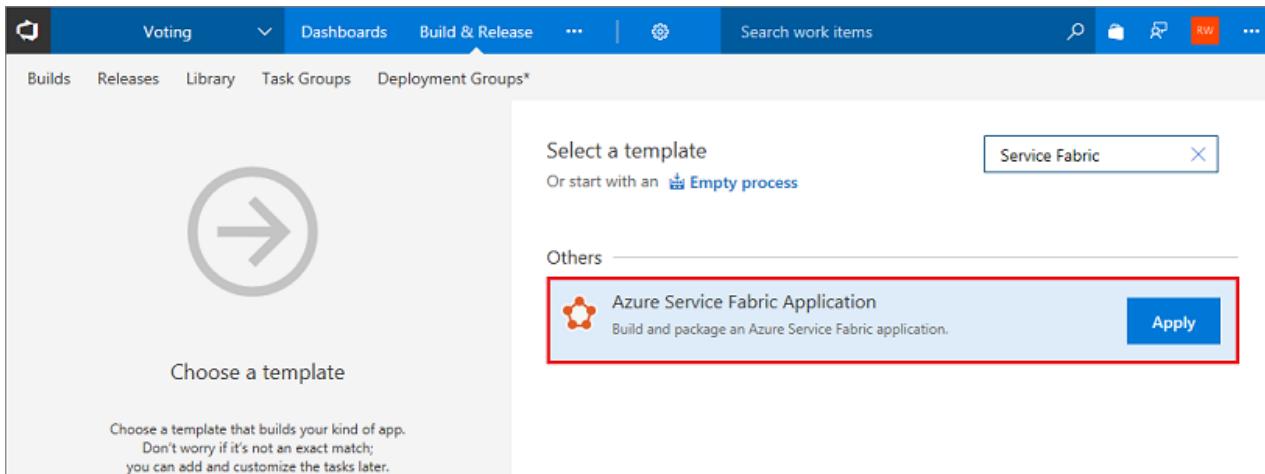
A Team Services release definition describes a workflow that deploys an application package to a cluster. When used together, the build definition and release definition execute the entire workflow starting with source files to ending with a running application in your cluster. Learn more about Team Services [release definitions](#).

### Create a build definition

Open a web browser and navigate to your new team project at:

[https://myaccount.visualstudio.com/Voting/Voting%20Team/\\_git/Voting](https://myaccount.visualstudio.com/Voting/Voting%20Team/_git/Voting) .

Select the **Build & Release** tab, then **Builds**, then **+ New definition**. In **Select a template**, select the **Azure Service Fabric Application** template and click **Apply**.



The voting application contains a .NET Core project, so add a task that restores the dependencies. In the **Tasks** view, select **+ Add Task** in the bottom left. Search on "Command Line" to find the command-line task, then click **Add**.

Process

Some settings need attention

- Get sources
- NuGet restore
- Build solution \*\*\\*.sln
- Build solution \*\*\\*.sfproj
- Copy Files to: \$(build.artifactstagingdirectory)\pdbs
- Delete files from \$(build.artifactstagingdirectory)\a...
- Update Service Fabric App Versions
- Copy Files to: \$(build.artifactstagingdirectory)\proj...
- Publish Artifact: drop

+ Add Task

Add tasks

Don't see what you need? Check out our Marketplace.

.NET Core .NET Core Build, test and publish using dotnet core command-line.

> Command Line Run a command line with arguments

Add

In the new task, enter "Run dotnet.exe" in **Display name**, "dotnet.exe" in **Tool**, and "restore" in **Arguments**.

Process

Some settings need attention

- Get sources
- NuGet restore
- Run dotnet.exe
- Build solution \*\*\\*.sln
- Build solution \*\*\\*.sfproj
- Copy Files to: \$(build.artifactstagingdirectory)\pdbs
- Delete files from \$(build.artifactstagingdirectory)\a...
- Update Service Fabric App Versions
- Copy Files to: \$(build.artifactstagingdirectory)\proj...
- Publish Artifact: drop

+ Add Task

Command Line

Link settings Remove

Version 1.1

Display name \* Run dotnet.exe

Tool \* dotnet.exe

Arguments restore

Advanced

Control Options

In the **Triggers** view, click the **Enable this trigger** switch under **Continuous Integration**.

Select **Save & queue** and enter "Hosted VS2017" as the **Agent queue**. Select **Queue** to manually start a build.

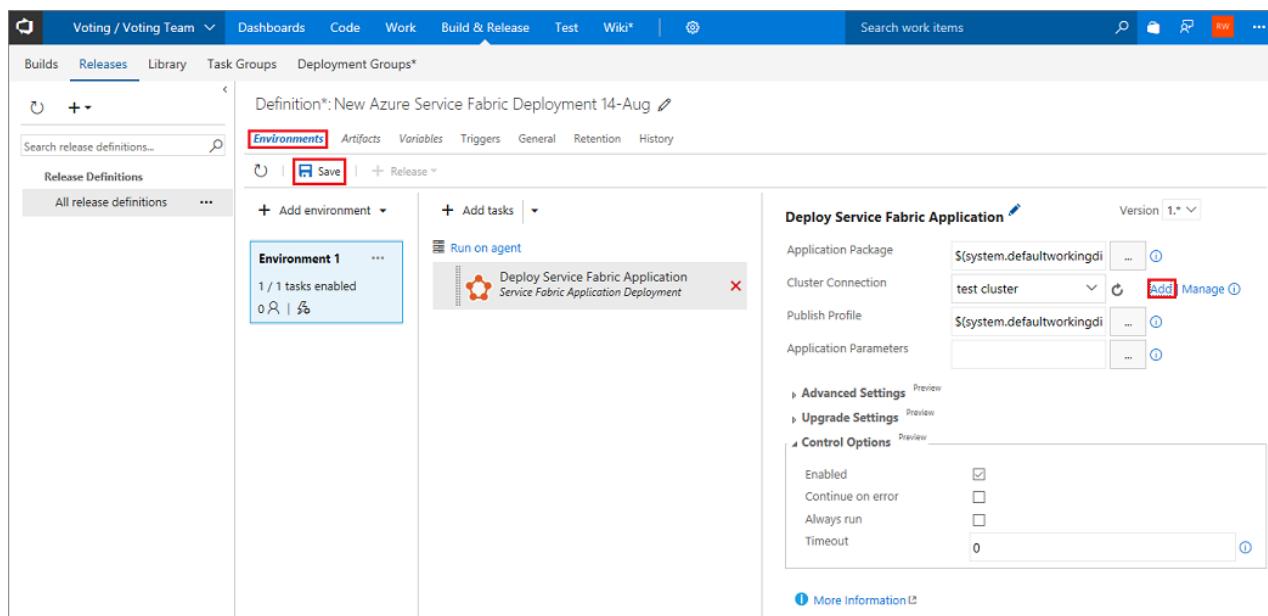
Builds also triggers upon push or check-in.

To check your build progress, switch to the **Builds** tab. Once you verify that the build executes successfully, define a release definition that deploys your application to a cluster.

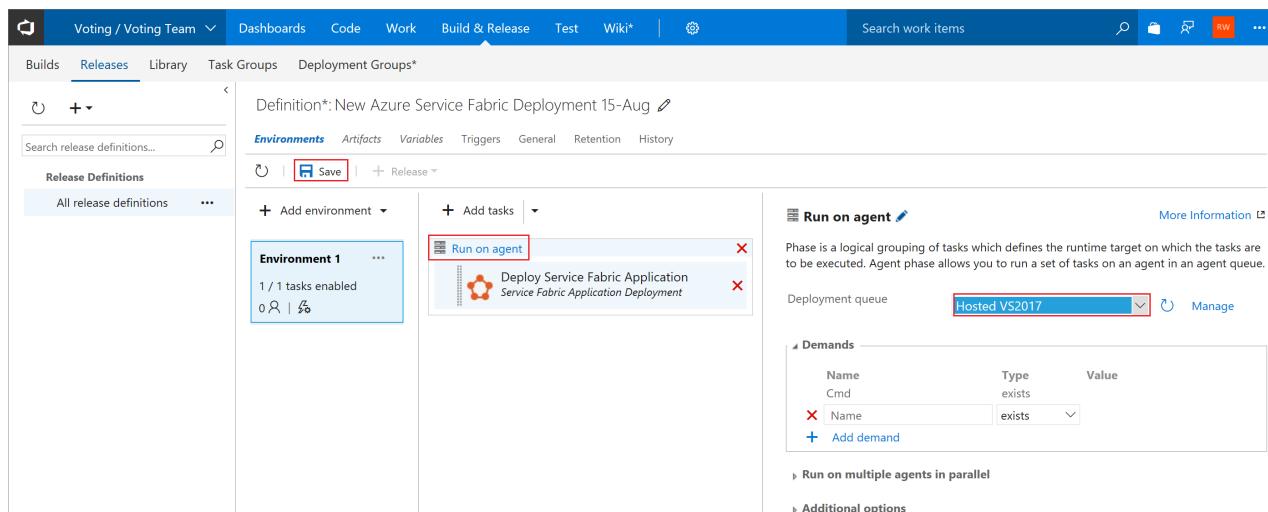
### Create a release definition

Select the **Build & Release** tab, then **Releases**, then **+ New definition**. In **Create release definition**, select the **Azure Service Fabric Deployment** template from the list and click **Next**. Select the **Build** source, check the **Continuous deployment** box, and click **Create**.

In the **Environments** view, click **Add** to the right of **Cluster Connection**. Specify a connection name of "mysftestcluster", a cluster endpoint of "tcp://mysftestcluster.westus.cloudapp.azure.com:19000", and the Azure Active Directory or certificate credentials for the cluster. For Azure Active Directory credentials, define the credentials you want to use to connect to the cluster in the **Username** and **Password** fields. For certificate-based authentication, define the Base64 encoding of the client certificate file in the **Client Certificate** field. See the help pop-up on that field for info on how to get that value. If your certificate is password-protected, define the password in the **Password** field. Click **Save** to save the release definition.



Click **Run on agent**, then select **Hosted VS2017** for **Deployment queue**. Click **Save** to save the release definition.



Select **+Release** -> **Create Release** -> **Create** to manually create a release. Verify that the deployment succeeded and the application is running in the cluster. Open a web browser and navigate to <http://mysftestcluster.westus.cloudapp.azure.com:19080/Explorer/>. Note the application version, in this example it

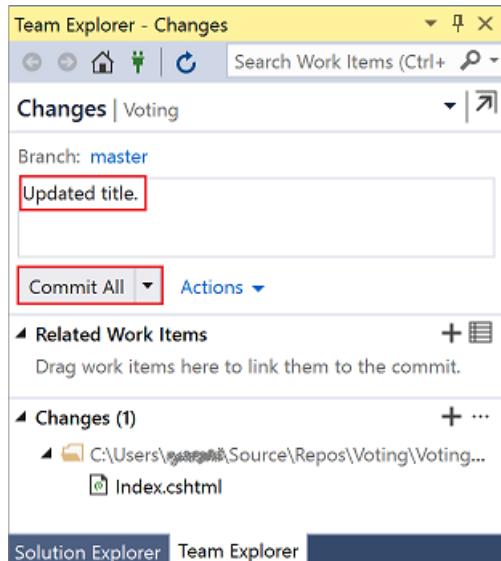
is "1.0.0.20170616.3".

## Commit and push changes, trigger a release

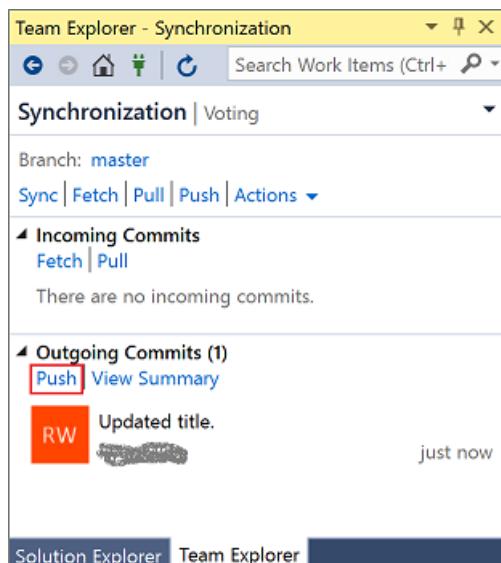
To verify that the continuous integration pipeline is functioning by checking in some code changes to Team Services.

As you write your code, your changes are automatically tracked by Visual Studio. Commit changes to your local Git repository by selecting the pending changes icon ( 1) from the status bar in the bottom right.

On the **Changes** view in Team Explorer, add a message describing your update and commit your changes.



Select the unpublished changes status bar icon ( 1) or the Sync view in Team Explorer. Select **Push** to update your code in Team Services/TFS.



Pushing the changes to Team Services automatically triggers a build. When the build definition successfully completes, a release is automatically created and starts upgrading the application on the cluster.

To check your build progress, switch to the **Builds** tab in **Team Explorer** in Visual Studio. Once you verify that the build executes successfully, define a release definition that deploys your application to a cluster.

Verify that the deployment succeeded and the application is running in the cluster. Open a web browser and navigate to <http://mysftestcluster.westus.cloudapp.azure.com:19080/Explorer/>. Note the application version, in this example it is "1.0.0.20170815.3".

The screenshot shows the Service Fabric Explorer interface for the 'fabric/Voting' application. The 'ESSENTIALS' tab is selected. In the left sidebar, under 'Cluster > Applications > VotingType > fabric/Voting', the status is 'OK'. The 'Version' field is highlighted with a red box and shows '1.0.0.20170815.3'. The 'Status' is 'Ready'. Below this, the 'UNHEALTHY EVALUATIONS' section shows 'No items to display.' The 'SERVICES' section lists two services: 'fabric/Voting/VotingData' (VotingDataType, Version 1.0.0, Stateful, OK, Active) and 'fabric/Voting/VotingWeb' (VotingWebType, Version 1.0.0, Stateless, OK, Active).

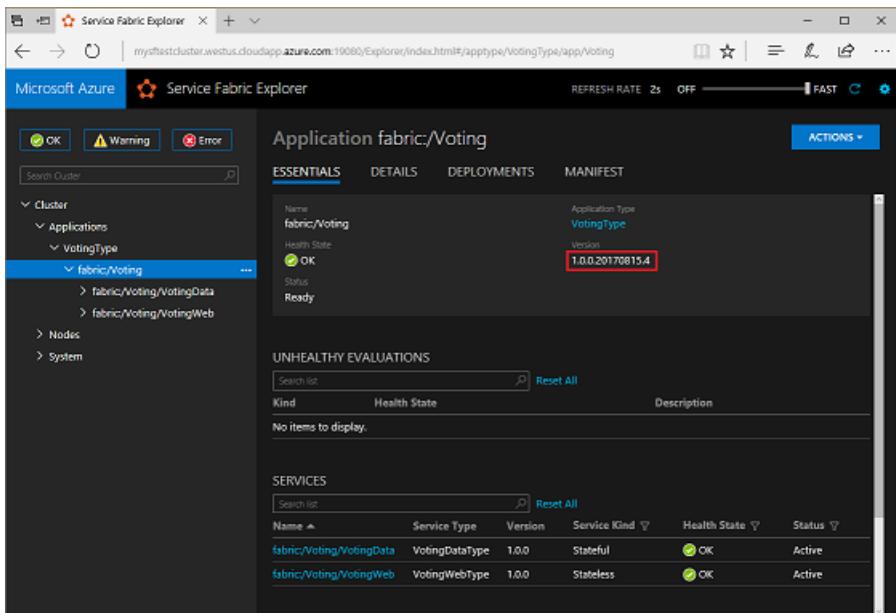
## Update the application

Make code changes in the application. Save and commit the changes, following the previous steps.

Once the upgrade of the application begins, you can watch the upgrade progress in Service Fabric Explorer:

The screenshot shows the Service Fabric Explorer interface for the 'fabric/Voting' application. The 'ESSENTIALS' tab is selected. In the left sidebar, under 'Cluster > Applications > VotingType > fabric/Voting', the status is 'Upgrading'. The 'Version' field shows '1.0.0.20170815.3'. The 'Status' is 'Upgrading'. A large red box highlights the 'UPGRADE IN PROGRESS' section. This section contains a table with columns: Current Version, Target Version, Progress by Upgrade Domain, and Upgrade State. The data is: Current Version 1.0.0.20170815.3, Target Version 1.0.0.20170815.4, Progress by Upgrade Domain [0, 1, 2, 3, 4], and Upgrade State RollingForwardInProgress. Below this, the 'UPGRADE IN PROGRESS' section has a 'Show upgrade details' link. The 'UNHEALTHY EVALUATIONS' and 'SERVICES' sections are also visible.

The application upgrade may take several minutes. When the upgrade is complete, the application will be running the next version. In this example "1.0.0.20170815.4".



## Next steps

In this tutorial, you learned how to:

- Add source control to your project
- Create a build definition
- Create a release definition
- Automatically deploy and upgrade an application

Advance to the next tutorial:

[Set up monitoring and diagnostics for the application](#)

# Monitor and diagnose an ASP.NET Core application on Service Fabric

9/25/2017 • 9 min to read • [Edit Online](#)

This tutorial is part four of a series. It goes through the steps to set up monitoring and diagnostics for an ASP.NET Core application running on a Service Fabric cluster using Application Insights. We will collect telemetry from the application developed in the first part of the tutorial, [Build a .NET Service Fabric application](#).

In part four of the tutorial series, you learn how to:

- Configure Application Insights for your application
- Collect response telemetry to trace HTTP-based communication between services
- Use the App Map feature in Application Insights
- Add custom events using the Application Insights API

In this tutorial series you learn how to:

- [Build a .NET Service Fabric application](#)
- [Deploy the application to a remote cluster](#)
- [Configure CI/CD using Visual Studio Team Services](#)
- Set up monitoring and diagnostics for the application

## Prerequisites

Before you begin this tutorial:

- If you don't have an Azure subscription, create a [free account](#)
- [Install Visual Studio 2017](#) and install the **Azure development** and **ASP.NET and web development** workloads.
- [Install the Service Fabric SDK](#)

## Download the Voting sample application

If you did not build the Voting sample application in [part one of this tutorial series](#), you can download it. In a command window or terminal, run the following command to clone the sample app repository to your local machine.

```
git clone https://github.com/Azure-Samples/service-fabric-dotnet-quickstart
```

## Set up an Application Insights resource

Application Insights is Azure's application performance management platform, and Service Fabric's recommended platform for application monitoring and diagnostics. To create an Application Insights resource, navigate to [Azure portal](#). Click **New** on the left navigation menu to open up Azure Marketplace. Click on **Monitoring + Management** and then **Application Insights**.

The screenshot shows the Microsoft Azure portal interface. On the left, a sidebar lists various service categories like Dashboard, All resources, App Services, etc. A red box highlights the '+ New' button at the top of the sidebar. The main area is titled 'New' and features a search bar labeled 'Search the Marketplace'. Below it, there are two tabs: 'Azure Marketplace' (selected) and 'Featured'. Under 'Azure Marketplace', there are several categories: Get started, Compute, Networking, Storage, Web + Mobile, Databases, Data + Analytics, AI + Cognitive Services, Internet of Things, Enterprise Integration, Security + Identity, Developer tools, Monitoring + Management (highlighted with a red box), Add-ons, Containers, and Blockchain. To the right of these categories are their respective icons and names. One item, 'Application Insights', has a red box around its 'Details' link.

You will now need to fill out required information about the attributes of the resource to be created. Enter an appropriate *Name*, *Resource Group*, and *Subscription*. Set the *Location* to where you would deploy your Service Fabric cluster in the future. In this tutorial, we will deploy the app to a local cluster so the *Location* field is irrelevant. The *Application Type* should be left as "ASP.NET web application."

Application Insights

Monitor web app performance and usage

Name \*  
servicefabric-ai ✓

\* Application Type \*  
ASP.NET web application

\* Subscription  
Windows Fabric - Internal Consumption (P1) ✓

\* Resource Group \*  
 Create new  Use existing  
servicefabric-ai ✓

\* Location  
East US

Pin to dashboard

Create Automation options

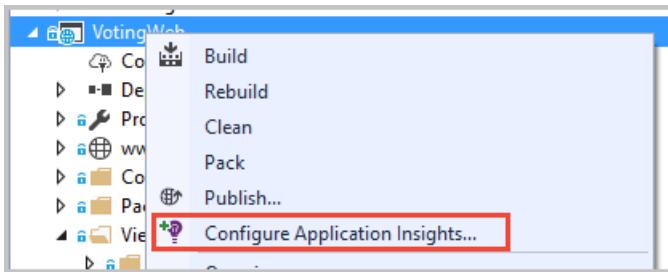
Once you've filled out the required information, click **Create** to provision the resource - it should take about a minute.

## Add Application Insights to the application's services

Launch Visual Studio 2017 with elevated privileges. You can do this by right-clicking the Visual Studio icon in the Start Menu and choosing **Run as administrator**. Click **File > Open > Project/Solution** and navigate to the Voting application (either created in part one of the tutorial or git cloned). Open *Voting.sln* and if prompted to restore the application's NuGet packages, click **Yes**.

Follow these steps to configure Application Insights for both VotingWeb and VotingData services:

1. Right-click on the name of the service, and click **Configure Application Insights...**



2. Click **Start Free**.
3. Sign in to your account (with which you also set up your Azure subscription) and select the subscription in which you created the Application Insights resource. Find the resource under *Existing Application Insights* resource in the "Resource" dropdown. Click **Register** to add Application Insights to your service.

Application Insights Configuration X

## Application Insights

SDK added

Register your app with Application Insights

Account

Microsoft  
dekapur@microsoft.com

Subscription

Windows Fabric - Internal Consumption (PM Team)

Resource

servicefabric-ai (Existing resource)

[Configure settings...](#)

---

Base Monthly Price	Free
Included Data	1 GB / Month
Additional Data	\$2.30 per GB*
Data retention (raw and aggregated data)	90 days

\*Pricing is subject to change. Visit our [pricing page](#) for most recent pricing details.

[Register](#)

4. Click **Finish** once the dialog box that pops up completes the action.

Make sure to do the above steps for **both** of the services in the application to finish configuring Application Insights for the application. The same Application Insights resource is used for both of the services in order to see incoming and outgoing requests and communication between the services.

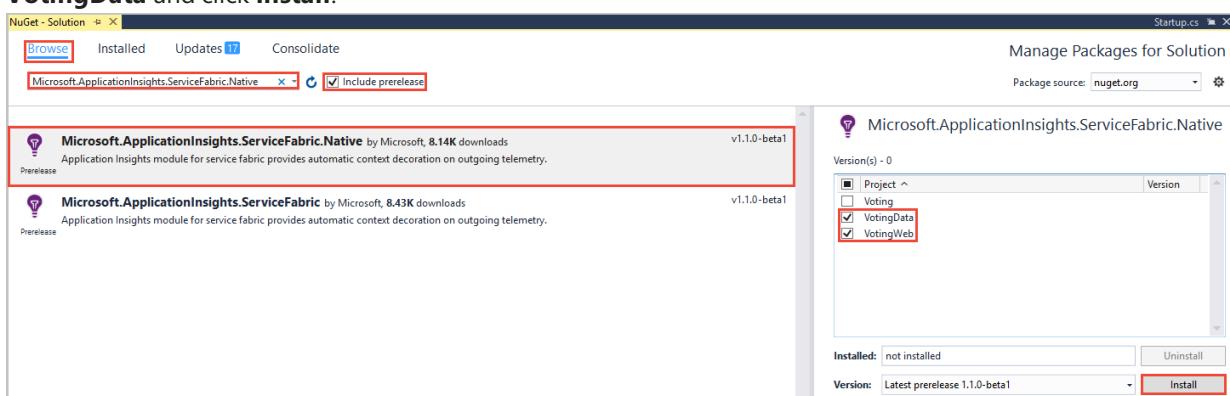
## Add the Microsoft.ApplicationInsights.ServiceFabric.Native NuGet to the services

Application Insights has two Service Fabric specific NuGets that can be used depending on the scenario. One is used with Service Fabric's native services, and the other with containers and guest executables. In this case, we'll be using the Microsoft.ApplicationInsights.ServiceFabric.Native NuGet to leverage the understanding of service context that it brings. To read more about the Application Insights SDK and the Service Fabric specific NuGets, see [Microsoft Application Insights for Service Fabric](#).

Here are the steps to set up the NuGet:

1. Right-click on the **Solution 'Voting'** at the top of your Solution Explorer, and click **Manage NuGet Packages for Solution....**
2. Click **Browse** on the top navigation menu of the "NuGet - Solution" window, and check the **Include prerelease** box next to the search bar.
3. Search for `Microsoft.ApplicationInsights.ServiceFabric.Native` and click on the appropriate NuGet package.

4. On the right, click on the two checkboxes next to the two services in the application, **VotingWeb** and **VotingData** and click **Install**.



5. Click **OK** on the *Review Changes* dialog box that pops up, and accept the *License Acceptance*. This will complete adding the NuGet to the services.
6. You now need to set up the telemetry initializer in the two services. To this, open up *VotingWeb.cs* and *VotingData.cs*. For both of them, do the following two steps:

- a. Add these two *using* statements at the top of each *<ServiceName>.cs*:

```
using Microsoft.ApplicationInsights.Extensibility;
using Microsoft.ApplicationInsights.ServiceFabric;
```

- b. In the nested *return* statement of *CreateServiceInstanceListeners()* or *CreateServiceReplicaListeners()*, under *ConfigureServices > services*, in between the two Singleton services declared, add:

```
.AddSingleton<ITelemetryInitializer>((serviceProvider) =>
    FabricTelemetryInitializerExtension.CreateFabricTelemetryInitializer(serviceContext))
```

This will add the *Service Context* to your telemetry, allowing you to better understand the source of your telemetry in Application Insights. Your nested *return* statement in *VotingWeb.cs* should look like this:

```
return new WebHostBuilder().UseWebListener()
    .ConfigureServices(
        services => services
            .AddSingleton<StatelessServiceContext>(serviceContext)
            .AddSingleton<ITelemetryInitializer>((serviceProvider) =>
                FabricTelemetryInitializerExtension.CreateFabricTelemetryInitializer(serviceContext))
            .AddSingleton<HttpClient>()
            .UseContentRoot(Directory.GetCurrentDirectory())
            .UseStartup<Startup>()
            .UseApplicationInsights()
            .UseServiceFabricIntegration(listener, ServiceFabricIntegrationOptions.None)
            .UseUrls(url)
            .Build();
```

Similarly, in *VotingData.cs*, you should have:

```
return new WebHostBuilder()
    .UseKestrel()
    .ConfigureServices(
        services => services
            .AddSingleton<StatefulServiceContext>(serviceContext)
            .AddSingleton<ITelemetryInitializer>((serviceProvider) =>
FabricTelemetryInitializerExtension.CreateFabricTelemetryInitializer(serviceContext))
            .AddSingleton<IReliableStateManager>(this.StateManager))
    .UseContentRoot(Directory.GetCurrentDirectory())
    .UseStartup<Startup>()
    .UseApplicationInsights()
    .UseServiceFabricIntegration(listener, ServiceFabricIntegrationOptions.UseUniqueServiceUrl)
    .UseUrls(url)
    .Build();
```

At this point, you are ready to deploy the application. Click **Start** at the top (or **F5**), and Visual Studio will build and package the application, set up your local cluster, and deploy the application to it.

Once the application is done deploying, head over to [localhost:8080](http://localhost:8080), where you should be able to see the Voting Sample single page application. Vote for a few different items of your choice to create some sample data and telemetry - I went for desserts!

Service Fabric Voting Sample

Add voting option  **+ Add**

Click to vote

Voting Option	Votes	Action
Chocolate	5 Votes	<b>✖ Remove</b>
Ice cream	3 Votes	<b>✖ Remove</b>
Pie	2 Votes	<b>✖ Remove</b>

Feel free to *Remove* some of the voting options as well when you're done adding a few votes.

## View telemetry and the App map in Application Insights

Head over to your Application Insights resource in Azure portal, and in the left navigation bar of the resource, click on **Previews** under *Configure*. Turn **On** the *Multi-role Application Map* in the list of available previews.

servicefabric-ai - Previews

Application Insights

Search (Ctrl+ /)

USAGE (PREVIEW)

- Users
- Sessions
- Events
- Funnels
- Retention
- Workbooks
- Cohorts
- User Flows
- Usage (Deprecated)

CONFIGURE

- Getting started
- Previews
- Properties
- Alerts
- Smart Detection settings
- Features + pricing
- Data volume management
- Continuous export
- Performance Testing
- API Access
- Work Items
- Scheduled Analytics (preview)

SETTINGS

Application Insights frequently releases new features. We usually begin with a restricted release and use the feedback we get to improve the feature before full release. Here you can choose whether you'd like to see our feature previews. Be aware that we might change or withdraw features. You can suggest features or log bugs on [User Voice](#).

Auto-enroll for future previews [?](#)

Always Auto Never

Available previews

Sort by Release date Descending [Enable all](#) [Disable all](#)

**Interactive Failures Investigation Experience** [Give feedback](#)  
Release date: 9/1/2017  
We have enabled this new failures investigation experience as an improvement to Application Insights. We'd love for you to try it and give us feedback. If you still need the old experience, it's just a click away in the Previews management blade. We'd value your feedback even if you choose to opt-out, as that will also help us make this new experience even better.  
[On](#) [Off](#)

**Interactive Performance Investigation Experience** [Give feedback](#)  
Release date: 7/17/2017  
We have enabled this new performance investigation experience as an improvement to Application Insights. We'd love for you to try it and give us feedback. If you still need the old experience, it's just a click away in the Previews management blade. We'd value your feedback even if you choose to opt-out, as that will also help us make this new experience even better.  
[On](#) [Off](#)

**Multi-role Application Map** [Give feedback](#)  
Release date: 5/10/2017  
With this preview, we've enabled you to see multiple server components reporting to the same Application Insights resource on the Application Map. Servers are segmented by their cloud role name.  
[On](#) [Off](#)

**Metrics (preview)** [Give feedback](#)  
Release date: 5/10/2017  
With this preview, you can try out our new metrics exploration experience. It includes the ability to visualize metrics from multiple Azure resources – both Application Insights and other types – on the same chart, higher fidelity data visualization, and a streamlined configuration experience. Note that at this point, many features of current Metrics Explorer have not yet been added to this new experience. Enabling this preview will not affect your ability to use current Metrics Explorer. [Details...](#)  
[On](#) [Off](#)

Click **Overview** to go back to the landing page of your resource. Then click **Search** in the top to see the traces coming in. It takes a few minutes for traces to appear in Application Insights. In the case that you do not see any, wait a minute and hit the **Refresh** button at the top.

Scrolling down on the *Search* window will show you all the incoming telemetry you get out-of-the-box with Application Insights. For each action that you took in the Voting application, there should be an outgoing PUT request from *VotingWeb* (PUT Votes/Put [name]), an incoming PUT request from *VotingData* (PUT VoteData/Put [name]), followed by a pair of GET requests for refreshing the data being displayed. There will also be a dependency trace for HTTP on localhost, since these are HTTP requests. Here is an example of what you will see for

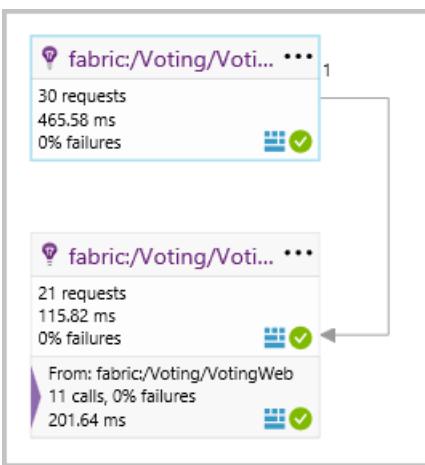
<b>9/17/2017 7:41:00 PM - REQUEST</b>
GET VoteData/Get
Request URL: http://localhost:61436/94936cd5-6277-4161-b694-356cce2b335/131501759642...
Response code: 200 Server response time: 4.39 ms
Request URL path: /94936cd5-6277-4161-b694-356cce2b335/131501759642254118/86159fb...
<b>9/17/2017 7:41:00 PM - DEPENDENCY</b>
localhost
Operation name: GET Votes/Get Dependency duration: 5 ms Successful call: true
Dependency name: GET /Voting/VotingData/api/VoteData
<b>9/17/2017 7:41:00 PM - REQUEST</b>
GET Votes/Get
Request URL: http://localhost:8080/api/Votes?c=1505702460567 Response code: 200
Server response time: 11.84 ms Request URL path: /api/Votes
<b>9/17/2017 7:41:00 PM - REQUEST</b>
PUT VoteData/Put [name]
Request URL: http://localhost:61436/94936cd5-6277-4161-b694-356cce2b335/131501759642...
Response code: 200 Server response time: 21.67 ms
Request URL path: /94936cd5-6277-4161-b694-356cce2b335/131501759642254118/86159fb...
<b>9/17/2017 7:41:00 PM - REQUEST</b>
PUT Votes/Put [name]
Request URL: http://localhost:8080/api/Votes/Ice cream Response code: 200
Server response time: 28.51 ms Request URL path: /api/Votes/Ice%20cream

how one vote is added:

You can click on one of the traces to view more details about it. There is useful information about the request that is provided by Application Insights, including the *Response time* and the *Request URL*. In addition, since you added the Service Fabric specific NuGet, you will also get data about your application in the context of a Service Fabric cluster in the *Custom Data* section below. This includes the service context, so you can see the *PartitionID* and *ReplicaId* of the source of the request, and better localize issues when diagnosing errors in your application.

The screenshot shows two Application Insights pages side-by-side. The left page is 'Search' with a time range of 'Last 33 minutes (1 minute granularity) - servicefabric-ai'. It lists several log entries for '9/17/2017 7:41:00 PM'. One entry is highlighted in blue: 'PUT VoteData/Put [name]' with a request URL of 'http://localhost:8080/api/Votes/Get?c=1505702460730'. The right page is 'PUT VoteData/Put [name]' showing 'Request Properties' for the same event. The properties include Event time (9/17/2017 7:41:00 PM), Request name (PUT VoteData/Put [name]), Response code (200), and various URL and client details.

Additionally, since we enabled the App map, on the *Overview* page, clicking on the **App map** icon will show you both your services connected.



The App map can help you understand your application topology better, especially as you start adding multiple different services that work together. It also gives you basic data on request success rates, and can help you diagnose failed request to understand where things may have gone wrong. To learn more about using the App map, see [Application Map in Application Insights](#).

## Add custom instrumentation to your application

Though Application Insights provides a lot of telemetry out of the box, you may want to add further custom

instrumentation. This could be based on your business needs or to improve diagnostics when things go wrong in your application. Application Insights has an API to ingest custom events and metrics, which you can read more about [here](#).

Let's add some custom events to *VoteDataController.cs* (under *VotingData > Controllers*) to track when votes are being added and deleted from the underlying *votesDictionary*.

1. Add `using Microsoft.ApplicationInsights;` at the end of the other using statements.
2. Declare a new *TelemetryClient* at the start of the class, under the creation of the *IReliableStateManager*:  
`private TelemetryClient telemetry = new TelemetryClient();`.
3. In the *Put()* function, add an event that confirms a vote has been added. Add  
`telemetry.TrackEvent($"Added a vote for {name}");` after the transaction has completed, right before the return *OkResult* statement.
4. In *Delete()*, there is an "if/else" based on the condition that the *votesDictionary* contains votes for a given voting option.
  - a. Add an event that confirms the deletion of a vote in the *if* statement, after the *await tx.CommitAsync()*:  
`telemetry.TrackEvent($"Deleted votes for {name}");`
  - b. Add an event to show that the deletion did not take place in the *else* statement, before the return statement: `telemetry.TrackEvent($"Unable to delete votes for {name}, voting option not found");`

Here's an example of what your *Put()* and *Delete()* functions may look like after adding the events:

```
// PUT api/VoteData/name
[HttpPut("{name}")]
public async Task<IActionResult> Put(string name)
{
    var votesDictionary = await this.stateManager.GetOrAddAsync<IReliableDictionary<string, int>>("counts");

    using (ITransaction tx = this.stateManager.CreateTransaction())
    {
        await votesDictionary.AddOrUpdateAsync(tx, name, 1, (key, oldvalue) => oldvalue + 1);
        await tx.CommitAsync();
    }

    telemetry.TrackEvent($"Added a vote for {name}");
    return new OkResult();
}

// DELETE api/VoteData/name
[HttpDelete("{name}")]
public async Task<IActionResult> Delete(string name)
{
    var votesDictionary = await this.stateManager.GetOrAddAsync<IReliableDictionary<string, int>>("counts");

    using (ITransaction tx = this.stateManager.CreateTransaction())
    {
        if (await votesDictionary.ContainsKeyAsync(tx, name))
        {
            await votesDictionary.TryRemoveAsync(tx, name);
            await tx.CommitAsync();
            telemetry.TrackEvent($"Deleted votes for {name}");
            return new OkResult();
        }
        else
        {
            telemetry.TrackEvent($"Unable to delete votes for {name}, voting option not found");
            return new NotFoundResult();
        }
    }
}
```

Once you're done making these changes, **Start** the application so that it builds and deploys the latest version of it. Once the application is done deploying, head over to [localhost:8080](http://localhost:8080), and add and delete some voting options. Then, go back to your Application Insights resource to see the traces for the latest run (as before, traces can take 1-2 min to show up in Application Insights). For all the votes you added and deleted, you should now see a "Custom Event\*" along with all the response telemetry.

The screenshot shows a log viewer with several traces. The traces are color-coded: blue for REQUESTS, yellow for DEPENDENCIES, and green for CUSTOM EVENTS. A red box highlights the CUSTOM EVENT trace from 9/18/2017 11:09:39 AM, which indicates a vote was added for 'Chocolate'. Another red box highlights the CUSTOM EVENT trace from 9/18/2017 11:09:10 AM, which indicates a vote was added for 'Chocolate'.

**9/18/2017 11:09:39 AM - REQUEST**  
GET VoteData/Get  
Request URL: http://localhost:54974/b5fdb650-1514-4eba-a58c-c63d72a91dad/131502290315...  
Response code: 200 Server response time: 6.22 ms  
Request URL path: /b5fdb650-1514-4eba-a58c-c63d72a91dad/131502290315882480/e232814...

**9/18/2017 11:09:39 AM - DEPENDENCY**  
localhost  
Operation name: GET Votes/Get Dependency duration: 7 ms Successful call: true  
Dependency name: GET /Voting/VotingData/api/VoteData

**9/18/2017 11:09:39 AM - REQUEST**  
GET Votes/Get  
Request URL: http://localhost:8080/api/Votes?c=1505758179373 Response code: 200  
Server response time: 13.71 ms Request URL path: /api/Votes

**9/18/2017 11:09:39 AM - CUSTOM EVENT**  
Deleted votes for Chocolate  
Device type: PC Application version: 1.0.0.0

**9/18/2017 11:09:39 AM - REQUEST**  
DELETE VoteData/Delete [name]  
Request URL: http://localhost:54974/b5fdb650-1514-4eba-a58c-c63d72a91dad/131502290315...  
Response code: 200 Server response time: 48.59 ms  
Request URL path: /b5fdb650-1514-4eba-a58c-c63d72a91dad/131502290315882480/e232814...

**9/18/2017 11:09:39 AM - REQUEST**  
DELETE Votes/Delete [name]  
Request URL: http://localhost:8080/api/Votes/Chocolate Response code: 200  
Server response time: 55.61 ms Request URL path: /api/Votes/Chocolate

**9/18/2017 11:09:10 AM - REQUEST**  
GET VoteData/Get  
Request URL: http://localhost:54974/b5fdb650-1514-4eba-a58c-c63d72a91dad/131502290315...  
Response code: 200 Server response time: 4.52 ms  
Request URL path: /b5fdb650-1514-4eba-a58c-c63d72a91dad/131502290315882480/e232814...

**9/18/2017 11:09:10 AM - DEPENDENCY**  
localhost  
Operation name: GET Votes/Get Dependency duration: 6 ms Successful call: true  
Dependency name: GET /Voting/VotingData/api/VoteData

**9/18/2017 11:09:10 AM - REQUEST**  
GET Votes/Get  
Request URL: http://localhost:8080/api/Votes?c=1505758150212 Response code: 200  
Server response time: 14.04 ms Request URL path: /api/Votes

**9/18/2017 11:09:10 AM - CUSTOM EVENT**  
Added a vote for Chocolate  
Device type: PC Application version: 1.0.0.0

**9/18/2017 11:09:10 AM - REQUEST**  
PUT VoteData/Put [name]  
Request URL: http://localhost:54974/b5fdb650-1514-4eba-a58c-c63d72a91dad/131502290315...  
Response code: 200 Server response time: 26.29 ms  
Request URL path: /b5fdb650-1514-4eba-a58c-c63d72a91dad/131502290315882480/e232814...

**9/18/2017 11:09:10 AM - REQUEST**  
PUT Votes/Put [name]  
Request URL: http://localhost:8080/api/Votes/Chocolate Response code: 200  
Server response time: 54.01 ms Request URL path: /api/Votes/Chocolate

## Next steps

In this tutorial, you learned how to:

- Configure Application Insights for your application
- Collect response telemetry to trace HTTP-based communication between services
- Use the App Map feature in Application Insights
- Add custom events using the Application Insights API

Now that you have completed setting up monitoring and diagnostics for your ASP.NET application, try the following:

- Explore monitoring and diagnostics in Service Fabric

- Service Fabric event analysis with Application Insights
- To learn more about Application Insights, see [Application Insights Documentation](#)

# Create a secure cluster in Azure by using PowerShell

10/26/2017 • 5 min to read • [Edit Online](#)

This article is the first in a series of tutorials that show you how to move a .NET application to the cloud by using Azure Service Fabric clusters and containers. In the following steps, you learn how to create a Service Fabric cluster (Windows or Linux) that runs in Azure. When you're finished, you have a secure cluster that runs in the cloud to which you can deploy applications.

## Prerequisites

Before you begin this tutorial:

- If you don't have an Azure subscription, create a [free account](#).
- Install the [Service Fabric SDK](#).
- Install [Azure Powershell module version 4.1 or higher](#). (If needed, [install Azure PowerShell](#) or [update to a newer version](#).)

## Create a Service Fabric cluster

This script creates a single-node, preview Service Fabric cluster. A self-signed certificate secures the cluster. The script creates the certificate along with the cluster, and then places the certificate in a key vault. You can't scale single-node clusters beyond one virtual machine, and you can't upgrade preview clusters to newer versions.

To calculate the cost incurred by running a Service Fabric cluster in Azure, use the [Azure pricing calculator](#). For more information on how to create Service Fabric clusters, see [Create a Service Fabric cluster by using Azure Resource Manager](#).

## Log in to Azure

Open a PowerShell console, log in to Azure, and select the subscription you want to deploy the cluster in:

```
Login-AzureRmAccount  
Select-AzureRmSubscription -SubscriptionId <subscription-id>
```

## Cluster parameters

This script uses the following parameters and concepts. Customize the parameters to fit your requirements.

PARAMETER	DESCRIPTION	SUGGESTED VALUE
Location	The Azure region where you deployed the cluster.	For example, <i>westeurope</i> , <i>eastasia</i> , or <i>eastus</i>
Name	The name of the cluster you want to create. The name must be 4-23 characters and can only have lowercase letters, numbers, and hyphens.	For example, <i>bobs-sfpreviewcluster</i>
ResourceGroupName	The name of the resource group in which to create the cluster.	For example, <i>myresourcegroup</i>

PARAMETER	DESCRIPTION	SUGGESTED VALUE
VmSku	The virtual machine SKU to use for the nodes.	Any valid virtual machine SKU
OS	The virtual machine OS to use for the nodes.	Any valid virtual machine OS
KeyVaultName	The name of the new key vault to associate with the cluster.	For example, <i>mykeyvault</i>
ClusterSize	The number of virtual machines in your cluster (can be 1 or 3-99).	Specify only one virtual machine for a preview cluster
CertificateSubjectName	The subject name of the certificate to be created.	Follows the format: ..cloudapp.azure.com

### Default parameter values

**Virtual Machine:** Optional settings. If you don't specify them, the admin username defaults to *vmadmin* and PowerShell prompts you for a virtual machine password before it creates the cluster.

**Ports:** Default to ports 80 and 8081. You can specify additional ports by following the guidance for [ports in Service Fabric clusters](#).

**Diagnostics:** Enabled by default.

**DNS service:** Not enabled by default.

**Reverse proxy:** Not enabled by default.

## Create the cluster with your parameters

After you decide on the parameters that fit your requirements, run the following command to generate a secure Service Fabric cluster and its certificate.

You can modify this script to include additional parameters. For more information on parameters for cluster creation, see the [New-AzureRmServiceFabricCluster](#) cmdlet.

#### NOTE

Before you run this command, you must create a folder where you can store the cluster certificate.

```

# Set the certificate variables. This creates and encrypts a password that Service Fabric will use.
$certpwd="Password#1234" | ConvertTo-SecureString -AsPlainText -Force

# You must create the folder where you want to store the certificate on your machine before you start this
# step.
$certfolder="c:\mycertificates\"

# Set the variables for common values. Change the values to fit your needs.
$clusterloc="WestUS"
$clustername = "mysfcluster"
$groupname="mysfclustergroup"
$vmsku = "Standard_D2_v2"
$vaultname = "mykeyvault"
$subname="$clustername.$clusterloc.cloudapp.azure.com"

# Set the number of cluster nodes. The possible values are 1 and 3-99.
$clustersize=1

# Create the Service Fabric cluster and its self-signed certificate. The OS you specify here lets you use this
# cluster with any applications that are also using containers.
New-AzureRmServiceFabricCluster -Name $clustername -ResourceGroupName $groupname -Location $clusterloc `

-ClusterSize $clustersize -CertificateSubjectName $subname `

-CertificatePassword $certpwd -CertificateOutputFolder $certfolder `

-OS WindowsServer2016DatacenterwithContainers -VmSku $vmsku -KeyVaultName $vaultname

```

The creation process can take several minutes. After the configuration finishes, it outputs information about the cluster created in Azure. It also copies the cluster certificate to the `-CertificateOutputFolder` directory on the path you specified for this parameter.

Take note of the **ManagementEndpoint** URL for your cluster, which might be like the following URL:  
<https://mycluster.westeurope.cloudapp.azure.com:19080>.

## Import the certificate

When the cluster is successfully created, run the following command to ensure that you can use the self-signed certificate:

```

# To connect to the cluster, install the certificate into the Personal (My) store of the current user on your
computer.
Import-PfxCertificate -Exportable -CertStoreLocation Cert:\CurrentUser\My `

-FilePath C:\mycertificates\mysfclustergroup20170531104310.pfx `

-Password $certpwd

```

This command installs the certificate on the current user of your machine. You need this certificate to access Service Fabric Explorer and view the health of your cluster.

## View your cluster (Optional)

After you have both the cluster and the imported certificate, you can connect to the cluster and view its health. There are multiple ways to connect, via either Service Fabric Explorer or PowerShell.

### Service Fabric Explorer

You can view the health of your cluster through Service Fabric Explorer. To do so, browse to the **ManagementEndpoint** URL for your cluster, and then select the certificate you saved on your machine.

#### NOTE

When you open Service Fabric Explorer, you see a certificate error, as you're using a self-signed certificate. In Edge, you have to click **Details**, and then click the **Go on to the webpage** link. In Chrome, you have to click **Advanced**, and then click the **proceed** link.

## PowerShell

The Service Fabric PowerShell module provides many cmdlets for managing Service Fabric clusters, applications, and services. Use the [Connect-ServiceFabricCluster](#) cmdlet to connect to the secure cluster. The certificate thumbprint and connection endpoint details can be found in the output from a previous step.

```
Connect-ServiceFabricCluster -ConnectionEndpoint mysfcluster.southcentralus.cloudapp.azure.com:19000 `  
-KeepAliveIntervalInSec 10 `  
-X509Credential -ServerCertThumbprint C4C1E541AD512B8065280292A8BA6079C3F26F10 `  
-FindType FindByThumbprint -FindValue C4C1E541AD512B8065280292A8BA6079C3F26F10 `  
-StoreLocation CurrentUser -StoreName My
```

You can also check that you're connected and that the cluster is healthy by using the [Get-ServiceFabricClusterHealth](#) cmdlet.

```
Get-ServiceFabricClusterHealth
```

## Next steps

In this tutorial, you learned how to create a secure Service Fabric cluster in Azure by using PowerShell.

Next, advance to the following tutorial to learn how to deploy an existing application:

[Deploy an existing .NET application with Docker Compose](#)

# Deploy a .NET application in a Windows container to Azure Service Fabric

9/25/2017 • 6 min to read • [Edit Online](#)

This tutorial shows you how to deploy an existing ASP.NET application in a Windows container on Azure.

In this tutorial, you learn how to:

- Create a Docker project in Visual Studio
- Containerize an existing application
- Setup continuous integration with Visual Studio and VSTS

## Prerequisites

1. Install [Docker CE for Windows](#) so that you can run containers on Windows 10.
2. Familiarize yourself with the [Windows 10 Containers quickstart](#).
3. Download the [Fabrikam Fiber CallCenter](#) sample application.
4. Install [Azure PowerShell](#)
5. Install the [Continuous Delivery Tools extension for Visual Studio 2017](#)
6. Create an [Azure subscription](#) and a [Visual Studio Team Services account](#).
7. [Create a cluster on Azure](#)

## Containerize the application

Now that you have a [Service Fabric cluster is running in Azure](#) you are ready to create and deploy a containerized application. To start running our application in a container, we need to add **Docker Support** to the project in Visual Studio. When you add **Docker support** to the application, two things happen. First, a *Dockerfile* is added to the project. This new file describes how the container image is to be built. Then second, a new *docker-compose* project is added to the solution. The new project contains a few docker-compose files. Docker-compose files can be used to describe how the container is run.

More info on working with [Visual Studio Container Tools](#).

### NOTE

If it is the first time you are running Windows container images on your computer, Docker CE must pull down the base images for your containers. The images used in this tutorial are 14 GB. Go ahead and run the following terminal command to pull the base images:

```
docker pull microsoft/mssql-server-windows-developer
docker pull microsoft/aspnet:4.6.2
```

### Add Docker support

Open the [FabrikamFiber.CallCenter.sln](#) file in Visual Studio.

Right-click the **FabrikamFiber.Web** project > **Add > Docker Support**.

### Add support for SQL

This application uses SQL as the data provider, so a SQL server is required to run the application. Reference a SQL

Server container image in our docker-compose.override.yml file.

In Visual Studio, open **Solution Explorer**, find **docker-compose**, and open the file **docker-compose.override.yml**.

Navigate to the `services:` node, add a node named `db:` that defines the SQL Server entry for the container.

```
db:  
  image: microsoft/mssql-server-windows-developer  
  environment:  
    SA_PASSWORD: "Password1"  
    ACCEPT_EULA: "Y"  
  ports:  
    - "1433"  
  healthcheck:  
    test: [ "CMD", "sqlcmd", "-U", "sa", "-P", "Password1", "-Q", "select 1" ]  
    interval: 1s  
    retries: 20
```

#### NOTE

You can use any SQL Server you prefer for local debugging, as long as it is reachable from your host. However, **localdb** does not support `container -> host` communication.

#### WARNING

Running SQL Server in a container does not support persisting data. When the container stops, your data is erased. Do not use this configuration for production.

Navigate to the `fabrikamfiber.web:` node and add a child node named `depends_on:`. This ensures that the `db` service (the SQL Server container) starts before our web application (`fabrikamfiber.web`).

```
fabrikamfiber.web:  
  depends_on:  
    - db
```

#### Update the web config

Back in the **FabrikamFiber.Web** project, update the connection string in the **web.config** file, to point to the SQL Server in the container.

```
<add name="FabrikamFiber-Express" connectionString="Data Source=db,1433;Database=FabrikamFiber;User  
Id=sa;Password=Password1;MultipleActiveResultSets=True" providerName="System.Data.SqlClient" />  
  
<add name="FabrikamFiber-DataWarehouse" connectionString="Data Source=db,1433;Database=FabrikamFiber;User  
Id=sa;Password=Password1;MultipleActiveResultSets=True" providerName="System.Data.SqlClient" />
```

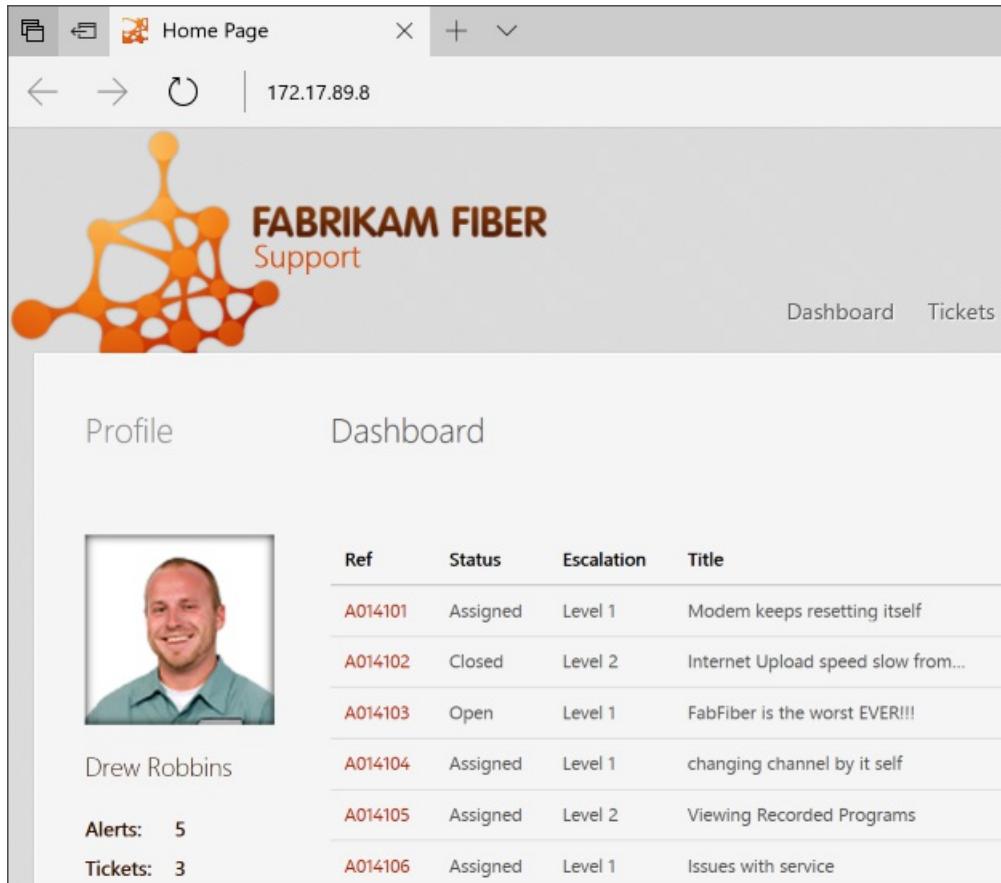
#### NOTE

If you want to use a different SQL Server when building a release build of your web application, add another connection string to your `web.release.config` file.

#### Test your container

Press **F5** to run and debug the application in your container.

Edge opens your application's defined launch page using the IP address of the container on the internal NAT network (typically 172.x.x.x). To learn more about debugging applications in containers using Visual Studio 2017, see [this article](#).



The container is now ready to be built and packaged in a Service Fabric application. Once you have the container image built on your machine, you can push it to any container registry and pull it down to any host to run.

## Get the application ready for the cloud

To get the application ready for running in Service Fabric in Azure, we need to complete two steps:

1. Expose the port where we want to be able to reach our web application in the Service Fabric cluster.
2. Provide a production ready SQL database for our application.

### Expose the port for the app

The Service Fabric cluster we have configured, has port 80 open by default in the Azure Load Balancer, that balances incoming traffic to the cluster. We can expose our container on this port via our docker-compose.yml file.

In Visual Studio, open **Solution Explorer**, find **docker-compose**, and open the file **docker-compose.yml**.

Modify the `fabrikamfiber.web:` node, add a child node named `ports:` .

Add a string entry `- "80:80"`. This is what your docker-compose.yml file should look like:

```
version: '3'

services:
  fabrikamfiber.web:
    image: fabrikamfiber.web
    build:
      context: .\FabrikamFiber.Web
      dockerfile: Dockerfile
    ports:
      - "80:80"
```

## Use a production SQL database

When running in production, we need our data persisted in our database. There is currently no way to guarantee persistent data in a container, therefore you cannot store production data in SQL Server in a container.

We recommend you utilize an Azure SQL Database. To set up and run a managed SQL Server in Azure, visit the [Azure SQL Database Quickstarts](#) article.

### NOTE

Remember to change the connection strings to the SQL server in the **web.release.config** file in the **FabrikamFiber.Web** project.

This application fails gracefully if no SQL database is reachable. You can choose to go ahead and deploy the application with no SQL server.

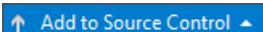
## Deploy with Visual Studio Team Services

To set up deployment using Visual Studio Team Services, you need to install the [Continuous Delivery Tools extension for Visual Studio 2017](#). This extension makes it easy to deploy to Azure by configuring Visual Studio Team Services and get your app deployed to your Service Fabric cluster.

To get started, your code must be hosted in source control. The rest of this section assumes **git** is being used.

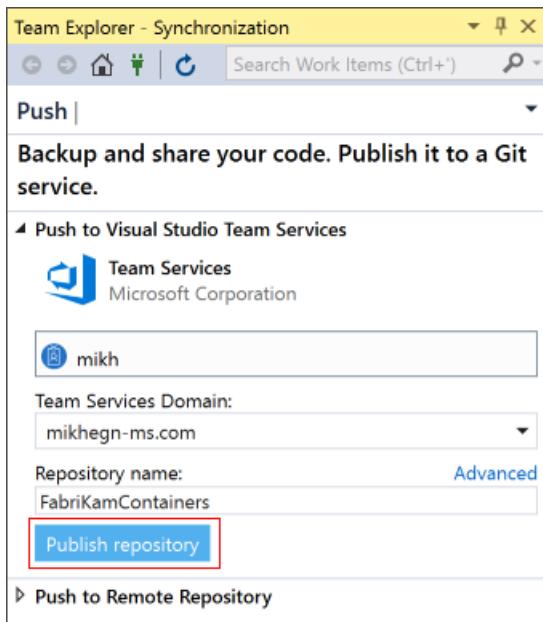
### Set up a VSTS repo

At the bottom-right corner of Visual Studio, click **Add to Source Control > Git** (or whichever option you prefer).



In the *Team Explorer* pane, press **Publish Git Repo**.

Select your VSTS repository name and press **Repository**.



Now that your code is synchronized with a VSTS source repository, you can configure continuous integration and continuous delivery.

### Setup continuous delivery

In *Solution Explorer*, right-click the **solution** > **Configure Continuous Delivery**.

Select the Azure Subscription.

Set **Host Type** to **Service Fabric Cluster**.

Set **Target Host** to the service fabric cluster you created in the previous section.

Choose a **Container Registry** to publish your container to.

#### TIP

Use the **Edit** button to create a container registry.

Press **OK**.

## Configure Continuous Delivery

Create the Team Services and Azure resources needed to continuously deliver your app



Microsoft  
mikh

### Select a Repository Branch

Repository: fabrikamcontainers  
Branch: master

### Select Target Azure Resources

Azure subscription: My Internal Subscription  
Host Type: Service Fabric Cluster  
Target Host: sfcluster56compose  
Cluster Certificate password:  
Container Registry: newRegistry (new) [Edit...](#)

#### Enabling continuous delivery will:

Create an Azure Container Registry if none exist

Create a build definition on Visual Studio Team Services targeting the selected branch

Create a release definition targeting your 'Dev' environment

Kick-off a build and deploy now and whenever you push code

[What is continuous delivery?](#)

This process may take several minutes to complete

OK

Cancel

Once the configuration is completed, your container is deployed to Service Fabric. Whenever you push updates to the repository a new build and release is executed.

#### NOTE

Building the container images take approximately 15 minutes. The first deployment to the Service Fabric cluster causes the base Windows Server Core container images to be downloaded. The download takes additional 5-10 minutes to complete.

Browse to the Fabrikam Call Center application using the url of your cluster: for example,  
<http://mycluster.westeurope.cloudapp.azure.com>

Now that you have containerized and deployed the Fabrikam Call Center solution, you can open the [Azure portal](#) and see the application running in Service Fabric. To try the application, open a web browser and go to the URL of your Service Fabric cluster.

## Next steps

In this tutorial, you learned how to:

- Create a Docker project in Visual Studio
- Containerize an existing application
- Setup continuous integration with Visual Studio and VSTS

In the next part of the tutorial, learn how to set up [monitoring for your container](#).

# Monitor Windows containers on Service Fabric using OMS

9/25/2017 • 6 min to read • [Edit Online](#)

This is part three of a tutorial, and walks you through setting up OMS to monitor your Windows containers orchestrated on Service Fabric.

In this tutorial, you learn how to:

- Configure OMS for your Service Fabric cluster
- Use an OMS workspace to view and query logs from your containers and nodes
- Configure the OMS agent to pick up container and node metrics

## Prerequisites

Before you begin this tutorial, you should:

- Have a cluster on Azure, or [create one with this tutorial](#)
- [Deploy a containerized application to it](#)

## Setting up OMS with your cluster in the Resource Manager template

In the case that you used the [template provided](#) in the first part of this tutorial, it should include the following additions to a generic Service Fabric Azure Resource Manager template. In case the case that you have a cluster of your own that you are looking to set up for monitoring containers with OMS:

- Make the following changes to your Resource Manager template.
- Deploy it using PowerShell to upgrade your cluster by [deploying the template](#). Azure Resource Manager realizes that the resource exists, so will roll it out as an upgrade.

### Adding OMS to your cluster template

Make the following changes in your *template.json*:

1. Add the OMS workspace location and name to your *parameters* section:

```

"omsWorkspacename": {
    "type": "string",
    "defaultValue": "[toLowerCase(concat('sf',uniqueString(resourceGroup().id)))]",
    "metadata": {
        "description": "Name of your OMS Log Analytics Workspace"
    }
},
"omsRegion": {
    "type": "string",
    "defaultValue": "East US",
    "allowedValues": [
        "West Europe",
        "East US",
        "Southeast Asia"
    ],
    "metadata": {
        "description": "Specify the Azure Region for your OMS workspace"
    }
}
}

```

To change the value used for either add the same parameters to your *template.parameters.json* and change the values used there.

2. Add the solution name and the solution to your *variables*:

```

"omsSolutionName": "[Concat('ServiceFabric', '(', parameters('omsWorkspacename'), ')')]",
"omsSolution": "ServiceFabric"

```

3. Add the OMS Microsoft Monitoring Agent as a virtual machine extension. Find virtual machine scale sets resource: *resources > "apiVersion": "[variables('vmssApiVersion')]"*. Under the *properties > virtualMachineProfile > extensionProfile > extensions*, add the following extension description under the *ServiceFabricNode* extension:

```

{
    "name": "[concat(variables('vmNodeType0Name'), 'OMS')]",
    "properties": {
        "publisher": "Microsoft.EnterpriseCloud.Monitoring",
        "type": "MicrosoftMonitoringAgent",
        "typeHandlerVersion": "1.0",
        "autoUpgradeMinorVersion": true,
        "settings": {
            "workspaceId": "[reference(resourceId('Microsoft.OperationalInsights/workspaces/',
parameters('omsWorkspacename')), '2015-11-01-preview').customerId]"
        },
        "protectedSettings": {
            "workspaceKey": "[listKeys(resourceId('Microsoft.OperationalInsights/workspaces/',
parameters('omsWorkspacename')),'2015-11-01-preview').primarySharedKey]"
        }
    }
},

```

4. Add the OMS workspace as an individual resource. In *resources*, after the virtual machine scale sets resource, add the following:

```

{
    "apiVersion": "2015-11-01-preview",
    "location": "[parameters('omsRegion')]",
    "name": "[parameters('omsWorkspacename')]",
    "type": "Microsoft.OperationalInsights/workspaces",
    "properties": {
        "sku": {

```

```

        "name": "Free"
    },
},
"resources": [
{
    "apiVersion": "2015-11-01-preview",
    "name": "",
[concat(variables('applicationDiagnosticsStorageAccountName'),parameters('omsWorkspacename'))]",
    "type": "storageinsightconfigs",
    "dependsOn": [
        "[concat('Microsoft.OperationalInsights/workspaces/', parameters('omsWorkspacename'))]",
        "[concat('Microsoft.Storage/storageAccounts/',
variables('applicationDiagnosticsStorageAccountName'))]"
    ],
    "properties": {
        "containers": [ ],
        "tables": [
            "WADServiceFabric*EventTable",
            "WADWindowsEventLogsTable",
            "WADETWEVENTTable"
        ],
        "storageAccount": {
            "id": "[resourceId('Microsoft.Storage/storageaccounts/',
variables('applicationDiagnosticsStorageAccountName'))]",
            "key": "[listKeys(resourceId('Microsoft.Storage/storageAccounts',
variables('applicationDiagnosticsStorageAccountName')), '2015-06-15').key1]"
        }
    }
},
{
    "apiVersion": "2015-11-01-preview",
    "name": "System",
    "type": "datasources",
    "dependsOn": [
        "[concat('Microsoft.OperationalInsights/workspaces/', parameters('omsWorkspacename'))]"
    ],
    "kind": "WindowsEvent",
    "properties": {
        "eventLogName": "System",
        "eventTypes": [
            {
                "eventType": "Error"
            },
            {
                "eventType": "Warning"
            },
            {
                "eventType": "Information"
            }
        ]
    }
},
]
},
{
    "apiVersion": "2015-11-01-preview",
    "location": "[parameters('omsRegion')]",
    "name": "[variables('omsSolutionName')]",
    "type": "Microsoft.OperationsManagement/solutions",
    "dependsOn": [
        "[concat('Microsoft.OperationalInsights/workspaces/', parameters('OMSWorkspacename'))]"
    ],
    "properties": {
        "workspaceResourceId": "[resourceId('Microsoft.OperationalInsights/workspaces/',
parameters('omsWorkspacename'))]"
    },
    "plan": {
        "name": "[variables('omsSolutionName')]",
        "publisher": "Microsoft"
    }
}
]
```

```

    "publisher": "Microsoft",
    "product": "[Concat('OMSGallery/', variables('omsSolution'))]]",
    "promotionCode": ""
}
},

```

Here is a sample template (used in part one of this tutorial) that has all of these changes that you can reference as needed. These changes will add an OMS Log Analytics workspace to your resource group. The workspace will be configured to pick up Service Fabric platform events from the storage tables configured with the [Windows Azure Diagnostics](#) agent. The OMS agent (Microsoft Monitoring Agent) has also been added to each node in your cluster as a virtual machine extension - this means that as you scale your cluster, the agent is automatically configured on each machine and hooked up to the same workspace.

Deploy the template with your new changes to upgrade your current cluster. You should see the OMS resources in your resource group once this has completed. When the cluster is ready, deploy your containerized application to it. In the next step, we will set up monitoring the containers.

## Add the Container Monitoring Solution to your OMS workspace

To set up the Container solution in your workspace, search for *Container Monitoring Solution* and create a Containers resource (under the Monitoring + Management category).

The screenshot shows the Microsoft Azure Marketplace interface. The left sidebar navigation includes 'New', 'Dashboard', 'All resources', 'Resource groups', 'App Services', 'SQL databases', 'SQL data warehouses', 'Azure Cosmos DB', 'Virtual machines', 'Load balancers', 'Storage accounts', 'Virtual networks', 'Azure Active Directory', 'Monitor', 'Advisor', 'Security Center', 'Billing', 'Help + support', and 'Subscriptions'. The main search bar contains 'Container Monitoring Solution'. The search results table has columns for NAME, PUBLISHER, and CATEGORY. The first result, 'Container Monitoring Solution' by Microsoft, is selected and highlighted with a red border. The right pane displays a detailed description of the solution, mentioning it helps see container usage and diagnose failures across public and private cloud environments. It lists features like seeing all container hosts, troubleshooting logs, finding noisy neighbors, and viewing CPU and memory usage. It also mentions data collected such as container name, computer name, ID, image, state, logs, and audit trail. A preview image of the solution's dashboard is shown, featuring various metrics and charts. At the bottom of the right pane, there are links for 'PUBLISHER', 'USEFUL LINKS', and a prominent 'Create' button.

When prompted for the *OMS Workspace*, select the workspace that was created in your resource group, and click **Create**. This adds a *Container Monitoring Solution* to your workspace, will automatically cause the OMS agent deployed by the template to start collecting docker logs and stats.

Navigate back to your *resource group*, where you should now see the newly added monitoring solution. If you click into it, the landing page should show you the number of container images you have running.

*Note that I ran 5 instances of my fabrikam container from part two of the tutorial*

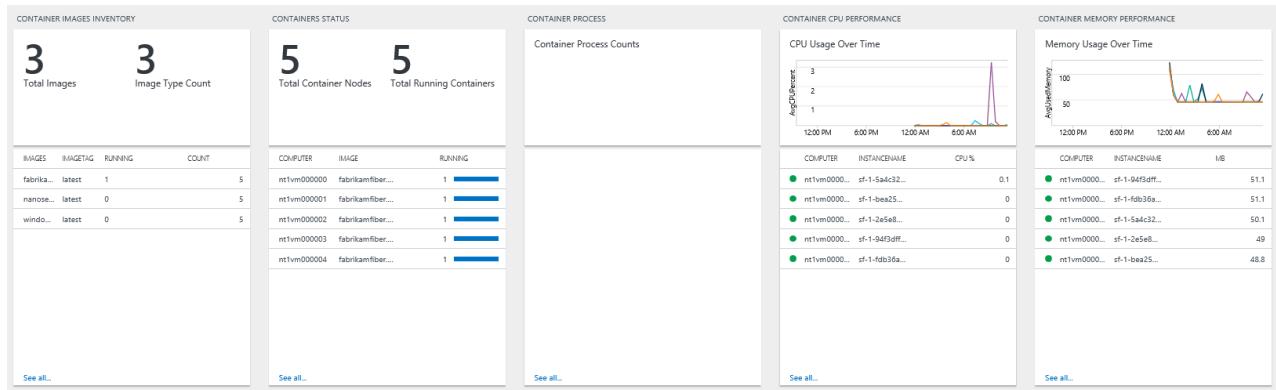
The screenshot shows the Azure Container Monitoring Solution dashboard in the OMS Portal. Key details include:

- Solution:** Containers([ContainerMonitoring](#))
- Type:** Microsoft.OperationsManagement/solutions
- Subscription name:** [dakapurtutorial](#)
- Location:** East US
- Containers:** 1 Running (5 Total)
- Solution Resources:** 1 Container

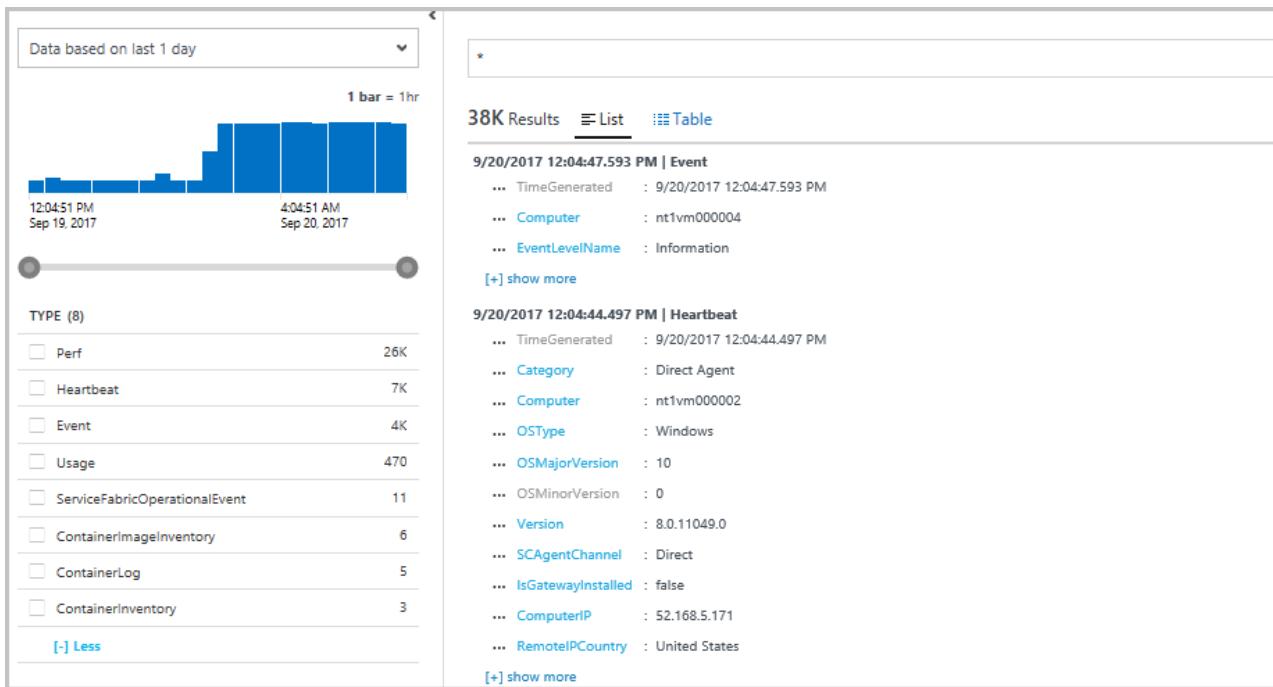
Clicking into the **Container Monitor Solution** will take you to a more detailed dashboard, which allows you to scroll through multiple panels as well as run queries in Log Analytics.

*Note that as of September, 2017, the solution is going through some updates - ignore any errors you may get about Kubernetes events as we work on integrating multiple orchestrators into the same solution.*

Since the agent is picking up docker logs, it defaults to showing *stdout* and *stderr*. If you scroll to the right, you will see container image inventory, status, metrics, and sample queries that you could run to get more helpful data.



Clicking into any of these panels will take you to the Log Analytics query that is generating the displayed value. Change the query to \*\*\* to see all the different kinds of logs that are being picked up. From here, you can query or filter for container performance, logs, or look at Service Fabric platform events. Your agents are also constantly emitting a heartbeat from each node, that you can look at to make sure data is still being gathered from all your machines if your cluster configuration changes.



## Configure OMS agent to pick up performance counters

Another benefit of using the OMS agent is the ability to change the performance counters you want to pick up through the OMS UI experience, rather than having to configure the Azure diagnostics agent and do a Resource Manager template based upgrade each time. To do this, click on **OMS Portal** on the landing page of your Container Monitoring (or Service Fabric) solution.

The screenshot shows the OMS Portal landing page. At the top, there are buttons for 'OMS Portal' (which is highlighted with a red box) and 'Delete'. Below that is a 'Essentials' dropdown menu. The main area is titled 'Summary' and contains a section for 'Container Monitoring Solution'. It features a large blue circle with the number '5 TOTAL' in the center, and below it, a bar chart showing 'Running' status with the value '5'. To the right, there's a 'Solution Resources' section showing '1' container and a link to 'Containers(https://go.microsoft.com/fwlink/?linkid=846121)'.

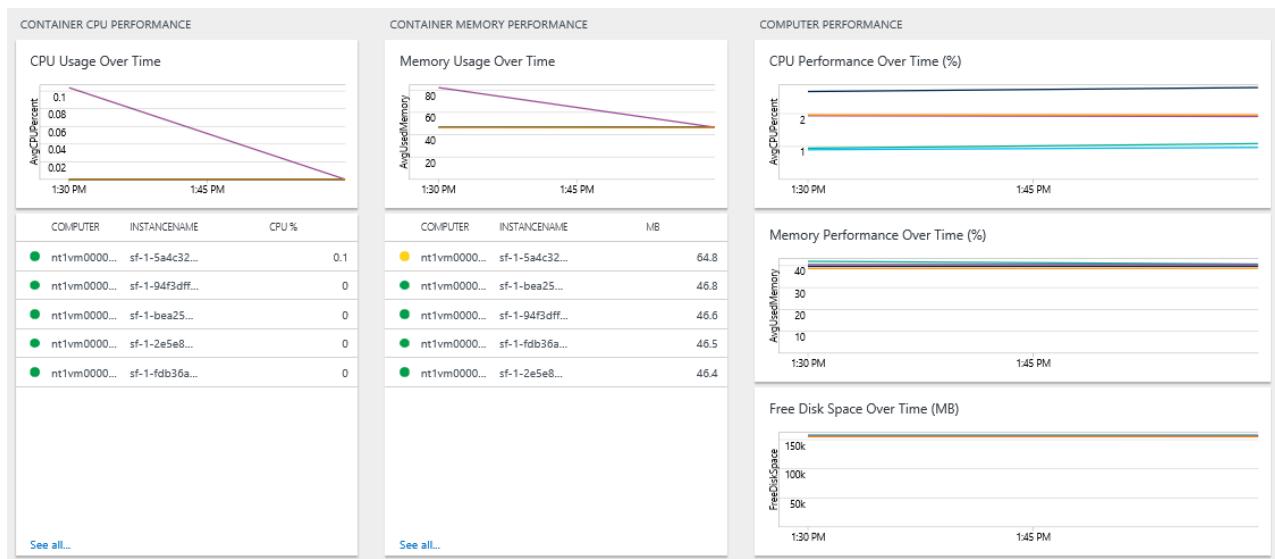
This will take you to your workspace in the OMS portal, where you can view your solutions, create custom dashboards, as well as configure the OMS agent.

- Click on the **cog wheel** on the top right corner of your screen to open up the *Settings* menu.
- Click on **Connected Sources > Windows Servers** to verify that you have *5 Windows Computers Connected*.
- Click on **Data > Windows Performance Counters** to search for and add new performance counters. Here you will see a list of recommendations from OMS for perf counters you could collect as well as the option to search for other counters. Click **Add the selected performance counters** to start collecting the suggested metrics.

The screenshot shows the OMS Settings page. On the left, there's a sidebar with options like Solutions, Connected Sources, Data (which is selected and highlighted with a red box), Computer Groups, Accounts, Alerts, and Preview Features. The main area is titled "Windows Performance Counters" and contains a list of performance counters with checkboxes. A red box highlights the "Add the selected performance counters" button and the first few items in the list.

Back in the Azure portal, **refresh** your Container Monitoring Solution in a few minutes, and you should start to see *Computer Performance* data coming in. This will help you understand how your resources are being used. You can also use these metrics to make appropriate decisions about scaling your cluster, or to confirm if a cluster is balancing your load as expected.

*Note: Make sure your time filters are set appropriately for you to consume these metrics.*



## Next steps

In this tutorial, you learned how to:

- Configure OMS for your Service Fabric cluster
- Use an OMS workspace to view and query logs from your containers and nodes
- Configure the OMS agent to pick up container and node metrics

Now that you have set up monitoring for your containerized application, try the following:

- Set up the OMS for a Linux cluster, following similar steps as above. Reference [this template](#) to make changes in your Resource Manager template.
- Configure OMS to set up [automated alerting](#) to aid in detecting and diagnostics.
- Explore Service Fabric's list of [recommended performance counters](#) to configure for your clusters.
- Get familiarized with the [log search and querying](#) features offered as part of Log Analytics.

# Create container images for Service Fabric

10/4/2017 • 5 min to read • [Edit Online](#)

This tutorial is part one of a tutorial series that demonstrates how to use containers in a Linux Service Fabric cluster. In this tutorial, a multi-container application is prepared for use with Service Fabric. In subsequent tutorials, these images are used as part of a Service Fabric application. In this tutorial you learn how to:

- Clone application source from GitHub
- Create a container image from the application source
- Deploy an Azure Container Registry (ACR) instance
- Tag a container image for ACR
- Upload the image to ACR

In this tutorial series, you learn how to:

- Create container images for Service Fabric
- [Build and Run a Service Fabric Application with Containers](#)
- [How failover and scaling are handled in Service Fabric](#)

## Prerequisites

- Linux development environment set up for Service Fabric. Follow the instructions [here](#) to set up your Linux environment.
- This tutorial requires that you are running the Azure CLI version 2.0.4 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI 2.0](#).
- Additionally, it requires that you have an Azure subscription available. For more information on a free trial version, go [here](#).

## Get application code

The sample application used in this tutorial is a voting app. The application consists of a front-end web component and a back-end Redis instance. The components are packaged into container images.

Use git to download a copy of the application to your development environment.

```
git clone https://github.com/Azure-Samples/service-fabric-dotnet-containers.git  
cd service-fabric-dotnet-containers/Linux/container-tutorial/
```

The 'container-tutorial' directory contains a folder named 'azure-vote'. This 'azure-vote' folder contains the front-end source code and a Dockerfile to build the front-end. The 'container-tutorial' directory also contains the 'redis' directory which has the Dockerfile to build the redis image. These directories contain the necessary assets for this tutorial set.

## Create container images

Inside the 'azure-vote' directory, run the following command to build the image for the front-end web component. This command uses the Dockerfile in this directory to build the image.

```
docker build -t azure-vote-front .
```

Inside, the 'redis' directory, run the following command to build the image for the redis backend. This command uses the Dockerfile in the directory to build the image.

```
docker build -t azure-vote-back .
```

When completed, use the [docker images](#) command to see the created images.

```
docker images
```

Notice that four images have been downloaded or created. The *azure-vote-front* image contains the application. It was derived from a *python* image from Docker Hub. The Redis image was downloaded from Docker Hub.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
azure-vote-back	latest	bf9a858a9269	3 seconds ago	107MB
azure-vote-front	latest	052c549a75bf	About a minute ago	708MB
redis	latest	9813a7e8fcc0	2 days ago	107MB
tiangolo/uwsgi-nginx-flask	python3.6	590e17342131	5 days ago	707MB

## Deploy Azure Container Registry

First run the [az login](#) command to log in to your Azure account.

Next, use the [az account](#) command to choose your subscription to create the Azure Container registry.

```
az account set --subscription <subscription_id>
```

When deploying an Azure Container Registry, you first need a resource group. An Azure resource group is a logical container into which Azure resources are deployed and managed.

Create a resource group with the [az group create](#) command. In this example, a resource group named *myResourceGroup* is created in the *westus* region. Please choose the resource group in a region near you.

```
az group create --name myResourceGroup --location westus
```

Create an Azure Container registry with the [az acr create](#) command. The name of a Container Registry **must be unique**.

```
az acr create --resource-group myResourceGroup --name <acrName> --sku Basic --admin-enabled true
```

Throughout the rest of this tutorial, we use "acrname" as a placeholder for the container registry name that you chose.

## Log in to your container registry

Log in to your ACR instance before pushing images to it. Use the [az acr login](#) command to complete the operation. Provide the unique name given to the container registry when it was created.

```
az acr login --name <acrName>
```

The command returns a 'Login Succeeded' message once completed.

## Tag container images

Each container image needs to be tagged with the loginServer name of the registry. This tag is used for routing when pushing container images to an image registry.

To see a list of current images, use the [docker images](#) command.

```
docker images
```

Output:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
azure-vote-back	latest	bf9a858a9269	3 seconds ago	107MB
azure-vote-front	latest	052c549a75bf	About a minute ago	708MB
redis	latest	9813a7e8fcc0	2 days ago	107MB
tiangolo/uwsgi-nginx-flask	python3.6	590e17342131	5 days ago	707MB

To get the loginServer name, run the following command:

```
az acr show --name <acrName> --query loginServer --output table
```

Now, tag the *azure-vote-front* image with the loginServer of the container registry. Also, add `:v1` to the end of the image name. This tag indicates the image version.

```
docker tag azure-vote-front <acrLoginServer>/azure-vote-front:v1
```

Next, tag the *azure-vote-back* image with the loginServer of the container registry. Also, add `:v1` to the end of the image name. This tag indicates the image version.

```
docker tag azure-vote-back <acrLoginServer>/azure-vote-back:v1
```

Once tagged, run 'docker images' to verify the operation.

Output:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
azure-vote-back	latest	bf9a858a9269	22 minutes ago	107MB
<acrName>.azurecr.io/azure-vote-back	v1	bf9a858a9269	22 minutes ago	107MB
azure-vote-front	latest	052c549a75bf	23 minutes ago	708MB
<acrName>.azurecr.io/azure-vote-front	v1	052c549a75bf	23 minutes ago	708MB
redis	latest	9813a7e8fcc0	2 days ago	107MB
tiangolo/uwsgi-nginx-flask	python3.6	590e17342131	5 days ago	707MB

## Push images to registry

Push the *azure-vote-front* image to the registry.

Using the following example, replace the ACR loginServer name with the loginServer from your environment.

```
docker push <acrLoginServer>/azure-vote-front:v1
```

Push the *azure-vote-back* image to the registry.

Using the following example, replace the ACR loginServer name with the loginServer from your environment.

```
docker push <acrLoginServer>/azure-vote-back:v1
```

The docker push commands take a couple of minutes to complete.

## List images in registry

To return a list of images that have been pushed to your Azure Container registry, use the [az acr repository list](#) command. Update the command with the ACR instance name.

```
az acr repository list --name <acrName> --output table
```

Output:

```
Result
-----
azure-vote-back
azure-vote-front
```

At tutorial completion, the container image has been stored in a private Azure Container Registry instance. This image is deployed from ACR to a Service Fabric cluster in subsequent tutorials.

## Next steps

In this tutorial, an application was pulled from Github and container images were created and pushed to a registry. The following steps were completed:

- Clone application source from GitHub
- Create a container image from the application source
- Deploy an Azure Container Registry (ACR) instance
- Tag a container image for ACR
- Upload the image to ACR

Advance to the next tutorial to learn about packaging containers into a Service Fabric application using Yeoman.

[Package and deploy containers as a Service Fabric application](#)

# Package and deploy containers as a Service Fabric application

10/3/2017 • 8 min to read • [Edit Online](#)

This tutorial is part two in a series. In this tutorial, a template generator tool (Yeoman) is used to generate a Service Fabric application definition. This application can then be used to deploy containers to Service Fabric. In this tutorial you learn how to:

- Install Yeoman
- Create an application package using Yeoman
- Configure settings in the application package for use with containers
- Build the application
- Deploy and run the application
- Clean up the application

## Prerequisites

- The container images pushed to Azure Container Registry created in [Part 1](#) of this tutorial series are used.
- Linux development environment is [set up](#).

## Install Yeoman

Service fabric provides scaffolding tools to help create applications from terminal using Yeoman template generator. Follow the steps below to ensure you have the Yeoman template generator.

1. Install nodejs and NPM on your machine. Note that, Mac OSX users will have to use the package manager Homebrew

```
sudo apt-get install npm && sudo apt install nodejs-legacy
```

2. Install Yeoman template generator on your machine from NPM

```
sudo npm install -g yo
```

3. Install the Service Fabric Yeoman container generator

```
sudo npm install -g generator-azuresfcontainer
```

## Package a Docker image container with Yeoman

1. To create a Service Fabric container application, in the 'container-tutorial' directory of the cloned repository, run the following command.

```
yo azuresfcontainer
```

2. Name your application "TestContainer" and name the application service "azurevotefront".

3. Provide the container image path in ACR for the frontend repo - for example 'test.azurecr.io/azure-vote-front:v1'.
4. Press Enter to leave the Commands section empty.
5. Specify an instance count of 1.

The following shows the input and output of running the yo command:

```
? Name your application TestContainer
? Name of the application service: azurevotefront
? Input the Image Name: <acrName>.azurecr.io/azure-vote-front:v1
? Commands:
? Number of instances of guest container application: 1
  create TestContainer/TestContainer/ApplicationManifest.xml
  create TestContainer/TestContainer/azurevotefrontPkg/ServiceManifest.xml
  create TestContainer/TestContainer/azurevotefrontPkg/config/Settings.xml
  create TestContainer/TestContainer/azurevotefrontPkg/code/Dummy.txt
  create TestContainer/install.sh
  create TestContainer/uninstall.sh
```

To add another container service to an application already created using yeoman, perform the following steps:

1. Change directory to the **TestContainer** directory
2. Run `yo azuresfcontainer:AddService`
3. Name the service 'azurevoteback'
4. Provide the container image path in ACR for the backend repo - for example 'test.azurecr.io/azure-vote-back:v1'
5. Press Enter to leave the Commands section empty
6. Specify an instance count of "1".

The entries for adding the service used are all shown:

```
? Name of the application service: azurevoteback
? Input the Image Name: <acrName>.azurecr.io/azure-vote-back:v1
? Commands:
? Number of instances of guest container application: 1
  create TestContainer/azurevotebackPkg/ServiceManifest.xml
  create TestContainer/azurevotebackPkg/config/Settings.xml
  create TestContainer/azurevotebackPkg/code/Dummy.txt
```

For the remainder of this tutorial, we are working in the **TestContainer** directory.

## Configure the application manifest with credentials for Azure Container Registry

For Service Fabric to pull the container images from Azure Container Registry, we need to provide the credentials in the **ApplicationManifest.xml**.

Log in to your ACR instance. Use the [az acr login](#) command to complete the operation. Provide the unique name given to the container registry when it was created.

```
az acr login --name <acrName>
```

The command returns a **Login Succeeded** message once completed.

Next, run the following command to get the password of your container registry. This password is used by Service Fabric to authenticate with ACR to pull the container images.

```
az acr credential show -n <acrName> --query passwords[0].value
```

In the **ApplicationManifest.xml**, add the code snippet under the **ServiceManifestImport** element for each of the services. Insert your **acrName** for the **AccountName** field and the password returned from the previous command is used for the **Password** field. A full **ApplicationManifest.xml** is provided at the end of this document.

```
<Policies>
  <ContainerHostPolicies CodePackageRef="Code">
    <RepositoryCredentials AccountName="<acrName>" Password="<password>" PasswordEncrypted="false"/>
  </ContainerHostPolicies>
</Policies>
```

## Configure communication and container port-to-host port mapping

### Configure communication port

Configure an HTTP endpoint so clients can communicate with your service. Open the `./TestContainer/azurevotefrontPkg/ServiceManifest.xml` file and declare an endpoint resource in the **ServiceManifest** element. Add the protocol, port, and name. For this tutorial, the service listens on port 80.

```
<Resources>
  <Endpoints>
    <!-- This endpoint is used by the communication listener to obtain the port on which to
        listen. Please note that if your service is partitioned, this port is shared with
        replicas of different partitions that are placed in your code. -->
    <Endpoint Name="azurevotefrontTypeEndpoint" UriScheme="http" Port="80" Protocol="http"/>
  </Endpoints>
</Resources>
```

Similarly, modify the Service Manifest for the backend service. For this tutorial, the redis default of 6379 is maintained.

```
<Resources>
  <Endpoints>
    <!-- This endpoint is used by the communication listener to obtain the port on which to
        listen. Please note that if your service is partitioned, this port is shared with
        replicas of different partitions that are placed in your code. -->
    <Endpoint Name="azurevotebackTypeEndpoint" UriScheme="http" Port="6379" Protocol="http"/>
  </Endpoints>
</Resources>
```

Providing the **UriScheme** automatically registers the container endpoint with the Service Fabric Naming service for discoverability. A full ServiceManifest.xml example file for the backend service is provided at the end of this article as an example.

### Map container ports to a service

In order to expose the containers in the cluster, we also need to create a port binding in the 'ApplicationManifest.xml'. The **PortBinding** policy references the **Endpoints** we defined in the **ServiceManifest.xml** files. Incoming requests to these endpoints get mapped to the container ports that are opened and bounded here. In the **ApplicationManifest.xml** file, add the following code to bind port 80 and 6379 to the endpoints. A full **ApplicationManifest.xml** is available at the end of this document.

```
<ContainerHostPolicies CodePackageRef="Code">
  <PortBinding ContainerPort="80" EndpointRef="azurevotefrontTypeEndpoint"/>
</ContainerHostPolicies>
```

```
<ContainerHostPolicies CodePackageRef="Code">
  <PortBinding ContainerPort="6379" EndpointRef="azurevotebackTypeEndpoint"/>
</ContainerHostPolicies>
```

## Add a DNS name to the backend service

For Service Fabric to assign this DNS name to the backend service, the name needs to be specified in the **ApplicationManifest.xml**. Add the **ServiceDnsName** attribute to the **Service** element as shown:

```
<Service Name="azurevoteback" ServiceDnsName="redisbackend.testapp">
  <StatelessService ServiceTypeName="azurevotebackType" InstanceCount="1">
    <SingletonPartition/>
  </StatelessService>
</Service>
```

The frontend service reads an environment variable to know the DNS name of the Redis instance. The environment variable is defined in the Dockerfile as shown:

```
ENV REDIS redisbackend.testapp
```

The python script that renders the front end uses this DNS name to resolve and connect to the backend redis store as shown:

```
# Get DNS Name
redis_server = os.environ['REDIS']

# Connect to the Redis store
r = redis.StrictRedis(host=redis_server, port=6379, db=0)
```

At this point in the tutorial, the template for a Service Package application is available for deployment to a cluster. In the subsequent tutorial, this application is deployed and ran in a Service Fabric cluster.

## Create a Service Fabric cluster

To deploy the application to a cluster in Azure, use your own cluster, or use a Party Cluster.

Party clusters are free, limited-time Service Fabric clusters hosted on Azure. It is maintained by the Service Fabric team where anyone can deploy applications and learn about the platform. To get access to a Party Cluster, [follow the instructions](#).

For information about creating your own cluster, see [Create your first Service Fabric cluster on Azure](#).

## Build and deploy the application to the cluster

You can deploy the application the Azure cluster using the Service Fabric CLI. If Service Fabric CLI is not installed on your machine, follow instructions [here](#) to install it.

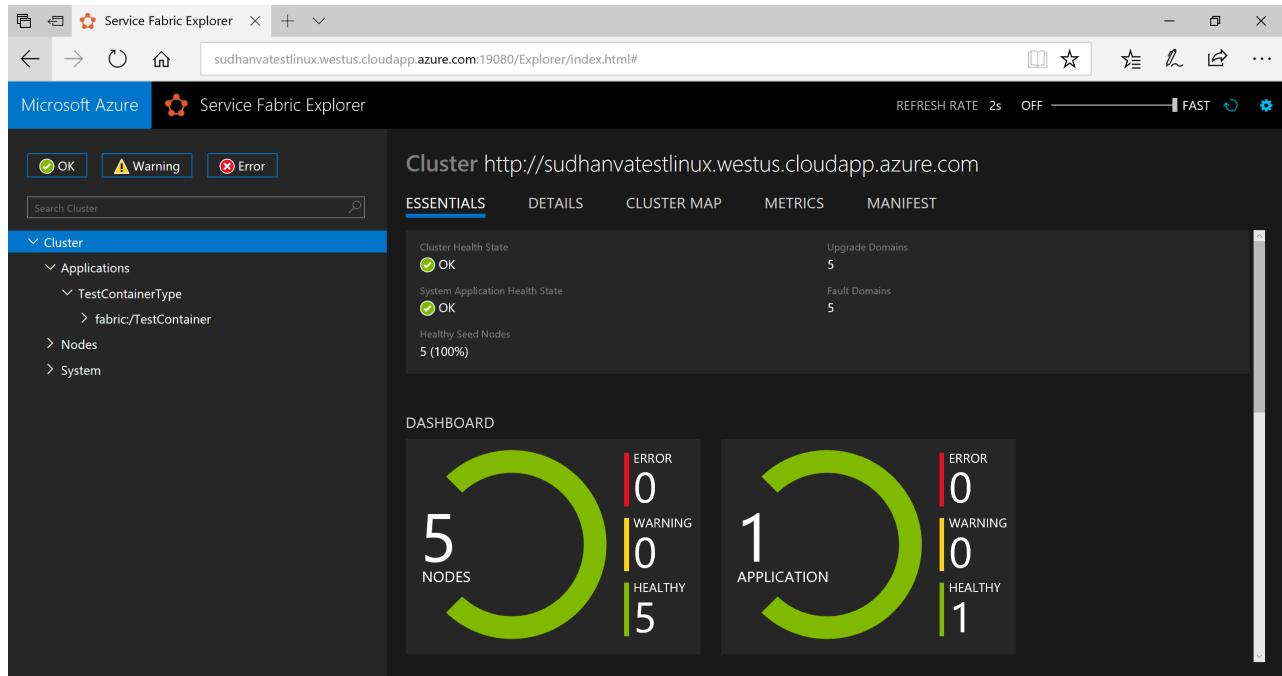
Connect to the Service Fabric cluster in Azure.

```
sfctl cluster select --endpoint http://lin4hjim3l4.westus.cloudapp.azure.com:19080
```

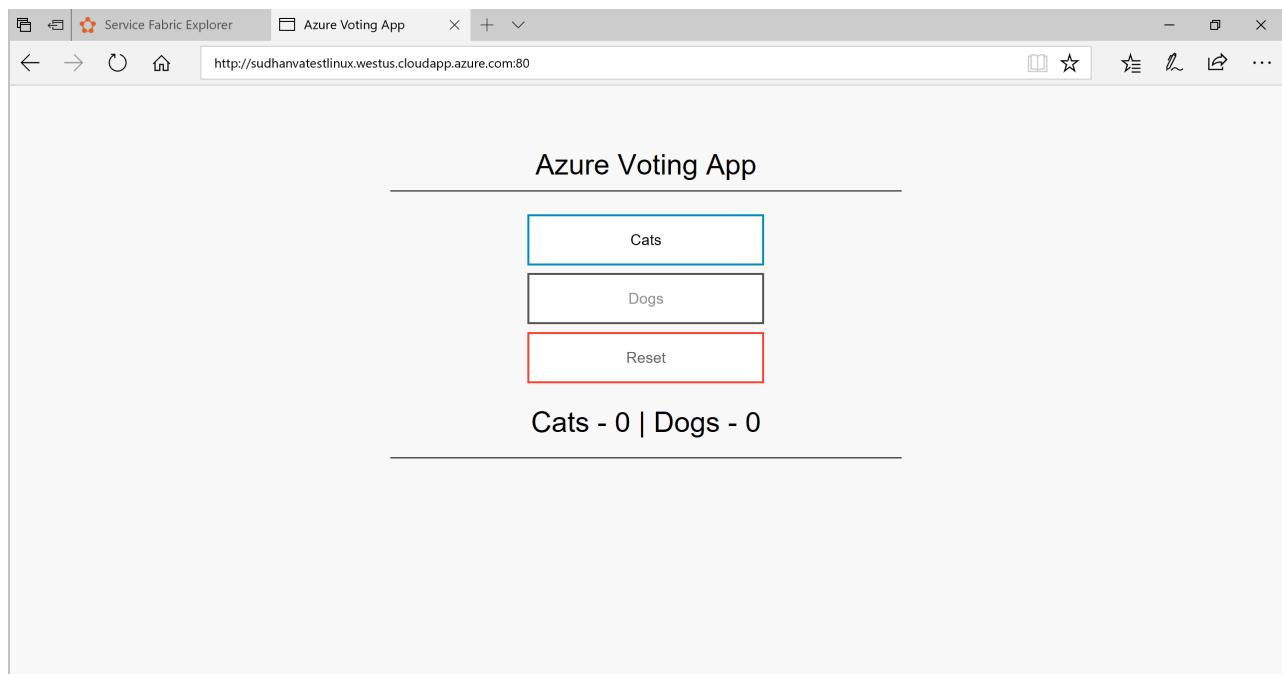
Use the install script provided in the **TestContainer** directory to copy the application package to the cluster's image store, register the application type, and create an instance of the application.

```
./install.sh
```

Open a browser and navigate to Service Fabric Explorer at <http://lin4hjim3l4.westus.cloudapp.azure.com:19080/Explorer>. Expand the Applications node and note that there is an entry for your application type and another for the instance.



In order to connect to the running application, open a web browser and go to the cluster url - for example <http://lin0823ryf2he.cloudapp.azure.com:80>. You should see the Voting application in the web UI.



## Clean up

Use the uninstall script provided in the template to delete the application instance from the cluster and unregister the application type. This command takes some time to clean up the instance and the 'install.sh' command cannot be run immediately after this script.

```
./uninstall.sh
```

## Examples of completed manifests

### ApplicationManifest.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ApplicationManifest ApplicationTypeName="TestContainerType" ApplicationTypeVersion="1.0.0"
xmlns="http://schemas.microsoft.com/2011/01/fabric" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <ServiceManifestImport>
        <ServiceManifestRef ServiceManifestName="azurevotefrontPkg" ServiceManifestVersion="1.0.0"/>
        <Policies>
            <ContainerHostPolicies CodePackageRef="Code">
                <RepositoryCredentials AccountName="myaccountname" Password="<password>" PasswordEncrypted="false"/>
                <PortBinding ContainerPort="80" EndpointRef="azurevotefrontTypeEndpoint"/>
            </ContainerHostPolicies>
        </Policies>
    </ServiceManifestImport>
    <ServiceManifestImport>
        <ServiceManifestRef ServiceManifestName="azurevotebackPkg" ServiceManifestVersion="1.0.0"/>
        <Policies>
            <ContainerHostPolicies CodePackageRef="Code">
                <RepositoryCredentials AccountName="myaccountname" Password="<password>" PasswordEncrypted="false"/>
                <PortBinding ContainerPort="6379" EndpointRef="azurevotebackTypeEndpoint"/>
            </ContainerHostPolicies>
        </Policies>
    </ServiceManifestImport>
    <DefaultServices>
        <Service Name="azurevotefront">
            <StatelessService ServiceTypeName="azurevotefrontType" InstanceCount="1">
                <SingletonPartition/>
            </StatelessService>
        </Service>
        <Service Name="azurevoteback" ServiceDnsName="redisbackend.testapp">
            <StatelessService ServiceTypeName="azurevotebackType" InstanceCount="1">
                <SingletonPartition/>
            </StatelessService>
        </Service>
    </DefaultServices>
</ApplicationManifest>
```

### Front-end ServiceManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<ServiceManifest Name="azurevotefrontPkg" Version="1.0.0"
    xmlns="http://schemas.microsoft.com/2011/01/fabric"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >

    <ServiceTypes>
        <StatelessServiceType ServiceTypeName="azurevotefrontType" UseImplicitHost="true">
        </StatelessServiceType>
    </ServiceTypes>

    <CodePackage Name="code" Version="1.0.0">
        <EntryPoint>
            <ContainerHost>
                <ImageName>my.azurecr.io/azure-vote-front:v1</ImageName>
                <Commands></Commands>
            </ContainerHost>
        </EntryPoint>
        <EnvironmentVariables>
        </EnvironmentVariables>
    </CodePackage>

    <Resources>
        <Endpoints>
            <!-- This endpoint is used by the communication listener to obtain the port on which to
                listen. Please note that if your service is partitioned, this port is shared with
                replicas of different partitions that are placed in your code. -->
            <Endpoint Name="azurevotefrontTypeEndpoint" UriScheme="http" Port="8080" Protocol="http"/>
        </Endpoints>
    </Resources>
</ServiceManifest>

```

## Redis ServiceManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<ServiceManifest Name="azurevotebackPkg" Version="1.0.0"
    xmlns="http://schemas.microsoft.com/2011/01/fabric"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >

    <ServiceTypes>
        <StatelessServiceType ServiceTypeName="azurevotebackType" UseImplicitHost="true">
        </StatelessServiceType>
    </ServiceTypes>

    <CodePackage Name="code" Version="1.0.0">
        <EntryPoint>
            <ContainerHost>
                <ImageName>my.azurecr.io/azure-vote-back:v1</ImageName>
                <Commands></Commands>
            </ContainerHost>
        </EntryPoint>
        <EnvironmentVariables>
        </EnvironmentVariables>
    </CodePackage>
    <Resources>
        <Endpoints>
            <!-- This endpoint is used by the communication listener to obtain the port on which to
                listen. Please note that if your service is partitioned, this port is shared with
                replicas of different partitions that are placed in your code. -->
            <Endpoint Name="azurevotebackTypeEndpoint" UriScheme="http" Port="6379" Protocol="http"/>
        </Endpoints>
    </Resources>
</ServiceManifest>

```

## Next steps

In this tutorial, multiple containers were packaged into a Service Fabric application using Yeoman. This application was then deployed and run on a Service Fabric cluster. The following steps were completed:

- Install Yeoman
- Create an application package using Yeoman
- Configure settings in the application package for use with containers
- Build the application
- Deploy and run the application
- Clean up the application

Advance to the next tutorial to learn about failover and scaling of the application in Service Fabric.

[Learn about failover and scaling applications](#)

# Demonstrate fail over and scaling of container services with Service Fabric

9/25/2017 • 2 min to read • [Edit Online](#)

This tutorial is part three of a series. In this tutorial, you learn how failover is handled in Service Fabric container applications. Additionally, you learn how to scale containers. In this tutorial, you:

- Learn about container failover in a Service Fabric cluster
- Scale the web front-end containers in an application

## Prerequisites

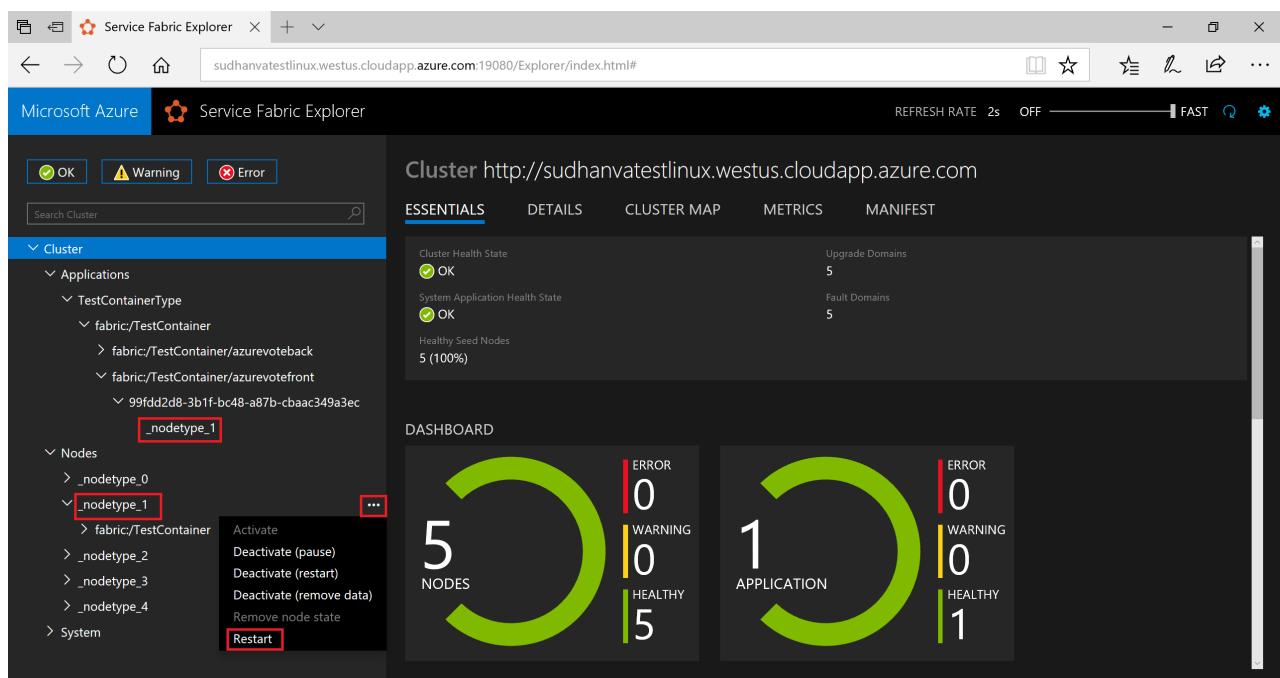
The application from [Part 2](#) is running in an active Service Fabric cluster.

## Fail over a container in a cluster

Service Fabric makes sure your container instances automatically moves to other nodes in the cluster, should a failure occur. You can also manually drain a node of containers and move them gracefully to other nodes in the cluster. You have multiple ways of scaling your services, in this example, we are using Service Fabric Explorer.

To fail over the front-end container, do the following steps:

1. Open Service Fabric Explorer in your cluster - for example, <http://lin4hjim314.westus.cloudapp.azure.com:19080>.
2. Click on the **fabric:/TestContainer/azurevotefront** node in the tree-view and expand the partition node (represented by a GUID). Notice the node name in the treeview, which shows you the nodes that container is currently running on - for example `_nodetype_1`
3. Expand the **Nodes** node in the treeview. Click on the ellipsis (three dots) next to the node, which is running the container.
4. Choose **Restart** to restart that node and confirm the restart action. The restart causes the container to fail over to another node in the cluster.



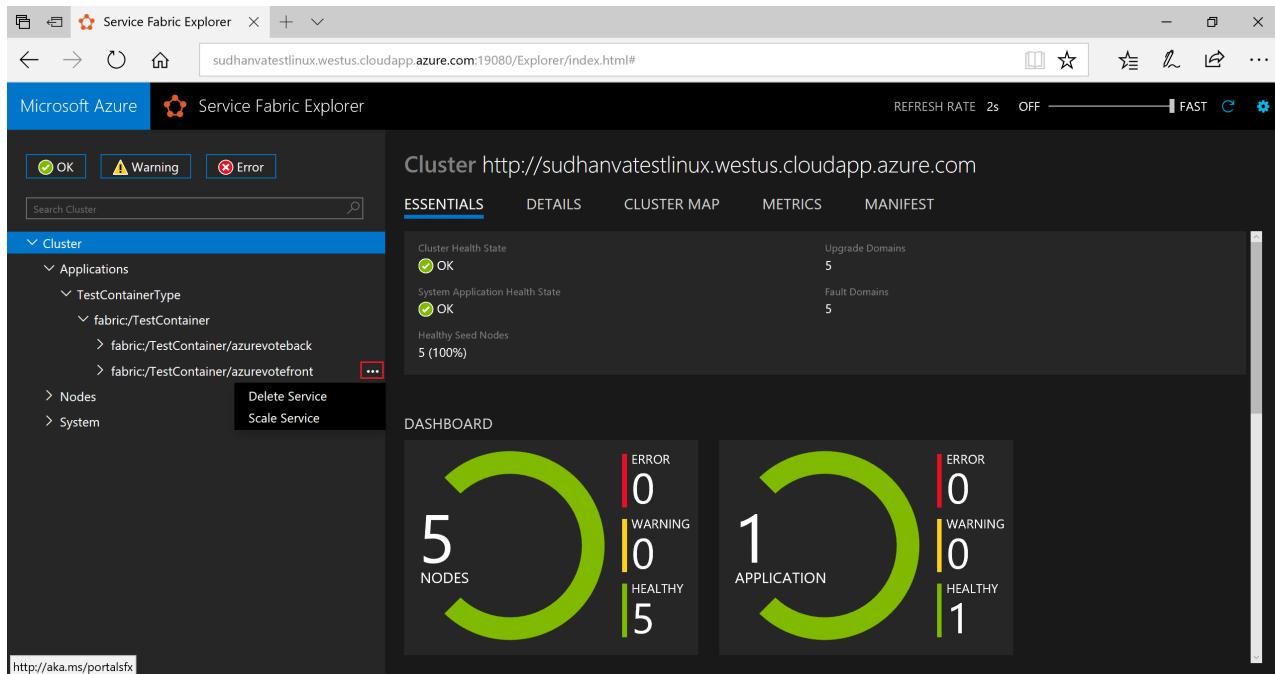
Notice how the node name indicating where the front-end containers runs, now changes to another node in the cluster. After a few moments, you should be able to browse to the application again and see the application now running on a different node.

## Scale containers and services in a cluster

Service Fabric containers can be scaled across a cluster to accommodate for the load on the services. You scale a container by changing the number of instances running in the cluster.

To scale the web front-end, do the following steps:

1. Open Service Fabric Explorer in your cluster - for example, <http://lin4hjim314.westus.cloudapp.azure.com:19080>.
2. Click on the ellipsis (three dots) next to the **fabric:/TestContainer/azurevotefront** node in the tree view and choose **Scale Service**.



You can now choose to scale the number of instances of the web front-end.

1. Change the number to **2** and click **Scale Service**.
2. Click on the **fabric:/TestContainer/azurevotefront** node in the tree view and expand the partition node (represented by a GUID).

The screenshot shows the Microsoft Azure Service Fabric Explorer interface. On the left, a tree view displays the cluster structure under 'Cluster' > 'Applications' > 'TestContainerType' > 'fabric/TestContainer' > 'fabric/TestContainer/azurevoteback' > 'fabric/TestContainer/azurevotefront'. Under 'fabric/TestContainer/azurevotefront', there are two nodes: '\_nodetype\_1' and '\_nodetype\_4'. The main pane shows the 'ESSENTIALS' tab for the service 'Service fabric:/TestContainer/azurevotefront'. The service has the name 'fabric:/TestContainer/azurevotefront', is of type 'azurevotefrontType', version 1.0.0, and is active. It has an instance count of 2. The 'HEALTHY EVALUATIONS' section shows 'No items to display.' The 'PARTITIONS' section also shows 'No items to display.'

You can now see that the service has two instances. In the tree view, you see which nodes the instances run on.

By this simple management task, we doubled the resources available for our front-end service to process user load. It's important to understand that you do not need multiple instances of a service to have it run reliably. If a service fails, Service Fabric makes sure a new service instance runs in the cluster.

## Next steps

In this tutorial, container failover was demonstrated as well as scaling of an application. The following steps were completed:

- Learn about container failover in a Service Fabric cluster
- Scale the web front-end containers in an application

In this tutorial series, you learned how to:

- Create container images
- Push container images to Azure Container Registry
- Package Containers for Service Fabric using Yeoman
- Build and Run a Service Fabric Application with Containers
- How failover and scaling are handled in Service Fabric

# Deploy a Service Fabric Windows cluster into an Azure virtual network

11/2/2017 • 6 min to read • [Edit Online](#)

This tutorial is part one of a series. You will learn how to deploy a Windows Service Fabric cluster into an existing Azure virtual network (VNET) and sub-net using PowerShell. When you're finished, you have a cluster running in the cloud that you can deploy applications to. To create a Linux cluster using Azure CLI, see [Create a secure Linux cluster on Azure](#).

In this tutorial, you learn how to:

- Create a VNET in Azure using PowerShell
- Create a key vault and upload a certificate
- Create a secure Service Fabric cluster in Azure PowerShell
- Secure the cluster with an X.509 certificate
- Connect to the cluster using PowerShell
- Remove a cluster

In this tutorial series you learn how to:

- Create a secure cluster on Azure
- [Deploy API Management with Service Fabric](#)

## Prerequisites

Before you begin this tutorial:

- If you don't have an Azure subscription, create a [free account](#)
- Install the [Service Fabric SDK and PowerShell module](#)
- Install the [Azure Powershell module version 4.1 or higher](#)

The following procedures create a five-node Service Fabric cluster. To calculate cost incurred by running a Service Fabric cluster in Azure use the [Azure Pricing Calculator](#).

## Sign-in to Azure and select your subscription

This guide uses Azure PowerShell. When you start a new PowerShell session, sign in to your Azure account and select your subscription before you execute Azure commands.

Run the following script to sign in to your Azure account select your subscription:

```
Login-AzureRmAccount  
Get-AzureRmSubscription  
Set-AzureRmContext -SubscriptionId <guid>
```

## Create a resource group

Create a new resource group for your deployment and give it a name and a location.

```
$groupname = "sfclustertutorialgroup"
$clusterloc="southcentralus"
New-AzureRmResourceGroup -Name $groupname -Location $clusterloc
```

## Deploy the network topology

Next, set up the network topology to which API Management and the Service Fabric cluster will be deployed. The [network.json](#) Resource Manager template is configured to create a virtual network (VNET) and also a subnet and network security group (NSG) for Service Fabric and a subnet and NSG for API Management. Learn more about VNETs, subnets, and NSGs [here](#).

The [network.parameters.json](#) parameters file contains the names of the subnets and NSGs that Service Fabric and API Management deploy to. API Management is deployed in the [following tutorial](#). For this guide, the parameter values do not need to be changed. The Service Fabric Resource Manager templates use these values. If the values are modified here, you must modify them in the other Resource Manager templates used in this tutorial and the [Deploy API Management tutorial](#).

Download the following Resource Manager template and parameters file:

- [network.json](#)
- [network.parameters.json](#)

Use the following PowerShell command to deploy the Resource Manager template and parameter files for the network setup:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName $groupname -TemplateFile
C:\winclustertutorial\network.json -TemplateParameterFile C:\winclustertutorial\network.parameters.json -
Verbose
```

## Deploy the Service Fabric cluster

Once the network resources have finished deploying, the next step is to deploy a Service Fabric cluster to the VNET in the subnet and NSG designated for the Service Fabric cluster. Deploying a cluster to an existing VNET and subnet (deployed previously in this article) requires a Resource Manager template. For this tutorial series, the template is pre-configured to use the names of the VNET, subnet, and NSG that you set up in a previous step.

Download the following Resource Manager template and parameters file:

- [cluster.json](#)
- [cluster.parameters.json](#)

Use this template to create a secure cluster. A cluster certificate is an X.509 certificate used to secure node-to-node communication and authenticate the cluster management endpoints to a management client. The cluster certificate also provides an SSL for the HTTPS management API and for Service Fabric Explorer over HTTPS. Azure Key Vault is used to manage certificates for Service Fabric clusters in Azure. When a cluster is deployed in Azure, the Azure resource provider responsible for creating Service Fabric clusters pulls certificates from Key Vault and installs them on the cluster VMs.

You can use a certificate from a certificate authority (CA) as the cluster certificate or, for testing purposes, create a self-signed certificate. The cluster certificate must:

- contain a private key.
- be created for key exchange, which is exportable to a Personal Information Exchange (.pfx) file.
- have a subject name that matches the domain that you use to access the Service Fabric cluster. This matching is

required to provide SSL for the cluster's HTTPS management endpoints and Service Fabric Explorer. You cannot obtain an SSL certificate from a certificate authority (CA) for the .cloudapp.azure.com domain. You must obtain a custom domain name for your cluster. When you request a certificate from a CA, the certificate's subject name must match the custom domain name that you use for your cluster.

Fill in the empty *location*, *clusterName*, *adminUserName*, and *adminPassword* parameters in the *cluster.parameters.json* file for your deployment. Leave the *certificateThumbprint*, *certificateUrlValue*, and *sourceVaultValue* parameters blank to create a self-signed certificate. If you want to use an existing certificate previously uploaded to a key vault, fill in those parameter values.

The following script uses the [New-AzureRmServiceFabricCluster](#) cmdlet and template to deploy a new cluster in Azure. The cmdlet also creates a new key vault in Azure, adds a new self-signed certificate to the key vault, and downloads the certificate file locally. You can specify an existing certificate and/or key vault by using other parameters of the [New-AzureRmServiceFabricCluster](#) cmdlet.

```
# Variables.  
$certpwd="q6D7nN%6ck@6" | ConvertTo-SecureString -AsPlainText -Force  
$certfolder="c:\mycertificates\"  
$clustername = "mysfcluster"  
$vaultname = "clusterkeyvault111"  
$vaultgroupname="clusterkeyvaultgroup111"  
$subname="$clustername.$clusterloc.cloudapp.azure.com"  
  
# Create the Service Fabric cluster.  
New-AzureRmServiceFabricCluster -ResourceGroupName $groupname -TemplateFile  
'C:\winclustertutorial\cluster.json' `  
-ParameterFile 'C:\winclustertutorial\cluster.parameters.json' -CertificatePassword $certpwd `  
-CertificateOutputFolder $certfolder -KeyVaultName $vaultname -KeyVaultResouceGroupName $vaultgroupname -  
CertificateSubjectName $subname
```

## Connect to the secure cluster

Connect to the cluster using the Service Fabric PowerShell module installed with the Service Fabric SDK. First, install the certificate into the Personal (My) store of the current user on your computer. Run the following PowerShell command:

```
$certpwd="q6D7nN%6ck@6" | ConvertTo-SecureString -AsPlainText -Force  
Import-PfxCertificate -Exportable -CertStoreLocation Cert:\CurrentUser\My `  
-FilePath C:\mycertificates\mysfcluster20170531104310.pfx `  
-Password $certpwd
```

You are now ready to connect to your secure cluster.

The **Service Fabric** PowerShell module provides many cmdlets for managing Service Fabric clusters, applications, and services. Use the [Connect-ServiceFabricCluster](#) cmdlet to connect to the secure cluster. The certificate thumbprint and connection endpoint details are found in the output from the previous step.

```
Connect-ServiceFabricCluster -ConnectionEndpoint mysfcluster.southcentralus.cloudapp.azure.com:19000 `  
-KeepAliveIntervalInSec 10 `  
-X509Credential -ServerCertThumbprint C4C1E541AD512B8065280292A8BA6079C3F26F10 `  
-FindType FindByThumbprint -FindValue C4C1E541AD512B8065280292A8BA6079C3F26F10 `  
-StoreLocation CurrentUser -StoreName My
```

Check that you are connected and the cluster is healthy using the [Get-ServiceFabricClusterHealth](#) cmdlet.

```
Get-ServiceFabricClusterHealth
```

## Clean up resources

The other articles in this tutorial series use the cluster you just created. If you're not immediately moving on to the next article, you might want to delete the cluster and key vault to avoid incurring charges. The simplest way to delete the cluster and all the resources it consumes is to delete the resource group.

Delete the resource group and all the cluster resources using the [Remove-AzureRMResourceGroup cmdlet](#). Also delete the resource group containing the key vault.

```
Remove-AzureRmResourceGroup -Name $groupname -Force  
Remove-AzureRmResourceGroup -Name $vaultgroupname -Force
```

## Next steps

In this tutorial, you learned how to:

- Create a VNET in Azure using PowerShell
- Create a key vault and upload a certificate
- Create a secure Service Fabric cluster in Azure using PowerShell
- Secure the cluster with an X.509 certificate
- Connect to the cluster using PowerShell
- Remove a cluster

Next, advance to the following tutorial to learn how to deploy API Management with Service Fabric.

[Deploy API Management](#)

# Deploy a Service Fabric Linux cluster into an Azure virtual network

11/2/2017 • 5 min to read • [Edit Online](#)

This tutorial is part one of a series. You will learn how to deploy a Linux Service Fabric cluster into an existing Azure virtual network (VNET) and sub-net using Azure CLI. When you're finished, you have a cluster running in the cloud that you can deploy applications to. To create a Windows cluster using PowerShell, see [Create a secure Windows cluster on Azure](#).

In this tutorial, you learn how to:

- Create a VNET in Azure using Azure CLI
- Create a secure Service Fabric cluster in Azure using Azure CLI
- Secure the cluster with an X.509 certificate
- Connect to the cluster using Service Fabric CLI
- Remove a cluster

In this tutorial series you learn how to:

- Create a secure cluster on Azure
- [Deploy API Management with Service Fabric](#)

## Prerequisites

Before you begin this tutorial:

- If you don't have an Azure subscription, create a [free account](#)
- Install the [Service Fabric CLI](#)
- Install the [Azure CLI 2.0](#)

The following procedures create a five-node Service Fabric cluster. To calculate cost incurred by running a Service Fabric cluster in Azure use the [Azure Pricing Calculator](#).

## Sign-in to Azure and select your subscription

This guide uses Azure CLI. When you start a new session, sign in to your Azure account and select your subscription before you execute Azure commands.

Run the following script to sign in to your Azure account select your subscription:

```
az login  
az account set --subscription <guid>
```

## Create a resource group

Create a new resource group for your deployment and give it a name and a location.

```
ResourceGroupName="sflinuxclustergroup"
Location="southcentralus"
az group create --name $ResourceGroupName --location $Location
```

## Deploy the network topology

Next, set up the network topology to which API Management and the Service Fabric cluster will be deployed. The [network.json](#) Resource Manager template is configured to create a virtual network (VNET) and also a subnet and network security group (NSG) for Service Fabric and a subnet and NSG for API Management. Learn more about VNETs, subnets, and NSGs [here](#).

The [network.parameters.json](#) parameters file contains the names of the subnets and NSGs that Service Fabric and API Management deploy to. API Management is deployed in the [following tutorial](#). For this guide, the parameter values do not need to be changed. The Service Fabric Resource Manager templates use these values. If the values are modified here, you must modify them in the other Resource Manager templates used in this tutorial and the [Deploy API Management tutorial](#).

Download the following Resource Manager template and parameters file:

- [network.json](#)
- [network.parameters.json](#)

Use the following script to deploy the Resource Manager template and parameter files for the network setup:

```
az group deployment create \
--name VnetDeployment \
--resource-group $ResourceGroupName \
--template-file network.json \
--parameters @network.parameters.json
```

## Deploy the Service Fabric cluster

Once the network resources have finished deploying, the next step is to deploy a Service Fabric cluster to the VNET in the subnet and NSG designated for the Service Fabric cluster. Deploying a cluster to an existing VNET and subnet (deployed previously in this article) requires a Resource Manager template. For more information, see [Create a cluster by using Azure Resource Manager](#). For this tutorial series, the template is pre-configured to use the names of the VNET, subnet, and NSG that you set up in a previous step.

Download the following Resource Manager template and parameters file:

- [linuxcluster.json](#)
- [linuxcluster.parameters.json](#)

Use this template to create a secure cluster. A cluster certificate is an X.509 certificate used to secure node-to-node communication and authenticate the cluster management endpoints to a management client. The cluster certificate also provides an SSL for the HTTPS management API and for Service Fabric Explorer over HTTPS. Azure Key Vault is used to manage certificates for Service Fabric clusters in Azure. When a cluster is deployed in Azure, the Azure resource provider responsible for creating Service Fabric clusters pulls certificates from Key Vault and installs them on the cluster VMs.

You can use a certificate from a certificate authority (CA) as the cluster certificate or, for testing purposes, create a self-signed certificate. The cluster certificate must:

- contain a private key.
- be created for key exchange, which is exportable to a Personal Information Exchange (.pfx) file.

- have a subject name that matches the domain that you use to access the Service Fabric cluster. This matching is required to provide SSL for the cluster's HTTPS management endpoints and Service Fabric Explorer. You cannot obtain an SSL certificate from a certificate authority (CA) for the .cloudapp.azure.com domain. You must obtain a custom domain name for your cluster. When you request a certificate from a CA, the certificate's subject name must match the custom domain name that you use for your cluster.

Fill in the empty **clusterName**, **adminUserName**, and **adminPassword** parameters in the *linuxcluster.parameters.json* file for your deployment. Leave the **certificateThumbprint**, **certificateUrlValue**, and **sourceVaultValue** parameters blank to create a self-signed certificate. If you want to use an existing certificate previously uploaded to a key vault, fill in those parameter values.

The following script uses the [az sf cluster create](#) command and template to deploy a new cluster in Azure. The cmdlet also creates a new key vault in Azure, adds a new self-signed certificate to the key vault, and downloads the certificate file locally. You can specify an existing certificate and/or key vault by using other parameters of the [az sf cluster create](#) command.

```
Password="q6D7nN%6ck@6"
Subject="mysfcluster.southcentralus.cloudapp.azure.com"
VaultName="linuxclusterkeyvault"
az group create --name $ResourceGroupName --location $Location

az sf cluster create --resource-group $ResourceGroupName --location $Location \
--certificate-output-folder . --certificate-password $Password --certificate-subject-name $Subject \
--vault-name $VaultName --vault-resource-group $ResourceGroupName \
--template-file linuxcluster.json --parameter-file linuxcluster.parameters.json
```

## Connect to the secure cluster

Connect to the cluster using the Service Fabric CLI `sfctl cluster select` command using your key. Note, only use the **--no-verify** option for a self-signed certificate.

```
sfctl cluster select --endpoint https://aztestcluster.southcentralus.cloudapp.azure.com:19080 \
--pem ./aztestcluster201709151446.pem --no-verify
```

Check that you are connected and the cluster is healthy using the `sfctl cluster health` command.

```
sfctl cluster health
```

## Clean up resources

The other articles in this tutorial series use the cluster you just created. If you're not immediately moving on to the next article, you might want to delete the cluster to avoid incurring charges. The simplest way to delete the cluster and all the resources it consumes is to delete the resource group.

Log in to Azure and select the subscription ID with which you want to remove the cluster. You can find your subscription ID by logging in to the [Azure portal](#). Delete the resource group and all the cluster resources using the [az group delete](#) command.

```
az group delete --name $ResourceGroupName
```

## Next steps

In this tutorial, you learned how to:

- Create a VNET in Azure using Azure CLI
- Create a secure Service Fabric cluster in Azure using Azure CLI
- Secure the cluster with an X.509 certificate
- Connect to the cluster using Service Fabric CLI
- Remove a cluster

Next, advance to the following tutorial to learn how to deploy API Management with Service Fabric.

[Deploy API Management](#)

# Scale a Service Fabric cluster

11/2/2017 • 5 min to read • [Edit Online](#)

This tutorial is part three of a series, and shows you how to scale your existing cluster out and in. When you've finished, you will know how to scale your cluster and how to clean up any left-over resources.

In this tutorial, you learn how to:

- Read the cluster node count
- Add cluster nodes (scale out)
- Remove cluster nodes (scale in)

## Prerequisites

Before you begin this tutorial:

- If you don't have an Azure subscription, create a [free account](#)
- Install the [Azure Powershell module version 4.1 or higher](#) or [Azure CLI 2.0](#).
- Create a secure [Windows cluster](#) or [Linux cluster](#) on Azure
- If you deploy a Windows cluster, set up a Windows development environment. Install [Visual Studio 2017](#) and the [Azure development, ASP.NET and web development](#), and [.NET Core cross-platform development](#) workloads. Then set up a [.NET development environment](#).
- If you deploy a Linux cluster, set up a Java development environment on [Linux](#) or [MacOS](#). Install the [Service Fabric CLI](#).

## Sign in to Azure

Sign in to your Azure account select your subscription before you execute Azure commands.

```
Login-AzureRmAccount  
Get-AzureRmSubscription  
Set-AzureRmContext -SubscriptionId <guid>
```

## Connect to the cluster

To successfully complete this part of the tutorial, you need to connect to both the Service Fabric cluster and the virtual machine scale set (that hosts the cluster). The virtual machine scale set is the Azure resource that hosts Service Fabric on Azure.

When you connect to a cluster, you can query it for information. You can use the cluster to learn about what nodes the cluster is aware of. Connecting to the cluster in the following code uses the same certificate that was created in the first part of this series. Make sure you set the `$endpoint` and `$thumbprint` variables to your values.

```

$endpoint = "<mycluster>.southcentralus.cloudapp.azure.com:19000"
$thumbprint = "63EB5BA4BC2A3BADC42CA6F93D6F45E5AD98A1E4"

Connect-ServiceFabricCluster -ConnectionEndpoint $endpoint ` 
    -KeepAliveIntervalInSec 10 ` 
    -X509Credential -ServerCertThumbprint $thumbprint ` 
    -FindType FindByThumbprint -FindValue $thumbprint ` 
    -StoreLocation CurrentUser -StoreName My

Get-ServiceFabricClusterHealth

```

With the `Get-ServiceFabricClusterHealth` command, status is returned to you with details about the health of each node in the cluster.

## Scale out

When you scale out, you add more virtual machine instances to the scale set. These instances become the nodes that Service Fabric uses. Service Fabric knows when the scale set has more instances added (by scaling out) and reacts automatically. The following code gets a scale set by name and increases the **capacity** of the scale set by 1.

```

$scaleset = Get-AzureRmVmss -ResourceGroupName SFCLUSTERTUTORIALGROUP -VMScaleSetName nt1vm
$scaleset.Sku.Capacity += 1

Update-AzureRmVmss -ResourceGroupName SFCLUSTERTUTORIALGROUP -VMScaleSetName nt1vm -VirtualMachineScaleSet
$scaleset

```

After the update operation is completed, run the `Get-ServiceFabricClusterHealth` command to see the new node information.

## Scale in

Scaling in is the same as scaling out, except you use a lower **capacity** value. When you scale in the scale set, you remove virtual machine instances from the scale set. Normally, Service Fabric is unaware what has happened, and it thinks that a node has gone missing. Service Fabric then reports an unhealthy state for the cluster. To prevent that bad state, you must inform service fabric that you expect the node to disappear.

### Remove the service fabric node

#### NOTE

This part only applies to the *Bronze* durability tier. For more information about durability, see [Service Fabric cluster capacity planning](#).

When you scale in a virtual machine scale set, the scale set (in most cases) removes the virtual machine instance that was last created. So you need to find the matching, last created, service fabric node. You can find this last node by checking the biggest `NodeId` property value on the service fabric nodes.

```
Get-ServiceFabricNode | Sort-Object NodeInstanceId -Descending | Select-Object -First 1
```

The service fabric cluster needs to know that this node is going to be removed. There are three steps you need to take:

1. Disable the node so that it no longer is a replicate for data.

```
Disable-ServiceFabricNode
```

2. Stop the node so that the service fabric runtime shuts down cleanly, and your app gets a terminate request.

```
Start-ServiceFabricNodeTransition -Stop
```

3. Remove the node from the cluster.

```
Remove-ServiceFabricNodeState
```

Once these three steps have been applied to the node, it can be removed from the scale set. If you're using any durability tier besides [bronze](#), these steps are done for you when the scale set instance is removed.

The following code block gets the last created node, disables, stops, and removes the node from the cluster.

```

# Get the node that was created last
$node = Get-ServiceFabricNode | Sort-Object NodeInstanceId -Descending | Select-Object -First 1

# Node details for the disable/stop process
$nodename = $node.NodeName
$nodeid = $node.NodeInstanceId

$loopTimeout = 10

# Run disable logic
Disable-ServiceFabricNode -NodeName $nodename -Intent RemoveNode -TimeoutSec 300 -Force

$state = Get-ServiceFabricNode | Where-Object NodeName -eq $nodename | Select-Object -ExpandProperty NodeStatus

while (($state -ne [System.Fabric.Query.NodeStatus]::Disabled) -and ($loopTimeout -ne 0))
{
    Start-Sleep 5
    $loopTimeout -= 1
    $state = Get-ServiceFabricNode | Where-Object NodeName -eq $nodename | Select-Object -ExpandProperty NodeStatus
    Write-Host "Checking state... $state found"
}

# Exit if the node was unable to be disabled
if ($state -ne [System.Fabric.Query.NodeStatus]::Disabled)
{
    Write-Error "Disable failed with state $state"
}
else
{
    # Stop node
    $stopid = New-Guid
    Start-ServiceFabricNodeTransition -Stop -OperationId $stopid -NodeName $nodename -NodeInstanceId $nodeid -StopDurationInSeconds 300

    $state = (Get-ServiceFabricNodeTransitionProgress -OperationId $stopid).State
    $loopTimeout = 10

    # Watch the transaction
    while (($state -eq [System.Fabric.TestCommandProgressState]::Running) -and ($loopTimeout -ne 0))
    {
        Start-Sleep 5
        $state = (Get-ServiceFabricNodeTransitionProgress -OperationId $stopid).State
        Write-Host "Checking state... $state found"
    }

    if ($state -ne [System.Fabric.TestCommandProgressState]::Completed)
    {
        Write-Error "Stop transaction failed with $state"
    }
    else
    {
        # Remove the node from the cluster
        Remove-ServiceFabricNodeState -NodeName $nodename -TimeoutSec 300 -Force
    }
}
}

```

## Scale in the scale set

Now that the service fabric node has been removed from the cluster, the virtual machine scale set can be scaled in. In the example below, the scale set capacity is reduced by 1.

```
$scaleSet = Get-AzureRmVmss -ResourceGroupName SFCLUSTERTUTORIALGROUP -VMScaleSetName nt1vm
	scaleSet.Sku.Capacity -= 1

Update-AzureRmVmss -ResourceGroupName SFCLUSTERTUTORIALGROUP -VMScaleSetName nt1vm -VirtualMachineScaleSet
$scaleSet
```

## Next steps

In this tutorial, you learned how to:

- Read the cluster node count
- Add cluster nodes (scale out)
- Remove cluster nodes (scale in)

Next, advance to the following tutorial to learn how to deploy an application and use API management.

[Deploy an application](#)

# Deploy API Management with Service Fabric

11/2/2017 • 12 min to read • [Edit Online](#)

This tutorial is part two of a series. This tutorial shows you how to set up [Azure API Management](#) with Service Fabric to route traffic to a back-end service in Service Fabric. When you're finished, you have deployed API Management to a VNET, configured an API operation to send traffic to back-end stateless services. To learn more about Azure API Management scenarios with Service Fabric, see the [overview](#) article.

In this tutorial, you learn how to:

- Deploy API Management
- Configure API Management
- Create an API operation
- Configure a backend policy
- Add the API to a product

In this tutorial series you learn how to:

- Create a secure [Windows cluster](#) or [Linux cluster](#) on Azure using a template
- Deploy API Management with Service Fabric

## Prerequisites

Before you begin this tutorial:

- If you don't have an Azure subscription, create a [free account](#)
- Install the [Azure Powershell module version 4.1 or higher](#) or [Azure CLI 2.0](#).
- Create a secure [Windows cluster](#) or [Linux cluster](#) on Azure
- If you deploy a Windows cluster, set up a Windows development environment. Install [Visual Studio 2017](#) and the **Azure development, ASP.NET and web development**, and **.NET Core cross-platform development** workloads. Then set up a [.NET development environment](#).
- If you deploy a Linux cluster, set up a Java development environment on [Linux](#) or [MacOS](#). Install the [Service Fabric CLI](#).

## Sign in to Azure and select your subscription

Sign in to your Azure account select your subscription before you execute Azure commands.

```
Login-AzureRmAccount  
Get-AzureRmSubscription  
Set-AzureRmContext -SubscriptionId <guid>
```

```
az login  
az account set --subscription <guid>
```

## Deploy API Management

Cloud applications typically need a front-end gateway to provide a single point of ingress for users, devices, or other applications. In Service Fabric, a gateway can be any stateless service such as an ASP.NET Core application,

or another service designed for traffic ingress, such as Event Hubs, IoT Hub, or Azure API Management. This tutorial is an introduction to using Azure API Management as a gateway to your Service Fabric applications. API Management integrates directly with Service Fabric, allowing you to publish APIs with a rich set of routing rules to your back-end Service Fabric services.

Now that you have a secure [Windows cluster](#) or [Linux cluster](#) on Azure, deploy API Management to the virtual network (VNET) in the subnet and NSG designated for API Management. For this tutorial, the API Management Resource Manager template is pre-configured to use the names of the VNET, subnet, and NSG that you set up in the previous [Windows cluster tutorial](#) or [Linux cluster tutorial](#).

Download the following Resource Manager template and parameters file:

- [apim.json](#)
- [apim.parameters.json](#)

Fill in the empty parameters in the `apim.parameters.json` for your deployment.

Use the following script to deploy the Resource Manager template and parameter files for API Management:

```
$ResourceGroupName = "tutorialgroup"
New-AzureRmResourceGroupDeployment -ResourceGroupName $ResourceGroupName -TemplateFile .\apim.json -
TemplateParameterFile .\apim.parameters.json -Verbose
```

```
ResourceGroupName="tutorialgroup"
az group deployment create --name ApiMgmtDeployment --resource-group $ResourceGroupName --template-file
apim.json --parameters @apim.parameters.json
```

## Configure API Management

Once your API Management and Service Fabric cluster are deployed, you can configure security settings and a Service Fabric backend in API Management. This allows you to create a backend service policy that sends traffic to your Service Fabric cluster.

### Configure API Management Security

To configure the Service Fabric backend, you first need to configure API Management security settings. To configure security settings, go to your API Management service in the Azure portal.

#### Enable the API Management REST API

The API Management REST API is currently the only way to configure a backend service. The first step is to enable the API Management REST API and secure it.

1. In the API Management service, select **Management API** under **Security**.
2. Check the **Enable API Management REST API** checkbox.
3. Note the **Management API URL**, which we use later to set up the Service Fabric backend.
4. Generate an **Access Token** by selecting an expiry date and a key, then click the **Generate** button toward the bottom of the page.
5. Copy the **access token** and save it. We use the access token in the following steps. Note this is different from the primary key and secondary key.

#### Upload a Service Fabric client certificate

API Management must authenticate with your Service Fabric cluster for service discovery using a client certificate that has access to your cluster. For simplicity, this tutorial uses the same certificate specified previously when creating the [Windows cluster](#) or [Linux cluster](#), which by default can be used to access your cluster.

1. In the API Management service, select **Client certificates** under **Security**.

2. Click the **+ Add** button.
3. Select the private key file (.pfx) of the cluster certificate that you specified when creating your Service Fabric cluster, give it a name, and provide the private key password.

**NOTE**

This tutorial uses the same certificate for client authentication and cluster node-to-node security. You may use a separate client certificate if you have one configured to access your Service Fabric cluster.

## Configure the backend

Now that API Management security is configured, you can configure the Service Fabric backend. For Service Fabric backends, the Service Fabric cluster is the backend, rather than a specific Service Fabric service. This allows a single policy to route to more than one service in the cluster.

This step requires the access token you generated previously and the thumbprint for your cluster certificate that you previously uploaded to API Management.

**NOTE**

If you used a separate client certificate in the previous step for API Management, you need the thumbprint for the client certificate in addition to the cluster certificate thumbprint in this step.

Send the following HTTP PUT request to the API Management API URL you noted earlier when enabling the API Management REST API to configure the Service Fabric backend service. You should see an **HTTP 201 Created** response when the command succeeds. For more information on each field, see the API Management [backend API reference documentation](#).

HTTP command and URL:

```
PUT https://your-apim.management.azure-api.net/backends/servicefabric?api-version=2016-10-10
```

Request headers:

```
Authorization: SharedAccessSignature <your access token>
Content-Type: application/json
```

Request body:

```
{
    "description": "<description>",
    "url": "<fallback service name>",
    "protocol": "http",
    "resourceId": "<cluster HTTP management endpoint>",
    "properties": {
        "serviceFabricCluster": {
            "managementEndpoints": [ "<cluster HTTP management endpoint>" ],
            "clientCertificateThumbprint": "<client cert thumbprint>",
            "serverCertificateThumbprints": [ "<cluster cert thumbprint>" ],
            "maxPartitionResolutionRetries" : 5
        }
    }
}
```

The **url** parameter here is a fully qualified service name of a service in your cluster that all requests are routed to

by default if no service name is specified in a backend policy. You may use a fake service name, such as "fabric:/fake/service" if you do not intend to have a fallback service.

Refer to the API Management [backend API reference documentation](#) for more details on each field.

```
#import requests library for making REST calls
import requests
import json

#specify url
url = 'https://your-apim.management.azure-api.net/backends/servicefabric?api-version=2016-10-10'

token = "SharedAccessSignature
integration&201710140514&Lqnbei7n2Sot6doiNtxMFPUi/m9LsNa+1ZK/PdxqF149JFwjXh1wW5Gd99r/tIOeHp6dU8UV5iZUdOPfcrm
5hg=="

payload = {
    "description": "My Service Fabric backend",
    "url": "fabric:/ApiApplication/ApiWebService",
    "protocol": "http",
    "resourceId": "https://tutorialcluster.eastus.cloudapp.azure.com:19080",
    "properties": {
        "serviceFabricCluster": {
            "managementEndpoints": [ "https://tutorialcluster.eastus.cloudapp.azure.com:19080" ],
            "clientCertificateThumbprint": "97EDD7E4979FB072AF3E480A5E5EE34B1B89DD80",
            "serverCertificateThumbprints": [ "97EDD7E4979FB072AF3E480A5E5EE34B1B89DD80" ],
            "maxPartitionResolutionRetries" : 5
        }
    }
}

headers = {'Authorization': token, "Content-Type": "application/json"}

#Call REST API
response = requests.put(url, data=json.dumps(payload), headers=headers)

#print Response
print(response.status_code)
print(response.text)
```

## Deploy a Service Fabric back-end service

Now that you have the Service Fabric configured as a backend to API Management, you can author backend policies for your APIs that send traffic to your Service Fabric services. But first you need a service running in Service Fabric to accept requests. If you previously created a [Windows cluster](#), deploy a .NET Service Fabric service. If you previously created a [Linux cluster](#), deploy a Java Service Fabric service.

### Deploy a .NET Service Fabric service

For this tutorial, we create a basic stateless ASP.NET Core Reliable Service using the default Web API project template. This creates an HTTP endpoint for your service, which you expose through Azure API Management:

```
/api/values
```

Start Visual Studio as Administrator and create an ASP.NET Core service:

1. In Visual Studio, select File -> New Project.
2. Select the Service Fabric Application template under Cloud and name it "**ApiApplication**".
3. Select the ASP.NET Core service template and name the project "**Web ApiService**".
4. Select the Web API ASP.NET Core 1.1 project template.

- Once the project is created, open `PackageRoot\ServiceManifest.xml` and remove the `Port` attribute from the endpoint resource configuration:

```
<Resources>
  <Endpoints>
    <Endpoint Protocol="http" Name="ServiceEndpoint" Type="Input" />
  </Endpoints>
</Resources>
```

This allows Service Fabric to specify a port dynamically from the application port range, which we opened through the Network Security Group in the cluster Resource Manager template, allowing traffic to flow to it from API Management.

- Press F5 in Visual Studio to verify the web API is available locally.

Open Service Fabric Explorer and drill down to a specific instance of the ASP.NET Core service to see the base address the service is listening on. Add `/api/values` to the base address and open it in a browser. This invokes the Get method on the ValuesController in the Web API template. It returns the default response that is provided by the template, a JSON array that contains two strings:

```
["value1", "value2"]`
```

This is the endpoint that you expose through API Management in Azure.

- Finally, deploy the application to your cluster in Azure. [Using Visual Studio](#), right-click the Application project and select **Publish**. Provide your cluster endpoint (for example, `mycluster.southcentralus.cloudapp.azure.com:19000`) to deploy the application to your Service Fabric cluster in Azure.

An ASP.NET Core stateless service named `fabric:/ApiApplication/Web ApiService` should now be running in your Service Fabric cluster in Azure.

## Create a Java Service Fabric service

For this tutorial, we deploy a basic web server which echoes messages back to the user. The echo server sample application contains an HTTP endpoint for your service, which you expose through Azure API Management.

- Clone the Java getting started samples.

```
git clone https://github.com/Azure-Samples/service-fabric-java-getting-started.git
cd service-fabric-java-getting-started
```

- Edit `Services/EchoServer/EchoServer1.0/EchoServerApplication/EchoServerPkg/ServiceManifest.xml`.

Update the endpoint so the service listens on port 8081.

```
<Endpoint Name="WebEndpoint" Protocol="http" Port="8081" />
```

- Save `ServiceManifest.xml`, then build the EchoServer1.0 application.

```
cd Services/EchoServer/EchoServer1.0/
gradle
```

- Deploy the application to the cluster.

```
cd Scripts  
sfctl cluster select --endpoint http://mycluster.southcentralus.cloudapp.azure.com:19080  
.install.sh
```

A Java stateless service named `fabric:/EchoServerApplication/EchoServerService` should now be running in your Service Fabric cluster in Azure.

5. Open a browser and type in <http://mycluster.southcentralus.cloudapp.azure.com:8081/getMessage>, you should see "[version 1.0]Hello World !!!" displayed.

## Create an API operation

Now we're ready to create an operation in API Management that external clients use to communicate with the ASP.NET Core stateless service running in the Service Fabric cluster.

1. Log in to the Azure portal and navigate to your API Management service deployment.
2. In the API Management service blade, select **APIs**
3. Add a new API by clicking **+API**, then **Blank API** box, and filling out the dialog box:
  - **Web service URL:** For Service Fabric backends, this URL value is not used. You can put any value here. For this tutorial, use: "<http://servicefabric>".
  - **Display Name:** Provide any name for your API. For this tutorial, use "Service Fabric App".
  - **Name:** Enter "service-fabric-app".
  - **URL scheme:** Select either **HTTP**, **HTTPS**, or **both**. For this tutorial, use **both**.
  - **API URL Suffix:** Provide a suffix for our API. For this tutorial, use "myapp".
4. Select **Service Fabric App** from the list of APIs and click **+ Add operation** to add a front-end API operation. Fill out the values:
  - **URL:** Select **GET** and provide a URL path for the API. For this tutorial, use "/api/values". By default, the URL path specified here is the URL path sent to the backend Service Fabric service. If you use the same URL path here that your service uses, in this case "/api/values", then the operation works without further modification. You may also specify a URL path here that is different from the URL path used by your backend Service Fabric service, in which case you also need to specify a path rewrite in your operation policy later.
  - **Display name:** Provide any name for the API. For this tutorial, use "Values".
5. Click **Save**.

## Configure a backend policy

The backend policy ties everything together. This is where you configure the backend Service Fabric service to which requests are routed. You can apply this policy to any API operation. The [backend configuration for Service Fabric](#) provides the following request routing controls:

- Service instance selection by specifying a Service Fabric service instance name, either hardcoded (for example, `"fabric:/myapp/myservice"` ) or generated from the HTTP request (for example, `"fabric:/myapp/users/" + context.Request.MatchedParameters["name"]` ).
- Partition resolution by generating a partition key using any Service Fabric partitioning scheme.
- Replica selection for stateful services.
- Resolution retry conditions that allow you to specify the conditions for re-resolving a service location and resending a request.

For this tutorial, create a backend policy that routes requests directly to the ASP.NET Core stateless service deployed earlier:

1. Select and edit the **inbound policies** for the Values operation by clicking the edit icon, and then selecting **Code View**.
2. In the policy code editor, add a `set-backend-service` policy under inbound policies as shown here and click the **Save** button:

```
<policies>
  <inbound>
    <base/>
    <set-backend-service
      backend-id="servicefabric"
      sf-service-instance-name="fabric:/ApiApplication/WebApiService"
      sf-resolve-condition="@((int)context.Response.StatusCode != 200)" />
  </inbound>
  <backend>
    <base/>
  </backend>
  <outbound>
    <base/>
  </outbound>
</policies>
```

For a full set of Service Fabric back-end policy attributes, refer to the [API Management back-end documentation](#)

### Add the API to a product

Before you can call the API, it must be added to a product where you can grant access to users.

1. In the API Management service, select **Products**.
2. By default, API Management providers two products: Starter and Unlimited. Select the Unlimited product.
3. Select **+Add APIs**.
4. Select the `Service Fabric App` API you created in the previous steps and click the **Select** button.

### Test it

You can now try sending a request to your back-end service in Service Fabric through API Management directly from the Azure portal.

1. In the API Management service, select **API**.
2. In the **Service Fabric App** API you created in the previous steps, select the **Test** tab and then the **Values** operation.
3. Click the **Send** button to send a test request to the backend service. You should see an HTTP response similar to:

```
HTTP/1.1 200 OK

Transfer-Encoding: chunked

Content-Type: application/json; charset=utf-8

Vary: Origin

Access-Control-Allow-Origin: https://apimanagement.hosting.portal.azure.net

Access-Control-Allow-Credentials: true

Access-Control-Expose-Headers: Transfer-Encoding,Date,Server,Vary,Ocp-Apim-Trace-Location

Ocp-Apim-Trace-Location:
https://apimgmtstuvyx3wa3oqhdwy.blob.core.windows.net/apiinspectorcontainer/RaVuJBQ9yxtdyH55BAsjQ2-1?sv=2015-07-08&sr=b&sig=Ab6dPyLpTGAU6Tdm1EVu32DMfdCXTiKAASUlwSq3jcY%3D&se=2017-09-15T05%3A49%3A53Z&sp=r&traceId=ed9f1f4332e34883a774c34aa899b832

Date: Thu, 14 Sep 2017 05:49:56 GMT
```

```
[  
 "value1",  
 "value2"  
]  
```
```

## Clean up resources

A cluster is made up of other Azure resources in addition to the cluster resource itself. The simplest way to delete the cluster and all the resources it consumes is to delete the resource group.

Log in to Azure and select the subscription ID with which you want to remove the cluster. You can find your subscription ID by logging in to the [Azure portal](#). Delete the resource group and all the cluster resources using the [Remove-AzureRMResourceGroup cmdlet](#).

```
$ResourceGroupName = "tutorialgroup"  
Remove-AzureRmResourceGroup -Name $ResourceGroupName -Force
```

```
ResourceGroupName="tutorialgroup"  
az group delete --name $ResourceGroupName
```

## Conclusion

In this tutorial, you learned how to:

- Deploy API Management
- Configure API Management
- Create an API operation
- Configure a backend policy
- Add the API to a product

# Azure PowerShell samples

10/10/2017 • 1 min to read • [Edit Online](#)

The following table includes links to PowerShell scripts samples that create and manage Service Fabric clusters, applications, and services.

## IMPORTANT

There are two PowerShell modules used to interact with Service Fabric. [Azure PowerShell](#) is used to manage Azure resources, such as an Azure-hosted Service Fabric cluster. [Azure Service Fabric SDK](#) is used to directly connect to the Service Fabric cluster (regardless of where it's hosted) and manage the cluster, applications, and services.

| <b>Create cluster</b>                            |                                                                  |
|--------------------------------------------------|------------------------------------------------------------------|
| <a href="#">Create a cluster (Azure)</a>         | Creates an Azure Service Fabric cluster.                         |
| <b>Manage cluster</b>                            |                                                                  |
| <a href="#">Add an application certificate</a>   | Adds an application X.509 certificate to all nodes in a cluster. |
| <b>Manage applications</b>                       |                                                                  |
| <a href="#">Deploy an application</a>            | Deploy an application to a cluster.                              |
| <a href="#">Upgrade an application</a>           | Upgrade an application                                           |
| <a href="#">Remove an application</a>            | Remove an application from a cluster.                            |
| <a href="#">Open a port in the load balancer</a> | Open an application port in the Azure load balancer.             |

# Azure Service Fabric CLI Samples

10/9/2017 • 1 min to read • [Edit Online](#)

The following table includes links to [Service Fabric CLI](#) script samples that manage Service Fabric clusters, applications, and services.

## IMPORTANT

There are two CLI utilities used to interact with Service Fabric. [Azure CLI](#) is used to manage Azure resources, such as an Azure-hosted Service Fabric cluster. [Service Fabric CLI](#) is used to directly connect to the Service Fabric cluster (regardless of where it's hosted) and manage the cluster, applications, and services.

| <b>Create cluster</b>                                 |                                             |
|-------------------------------------------------------|---------------------------------------------|
| <a href="#">Create a secure Linux cluster (Azure)</a> | Creates an Azure Service Fabric cluster.    |
| <b>Manage applications</b>                            |                                             |
| <a href="#">List applications</a>                     | List applications provisioned to a cluster. |
| <a href="#">Deploy an application</a>                 | Deploy an application to a cluster.         |
| <a href="#">Remove an application</a>                 | Remove an application from a cluster.       |
| <a href="#">Upgrade an application</a>                | Upgrade an already deployed application.    |

# Why a microservices approach to building applications?

10/19/2017 • 17 min to read • [Edit Online](#)

As software developers, there is nothing new in how we think about factoring an application into component parts. It is the central paradigm of object orientation, software abstractions, and componentization. Today, this factorization tends to take the form of classes and interfaces between shared libraries and technology layers. Typically, a tiered approach is taken with a back-end store, middle-tier business logic, and a front-end user interface (UI). What *has* changed over the last few years is that we, as developers, are building distributed applications that are for the cloud and driven by the business.

The changing business needs are:

- A service that's built and operates at scale to reach customers in new geographical regions (for example).
- Faster delivery of features and capabilities to be able to respond to customer demands in an agile way.
- Improved resource utilization to reduce costs.

These business needs are affecting *how* we build applications.

For more information about the approach of Azure to microservices, read [Microservices: An application revolution powered by the cloud](#).

## Monolithic vs. microservice design approach

All applications evolve over time. Successful applications evolve by being useful to people. Unsuccessful applications do not evolve and eventually are deprecated. The question becomes: How much do you know about your requirements today, and what will they be in the future? For example, let's say that you are building a reporting application for a department. You are sure that the application remains within the scope of your company and that the reports are short-lived. Your choice of approach is different from, say, building a service that delivers video content to tens of millions of customers.

Sometimes, getting something out the door as proof of concept is the driving factor, while you know that the application can be redesigned later. There is little point in over-engineering something that never gets used. It's the usual engineering trade-off. On the other hand, when companies talk about building for the cloud, the expectation is growth and usage. The issue is that growth and scale are unpredictable. We would like to be able to prototype quickly while also knowing that we are on a path to deal with future success. This is the lean startup approach: build, measure, learn, and iterate.

During the client-server era, we tended to focus on building tiered applications by using specific technologies in each tier. The term *monolithic* application has emerged for these approaches. The interfaces tended to be between the tiers, and a more tightly coupled design was used between components within each tier. Developers designed and factored classes that were compiled into libraries and linked together into a few executables and DLLs.

There are benefits to such a monolithic design approach. It's often simpler to design, and it has faster calls between components, because these calls are often over interprocess communication (IPC). Also, everyone tests a single product, which tends to be more people-resource efficient. The downside is that there's a tight coupling between tiered layers, and you cannot scale individual components. If you need to perform fixes or upgrades, you have to wait for others to finish their testing. It is more difficult to be agile.

Microservices address these downsides and more closely align with the preceding business requirements, but they also have both benefits and liabilities. The benefits of microservices are that each one typically encapsulates

simpler business functionality, which you scale up or down, test, deploy, and manage independently. One important benefit of a microservice approach is that teams are driven more by business scenarios than by technology, which the tiered approach encourages. In practice, smaller teams develop a microservice based on a customer scenario and use any technologies they choose.

In other words, the organization doesn't need to standardize tech to maintain microservice applications. Individual teams that own services can do what makes sense for them based on team expertise or what's most appropriate to solve the problem. In practice, a set of recommended technologies, such as a particular NoSQL store or web application framework, is preferable.

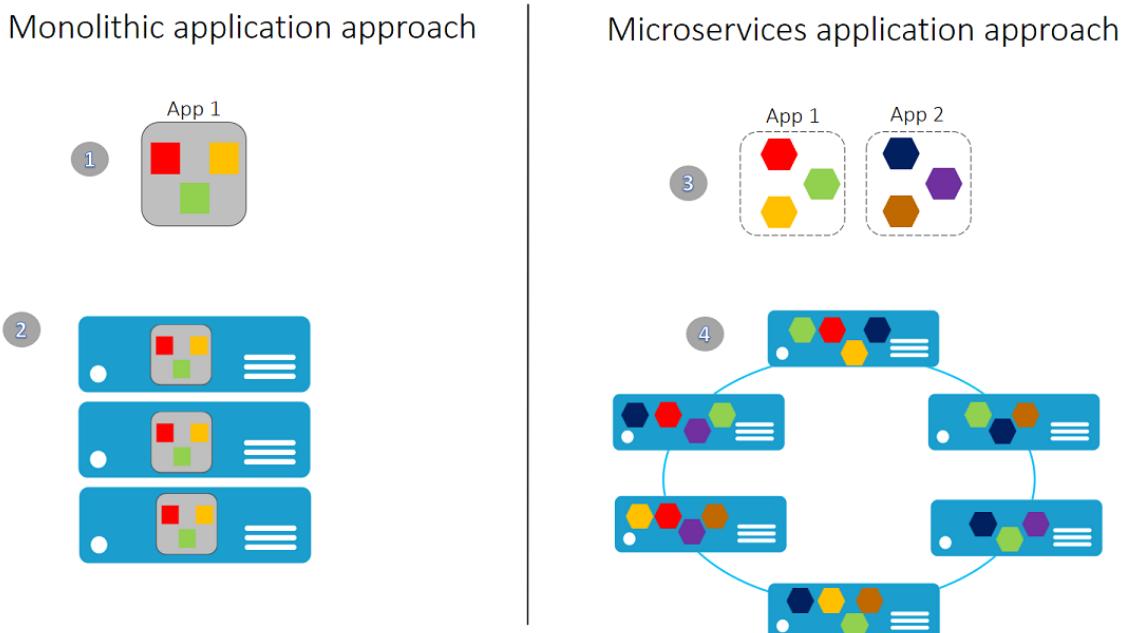
The downside of microservices comes in managing the increased number of separate entities and dealing with more complex deployments and versioning. Network traffic between the microservices increases as well as the corresponding network latencies. Lots of chatty, granular services are a recipe for a performance nightmare. Without tools to help view these dependencies, it is hard to "see" the whole system.

Standards make the microservice approach work by agreeing on how to communicate and being tolerant of only the things you need from a service, rather than rigid contracts. It is important to define these contracts up front in the design, because services update independently of each other. Another description coined for designing with a microservices approach is "fine-grained service-oriented architecture (SOA)."

***At its simplest, the microservices design approach is about a decoupled federation of services, with independent changes to each, and agreed-upon standards for communication.***

As more cloud apps are produced, people discover that this decomposition of the overall app into independent, scenario-focused services is a better long-term approach.

## Comparison between application development approaches



- 1) A monolithic app contains domain-specific functionality and is normally divided by functional layers, such as web, business, and data.
- 2) You scale a monolithic app by cloning it on multiple servers/virtual machines/containers.
- 3) A microservice application separates functionality into separate smaller services.
- 4) The microservices approach scales out by deploying each service independently, creating instances of these services across servers/virtual machines/containers.

Designing with a microservice approach is not a panacea for all projects, but it does align more closely with the business objectives described earlier. Starting with a monolithic approach might be acceptable if you know that you will have the opportunity to rework the code later into a microservices design. More commonly, you begin with a monolithic application and slowly break it up in stages, starting with the functional areas that need to be more scalable or agile.

To summarize, the microservice approach is to compose your application of many small services. The services run in containers that are deployed across a cluster of machines. Smaller teams develop a service that focuses on a scenario and independently test, version, deploy, and scale each service so that the entire application can evolve.

## What is a microservice?

There are different definitions of microservices. If you search the Internet, you'll find many useful resources that provide their own viewpoints and definitions. However, most of the following characteristics of microservices are widely agreed upon:

- Encapsulate a customer or business scenario. What is the problem you are solving?
- Developed by a small engineering team.
- Written in any programming language and use any framework.
- Consist of code and (optionally) state, both of which are independently versioned, deployed, and scaled.
- Interact with other microservices over well-defined interfaces and protocols.
- Have unique names (URLs) used to resolve their location.
- Remain consistent and available in the presence of failures.

You can summarize these characteristics into:

***Microservice applications are composed of small, independently versioned, and scalable customer-focused services that communicate with each other over standard protocols with well-defined interfaces.***

We covered the first two points in the preceding section, and now we expand on and clarify the others.

### **Written in any programming language and use any framework**

As developers, we should be free to choose a language or framework that we want, depending on our skills or the needs of the service. In some services, you might value the performance benefits of C++ above all else. In other services, the ease of managed development in C# or Java might be most important. In some cases, you may need to use a specific partner library, data storage technology, or means of exposing the service to clients.

After you have chosen a technology, you come to the operational or lifecycle management and scaling of the service.

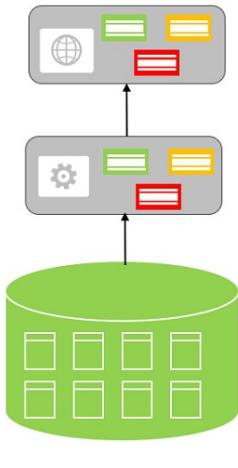
### **Allows code and state to be independently versioned, deployed, and scaled**

However you choose to write your microservices, the code and optionally the state should independently deploy, upgrade, and scale. This is actually one of the harder problems to solve, because it comes down to your choice of technologies. For scaling, understanding how to partition (or shard) both the code and state is challenging. When the code and state use separate technologies, which is common today, the deployment scripts for your microservice need to be able to cope with scaling them both. This is also about agility and flexibility, so you can upgrade some of the microservices without having to upgrade all of them at once.

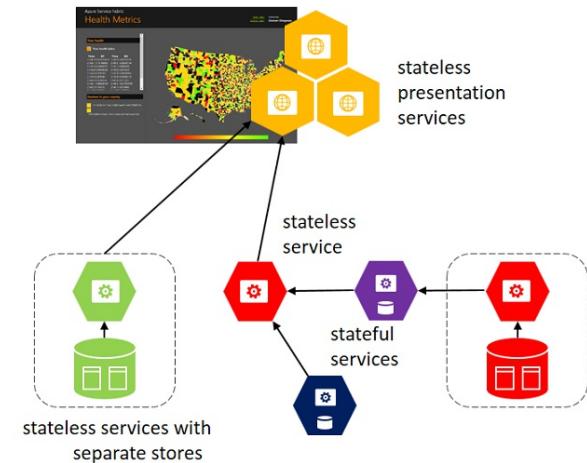
Returning to the monolithic versus microservice approach for a moment, the following diagram shows the differences in the approach to storing state.

### **State storage between application styles**

## State in Monolithic approach



## State in Microservices approach



**The monolithic approach on the left has a single database and tiers of specific technologies.**

**The microservices approach on the right has a graph of interconnected microservices where state is typically scoped to the microservice and various technologies are used.**

In a monolithic approach, typically the application uses a single database. The advantage is that it is a single location, which makes it easy to deploy. Each component can have a single table to store its state. Teams need to strictly separate state, which is a challenge. Inevitably there are temptations to add a new column to an existing customer table, do a join between tables, and create dependencies at the storage layer. After this happens, you can't scale individual components.

In the microservices approach, each service manages and stores its own state. Each service is responsible for scaling both code and state together to meet the demands of the service. A downside is that when there is a need to create views, or queries, of your application's data, you need to query across disparate state stores. Typically, this is solved by having a separate microservice that builds a view across a collection of microservices. If you need to perform multiple impromptu queries on the data, each microservice should consider writing its data to a data warehousing service for offline analytics.

Versioning is specific to the deployed version of a microservice so that multiple, different versions deploy and run side by side. Versioning addresses the scenarios where a newer version of a microservice fails during upgrade and needs to roll back to an earlier version. The other scenario for versioning is performing A/B-style testing, where different users experience different versions of the service. For example, it is common to upgrade a microservice for a specific set of customers to test new functionality before rolling it out more widely. After lifecycle management of microservices, this now brings us to communication between them.

### Interacts with other microservices over well-defined interfaces and protocols

This topic needs little attention here, because extensive literature about service-oriented architecture that has been published over the past 10 years describes communication patterns. Generally, service communication uses a REST approach with HTTP and TCP protocols and XML or JSON as the serialization format. From an interface perspective, it is about embracing the web design approach. But nothing stops you from using binary protocols or your own data formats. Be prepared for people to have a harder time using your microservices if these protocols and formats are not openly available.

### Has a unique name (URL) used to resolve its location

Remember how we keep saying that the microservice approach is like the web? Like the web, your microservice needs to be addressable wherever it is running. If you are thinking about machines and which one is running a particular microservice, things go bad quickly.

In the same way that DNS resolves a particular URL to a particular machine, your microservice needs to have a unique name so that its current location is discoverable. Microservices need addressable names that make them independent from the infrastructure that they are running on. This implies that there is an interaction between how your service is deployed and how it is discovered, because there needs to be a service registry. Equally, when a machine fails, the registry service must tell you where the service is now running.

This brings us to the next topic: resilience and consistency.

### **Remains consistent and available in the presence of failures**

Dealing with unexpected failures is one of the hardest problems to solve, especially in a distributed system. Much of the code that we write as developers is handling exceptions, and this is also where the most time is spent in testing. The problem is more involved than writing code to handle failures. What happens when the machine where the microservice is running fails? Not only do you need to detect this microservice failure (a hard problem on its own), but you also need something to restart your microservice.

A microservice needs to be resilient to failures and restart often on another machine for availability reasons. This also comes down to the state that was saved on behalf of the microservice, where the microservice can recover this state from, and whether the microservice is able to restart successfully. In other words, there needs to be resilience in the compute (the process restarts) as well as resilience in the state or data (no data loss and the data remains consistent).

The problems of resiliency are compounded during other scenarios, such as when failures happen during an application upgrade. The microservice, working with the deployment system, doesn't need to recover. It also needs to then decide whether it can continue to move forward to the newer version or instead roll back to a previous version to maintain a consistent state. Questions such as whether enough machines are available to keep moving forward and how to recover previous versions of the microservice need to be considered. This requires the microservice to emit health information to be able to make these decisions.

### **Reports health and diagnostics**

It may seem obvious, and it is often overlooked, but a microservice must report its health and diagnostics. Otherwise, there is little insight from an operations perspective. Correlating diagnostic events across a set of independent services and dealing with machine clock skews to make sense of the event order is challenging. In the same way that you interact with a microservice over agreed-upon protocols and data formats, there emerges a need for standardization in how to log health and diagnostic events that ultimately end up in an event store for querying and viewing. In a microservices approach, it is key that different teams agree on a single logging format. There needs to be a consistent approach to viewing diagnostic events in the application as a whole.

Health is different from diagnostics. Health is about the microservice reporting its current state to take appropriate actions. A good example is working with upgrade and deployment mechanisms to maintain availability. Although a service may be currently unhealthy due to a process crash or machine reboot, the service might still be operational. The last thing you need is to make this worse by performing an upgrade. The best approach is to do an investigation first or allow time for the microservice to recover. Health events from a microservice help us make informed decisions and, in effect, help create self-healing services.

## **Service Fabric as a microservices platform**

Azure Service Fabric emerged from a transition by Microsoft from delivering box products, which were typically monolithic in style, to delivering services. The experience of building and operating large services, such as Azure SQL Database and Azure Cosmos DB, shaped Service Fabric. The platform evolved over time as more and more services adopted it. Importantly, Service Fabric had to run not only in Azure but also in standalone Windows Server deployments.

***The aim of Service Fabric is to solve the hard problems of building and running a service and utilize infrastructure resources efficiently, so that teams can solve business problems using a microservices approach.***

Service Fabric provides three broad areas to help you build applications that use a microservices approach:

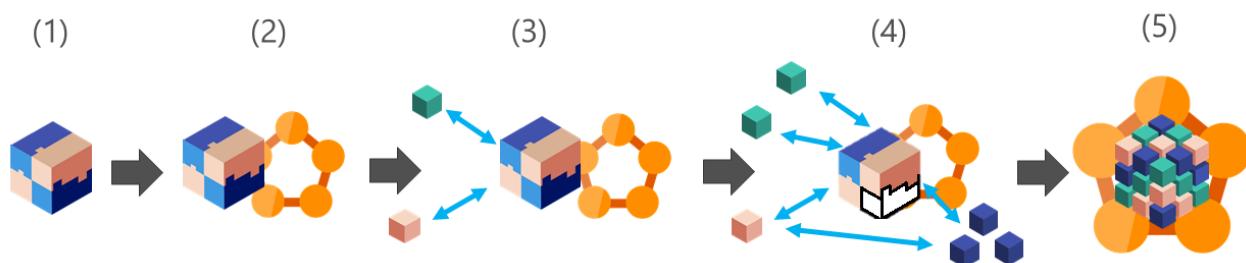
- A platform that provides system services to deploy, upgrade, detect, and restart failed services, discover services, route messages, manage state, and monitor health. These system services in effect enable many of the characteristics of microservices previously described.
- Ability to deploy applications either running in containers or as processes. Service Fabric is a container and process orchestrator.
- Productive programming APIs, to help you build applications as microservices: [ASP.NET Core](#), [Reliable Actors](#), and [Reliable Services](#). You can choose any code to build your microservice. But these APIs make the job more straightforward, and they integrate with the platform at a deeper level. This way, for example, you can get health and diagnostics information, or you can take advantage of built-in high availability.

***Service Fabric is agnostic on how you build your service, and you can use any technology. However, it does provide built-in programming APIs that make it easier to build microservices.***

### Migrating existing applications to Service Fabric

A key approach to Service Fabric is to reuse existing code, which can then be modernized with new microservices. There are five stages to application modernization, and you can start and stop at any of the stages. These are;

- 1) Take a traditional monolithic application
- 2) Lift and Shift - Use containers or guest executables to host existing code in Service Fabric.
- 3) Modernization - New microservices added alongside existing containerized code.
- 4) Innovate - Break the monolithic into microservices purely based on need.
- 5) Transformed into microservices - the transformation of existing monolithic applications or building new greenfield applications.



It is important to emphasize again that you can **start and stop at any of these stages**. You are not compelled to progress to the next stage. Let's now look at examples for each of these stages.

**Lift and Shift** - large numbers of companies are lifting and shifting existing monolithic applications into containers for two reasons;

- Cost reduction either due to consolidation and removal of existing hardware or running applications at higher density.
- Consistent deployment contract between development and operations.

Cost reductions are understandable, and within Microsoft, large numbers of existing applications are being containerized simply to save millions of dollars. Consistent deployment is harder to evaluate, but equally as important. It says that developers can still be free to choose the technology that suites them, however the operations will only accept a single way to deploy and manage these applications. It alleviates the operations from having to deal with the complexity of many different technologies or forcing developers to only choose certain ones. Essentially every application is containerized into self-contained deployment images.

Many organizations stop here. They already have the benefits of containers and Service Fabric provides the complete management experience from deployment, upgrades, versioning, rollbacks, health monitoring etc.

**Modernization** - is the addition of new services alongside existing containerized code. If you are going to write new code, it is best to decide to take small steps down the microservices path. This could be adding a new REST

API endpoint, or new business logic. This way, you start on the journey of building new microservices and practice developing and deploying them.

**Innovate** - remember those original changing business needs at the start of this article, that are driving the microservices approach? At this stage the decision is, are these happening to my current application and if so, I need to start splitting the monolith, or innovating. An example here is when a database becomes a processing bottleneck, since it is being used as a workflow queue. As the number of workflow requests increase the work needs to be distributed for scale. So for that particular piece of the application that is not scaling, or you need to update more frequently, split this out into a microservice and innovate.

**Transformed into microservices** - this is where your application is fully composed of (or decomposed into) microservices. To reach here, you have made the microservices journey. You can start here, but to do this without a microservices platform to help you is a significant investment.

### **Are microservices right for my application?**

Maybe. What we experienced was that as more and more teams in Microsoft began to build for the cloud for business reasons, many of them realized the benefits of taking a microservice-like approach. Bing, for example, has been developing microservices in search for years. For other teams, the microservices approach was new. Teams found that there were hard problems to solve outside of their core areas of strength. This is why Service Fabric gained traction as the technology of choice for building services.

The objective of Service Fabric is to reduce the complexities of building applications with a microservice approach, so that you do not have to go through as many costly redesigns. Start small, scale when needed, deprecate services, add new ones, and evolve with customer usage is the approach. We also know that there are many other problems yet to be solved to make microservices more approachable for most developers. Containers and the actor programming model are examples of small steps in that direction, and we are sure that more innovations will emerge to make this easier.

## Next steps

- [Service Fabric terminology overview](#)
- [Microservices: An application revolution powered by the cloud](#)

# So you want to learn about Service Fabric?

10/10/2017 • 18 min to read • [Edit Online](#)

Azure Service Fabric is a distributed systems platform that makes it easy to package, deploy, and manage scalable and reliable microservices. Service Fabric has a large surface area, however, and there's a lot to learn. This article provides a synopsis of Service Fabric and describes the core concepts, programming models, application lifecycle, testing, clusters, and health monitoring. Read the [Overview](#) and [What are microservices?](#) for an introduction and how Service Fabric can be used to create microservices. This article does not contain a comprehensive content list, but does link to overview and getting started articles for every area of Service Fabric.

## Core concepts

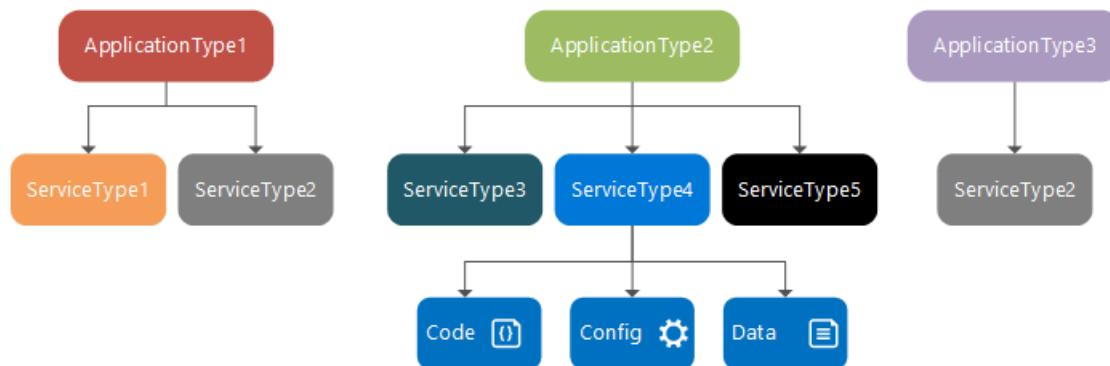
[Service Fabric terminology](#), [Application model](#), and [Supported programming models](#) provide more concepts and descriptions, but here are the basics.

| CORE CONCEPTS                                                                                                                                                                                                                                                                                                                                             | DESIGN TIME                                                                        | RUN TIME                                                                             |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| <p>Service Fabric: A Microservices Platform</p> <p>Build Applications with many Languages, Frameworks, &amp; Runtimes</p> <p>Service Fabric: Microservices Platform</p> <p>Lifecycle Mgmt Independent Scaling Rolling Upgrades Always On Availability Resource Efficient Stateless/ Stateful</p> <p>Public Clouds On Premises Private cloud Developer</p> |  |  |

### Design time: application type, service type, application package and manifest, service package and manifest

An application type is the name/version assigned to a collection of service types. This is defined in an *ApplicationManifest.xml* file, which is embedded in an application package directory. The application package is then copied to the Service Fabric cluster's image store. You can then create a named application from this application type, which then runs within the cluster.

A service type is the name/version assigned to a service's code packages, data packages, and configuration packages. This is defined in a *ServiceManifest.xml* file, which is embedded in a service package directory. The service package directory is then referenced by an application package's *ApplicationManifest.xml* file. Within the cluster, after creating a named application, you can create a named service from one of the application type's service types. A service type is described by its *ServiceManifest.xml* file. The service type is composed of executable code service configuration settings, which are loaded at run time, and static data that is consumed by the service.



The application package is a disk directory containing the application type's *ApplicationManifest.xml* file, which references the service packages for each service type that makes up the application type. For example, an application package for an email application type could contain references to a queue service package, a frontend

service package, and a database service package. The files in the application package directory are copied to the Service Fabric cluster's image store.

A service package is a disk directory containing the service type's *ServiceManifest.xml* file, which references the code, static data, and configuration packages for the service type. The files in the service package directory are referenced by the application type's *ApplicationManifest.xml* file. For example, a service package could refer to the code, static data, and configuration packages that make up a database service.

### Run time: clusters and nodes, named applications, named services, partitions, and replicas

A [Service Fabric cluster](#) is a network-connected set of virtual or physical machines into which your microservices are deployed and managed. Clusters can scale to thousands of machines.

A machine or VM that is part of a cluster is called a node. Each node is assigned a node name (a string). Nodes have characteristics such as placement properties. Each machine or VM has an auto-start Windows service, `FabricHost.exe`, which starts running upon boot and then starts two executables: `Fabric.exe` and `FabricGateway.exe`. These two executables make up the node. For development or testing scenarios, you can host multiple nodes on a single machine or VM by running multiple instances of `Fabric.exe` and `FabricGateway.exe`.

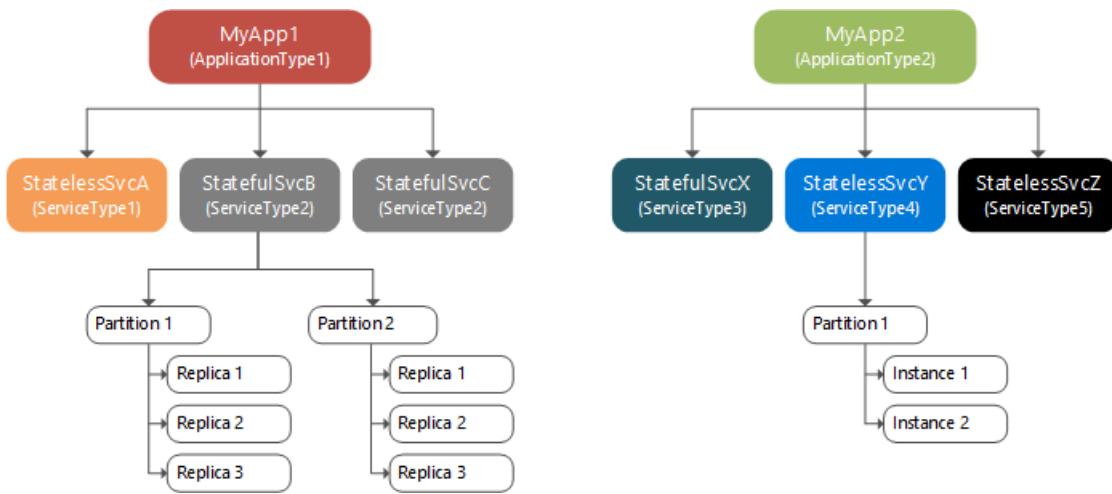
A named application is a collection of named services that performs a certain function or functions. A service performs a complete and standalone function (it can start and run independently of other services) and is composed of code, configuration, and data. After an application package is copied to the image store, you create an instance of the application within the cluster by specifying the application package's application type (using its name/version). Each application type instance is assigned a URI name that looks like *fabric:/MyNamedApp*. Within a cluster, you can create multiple named applications from a single application type. You can also create named applications from different application types. Each named application is managed and versioned independently.

After creating a named application, you can create an instance of one of its service types (a named service) within the cluster by specifying the service type (using its name/version). Each service type instance is assigned a URI name scoped under its named application's URI. For example, if you create a "MyDatabase" named service within a "MyNamedApp" named application, the URI looks like: *fabric:/MyNamedApp/MyDatabase*. Within a named application, you can create one or more named services. Each named service can have its own partition scheme and instance/replica counts.

There are two types of services: stateless and stateful. Stateless services can store persistent state in an external storage service such as Azure Storage, Azure SQL Database, or Azure Cosmos DB. Use a stateless service when the service has no persistent storage at all. A stateful service uses Service Fabric to manage your service's state via its Reliable Collections or Reliable Actors programming models.

When creating a named service, you specify a partition scheme. Services with large amounts of state split the data across partitions. Each partition is responsible for a portion of the complete state of the service, which is spread across the cluster's nodes.

The following diagram shows the relationship between applications and service instances, partitions, and replicas.



### Partitioning, scaling, and availability

[Partitioning](#) is not unique to Service Fabric. A well known form of partitioning is data partitioning, or sharding. Stateful services with large amounts of state split the data across partitions. Each partition is responsible for a portion of the complete state of the service.

The replicas of each partition are spread across the cluster's nodes, which allows your named service's state to [scale](#). As the data needs grow, partitions grow, and Service Fabric rebalances partitions across nodes to make efficient use of hardware resources. If you add new nodes to the cluster, Service Fabric will rebalance the partition replicas across the increased number of nodes. Overall application performance improves and contention for access to memory decreases. If the nodes in the cluster are not being used efficiently, you can decrease the number of nodes in the cluster. Service Fabric again rebalances the partition replicas across the decreased number of nodes to make better use of the hardware on each node.

Within a partition, stateless named services have instances while stateful named services have replicas. Usually, stateless named services only ever have one partition since they have no internal state. The partition instances provide for [availability](#). If one instance fails, other instances continue to operate normally and then Service Fabric creates a new instance. Stateful named services maintain their state within replicas and each partition has its own replica set. Read and write operations are performed at one replica (called the Primary). Changes to state from write operations are replicated to multiple other replicas (called Active Secondaries). Should a replica fail, Service Fabric builds a new replica from the existing replicas.

## Stateless and stateful microservices for Service Fabric

Service Fabric enables you to build applications that consist of microservices or containers. Stateless microservices (such as protocol gateways and web proxies) do not maintain a mutable state outside a request and its response from the service. Azure Cloud Services worker roles are an example of a stateless service. Stateful microservices (such as user accounts, databases, devices, shopping carts, and queues) maintain a mutable, authoritative state beyond the request and its response. Today's Internet-scale applications consist of a combination of stateless and stateful microservices.

A key differentiation with Service Fabric is its strong focus on building stateful services, either with the [built in programming models](#) or with containerized stateful services. The [application scenarios](#) describe the scenarios where stateful services are used.

Why have stateful microservices along with stateless ones? The two main reasons are:

- You can build high-throughput, low-latency, failure-tolerant online transaction processing (OLTP) services by keeping code and data close on the same machine. Some examples are interactive storefronts, search, Internet of Things (IoT) systems, trading systems, credit card processing and fraud detection systems, and personal record management.
- You can simplify application design. Stateful microservices remove the need for additional queues and caches,

which are traditionally required to address the availability and latency requirements of a purely stateless application. Stateful services are naturally high-availability and low-latency, which reduces the number of moving parts to manage in your application as a whole.

## Supported programming models

Service Fabric offers multiple ways to write and manage your services. Services can use the Service Fabric APIs to take full advantage of the platform's features and application frameworks. Services can also be any compiled executable program written in any language and hosted on a Service Fabric cluster. For more information, see [Supported programming models](#).

### Containers

By default, Service Fabric deploys and activates services as processes. Service Fabric can also deploy services in [containers](#). Importantly, you can mix services in processes and services in containers in the same application. Service Fabric supports deployment of Linux containers Windows containers on Windows Server 2016. You can deploy existing applications, stateless services, or stateful services in containers.

### Reliable Services

[Reliable Services](#) is a lightweight framework for writing services that integrate with the Service Fabric platform and benefit from the full set of platform features. Reliable Services can be stateless (similar to most service platforms, such as web servers or Worker Roles in Azure Cloud Services), where state is persisted in an external solution, such as Azure DB or Azure Table Storage. Reliable Services can also be stateful, where state is persisted directly in the service itself using Reliable Collections. State is made [highly available](#) through replication and distributed through [partitioning](#), all managed automatically by Service Fabric.

### Reliable Actors

Built on top of Reliable Services, the [Reliable Actor](#) framework is an application framework that implements the Virtual Actor pattern, based on the actor design pattern. The Reliable Actor framework uses independent units of compute and state with single-threaded execution called actors. The Reliable Actor framework provides built in communication for actors and pre-set state persistence and scale-out configurations.

### ASP.NET Core

Service Fabric integrates with [ASP.NET Core](#) as a first class programming model for building web and API applications. ASP.NET Core can be used in two different ways in Service Fabric:

- Hosted as a guest executable. This is primarily used to run existing ASP.NET Core applications on Service Fabric with no code changes.
- Run inside a Reliable Service. This allows better integration with the Service Fabric runtime and allows stateful ASP.NET Core services.

### Guest executables

A [guest executable](#) is an existing, arbitrary executable (written in any language) hosted on a Service Fabric cluster alongside other services. Guest executables do not integrate directly with Service Fabric APIs. However they still benefit from features the platform offers, such as custom health and load reporting and service discoverability by calling REST APIs. They also have full application lifecycle support.

## Application lifecycle

As with other platforms, an application on Service Fabric usually goes through the following phases: design, development, testing, deployment, upgrade, maintenance, and removal. Service Fabric provides first-class support for the full application lifecycle of cloud applications, from development through deployment, daily management, and maintenance to eventual decommissioning. The service model enables several different roles to participate independently in the application lifecycle. [Service Fabric application lifecycle](#) provides an overview of the APIs and how they are used by the different roles throughout the phases of the Service Fabric application lifecycle.

The entire app lifecycle can be managed using [PowerShell cmdlets](#), [C# APIs](#), [Java APIs](#), and [REST APIs](#). You can also set up continuous integration/continuous deployment pipelines using tools such as [Visual Studio Team Services](#) or [Jenkins](#).

The following Microsoft Virtual Academy video describes how to manage your application lifecycle:



## Test applications and services

To create truly cloud-scale services, it is critical to verify that your applications and services can withstand real-world failures. The Fault Analysis Service is designed for testing services that are built on Service Fabric. With the [Fault Analysis Service](#), you can induce meaningful faults and run complete test scenarios against your applications. These faults and scenarios exercise and validate the numerous states and transitions that a service will experience throughout its lifetime, all in a controlled, safe, and consistent manner.

[Actions](#) target a service for testing using individual faults. A service developer can use these as building blocks to write complicated scenarios. Examples of simulated faults are:

- Restart a node to simulate any number of situations where a machine or VM is rebooted.
- Move a replica of your stateful service to simulate load balancing, failover, or application upgrade.
- Invoke quorum loss on a stateful service to create a situation where write operations can't proceed because there aren't enough "back-up" or "secondary" replicas to accept new data.
- Invoke data loss on a stateful service to create a situation where all in-memory state is completely wiped out.

[Scenarios](#) are complex operations composed of one or more actions. The Fault Analysis Service provides two built-in complete scenarios:

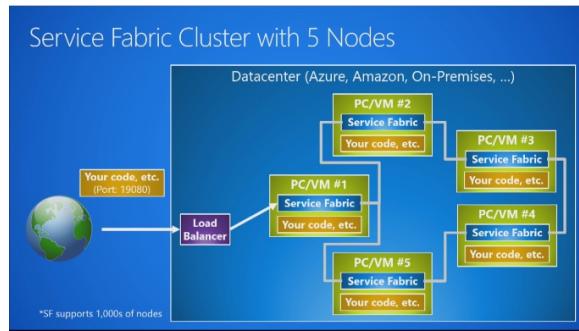
- [Chaos scenario](#)- simulates continuous, interleaved faults (both graceful and ungraceful) throughout the cluster over extended periods of time.
- [Failover scenario](#)- a version of the chaos test scenario that targets a specific service partition while leaving other services unaffected.

## Clusters

A [Service Fabric cluster](#) is a network-connected set of virtual or physical machines into which your microservices are deployed and managed. Clusters can scale to thousands of machines. A machine or VM that is part of a cluster is called a cluster node. Each node is assigned a node name (a string). Nodes have characteristics such as placement properties. Each machine or VM has an auto-start service, `FabricHost.exe`, which starts running upon boot and then starts two executables: `Fabric.exe` and `FabricGateway.exe`. These two executables make up the node. For testing scenarios, you can host multiple nodes on a single machine or VM by running multiple instances of `Fabric.exe` and `FabricGateway.exe`.

Service Fabric clusters can be created on virtual or physical machines running Windows Server or Linux. You are able to deploy and run Service Fabric applications in any environment where you have a set of Windows Server or Linux computers that are interconnected: on-premises, on Microsoft Azure, or on any cloud provider.

The following Microsoft Virtual Academy video describes Service Fabric clusters:



## Clusters on Azure

Running Service Fabric clusters on Azure provides integration with other Azure features and services, which makes operations and management of the cluster easier and more reliable. A cluster is an Azure Resource Manager resource, so you can model clusters like any other resources in Azure. Resource Manager also provides easy management of all resources used by the cluster as a single unit. Clusters on Azure are integrated with Azure diagnostics and Log Analytics. Cluster node types are [virtual machine scale sets](#), so autoscaling functionality is built in.

You can create a cluster on Azure through the [Azure portal](#), from a [template](#), or from [Visual Studio](#).

Service Fabric on Linux enables you to build, deploy, and manage highly available, highly scalable applications on Linux just as you would on Windows. The Service Fabric frameworks (Reliable Services and Reliable Actors) are available in Java on Linux, in addition to C# (.NET Core). You can also build [guest executable services](#) with any language or framework. Orchestrating Docker containers is also supported. Docker containers can run guest executables or native Service Fabric services, which use the Service Fabric frameworks. For more information, read about [Service Fabric on Linux](#).

There are some features that are supported on Windows, but not on Linux. To learn more, read [Differences between Service Fabric on Linux and Windows](#).

## Standalone clusters

Service Fabric provides an installation package for you to create standalone Service Fabric clusters on-premises or on any cloud provider. Standalone clusters give you the freedom to host a cluster wherever you want. If your data is subject to compliance or regulatory constraints, or you want to keep your data local, you can host your own cluster and applications. Service Fabric applications can run in multiple hosting environments with no changes, so your knowledge of building applications carries over from one hosting environment to another.

### Create your first Service Fabric standalone cluster

Linux standalone clusters are not yet supported.

## Cluster security

Clusters must be secured to prevent unauthorized users from connecting to your cluster, especially when it has production workloads running on it. Although it is possible to create an unsecured cluster, doing so allows anonymous users to connect to it if management endpoints are exposed to the public internet. It is not possible to later enable security on an unsecured cluster: cluster security is enabled at cluster creation time.

The cluster security scenarios are:

- Node-to-node security
- Client-to-node security
- Role-based access control (RBAC)

For more information, read [Secure a cluster](#).

## Scaling

If you add new nodes to the cluster, Service Fabric rebalances the partition replicas and instances across the

increased number of nodes. Overall application performance improves and contention for access to memory decreases. If the nodes in the cluster are not being used efficiently, you can decrease the number of nodes in the cluster. Service Fabric again rebalances the partition replicas and instances across the decreased number of nodes to make better use of the hardware on each node. You can scale clusters on Azure either [manually](#) or [programmatically](#). Standalone clusters can be scaled [manually](#).

## Cluster upgrades

Periodically, new versions of the Service Fabric runtime are released. Perform runtime, or fabric, upgrades on your cluster so that you are always running a [supported version](#). In addition to fabric upgrades, you can also update cluster configuration such as certificates or application ports.

A Service Fabric cluster is a resource that you own, but is partly managed by Microsoft. Microsoft is responsible for patching the underlying OS and performing fabric upgrades on your cluster. You can set your cluster to receive automatic fabric upgrades, when Microsoft releases a new version, or choose to select a supported fabric version that you want. Fabric and configuration upgrades can be set through the Azure portal or through Resource Manager. For more information, read [Upgrade a Service Fabric cluster](#).

A standalone cluster is a resource that you entirely own. You are responsible for patching the underlying OS and initiating fabric upgrades. If your cluster can connect to <https://www.microsoft.com/download>, you can set your cluster to automatically download and provision the new Service Fabric runtime package. You would then initiate the upgrade. If your cluster can't access <https://www.microsoft.com/download>, you can manually download the new runtime package from an internet connected machine and then initiate the upgrade. For more information, read [Upgrade a standalone Service Fabric cluster](#).

## Health monitoring

Service Fabric introduces a [health model](#) designed to flag unhealthy cluster and application conditions on specific entities (such as cluster nodes and service replicas). The health model uses health reporters (system components and watchdogs). The goal is easy and fast diagnosis and repair. Service writers need to think upfront about health and how to [design health reporting](#). Any condition that can impact health should be reported on, especially if it can help flag problems close to the root. The health information can save time and effort on debugging and investigation once the service is up and running at scale in production.

The Service Fabric reporters monitor identified conditions of interest. They report on those conditions based on their local view. The [health store](#) aggregates health data sent by all reporters to determine whether entities are globally healthy. The model is intended to be rich, flexible, and easy to use. The quality of the health reports determines the accuracy of the health view of the cluster. False positives that wrongly show unhealthy issues can negatively impact upgrades or other services that use health data. Examples of such services are repair services and alerting mechanisms. Therefore, some thought is needed to provide reports that capture conditions of interest in the best possible way.

Reporting can be done from:

- The monitored Service Fabric service replica or instance.
- Internal watchdogs deployed as a Service Fabric service (for example, a Service Fabric stateless service that monitors conditions and issues reports). The watchdogs can be deployed on all nodes or can be affinitized to the monitored service.
- Internal watchdogs that run on the Service Fabric nodes but are not implemented as Service Fabric services.
- External watchdogs that probe the resource from outside the Service Fabric cluster (for example, monitoring service like Gomez).

Out of the box, Service Fabric components report health on all entities in the cluster. [System health reports](#) provide visibility into cluster and application functionality and flag issues through health. For applications and services, system health reports verify that entities are implemented and are behaving correctly from the perspective of the Service Fabric runtime. The reports do not provide any health monitoring of the business logic of the service or

detect hung processes. To add health information specific to your service's logic, [implement custom health reporting](#) in your services.

Service Fabric provides multiple ways to [view health reports](#) aggregated in the health store:

- [Service Fabric Explorer](#) or other visualization tools.
- Health queries (through [PowerShell](#), the [C# FabricClient APIs](#) and [Java FabricClient APIs](#), or [REST APIs](#)).
- General queries that return a list of entities that have health as one of the properties (through [PowerShell](#), the API, or REST).

The following Microsoft Virtual Academy video describes the Service Fabric health model and how it's used:



## Next steps

- Learn how to create a [cluster in Azure](#) or a [standalone cluster on Windows](#).
- Try creating a service using the [Reliable Services](#) or [Reliable Actors](#) programming models.
- Learn how to [migrate from Cloud Services](#).
- Learn to [monitor and diagnose services](#).
- Learn to [test your apps and services](#).
- Learn to [manage and orchestrate cluster resources](#).
- Look through the [Service Fabric samples](#).
- Learn about [Service Fabric support options](#).
- Read the [team blog](#) for articles and announcements.

# Service Fabric application scenarios

7/12/2017 • 4 min to read • [Edit Online](#)

Azure Service Fabric offers a reliable and flexible platform that enables you to write and run many types of business applications and services. These applications and microservices can be stateless or stateful, and they are resource-balanced across virtual machines to maximize efficiency. The unique architecture of Service Fabric enables you to perform near real-time data analysis, in-memory computation, parallel transactions, and event processing in your applications. You can easily scale your applications up or down (really in or out), depending on your changing resource requirements.

The Service Fabric platform in Azure is ideal for the following categories of applications:

- **Highly available services:** Service Fabric services provide fast failover by creating multiple secondary service replicas. If a node, process, or individual service goes down due to hardware or other failure, one of the secondary replicas is promoted to a primary replica with minimal loss of service.
- **Scalable services:** Individual services can be partitioned, allowing for state to be scaled out across the cluster. In addition, individual services can be created and removed on the fly. Services can be quickly and easily scaled out from a few instances on a few nodes to thousands of instances on many nodes, and then scaled in again, depending on your resource needs. You can use Service Fabric to build these services and manage their complete lifecycles.
- **Computation on nonstatic data:** Service Fabric enables you to build data, input/output, and compute intensive stateful applications. Service Fabric allows the collocation of processing (computation) and data in applications. Normally, when your application requires access to data, there is network latency associated with an external data cache or storage tier. With stateful Service Fabric services, that latency is eliminated, enabling more performant reads and writes. Say for example that you have an application that performs near real-time recommendation selections for customers, with a round-trip time requirement of less than 100 milliseconds. The latency and performance characteristics of Service Fabric services (where the computation of recommendation selection is collocated with the data and rules) provides a responsive experience to the user compared with the standard implementation model of having to fetch the necessary data from remote storage.
- **Session-based interactive applications:** Service Fabric is useful if your applications, such as online gaming or instant messaging, require low latency reads and writes. Service Fabric enables you to build these interactive, stateful applications without having to create a separate store or cache, as required for stateless apps. (This increases latency and potentially introduces consistency issues.).
- **Data analytics and workflows:** The fast reads and writes of Service Fabric enable applications that must reliably process events or streams of data. Service Fabric also enables applications that describe processing pipelines, where results must be reliable and passed on to the next processing stage without loss. These include transactional and financial systems, where data consistency and computation guarantees are essential.
- **Data gathering, processing and IoT:** Since Service Fabric handles large scale and has low latency through its stateful services, it is ideal for data processing on millions of devices where the data for the device and the computation are co-located. We have seen several customers who have built IoT systems using Service Fabric including [BMW](#), [Schneider Electric](#) and [Mesh Systems](#).

## Application design case studies

A number of case studies showing how Service Fabric is used to design applications are published on the [Service Fabric team blog](#) and the [microservices solutions site](#).

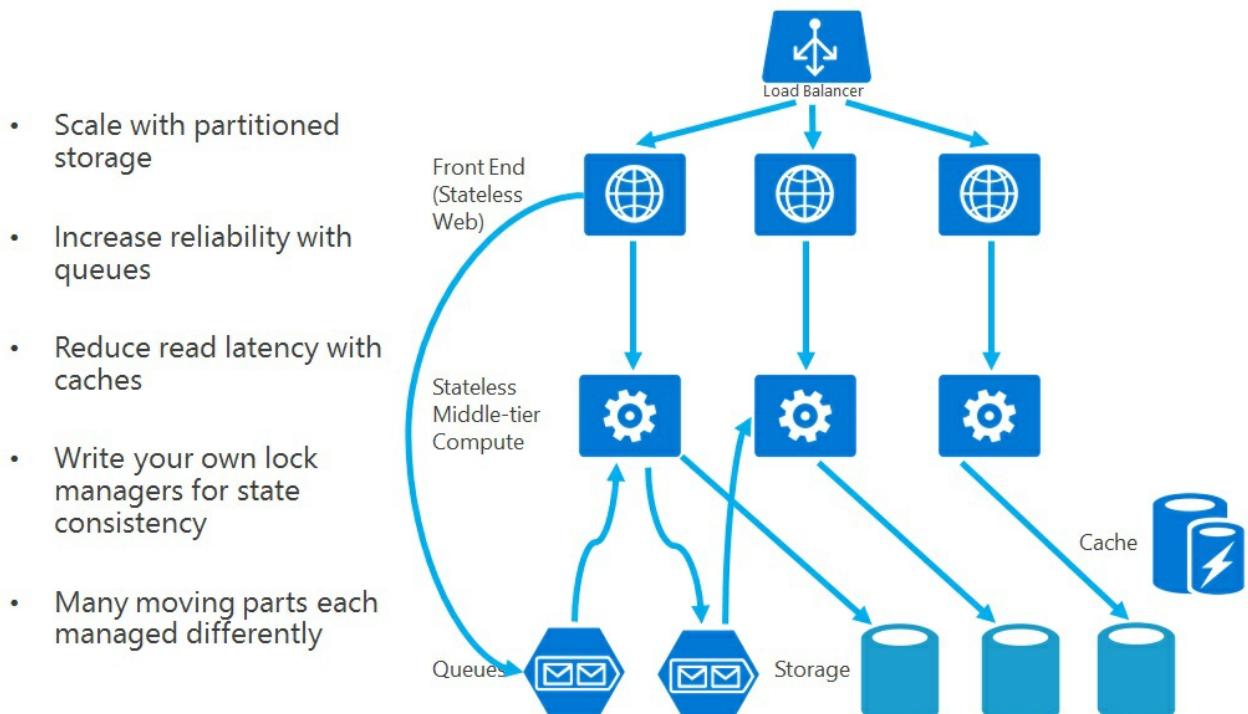
## Design applications composed of stateless and stateful microservices

Building applications with Azure Cloud Service worker roles is an example of a stateless service. In contrast, stateful microservices maintain their authoritative state beyond the request and its response. This provides high availability and consistency of the state through simple APIs that provide transactional guarantees backed by replication. Service Fabric's stateful services democratize high availability, bringing it to all types of applications, not just databases and other data stores. This is a natural progression. Applications have already moved from using purely relational databases for high availability to NoSQL databases. Now the applications themselves can have their "hot" state and data managed within them for additional performance gains without sacrificing reliability, consistency, or availability.

When building applications consisting of microservices, you typically have a combination of stateless web apps (ASP.NET, Node.js, etc.) calling onto stateless and stateful business middle-tier services, all deployed into the same Service Fabric cluster using the Service Fabric deployment commands. Each of these services is independent with regard to scale, reliability, and resource usage, greatly improving agility in development and lifecycle management.

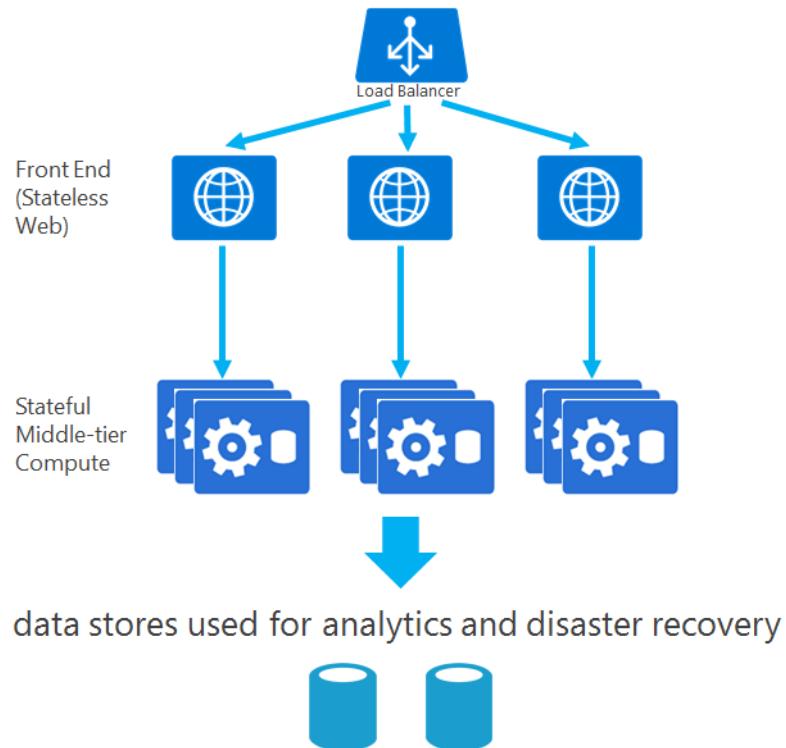
Stateful microservices simplify application designs because they remove the need for the additional queues and caches that have traditionally been required to address the availability and latency requirements of purely stateless applications. Since stateful services are naturally highly available and low latency, this means that there are fewer moving parts to manage in your application as a whole. The diagrams below illustrate the differences between designing an application that is stateless and one that is stateful. By taking advantage of the [Reliable Services](#) and [Reliable Actors](#) programming models, stateful services reduce application complexity while achieving high throughput and low latency.

## An application built using stateless services



## An application built using stateful services

- Application state lives in the compute tier
- Low Latency reads and writes
- Partitions are first class for scale-out
- Built in lock managers based on primary election
- Fewer moving parts



## Next steps

- Listen to [customer case studies](#)
- Read about [customer case studies](#)
- Learn more about [patterns and scenarios](#)
- Get started building stateless and stateful services with the Service Fabric [reliable services](#) and [reliable actors](#) programming models.
- Also see the following topics:
  - [Tell me about microservices](#)
  - [Define and manage service state](#)
  - [Availability of Service Fabric services](#)
  - [Scale Service Fabric services](#)
  - [Partition Service Fabric services](#)

# Service Fabric patterns and scenarios

10/5/2017 • 1 min to read • [Edit Online](#)

If you're looking at building large-scale microservices using Azure Service Fabric, learn from the experts who designed and built this platform as a service (PaaS). Get started with proper architecture, and then learn how to optimize resources for your application. The [Service Fabric Patterns and Practices](#) course answers the questions most often asked by real-world customers about Service Fabric scenarios and application areas.

Find out how to design, develop, and operate your microservices on Service Fabric using best practices and proven, reusable patterns. Get an overview of Service Fabric and then dive deep into topics that cover cluster optimization and security, migrating legacy apps, IoT at scale, hosting game engines, and more. Look at continuous delivery for various workloads, and even get the details on Linux support and containers.

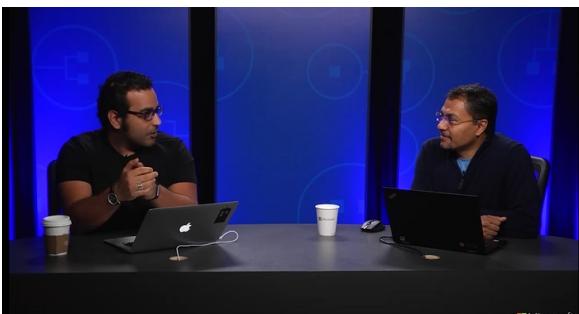
## Introduction

Explore best practices, and learn about choosing platform as a service (PaaS) over infrastructure as a service (IaaS). Get the details on following proven application design principles.

| VIDEO                                                                              | POWERPOINT DECK                                |
|------------------------------------------------------------------------------------|------------------------------------------------|
|  | <a href="#">Introduction to Service Fabric</a> |

## Cluster planning and management

Learn about capacity planning, cluster optimization, and cluster security, in this look at Azure Service Fabric.

| VIDEO                                                                               | POWERPOINT DECK                                 |
|-------------------------------------------------------------------------------------|-------------------------------------------------|
|  | <a href="#">Cluster Planning and Management</a> |

## Hyper-scale web

Review concepts around hyper-scale web, including availability and reliability, hyper-scale, and state management.

| VIDEO                                                                             | POWERPOINT DECK |
|-----------------------------------------------------------------------------------|-----------------|
|  | Hyper-scale web |

## IoT

Explore the Internet of Things (IoT) in the context of Azure Service Fabric, including the Azure IoT pipeline, multi-tenancy, and IoT at scale.

| VIDEO                                                                              | POWERPOINT DECK |
|------------------------------------------------------------------------------------|-----------------|
|  | IoT             |

## Gaming

Look at turn-based games, interactive games, and hosting existing game engines.

| VIDEO                                                                               | POWERPOINT DECK |
|-------------------------------------------------------------------------------------|-----------------|
|  | Gaming          |

## Continuous delivery

Explore concepts, including continuous integration/continuous delivery with Visual Studio Team Services, build/package/publish workflow, multi-environment setup, and service package/share.

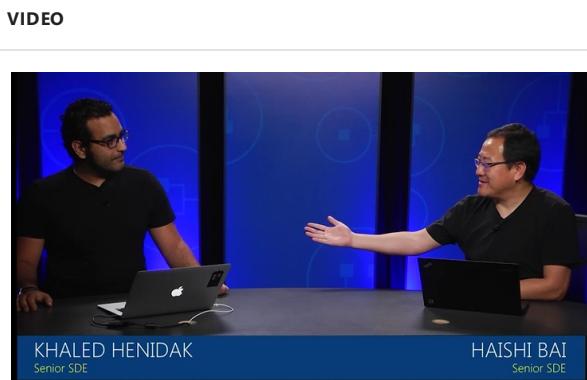
| VIDEO | POWERPOINT DECK |
|-------|-----------------|
|       |                 |



Continuous delivery

## Migration

Learn about migrating from a cloud service, in addition to migration of legacy apps.

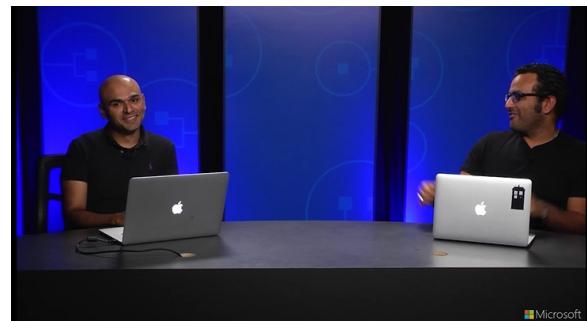


POWERPOINT DECK

Migration

## Containers and Linux support

Get the answer to the question, "Why containers?" Learn about the preview for Windows containers, Linux supports, and Linux containers orchestration. Plus, find out how to migrate .NET Core apps to Linux.



POWERPOINT DECK

Containers and Linux support

## Next steps

Now that you've learned about Service Fabric patterns and scenarios, read more about how to [create and manage clusters](#), [migrate Cloud Services apps to Service Fabric](#), [set up continuous delivery](#), and [deploy containers](#).

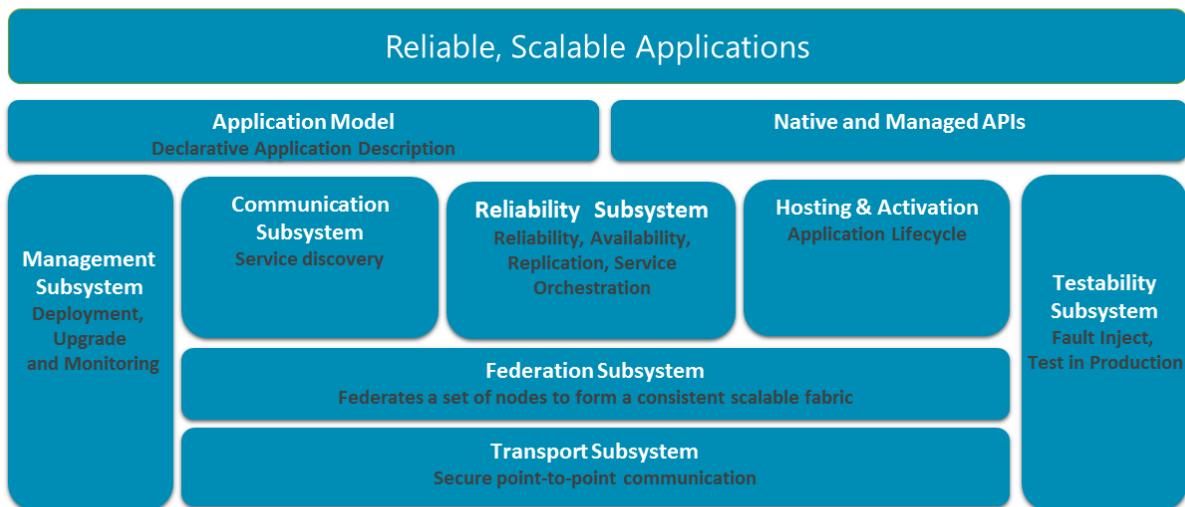
# Service Fabric architecture

10/30/2017 • 5 min to read • [Edit Online](#)

Service Fabric is built with layered subsystems. These subsystems enable you to write applications that are:

- Highly available
- Scalable
- Manageable
- Testable

The following diagram shows the major subsystems of Service Fabric.



In a distributed system, the ability to securely communicate between nodes in a cluster is crucial. At the base of the stack is the transport subsystem, which provides secure communication between nodes. Above the transport subsystem lies the federation subsystem, which clusters the different nodes into a single entity (named clusters) so that Service Fabric can detect failures, perform leader election, and provide consistent routing. The reliability subsystem, layered on top of the federation subsystem, is responsible for the reliability of Service Fabric services through mechanisms such as replication, resource management, and failover. The federation subsystem also underlies the hosting and activation subsystem, which manages the lifecycle of an application on a single node. The management subsystem manages the lifecycle of applications and services. The testability subsystem helps application developers test their services through simulated faults before and after deploying applications and services to production environments. Service Fabric provides the ability to resolve service locations through its communication subsystem. The application programming models exposed to developers are layered on top of these subsystems along with the application model to enable tooling.

## Transport subsystem

The transport subsystem implements a point-to-point datagram communication channel. This channel is used for communication within service fabric clusters and communication between the service fabric cluster and clients. It supports one-way and request-reply communication patterns, which provides the basis for implementing broadcast and multicast in the Federation layer. The transport subsystem secures communication by using X509

certificates or Windows security. This subsystem is used internally by Service Fabric and is not directly accessible to developers for application programming.

## Federation subsystem

In order to reason about a set of nodes in a distributed system, you need to have a consistent view of the system. The federation subsystem uses the communication primitives provided by the transport subsystem and stitches the various nodes into a single unified cluster that it can reason about. It provides the distributed systems primitives needed by the other subsystems - failure detection, leader election, and consistent routing. The federation subsystem is built on top of distributed hash tables with a 128-bit token space. The subsystem creates a ring topology over the nodes, with each node in the ring being allocated a subset of the token space for ownership. For failure detection, the layer uses a leasing mechanism based on heart beating and arbitration. The federation subsystem also guarantees through intricate join and departure protocols that only a single owner of a token exists at any time. This provides leader election and consistent routing guarantees.

## Reliability subsystem

The reliability subsystem provides the mechanism to make the state of a Service Fabric service highly available through the use of the *Replicator*, *Failover Manager*, and *Resource Balancer*.

- The Replicator ensures that state changes in the primary service replica will automatically be replicated to secondary replicas, maintaining consistency between the primary and secondary replicas in a service replica set. The replicator is responsible for quorum management among the replicas in the replica set. It interacts with the failover unit to get the list of operations to replicate, and the reconfiguration agent provides it with the configuration of the replica set. That configuration indicates which replicas the operations need to be replicated. Service Fabric provides a default replicator called Fabric Replicator, which can be used by the programming model API to make the service state highly available and reliable.
- The Failover Manager ensures that when nodes are added to or removed from the cluster, the load is automatically redistributed across the available nodes. If a node in the cluster fails, the cluster will automatically reconfigure the service replicas to maintain availability.
- The Resource Manager places service replicas across failure domain in the cluster and ensures that all failover units are operational. The Resource Manager also balances service resources across the underlying shared pool of cluster nodes to achieve optimal uniform load distribution.

## Management subsystem

The management subsystem provides end-to-end service and application lifecycle management. PowerShell cmdlets and administrative APIs enable you to provision, deploy, patch, upgrade, and de-provision applications without loss of availability. The management subsystem performs this through the following services.

- **Cluster Manager:** This is the primary service that interacts with the Failover Manager from reliability to place the applications on the nodes based on the service placement constraints. The Resource Manager in failover subsystem ensures that the constraints are never broken. The cluster manager manages the lifecycle of the applications from provision to de-provision. It integrates with the health manager to ensure that application availability is not lost from a semantic health perspective during upgrades.
- **Health Manager:** This service enables health monitoring of applications, services, and cluster entities. Cluster entities (such as nodes, service partitions, and replicas) can report health information, which is then aggregated into the centralized health store. This health information provides an overall point-in-time health snapshot of the services and nodes distributed across multiple nodes in the cluster, enabling you to take any needed corrective actions. Health query APIs enable you to query the health events reported to the health subsystem. The health query APIs return the raw health data stored in the health store or the aggregated, interpreted health data for a specific cluster entity.
- **Image Store:** This service provides storage and distribution of the application binaries. This service provides a

simple distributed file store where the applications are uploaded to and downloaded from.

## Hosting subsystem

The cluster manager informs the hosting subsystem (running on each node) which services it needs to manage for a particular node. The hosting subsystem then manages the lifecycle of the application on that node. It interacts with the reliability and health components to ensure that the replicas are properly placed and are healthy.

## Communication subsystem

This subsystem provides reliable messaging within the cluster and service discovery through the Naming service. The Naming service resolves service names to a location in the cluster and enables users to manage service names and properties. Using the Naming service, clients can securely communicate with any node in the cluster to resolve a service name and retrieve service metadata. Using a simple Naming client API, users of Service Fabric can develop services and clients capable of resolving the current network location despite node dynamism or the re-sizing of the cluster.

## Testability subsystem

Testability is a suite of tools specifically designed for testing services built on Service Fabric. The tools let a developer easily induce meaningful faults and run test scenarios to exercise and validate the numerous states and transitions that a service will experience throughout its lifetime, all in a controlled and safe manner. Testability also provides a mechanism to run longer tests that can iterate through various possible failures without losing availability. This provides you with a test-in-production environment.

# Service Fabric terminology overview

10/16/2017 • 9 min to read • [Edit Online](#)

Azure Service Fabric is a distributed systems platform that makes it easy to package, deploy, and manage scalable and reliable microservices. This article details the terminology used by Service Fabric to understand the terms used in the documentation.

The concepts listed in this section are also discussed in the following Microsoft Virtual Academy videos: [Core concepts](#), [Design-time concepts](#), and [Runtime concepts](#).

## Infrastructure concepts

**Cluster:** A network-connected set of virtual or physical machines into which your microservices are deployed and managed. Clusters can scale to thousands of machines.

**Node:** A machine or VM that's part of a cluster is called a *node*. Each node is assigned a node name (a string). Nodes have characteristics, such as placement properties. Each machine or VM has an auto-start Windows service, `FabricHost.exe`, that starts running upon boot and then starts two executables: `Fabric.exe` and `FabricGateway.exe`. These two executables make up the node. For testing scenarios, you can host multiple nodes on a single machine or VM by running multiple instances of `Fabric.exe` and `FabricGateway.exe`.

## Application concepts

**Application type:** The name/version assigned to a collection of service types. It is defined in an `ApplicationManifest.xml` file and embedded in an application package directory. The directory is then copied to the Service Fabric cluster's image store. You can then create a named application from this application type within the cluster.

Read the [Application model](#) article for more information.

**Application package:** A disk directory containing the application type's `ApplicationManifest.xml` file. References the service packages for each service type that makes up the application type. The files in the application package directory are copied to Service Fabric cluster's image store. For example, an application package for an email application type might contain references to a queue-service package, a frontend-service package, and a database-service package.

**Named application:** After you copy an application package to the image store, you create an instance of the application within the cluster. You create an instance when you specify the application package's application type, by using its name or version. Each application type instance is assigned a uniform resource identifier (URI) name that looks like: `"fabric:/MyNamedApp"`. Within a cluster, you can create multiple named applications from a single application type. You can also create named applications from different application types. Each named application is managed and versioned independently.

**Service type:** The name/version assigned to a service's code packages, data packages, and configuration packages. The service type is defined in the `ServiceManifest.xml` file and embedded in a service package directory. The service package directory is then referenced by an application package's `ApplicationManifest.xml` file. Within the cluster, after creating a named application, you can create a named service from one of the application type's service types. The service type's `ServiceManifest.xml` file describes the service.

Read the [Application model](#) article for more information.

There are two types of services:

- **Stateless:** Use a stateless service when the service's persistent state is stored in an external storage service, such as Azure Storage, Azure SQL Database, or Azure Cosmos DB. Use a stateless service when the service has no persistent storage. For example, for a calculator service where values are passed to the service, a computation is performed that uses these values, and then a result is returned.
- **Stateful:** Use a stateful service when you want Service Fabric to manage your service's state via its Reliable Collections or Reliable Actors programming models. When you create a named service, specify how many partitions you want to spread your state over for scalability. Also specify how many times to replicate your state across nodes, for reliability. Each named service has a single primary replica and multiple secondary replicas. You modify your named service's state when you write to the primary replica. Service Fabric then replicates this state to all the secondary replicas to keep your state in sync. Service Fabric automatically detects when a primary replica fails and promotes an existing secondary replica to a primary replica. Service Fabric then creates a new secondary replica.

**Replicas or instances** refer to code (and state for stateful services) of a service that's deployed and running. See [Replicas and instances](#).

**Reconfiguration** refers to the process of any change in the replica set of a service. See [Reconfiguration](#).

**Service package:** A disk directory containing the service type's `ServiceManifest.xml` file. This file references the code, static data, and configuration packages for the service type. The files in the service package directory are referenced by the application type's `ApplicationManifest.xml` file. For example, a service package might refer to the code, static data, and configuration packages that make up a database service.

**Named service:** After you create a named application, you can create an instance of one of its service types within the cluster. You specify the service type by using its name/version. Each service type instance is assigned a URI name scoped under its named application's URI. For example, if you create a "MyDatabase" named service within a "MyNamedApp" named application, the URI looks like: `"fabric:/MyNamedApp/MyDatabase"`. Within a named application, you can create several named services. Each named service can have its own partition scheme and instance or replica counts.

**Code package:** A disk directory containing the service type's executable files, typically EXE/DLL files. The files in the code package directory are referenced by the service type's `ServiceManifest.xml` file. When you create a named service, the code package is copied to the node or nodes selected to run the named service. Then the code starts to run. There are two types of code package executables:

- **Guest executables:** Executables that run as-is on the host operating system (Windows or Linux). These executables don't link to or reference any Service Fabric runtime files and therefore don't use any Service Fabric programming models. These executables are unable to use some Service Fabric features, such as the naming service for endpoint discovery. Guest executables can't report load metrics that are specific to each service instance.
- **Service host executables:** Executables that use Service Fabric programming models by linking to Service Fabric runtime files, enabling Service Fabric features. For example, a named service instance can register endpoints with Service Fabric's Naming Service and can also report load metrics.

**Data package:** A disk directory that contains the service type's static, read-only data files, typically photo, sound, and video files. The files in the data package directory are referenced by the service type's `ServiceManifest.xml` file. When you create a named service, the data package is copied to the node or nodes selected to run the named service. The code starts running and can now access the data files.

**Configuration package:** A disk directory that contains the service type's static, read-only configuration files, typically text files. The files in the configuration package directory are referenced by the service type's `ServiceManifest.xml` file. When you create a named service, the files in the configuration package are copied one or more nodes selected to run the named service. Then the code starts to run and can now access the configuration files.

**Containers:** By default, Service Fabric deploys and activates services as processes. Service Fabric can also deploy services in container images. Containers are a virtualization technology that virtualizes the underlying operating system from applications. An application and its runtime, dependencies, and system libraries run inside a container. The container has full, private access to the container's own isolated view of the operating system constructs. Service Fabric supports Docker containers on Linux and Windows Server containers. For more information, read [Service Fabric and containers](#).

**Partition scheme:** When you create a named service, you specify a partition scheme. Services with substantial amounts of state split the data across partitions, which spreads the state across the cluster's nodes. By splitting the data across partitions, your named service's state can scale. Within a partition, stateless named services have instances, whereas stateful named services have replicas. Usually, stateless named services have only one partition, because they have no internal state. The partition instances provide for availability. If one instance fails, other instances continue to operate normally, and then Service Fabric creates a new instance. Stateful named services maintain their state within replicas and each partition has its own replica set so the state is kept in sync. Should a replica fail, Service Fabric builds a new replica from the existing replicas.

Read the [Partition Service Fabric reliable services](#) article for more information.

## System services

There are system services that are created in every cluster that provide the platform capabilities of Service Fabric.

**Naming Service:** Each Service Fabric cluster has a Naming Service, which resolves service names to a location in the cluster. You manage the service names and properties, like an internet Domain Name System (DNS) for the cluster. Clients securely communicate with any node in the cluster by using the Naming Service to resolve a service name and its location. Applications move within the cluster. For example, this can be due to failures, resource balancing, or the resizing of the cluster. You can develop services and clients that resolve the current network location. Clients obtain the actual machine IP address and port where it's currently running.

Read [Communicate with services](#) for more information on the client and service communication APIs that work with the Naming Service.

**Image Store service:** Each Service Fabric cluster has an Image Store service where deployed, versioned application packages are kept. Copy an application package to the Image Store and then register the application type contained within that application package. After the application type is provisioned, you create a named application from it. You can unregister an application type from the Image Store service after all its named applications have been deleted.

Read [Understand the ImageStoreConnectionString setting](#) for more information about the Image Store service.

Read the [Deploy an application](#) article for more information on deploying applications to the Image Store service.

**Failover Manager service:** Each Service Fabric cluster has a Failover Manager service that is responsible for the following actions:

- Performs functions related to high availability and consistency of services.
- Orchestrates application and cluster upgrades.
- Interacts with other system components.

## Built-in programming models

There are .NET Framework programming models available for you to build Service Fabric services:

**Reliable Services:** An API to build stateless and stateful services. Stateful services store their state in Reliable Collections, such as a dictionary or a queue. You can also plug in various communication stacks, such as Web API and Windows Communication Foundation (WCF).

**Reliable Actors:** An API to build stateless and stateful objects through the virtual Actor programming model. This model is useful when you have lots of independent units of computation or state. This model uses a turn-based threading model, so it's best to avoid code that calls out to other actors or services because an individual actor can't process other incoming requests until all its outbound requests are finished.

Read the [Choose a programming model for your service](#) article for more information.

## Next steps

To learn more about Service Fabric:

- [Overview of Service Fabric](#)
- [Why a microservices approach to building applications?](#)
- [Application scenarios](#)

# Service Fabric programming model overview

10/25/2017 • 2 min to read • [Edit Online](#)

Service Fabric offers multiple ways to write and manage your services. Services can choose to use the Service Fabric APIs to take full advantage of the platform's features and application frameworks. Services can also be any compiled executable program written in any language or code running in a container simply hosted on a Service Fabric cluster.

## Guest executables

A [guest executable](#) is an existing, arbitrary executable (written in any language) that can be run as a service in your application. Guest executables do not call the Service Fabric SDK APIs directly. However they still benefit from features the platform offers, such as service discoverability, custom health and load reporting by calling REST APIs exposed by Service Fabric. They also have full application lifecycle support.

Get started with guest executables by deploying your first [guest executable application](#).

## Containers

By default, Service Fabric deploys and activates services as processes. Service Fabric can also deploy services in [containers](#). Service Fabric supports deployment of Linux containers and Windows containers on Windows Server 2016. Container images can be pulled from any container repository and deployed to the machine. You can deploy existing applications as guest executables, Service Fabric stateless or stateful Reliable services or Reliable Actors in containers, and you can mix services in processes and services in containers in the same application.

[Learn more about containerizing your services in Windows or Linux](#)

## Reliable Services

Reliable Services is a light-weight framework for writing services that integrate with the Service Fabric platform and benefit from the full set of platform features. Reliable Services provide a minimal set of APIs that allow the Service Fabric runtime to manage the lifecycle of your services and that allow your services to interact with the runtime. The application framework is minimal, giving you full control over design and implementation choices, and can be used to host any other application framework, such as ASP.NET Core.

Reliable Services can be stateless, similar to most service platforms, such as web servers, in which each instance of the service is created equal and state is persisted in an external solution, such as Azure DB or Azure Table Storage.

Reliable Services can also be stateful, exclusive to Service Fabric, where state is persisted directly in the service itself using Reliable Collections. State is made highly-available through replication and distributed through partitioning, all managed automatically by Service Fabric.

[Learn more about Reliable Services](#) or get started by [writing your first Reliable Service](#).

## Reliable Actors

Built on top of Reliable Services, the Reliable Actor framework is an application framework that implements the Virtual Actor pattern, based on the actor design pattern. The Reliable Actor framework uses independent units of compute and state with single-threaded execution called actors. The Reliable Actor framework provides built-in communication for actors and pre-set state persistence and scale-out configurations.

As Reliable Actors itself is an application framework built on Reliable Services, it is fully integrated with the Service Fabric platform and benefits from the full set of features offered by the platform.

[Learn more about Reliable Actors](#) or get started by [writing your first Reliable Actor service](#)

## ASP.NET Core

Service Fabric integrates with [ASP.NET Core](#) for building Web and API services that can be included as part of your application.

[Build a front end service using ASP.NET Core](#)

## Next steps

[Service Fabric and containers overview](#)

[Reliable Services overview](#)

[Reliable Actors overview](#)

[Service Fabric and ASP.NET Core](#)

# Service Fabric and containers

9/25/2017 • 4 min to read • [Edit Online](#)

## NOTE

Deploying containers to a Service Fabric cluster in Windows 10 isn't supported yet.

## Introduction

Azure Service Fabric is an [orchestrator](#) of services across a cluster of machines, with years of usage and optimization in massive scale services at Microsoft. Services can be developed in many ways, from using the [Service Fabric programming models](#) to deploying [guest executables](#). By default, Service Fabric deploys and activates these services as processes. Processes provide the fastest activation and highest density usage of the resources in a cluster. Service Fabric can also deploy services in container images. Importantly, you can mix services in processes and services in containers in the same application.

## What are containers?

Containers are encapsulated, individually deployable components that run as isolated instances on the same kernel to take advantage of virtualization that an operating system provides. Thus, each application and its runtime, dependencies, and system libraries run inside a container with full, private access to the container's own isolated view of operating system constructs. Along with portability, this degree of security and resource isolation is the main benefit for using containers with Service Fabric, which otherwise runs services in processes.

Containers are a virtualization technology that virtualizes the underlying operating system from applications.

Containers provide an immutable environment for applications to run with varying degrees of isolation.

Containers run directly on top of the kernel and have an isolated view of the file system and other resources.

Compared to virtual machines, containers have the following advantages:

- **Small:** Containers use a single storage space and layer versions and updates to increase efficiency.
- **Fast:** Containers don't have to boot an entire operating system, so they can start much faster, typically in seconds.
- **Portability:** A containerized application image can be ported to run in the cloud, on premises, inside virtual machines, or directly on physical machines.
- **Resource governance:** A container can limit the physical resources that it can consume on its host.

## Container types and supported environments

Service Fabric supports containers on both Linux and Windows, and also supports Hyper-V isolation mode on the latter.

## NOTE

Deploying containers to a Service Fabric cluster on Windows 10 isn't currently supported.

### Docker containers on Linux

Docker provides high-level APIs to create and manage containers on top of Linux kernel containers. Docker Hub is a central repository to store and retrieve container images. For a tutorial, see [Deploy a Docker container to Service Fabric](#).

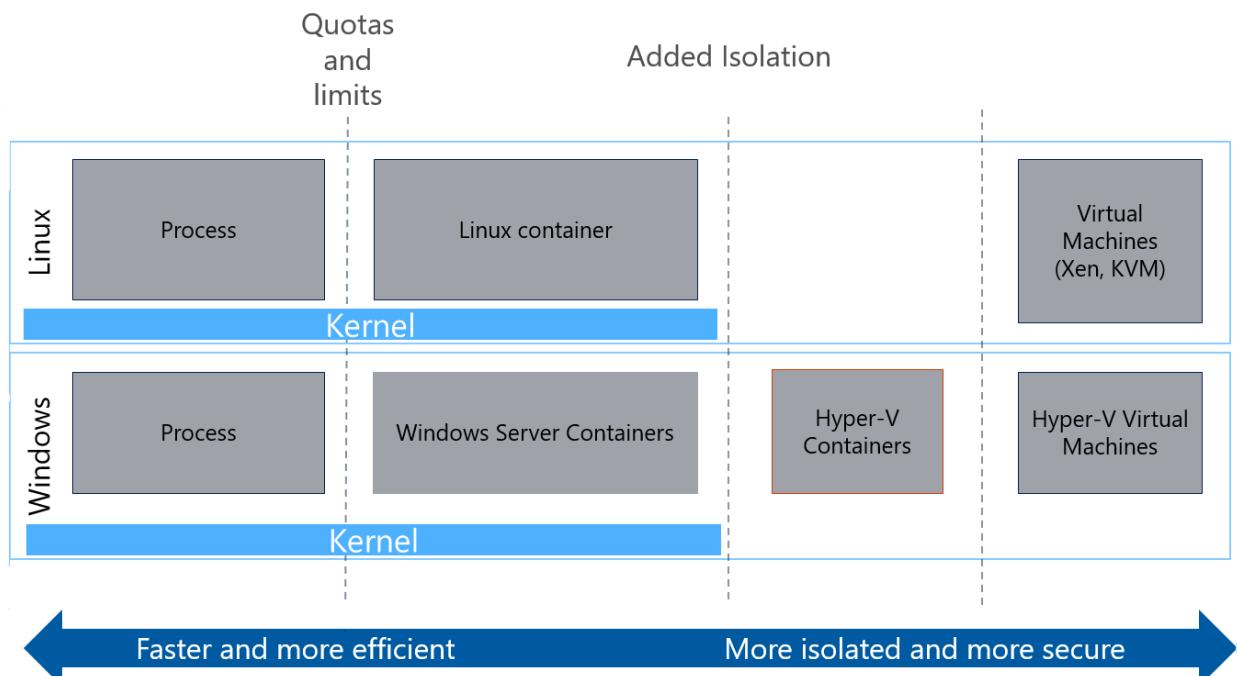
Service Fabric.

## Windows Server containers

Windows Server 2016 provides two different types of containers that differ in the level of provided isolation. Windows Server containers and Docker containers are similar because both have namespace and file system isolation but share the kernel with the host they are running on. On Linux, this isolation has traditionally been provided by `cgroups` and `namespaces`, and Windows Server containers behave similarly.

Windows containers with Hyper-V support provide more isolation and security because each container does not share the operating system kernel with other containers or with the host. With this higher level of security isolation, Hyper-V enabled containers are targeted at hostile, multitenant scenarios. For a tutorial, see [Deploy a Windows container to Service Fabric](#).

The following figure shows the different types of virtualization and isolation levels available in the operating system.



## Scenarios for using containers

Here are typical examples where a container is a good choice:

- **IIS lift and shift:** If you have existing [ASP.NET MVC](#) apps that you want to continue to use, put them in a container instead of migrating them to ASP.NET Core. These ASP.NET MVC apps depend on Internet Information Services (IIS). You can package these applications into container images from the precreated IIS image and deploy them with Service Fabric. See [Container Images on Windows Server](#) for information about Windows containers.
- **Mix containers and Service Fabric microservices:** Use an existing container image for part of your application. For example, you might use the [NGINX container](#) for the web front end of your application and stateful services for the more intensive back-end computation.
- **Reduce impact of "noisy neighbors" services:** You can use the resource governance ability of containers to restrict the resources that a service uses on a host. If services might consume many resources and affect the performance of others (such as a long-running, query-like operation), consider putting these services into containers that have resource governance.

# Service Fabric support for containers

Service Fabric supports the deployment of Docker containers on Linux and Windows Server containers on Windows Server 2016, along with support for Hyper-V isolation mode.

In the Service Fabric [application model](#), a container represents an application host in which multiple service replicas are placed. Service Fabric can run any containers, and the scenario is similar to the [guest executable scenario](#), where you package an existing application inside a container. This scenario is the common use-case for containers, and examples include running an application written using any language or frameworks, but not using the built-in Service Fabric programming models.

In addition, you can run [Service Fabric services inside containers](#) as well. Support for running Service Fabric services inside containers is currently limited, and will be improved in upcoming releases.

Service Fabric has several container capabilities that help you build applications that are composed of microservices that are containerized. Service Fabric offers the following capabilities for containerized services:

- Container image deployment and activation.
- Resource governance including setting resource values by default on Azure clusters.
- Repository authentication.
- Container port to host port mapping.
- Container-to-container discovery and communication.
- Ability to configure and set environment variables.
- Ability to set security credentials on the container.
- A choice of different networking modes for containers.

## Next steps

In this article, you learned about containers, that Service Fabric is a container orchestrator, and that Service Fabric has features that support containers. As a next step, we will go over examples of each of the features to show you how to use them.

[Create your first Service Fabric container application on Windows](#)

[Create your first Service Fabric container application on Linux](#)

[Learn more about Windows Containers](#)

# Docker Compose deployment support in Azure Service Fabric (Preview)

10/3/2017 • 4 min to read • [Edit Online](#)

Docker uses the `docker-compose.yml` file for defining multi-container applications. To make it easy for customers familiar with Docker to orchestrate existing container applications on Azure Service Fabric, we have included preview support for Docker Compose deployment natively in the platform. Service Fabric can accept version 3 and later of `docker-compose.yml` files.

Because this support is in preview, only a subset of Compose directives is supported. For example, application upgrades are not supported. However, you can always remove and deploy applications instead of upgrading them.

To use this preview, create your cluster with version 5.7 or greater of the Service Fabric runtime through the Azure portal along with the corresponding SDK.

## NOTE

This feature is in preview and is not supported in production. The examples below are based on runtime version 6.0 and SDK version 2.8.

## Deploy a Docker Compose file on Service Fabric

The following commands create a Service Fabric application (named `fabric:/TestContainerApp`), which you can monitor and manage like any other Service Fabric application. You can use the specified application name for health queries. Service Fabric recognizes "DeploymentName" as the identifier of the Compose deployment.

### Use PowerShell

Create a Service Fabric Compose deployment from a `docker-compose.yml` file by running the following command in PowerShell:

```
New-ServiceFabricComposeDeployment -DeploymentName TestContainerApp -Compose docker-compose.yml [-RegistryUserName <>] [-RegistryPassword <>] [-PasswordEncrypted]
```

`RegistryUserName` and `RegistryPassword` refer to the container registry username and password. After you've completed the deployment, you can check its status by using the following command:

```
Get-ServiceFabricComposeDeploymentStatus -DeploymentName TestContainerApp
```

To delete the Compose deployment through PowerShell, use the following command:

```
Remove-ServiceFabricComposeDeployment -DeploymentName TestContainerApp
```

To start a Compose deployment upgrade through PowerShell, use the following command:

```
Start-ServiceFabricComposeDeploymentUpgrade -DeploymentName TestContainerApp -Compose docker-compose-v2.yml -Monitored -FailureAction Rollback
```

After upgrade is accepted, the upgrade progress could be tracked using the following command:

```
Get-ServiceFabricComposeDeploymentUpgrade -Deployment TestContainerApp
```

## Use Azure Service Fabric CLI (sfctl)

Alternatively, you can use the following Service Fabric CLI command:

```
sfctl compose create --deployment-name TestContainerApp --file-path docker-compose.yml [ [ --user --encrypted-pass ] | [ --user --has-pass ] ] [ --timeout ]
```

After you've created the deployment, you can check its status by using the following command:

```
sfctl compose status --deployment-name TestContainerApp [ --timeout ]
```

To delete the compose deployment, use the following command:

```
sfctl compose remove --deployment-name TestContainerApp [ --timeout ]
```

To start a Compose deployment upgrade, use the following command:

```
sfctl compose upgrade --deployment-name TestContainerApp --file-path docker-compose-v2.yml [ [ --user --encrypted-pass ] | [ --user --has-pass ] ] [ --upgrade-mode Monitored ] [ --failure-action Rollback ] [ --timeout ]
```

After upgrade is accepted, the upgrade progress could be tracked using the following command:

```
sfctl compose upgrade-status --deployment-name TestContainerApp
```

## Supported Compose directives

This preview supports a subset of the configuration options from the Compose version 3 format, including the following primitives:

- Services > Deploy > Replicas
- Services > Deploy > Placement > Constraints
- Services > Deploy > Resources > Limits
  - -cpu-shares
  - -memory
  - -memory-swap
- Services > Commands
- Services > Environment
- Services > Ports
- Services > Image
- Services > Isolation (only for Windows)
- Services > Logging > Driver
- Services > Logging > Driver > Options
- Volume & Deploy > Volume

Set up the cluster for enforcing resource limits, as described in [Service Fabric resource governance](#). All other Docker Compose directives are unsupported for this preview.

## ServiceDnsName computation

If the service name that you specify in a Compose file is a fully qualified domain name (that is, it contains a dot [.]), the DNS name registered by Service Fabric is `<serviceName>` (including the dot). If not, each path segment in the application name becomes a domain label in the service DNS name, with the first path segment becoming the top-level domain label.

For example, if the specified application name is `fabric:/SampleApp/MyComposeApp`, `<ServiceName>.MyComposeApp.SampleApp` would be the registered DNS name.

## Compose deployment (instance definition) versus Service Fabric app model (type definition)

A docker-compose.yml file describes a deployable set of containers, including their properties and configurations. For example, the file can contain environment variables and ports. You can also specify deployment parameters, such as placement constraints, resource limits, and DNS names, in the docker-compose.yml file.

The [Service Fabric application model](#) uses service types and application types, where you can have many application instances of the same type. For example, you can have one application instance per customer. This type-based model supports multiple versions of the same application type that's registered with the runtime.

For example, customer A can have an application instantiated with type 1.0 of AppTypeA, and customer B can have another application instantiated with the same type and version. You define the application types in the application manifests, and you specify the application name and deployment parameters when you create the application.

Although this model offers flexibility, we are also planning to support a simpler, instance-based deployment model where types are implicit from the manifest file. In this model, each application gets its own independent manifest. We are previewing this effort by adding support for docker-compose.yml, which is an instance-based deployment format.

## Next steps

- Read up on the [Service Fabric application model](#)
- [Get started with Service Fabric CLI](#)

# Resource governance

10/10/2017 • 7 min to read • [Edit Online](#)

When you're running multiple services on the same node or cluster, it's possible that one service might consume more resources, starving other services in the process. This problem is referred to as the "noisy neighbor" problem. Azure Service Fabric enables the developer to specify reservations and limits per service to guarantee resources and limit resource usage.

Before you proceed with this article, we recommend that you get familiar with the [Service Fabric application model](#) and the [Service Fabric hosting model](#).

## Resource governance metrics

Resource governance is supported in Service Fabric in accordance with the [service package](#). The resources that are assigned to the service package can be further divided between code packages. The resource limits that are specified also mean the reservation of the resources. Service Fabric supports specifying CPU and memory per service package, with two built-in [metrics](#):

- *CPU* (metric name `servicefabric:/_CpuCores`): A logical core that's available on the host machine. All cores across all nodes are weighted the same.
- *Memory* (metric name `servicefabric:/_MemoryInMB`): Memory is expressed in megabytes, and it maps to physical memory that is available on the machine.

For these two metrics, [Cluster Resource Manager](#) tracks total cluster capacity, the load on each node in the cluster, and the remaining resources in the cluster. These two metrics are equivalent to any other user or custom metric. All existing features can be used with them:

- The cluster can be [balanced](#) according to these two metrics (default behavior).
- The cluster can be [defragmented](#) according to these two metrics.
- When [describing a cluster](#), buffered capacity can be set for these two metrics.

[Dynamic load reporting](#) is not supported for these metrics, and loads for these metrics are defined at creation time.

## Resource governance mechanism

The Service Fabric runtime currently does not provide reservation for resources. When a process or a container is opened, the runtime sets the resource limits to the loads that were defined at creation time. Furthermore, the runtime rejects the opening of new service packages that are available when resources are exceeded. To better understand how the process works, let's take an example of a node with two CPU cores (mechanism for memory governance is equivalent):

1. First, a container is placed on the node, requesting one CPU core. The runtime opens the container and sets the CPU limit to one core. The container won't be able to use more than one core.
2. Then, a replica of a service is placed on the node, and the corresponding service package specifies a limit of one CPU core. The runtime opens the code package and sets its CPU limit to one core.

At this point, the sum of limits is equal to the capacity of the node. A process and a container are running with one core each and not interfering with each other. Service Fabric doesn't place any more containers or replicas when they are specifying the CPU limit.

However, there are two situations in which other processes might contend for CPU. In these situations, a process and a container from our example might experience the noisy neighbor problem:

- *Mixing governed and non-governed services and containers*: If a user creates a service without any resource governance specified, the runtime sees it as consuming no resources, and can place it on the node in our example. In this case, this new process effectively consumes some CPU at the expense of the services that are already running on the node. There are two solution to this problem. Either don't mix governed and non-governed services on the same cluster, or use [placement constraints](#) so that these two types of services don't end up on the same set of nodes.
- *When another process is started on the node, outside Service Fabric (for example, an OS service)*: In this situation, the process outside Service Fabric also contends for CPU with existing services. The solution to this problem is to set up node capacities correctly to account for OS overhead, as shown in the next section.

## Cluster setup for enabling resource governance

When a node starts and joins the cluster, Service Fabric detects the available amount of memory and the available number of cores, and then sets the node capacities for those two resources.

To leave buffer space for the operating system, and for other processes might be running on the node, Service Fabric uses only 80% of the available resources on the node. This percentage is configurable, and can be changed in the cluster manifest.

Here is an example of how to instruct Service Fabric to use 50% of available CPU and 70% of available memory:

```
<Section Name="PlacementAndLoadBalancing">
    <!-- 0.0 means 0%, and 1.0 means 100%-->
    <Parameter Name="CpuPercentageNodeCapacity" Value="0.5" />
    <Parameter Name="MemoryPercentageNodeCapacity" Value="0.7" />
</Section>
```

If you need full manual setup of node capacities, you can use the regular mechanism for describing the nodes in the cluster. Here is an example of how to set up the node with four cores and 2 GB of memory:

```
<NodeType Name="MyNodeType">
    <Capacities>
        <Capacity Name="servicefabric:/_CpuCores" Value="4"/>
        <Capacity Name="servicefabric:/_MemoryInMB" Value="2048"/>
    </Capacities>
</NodeType>
```

When auto-detection of available resources is enabled, and node capacities are manually defined in the cluster manifest, Service Fabric checks that the node has enough resources to support the capacity that the user has defined:

- If node capacities that are defined in the manifest are less than or equal to the available resources on the node, then Service Fabric uses the capacities that are specified in the manifest.
- If node capacities that are defined in the manifest are greater than available resources, Service Fabric uses the available resources as node capacities.

Auto-detection of available resources can be turned off if it is not required. To turn it off, change the following setting:

```

<Section Name="PlacementAndLoadBalancing">
    <Parameter Name="AutoDetectAvailableResources" Value="false" />
</Section>

```

For optimal performance, the following setting should also be turned on in the cluster manifest:

```

<Section Name="PlacementAndLoadBalancing">
    <Parameter Name="PreventTransientOvercommit" Value="true" />
    <Parameter Name="AllowConstraintCheckFixesDuringApplicationUpgrade" Value="true" />
</Section>

```

## Specify resource governance

Resource governance limits are specified in the application manifest (ServiceManifestImport section) as shown in the following example:

```

<?xml version='1.0' encoding='UTF-8'?>
<ApplicationManifest ApplicationTypeName='TestAppTC1' ApplicationTypeVersion='vTC1'
xsi:schemaLocation='http://schemas.microsoft.com/2011/01/fabric ServiceFabricServiceModel.xsd'
xmlns='http://schemas.microsoft.com/2011/01/fabric' xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'>
    <Parameters>
    </Parameters>
    <!--
        ServicePackageA has the number of CPU cores defined, but doesn't have the MemoryInMB defined.
        In this case, Service Fabric sums the limits on code packages and uses the sum as
        the overall ServicePackage limit.
    -->
    <ServiceManifestImport>
        <ServiceManifestRef ServiceManifestName='ServicePackageA' ServiceManifestVersion='v1'/>
        <Policies>
            <ServicePackageResourceGovernancePolicy CpuCores="1"/>
            <ResourceGovernancePolicy CodePackageRef="CodeA1" CpuShares="512" MemoryInMB="1000" />
            <ResourceGovernancePolicy CodePackageRef="CodeA2" CpuShares="256" MemoryInMB="1000" />
        </Policies>
    </ServiceManifestImport>

```

In this example, the service package called **ServicePackageA** gets one core on the nodes where it is placed. This service package contains two code packages (**CodeA1** and **CodeA2**), and both specify the `CpuShares` parameter. The proportion of CpuShares 512:256 divides the core across the two code packages.

Thus, in this example, CodeA1 gets two-thirds of a core, and CodeA2 gets one-third of a core (and a soft-guarantee reservation of the same). If CpuShares are not specified for code packages, Service Fabric divides the cores equally among them.

Memory limits are absolute, so both code packages are limited to 1024 MB of memory (and a soft-guarantee reservation of the same). Code packages (containers or processes) can't allocate more memory than this limit, and attempting to do so results in an out-of-memory exception. For resource limit enforcement to work, all code packages within a service package should have memory limits specified.

## Other resources for containers

Besides CPU and memory, it's possible to specify other resource limits for containers. These limits are specified at the code-package level and are applied when the container is started. Unlike with CPU and memory, Cluster Resource Manager isn't aware of these resources, and won't do any capacity checks or load balancing for them.

- *MemorySwapInMB*: The amount of swap memory that a container can use.
- *MemoryReservationInMB*: The soft limit for memory governance that is enforced only when memory contention

is detected on the node.

- *CpuPercent*: The percentage of CPU that the container can use. If CPU limits are specified for the service package, this parameter is effectively ignored.
- *MaximumIOPS*: The maximum IOPS that a container can use (read and write).
- *MaximumIOPBytesps*: The maximum IO (bytes per second) that a container can use (read and write).
- *BlockIOWeight*: The block IO weight for relative to other containers.

These resources can be combined with CPU and memory. Here is an example of how to specify additional resources for containers:

```
<ServiceManifestImport>
    <ServiceManifestRef ServiceManifestName="FrontendServicePackage" ServiceManifestVersion="1.0"/>
    <Policies>
        <ResourceGovernancePolicy CodePackageRef="FrontendService.Code" CpuPercent="5"
            MemorySwapInMB="4084" MemoryReservationInMB="1024" MaximumIOPS="20" />
    </Policies>
</ServiceManifestImport>
```

## Next steps

- To learn more about Cluster Resource Manager, read [Introducing the Service Fabric cluster resource manager](#).
- To learn more about the application model, service packages, and code packages--and how replicas map to them--read [Model an application in Service Fabric](#).

# Reliable Services overview

8/15/2017 • 9 min to read • [Edit Online](#)

Azure Service Fabric simplifies writing and managing stateless and stateful Reliable Services. This topic covers:

- The Reliable Services programming model for stateless and stateful services.
- The choices you have to make when writing a Reliable Service.
- Some scenarios and examples of when to use Reliable Services and how they are written.

Reliable Services is one of the programming models available on Service Fabric. The other is the Reliable Actor programming model, which provides a virtual Actor programming model on top of the Reliable Services model. For more information on the Reliable Actors programming model, see [Introduction to Service Fabric Reliable Actors](#).

Service Fabric manages the lifetime of services, from provisioning and deployment through upgrade and deletion, via [Service Fabric application management](#).

## What are Reliable Services?

Reliable Services gives you a simple, powerful, top-level programming model to help you express what is important to your application. With the Reliable Services programming model, you get:

- Access to the rest of the Service Fabric programming APIs. Unlike Service Fabric Services modeled as [Guest Executables](#), Reliable Services get to use the rest of the Service Fabric APIs directly. This allows services to:
  - query the system
  - report health about entities in the cluster
  - receive notifications about configuration and code changes
  - find and communicate with other services,
  - (optionally) use the [Reliable Collections](#)
  - ...and giving them access to many other capabilities, all from a first class programming model in several programming languages.
- A simple model for running your own code that looks like programming models you are used to. Your code has a well-defined entry point and easily managed lifecycle.
- A pluggable communication model. Use the transport of your choice, such as HTTP with [Web API](#), WebSockets, custom TCP protocols, or anything else. Reliable Services provide some great out-of-the-box options you can use, or you can provide your own.
- For stateful services, the Reliable Services programming model allows you to consistently and reliably store your state right inside your service by using [Reliable Collections](#). Reliable Collections are a simple set of highly available and reliable collection classes that will be familiar to anyone who has used C# collections. Traditionally, services needed external systems for Reliable state management. With Reliable Collections, you can store your state next to your compute with the same high availability and reliability you've come to expect from highly available external stores. This model also improves latency because you are co-locating the compute and state it needs to function.

Watch this Microsoft Virtual Academy video for an overview of Reliable services:



## What makes Reliable Services different?

Reliable Services in Service Fabric are different from services you may have written before. Service Fabric provides reliability, availability, consistency, and scalability.

- **Reliability** - Your service stays up even in unreliable environments where your machines fail or hit network issues, or in cases where the services themselves encounter errors and crash or fail. For stateful services, your state is preserved even in the presence of network or other failures.
- **Availability** - Your service is reachable and responsive. Service Fabric maintains your desired number of running copies.
- **Scalability** - Services are decoupled from specific hardware, and they can grow or shrink as necessary through the addition or removal of hardware or other resources. Services are easily partitioned (especially in the stateful case) to ensure that the service can scale and handle partial failures. Services can be created and deleted dynamically via code, enabling more instances to be spun up as necessary, say in response to customer requests. Finally, Service Fabric encourages services to be lightweight. Service Fabric allows thousands of services to be provisioned within a single process, rather than requiring or dedicating entire OS instances or processes to a single instance of a service.
- **Consistency** - Any information stored in this service can be guaranteed to be consistent. This is true even across multiple reliable collections within a service. Changes across collections within a service can be made in a transactionally atomic manner.

## Service lifecycle

Whether your service is stateful or stateless, Reliable Services provide a simple lifecycle that lets you quickly plug in your code and get started. There are just one or two methods that you need to implement to get your service up and running.

- **CreateServiceReplicaListeners/CreateServiceInstanceListeners** - This method is where the service defines the communication stack(s) that it wants to use. The communication stack, such as [Web API](#), is what defines the listening endpoint or endpoints for the service (how clients reach the service). It also defines how the messages that appear interact with the rest of the service code.
- **RunAsync** - This method is where your service runs its business logic, and where it would kick off any background tasks that should run for the lifetime of the service. The cancellation token that is provided is a signal for when that work should stop. For example, if the service needs to pull messages out of a Reliable Queue and process them, this is where that work happens.

If you're learning about reliable services for the first time, read on! If you're looking for a detailed walkthrough of the lifecycle of reliable services, you can head over to [this article](#).

## Example services

Knowing this programming model, let's take a quick look at two different services to see how these pieces fit together.

### Stateless Reliable Services

A stateless service is one where there is no state maintained within the service across calls. Any state that is present is entirely disposable and doesn't require synchronization, replication, persistence, or high availability.

For example, consider a calculator that has no memory and receives all terms and operations to perform at once.

In this case, the `RunAsync()` (C#) or `runAsync()` (Java) of the service can be empty, since there is no background task-processing that the service needs to do. When the calculator service is created, it returns an `ICommunicationListener` (C#) or `CommunicationListener` (Java) (for example [Web API](#)) that opens up a listening endpoint on some port. This listening endpoint hooks up to the different calculation methods (example: "Add(n1, n2)") that define the calculator's public API.

When a call is made from a client, the appropriate method is invoked, and the calculator service performs the operations on the data provided and returns the result. It doesn't store any state.

Not storing any internal state makes this example calculator simple. But most services aren't truly stateless. Instead, they externalize their state to some other store. (For example, any web app that relies on keeping session state in a backing store or cache is not stateless.)

A common example of how stateless services are used in Service Fabric is as a front-end that exposes the public-facing API for a web application. The front-end service then talks to stateful services to complete a user request. In this case, calls from clients are directed to a known port, such as 80, where the stateless service is listening. This stateless service receives the call and determines whether the call is from a trusted party and which service it's destined for. Then, the stateless service forwards the call to the correct partition of the stateful service and waits for a response. When the stateless service receives a response, it replies to the original client. An example of such a service is in our samples [C# / Java](#). This is only one example of this pattern in the samples, there are others in other samples as well.

## Stateful Reliable Services

A stateful service is one that must have some portion of state kept consistent and present in order for the service to function. Consider a service that constantly computes a rolling average of some value based on updates it receives. To do this, it must have the current set of incoming requests it needs to process and the current average. Any service that retrieves, processes, and stores information in an external store (such as an Azure blob or table store today) is stateful. It just keeps its state in the external state store.

Most services today store their state externally, since the external store is what provides reliability, availability, scalability, and consistency for that state. In Service Fabric, services aren't required to store their state externally. Service Fabric takes care of these requirements for both the service code and the service state.

### NOTE

Support for Stateful Reliable Services is not available on Linux yet (for C# or Java).

Let's say we want to write a service that processes images. To do this, the service takes in an image and the series of conversions to perform on that image. This service returns a communication listener (let's suppose it's a WebAPI) that exposes an API like `ConvertImage(Image i, IList<Conversion> conversions)`. When it receives a request, the service stores it in a `IReliableQueue`, and returns some id to the client so it can track the request.

In this service, `RunAsync()` could be more complex. The service has a loop inside its `RunAsync()` that pulls requests out of `IReliableQueue` and performs the conversions requested. The results get stored in an `IReliableDictionary` so that when the client comes back they can get their converted images. To ensure that even if something fails the image isn't lost, this Reliable Service would pull out of the queue, perform the conversions, and store the result all in a single transaction. In this case, the message is removed from the queue and the results are stored in the result dictionary only when the conversions are complete. Alternatively, the service could pull the image out of the queue and immediately store it in a remote store. This reduces the amount of state the service has to manage, but increases complexity since the service has to keep the necessary metadata to manage the remote store. With either

approach, if something failed in the middle the request remains in the queue waiting to be processed.

One thing to note about this service is that it sounds like a normal .NET service! The only difference is that the data structures being used (`IReliableQueue` and `IReliableDictionary`) are provided by Service Fabric, and are highly reliable, available, and consistent.

## When to use Reliable Services APIs

If any of the following characterize your application service needs, then you should consider Reliable Services APIs:

- You want your service's code (and optionally state) to be highly available and reliable
- You need transactional guarantees across multiple units of state (for example, orders and order line items).
- Your application's state can be naturally modeled as Reliable Dictionaries and Queues.
- Your applications code or state needs to be highly available with low latency reads and writes.
- Your application needs to control the concurrency or granularity of transacted operations across one or more Reliable Collections.
- You want to manage the communications or control the partitioning scheme for your service.
- Your code needs a free-threaded runtime environment.
- Your application needs to dynamically create or destroy Reliable Dictionaries or Queues or whole Services at runtime.
- You need to programmatically control Service Fabric-provided backup and restore features for your service's state.
- Your application needs to maintain change history for its units of state.
- You want to develop or consume third-party-developed, custom state providers.

## Next steps

- [Reliable Services quick start](#)
- [Reliable Services advanced usage](#)
- [The Reliable Actors programming model](#)

# Reliable Services lifecycle overview

10/16/2017 • 8 min to read • [Edit Online](#)

When you're thinking about the lifecycles of Azure Service Fabric Reliable Services, the basics of the lifecycle are the most important. In general, the lifecycle includes the following:

- During startup:
  - Services are constructed.
  - The services have an opportunity to construct and return zero or more listeners.
  - Any returned listeners are opened, allowing communication with the service.
  - The service's **RunAsync** method is called, allowing the service to do long-running tasks or background work.
- During shutdown:
  - The cancellation token passed to **RunAsync** is canceled, and the listeners are closed.
  - After the listeners close, the service object itself is destructed.

There are details around the exact ordering of these events. The order of events can change slightly depending on whether the Reliable Service is stateless or stateful. In addition, for stateful services, we must deal with the Primary swap scenario. During this sequence, the role of Primary is transferred to another replica (or comes back) without the service shutting down. Finally, we must think about error or failure conditions.

## Stateless service startup

The lifecycle of a stateless service is straightforward. Here's the order of events:

1. The service is constructed.
2. Then, in parallel, two things happen:
  - `StatelessService.CreateServiceInstanceListeners()` is invoked and any returned listeners are opened.  
`ICommunicationListener.OpenAsync()` is called on each listener.
  - The service's `StatelessService.RunAsync()` method is called.
3. If present, the service's `StatelessService.OnOpenAsync()` method is called. This call is an uncommon override, but it is available.

Keep in mind that there is no ordering between the calls to create and open the listeners and **RunAsync**. The listeners can open before **RunAsync** is started. Similarly, you can invoke **RunAsync** before the communication listeners are open or even constructed. If any synchronization is required, it is left as an exercise to the implementer. Here are some common solutions:

- Sometimes listeners can't function until some other information is created or work is done. For stateless services, that work can usually be done in other locations, such as the following:
  - In the service's constructor.
  - During the `CreateServiceInstanceListeners()` call.
  - As a part of the construction of the listener itself.
- Sometimes the code in **RunAsync** doesn't start until the listeners are open. In this case, additional coordination is necessary. One common solution is that there is a flag within the listeners that indicates when they have finished. This flag is then checked in **RunAsync** before continuing to actual work.

## Stateless service shutdown

For shutting down a stateless service, the same pattern is followed, just in reverse:

1. In parallel:
  - Any open listeners are closed. `ICommunicationListener.CloseAsync()` is called on each listener.
  - The cancellation token passed to `RunAsync()` is canceled. A check of the cancellation token's `IsCancellationRequested` property returns true, and if called, the token's `ThrowIfCancellationRequested` method throws an `OperationCanceledException`.
2. After `CloseAsync()` finishes on each listener and `RunAsync()` also finishes, the service's `StatelessService.OnCloseAsync()` method is called, if present. It is uncommon to override `StatelessService.OnCloseAsync()`.
3. After `StatelessService.OnCloseAsync()` finishes, the service object is destructed.

## Stateful service startup

Stateful services have a similar pattern to stateless services, with a few changes. For starting up a stateful service, the order of events is as follows:

1. The service is constructed.
2. `StatefulServiceBase.OnOpenAsync()` is called. This call is not commonly overridden in the service.
3. The following things happen in parallel:
  - `StatefulServiceBase.CreateServiceReplicaListeners()` is invoked.
    - If the service is a Primary service, all returned listeners are opened. `ICommunicationListener.OpenAsync()` is called on each listener.
    - If the service is a Secondary service, only those listeners marked as `ListenOnSecondary = true` are opened. Having listeners that are open on secondaries is less common.
  - If the service is currently a Primary, the service's `StatefulServiceBase.RunAsync()` method is called.
4. After all the replica listener's `OpenAsync()` calls finish and `RunAsync()` is called, `StatefulServiceBase.OnChangeRoleAsync()` is called. This call is not commonly overridden in the service.

Similar to stateless services, there's no coordination between the order in which the listeners are created and opened and when **RunAsync** is called. If you need coordination, the solutions are much the same. There is one additional case for stateful service. Say that the calls that arrive at the communication listeners require information kept inside some [Reliable Collections](#). Because the communication listeners could open before the reliable collections are readable or writeable, and before **RunAsync** could start, some additional coordination is necessary. The simplest and most common solution is for the communication listeners to return an error code that the client uses to know to retry the request.

## Stateful service shutdown

Like stateless services, the lifecycle events during shutdown are the same as during startup, but reversed. When a stateful service is being shut down, the following events occur:

1. In parallel:
  - Any open listeners are closed. `ICommunicationListener.CloseAsync()` is called on each listener.
  - The cancellation token passed to `RunAsync()` is canceled. A check of the cancellation token's `IsCancellationRequested` property returns true, and if called, the token's `ThrowIfCancellationRequested` method throws an `OperationCanceledException`.
2. After `CloseAsync()` finishes on each listener and `RunAsync()` also finishes, the service's `StatefulServiceBase.OnChangeRoleAsync()` is called. This call is not commonly overridden in the service.

#### NOTE

The need to wait for `RunAsync` to finish is only necessary if this replica is a Primary replica.

3. After the `StatefulServiceBase.OnChangeRoleAsync()` method finishes, the `StatefulServiceBase.OnCloseAsync()` method is called. This call is an uncommon override, but it is available.
4. After `StatefulServiceBase.OnCloseAsync()` finishes, the service object is destructed.

## Stateful service Primary swaps

While a stateful service is running, only the Primary replicas of that stateful services have their communication listeners opened and their `RunAsync` method called. Secondary replicas are constructed, but see no further calls. While a stateful service is running, the replica that's currently the Primary can change. What does this mean in terms of the lifecycle events that a replica can see? The behavior the stateful replica sees depends on whether it is the replica being demoted or promoted during the swap.

### For the Primary that's demoted

For the Primary replica that's demoted, Service Fabric needs this replica to stop processing messages and quit any background work it is doing. As a result, this step looks like it did when the service is shut down. One difference is that the service isn't destructed or closed because it remains as a Secondary. The following APIs are called:

1. In parallel:
  - Any open listeners are closed. `ICommunicationListener.CloseAsync()` is called on each listener.
  - The cancellation token passed to `RunAsync()` is canceled. A check of the cancellation token's `IsCancellationRequested` property returns true, and if called, the token's `ThrowIfCancellationRequested` method throws an `OperationCancelledException`.
2. After `CloseAsync()` finishes on each listener and `RunAsync()` also finishes, the service's `StatefulServiceBase.OnChangeRoleAsync()` is called. This call is not commonly overridden in the service.

### For the Secondary that's promoted

Similarly, Service Fabric needs the Secondary replica that's promoted to start listening for messages on the wire and start any background tasks it needs to complete. As a result, this process looks like it did when the service is created, except that the replica itself already exists. The following APIs are called:

1. In parallel:
  - `StatefulServiceBase.CreateServiceReplicaListeners()` is invoked and any returned listeners are opened. `ICommunicationListener.OpenAsync()` is called on each listener.
  - The service's `StatefulServiceBase.RunAsync()` method is called.
2. After all the replica listener's `OpenAsync()` calls finish and `RunAsync()` is called, `StatefulServiceBase.OnChangeRoleAsync()` is called. This call is not commonly overridden in the service.

### Common issues during stateful service shutdown and Primary demotion

Service Fabric changes the Primary of a stateful service for a variety of reasons. The most common are [cluster rebalancing](#) and [application upgrade](#). During these operations (as well as during normal service shutdown, like you'd see if the service was deleted), it is important that the service respect the `CancellationToken`.

Services that do not handle cancellation cleanly can experience several issues. These operations are slow because Service Fabric waits for the services to stop gracefully. This can ultimately lead to failed upgrades that time out and roll back. Failure to honor the cancellation token can also cause imbalanced clusters. Clusters become unbalanced because nodes get hot, but the services can't be rebalanced because it takes too long to move them elsewhere.

Because the services are stateful, it is also likely that they use the [Reliable Collections](#). In Service Fabric, when a

Primary is demoted, one of the first things that happens is that write access to the underlying state is revoked. This leads to a second set of issues that can affect the service lifecycle. The collections return exceptions based on the timing and whether the replica is being moved or shut down. These exceptions should be handled correctly.

Exceptions thrown by Service Fabric fall into permanent ([FabricException](#)) and transient ([FabricTransientException](#)) categories. Permanent exceptions should be logged and thrown while the transient exceptions can be retried based on some retry logic.

Handling the exceptions that come from use of the `ReliableCollections` in conjunction with service lifecycle events is an important part of testing and validating a Reliable Service. We recommend that you always run your service under load while performing upgrades and [chaos testing](#) before deploying to production. These basic steps help ensure that your service is correctly implemented and handles lifecycle events correctly.

## Notes on the service lifecycle

- Both the `RunAsync()` method and the `CreateServiceReplicaListeners/CreateServiceInstanceListeners` calls are optional. A service can have one of them, both, or neither. For example, if the service does all its work in response to user calls, there is no need for it to implement `RunAsync()`. Only the communication listeners and their associated code are necessary. Similarly, creating and returning communication listeners is optional, as the service can have only background work to do, and so only needs to implement `RunAsync()`.
- It is valid for a service to complete `RunAsync()` successfully and return from it. Completing is not a failure condition. Completing `RunAsync()` indicates that the background work of the service has finished. For stateful reliable services, `RunAsync()` is called again if the replica is demoted from Primary to Secondary and then promoted back to Primary.
- If a service exits from `RunAsync()` by throwing some unexpected exception, this constitutes a failure. The service object is shut down and a health error is reported.
- Although there is no time limit on returning from these methods, you immediately lose the ability to write to Reliable Collections, and therefore, cannot complete any real work. We recommended that you return as quickly as possible upon receiving the cancellation request. If your service does not respond to these API calls in a reasonable amount of time, Service Fabric can forcibly terminate your service. Usually this only happens during application upgrades or when a service is being deleted. This timeout is 15 minutes by default.
- Failures in the `OnCloseAsync()` path result in `OnAbort()` being called, which is a last-chance best-effort opportunity for the service to clean up and release any resources that they have claimed.

## Next steps

- [Introduction to Reliable Services](#)
- [Reliable Services quick start](#)
- [Reliable Services advanced usage](#)
- [Replicas and instances](#)

# Reliable services lifecycle overview

8/18/2017 • 3 min to read • [Edit Online](#)

When thinking about the lifecycles of Reliable Services, the basics of the lifecycle are the most important. In general:

- During Startup
  - Services are constructed
  - They have an opportunity to construct and return zero or more listeners
  - Any returned listeners are opened, allowing communication with the service
  - The Service's runAsync method is called, allowing the service to do long running or background work
- During shutdown
  - The cancellation token passed to runAsync is canceled, and the listeners are closed
  - Once that is complete, the service object itself is destructed

There are details around the exact ordering of these events. In particular, the order of events may change slightly depending on whether the Reliable Service is Stateless or Stateful. In addition, for stateful services, we have to deal with the Primary swap scenario. During this sequence, the role of Primary is transferred to another replica (or comes back) without the service shutting down. Finally, we have to think about error or failure conditions.

## Stateless service startup

The lifecycle of a stateless service is fairly straightforward. Here's the order of events:

1. The Service is constructed
2. Then, in parallel two things happen:
  - `StatelessService.createServiceInstanceListeners()` is invoked and any returned listeners are Opened (`CommunicationListener.openAsync()` is called on each listener)
  - The service's runAsync method (`StatelessService.runAsync()`) is called
3. If present, the service's own onOpenAsync method is called (Specifically, `StatelessService.onOpenAsync()` is called. This is an uncommon override but it is available).

It is important to note that there is no ordering between the calls to create and open the listeners and runAsync. The listeners may open before runAsync is started. Similarly, runAsync may end up invoked before the communication listeners are open or have even been constructed. If any synchronization is required, it is left as an exercise to the implementer. Common solutions:

- Sometimes listeners can't function until some other information is created or work done. For stateless services that work can usually be done in the service's constructor, during the `createServiceInstanceListeners()` call, or as a part of the construction of the listener itself.
- Sometimes the code in runAsync does not want to start until the listeners are open. In this case additional coordination is necessary. One common solution is some flag within the listeners indicating when they have completed, which is checked in runAsync before continuing to actual work.

## Stateless service shutdown

When shutting down a stateless service, the same pattern is followed, just in reverse:

1. In parallel
  - Any open listeners are Closed (`CommunicationListener.closeAsync()` is called on each listener)

- The cancellation token passed to `runAsync()` is canceled (checking the cancellation token's `isCancelled` property returns true, and if called the token's `throwIfCancellationRequested` method throws a `CancellationException`)
- Once `closeAsync()` completes on each listener and `runAsync()` also completes, the service's `StatelessService.onCloseAsync()` method is called, if present (again this is an uncommon override).
  - After `StatelessService.onCloseAsync()` completes, the service object is destructed

## Notes on service lifecycle

- Both the `runAsync()` method and the `createServiceInstanceListeners` calls are optional. A service may have one of them, both, or neither. For example, if the service does all its work in response to user calls, there is no need for it to implement `runAsync()`. Only the communication listeners and their associated code are necessary. Similarly, creating and returning communication listeners is optional, as the service may have only background work to do, and so only needs to implement `runAsync()`
- It is valid for a service to complete `runAsync()` successfully and return from it. This is not considered a failure condition and would represent the background work of the service completing. For stateful reliable services `runAsync()` would be called again if the service were demoted from primary and then promoted back to primary.
- If a service exits from `runAsync()` by throwing some unexpected exception, this is a failure and the service object is shut down and a health error reported.
- While there is no time limit on returning from these methods, you immediately lose the ability to write and therefore cannot complete any real work. It is recommended that you return as quickly as possible upon receiving the cancellation request. If your service does not respond to these API calls in a reasonable amount of time Service Fabric may forcibly terminate your service. Usually this only happens during application upgrades or when a service is being deleted. This timeout is 15 minutes by default.
- Failures in the `onCloseAsync()` path result in `onAbort()` being called which is a last-chance best-effort opportunity for the service to clean up and release any resources that they have claimed.

### NOTE

Stateful reliable services are not supported in java yet.

## Next steps

- [Introduction to Reliable Services](#)
- [Reliable Services quick start](#)
- [Reliable Services advanced usage](#)

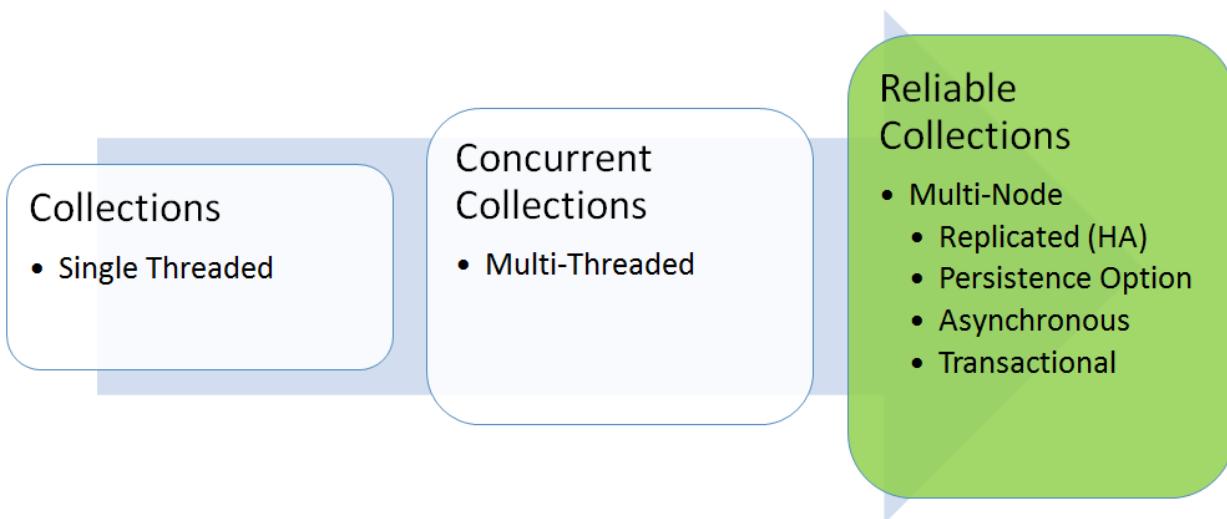
# Introduction to Reliable Collections in Azure Service Fabric stateful services

6/27/2017 • 2 min to read • [Edit Online](#)

Reliable Collections enable you to write highly available, scalable, and low-latency cloud applications as though you were writing single computer applications. The classes in the **Microsoft.ServiceFabric.Data.Collections** namespace provide a set of collections that automatically make your state highly available. Developers need to program only to the Reliable Collection APIs and let Reliable Collections manage the replicated and local state.

The key difference between Reliable Collections and other high-availability technologies (such as Redis, Azure Table service, and Azure Queue service) is that the state is kept locally in the service instance while also being made highly available. This means that:

- All reads are local, which results in low latency and high-throughput reads.
- All writes incur the minimum number of network IOs, which results in low latency and high-throughput writes.



Reliable Collections can be thought of as the natural evolution of the **System.Collections** classes: a new set of collections that are designed for the cloud and multi-computer applications without increasing complexity for the developer. As such, Reliable Collections are:

- Replicated: State changes are replicated for high availability.
- Persisted: Data is persisted to disk for durability against large-scale outages (for example, a datacenter power outage).
- Asynchronous: APIs are asynchronous to ensure that threads are not blocked when incurring IO.
- Transactional: APIs utilize the abstraction of transactions so you can manage multiple Reliable Collections within a service easily.

Reliable Collections provide strong consistency guarantees out of the box to make reasoning about application state easier. Strong consistency is achieved by ensuring transaction commits finish only after the entire transaction has been logged on a majority quorum of replicas, including the primary. To achieve weaker consistency, applications can acknowledge back to the client/requester before the asynchronous commit returns.

The Reliable Collections APIs are an evolution of concurrent collections APIs (found in the **System.Collections.Concurrent** namespace):

- Asynchronous: Returns a task since, unlike concurrent collections, the operations are replicated and persisted.

- No out parameters: Uses `ConditionalValue<T>` to return a bool and a value instead of out parameters.  
`ConditionalValue<T>` is like `Nullable<T>` but does not require T to be a struct.
- Transactions: Uses a transaction object to enable the user to group actions on multiple Reliable Collections in a transaction.

Today, **Microsoft.ServiceFabric.Data.Collections** contains three collections:

- **Reliable Dictionary**: Represents a replicated, transactional, and asynchronous collection of key/value pairs.  
Similar to **ConcurrentDictionary**, both the key and the value can be of any type.
- **Reliable Queue**: Represents a replicated, transactional, and asynchronous strict first-in, first-out (FIFO) queue.  
Similar to **ConcurrentQueue**, the value can be of any type.
- **Reliable Concurrent Queue**: Represents a replicated, transactional, and asynchronous best effort ordering queue for high throughput. Similar to the **ConcurrentQueue**, the value can be of any type.

## Next steps

- [Reliable Collection Guidelines & Recommendations](#)
- [Working with Reliable Collections](#)
- [Transactions and Locks](#)
- [Reliable State Manager and Collection Internals](#)
- Managing Data
  - [Backup and Restore](#)
  - [Notifications](#)
  - [Reliable Collection serialization](#)
  - [Serialization and Upgrade](#)
  - [Reliable State Manager configuration](#)
- Others
  - [Reliable Services quick start](#)
  - [Developer reference for Reliable Collections](#)

# Guidelines and recommendations for Reliable Collections in Azure Service Fabric

6/27/2017 • 2 min to read • [Edit Online](#)

This section provides guidelines for using Reliable State Manager and Reliable Collections. The goal is to help users avoid common pitfalls.

The guidelines are organized as simple recommendations prefixed with the terms *Do*, *Consider*, *Avoid* and *Do not*.

- Do not modify an object of custom type returned by read operations (for example, `TryPeekAsync` or `TryGetValueAsync`). Reliable Collections, just like Concurrent Collections, return a reference to the objects and not a copy.
- Do deep copy the returned object of a custom type before modifying it. Since structs and built-in types are pass-by-value, you do not need to do a deep copy on them.
- Do not use  `TimeSpan.MaxValue`  for time-outs. Time-outs should be used to detect deadlocks.
- Do not use a transaction after it has been committed, aborted, or disposed.
- Do not use an enumeration outside of the transaction scope it was created in.
- Do not create a transaction within another transaction's `using` statement because it can cause deadlocks.
- Do ensure that your `IComparable< TKey >` implementation is correct. The system takes dependency on `IComparable< TKey >` for merging checkpoints and rows.
- Do use Update lock when reading an item with an intention to update it to prevent a certain class of deadlocks.
- Consider keeping your items (for example, `TKey + TValue` for Reliable Dictionary) below 80 KBytes: smaller the better. This reduces the amount of Large Object Heap usage as well as disk and network IO requirements. Often, it reduces replicating duplicate data when only one small part of the value is being updated. Common way to achieve this in Reliable Dictionary, is to break your rows in to multiple rows.
- Consider using backup and restore functionality to have disaster recovery.
- Avoid mixing single entity operations and multi-entity operations (e.g `GetCountAsync` , `CreateEnumerableAsync` ) in the same transaction due to the different isolation levels.
- Do handle `InvalidOperationException`. User transactions can be aborted by the system for variety of reasons. For example, when the Reliable State Manager is changing its role out of Primary or when a long-running transaction is blocking truncation of the transactional log. In such cases, user may receive `InvalidOperationException` indicating that their transaction has already been terminated. Assuming, the termination of the transaction was not requested by the user, best way to handle this exception is to dispose the transaction, check if the cancellation token has been signaled (or the role of the replica has been changed), and if not create a new transaction and retry.

Here are some things to keep in mind:

- The default time-out is four seconds for all the Reliable Collection APIs. Most users should use the default time-out.
- The default cancellation token is `CancellationToken.None` in all Reliable Collections APIs.
- The key type parameter (`TKey`) for a Reliable Dictionary must correctly implement `GetHashCode()` and `Equals()` . Keys must be immutable.
- To achieve high availability for the Reliable Collections, each service should have at least a target and minimum replica set size of 3.
- Read operations on the secondary may read versions that are not quorum committed. This means that a version of data that is read from a single secondary might be false progressed. Reads from Primary are always stable:

can never be false progressed.

## Next steps

- [Working with Reliable Collections](#)
- [Transactions and Locks](#)
- [Reliable State Manager and Collection Internals](#)
- Managing Data
  - [Backup and Restore](#)
  - [Notifications](#)
  - [Serialization and Upgrade](#)
  - [Reliable State Manager configuration](#)
- Others
  - [Reliable Services quick start](#)
  - [Developer reference for Reliable Collections](#)

# Working with Reliable Collections

6/27/2017 • 10 min to read • [Edit Online](#)

Service Fabric offers a stateful programming model available to .NET developers via Reliable Collections. Specifically, Service Fabric provides reliable dictionary and reliable queue classes. When you use these classes, your state is partitioned (for scalability), replicated (for availability), and transacted within a partition (for ACID semantics). Let's look at a typical usage of a reliable dictionary object and see what it's actually doing.

```
///retry:  
  
try {  
    // Create a new Transaction object for this partition  
    using (ITransaction tx = base.StateManager.CreateTransaction()) {  
        // AddAsync takes key's write lock; if >4 secs, TimeoutException  
        // Key & value put in temp dictionary (read your own writes),  
        // serialized, redo/undo record is logged & sent to  
        // secondary replicas  
        await m_dic.AddAsync(tx, key, value, cancellationToken);  
  
        // CommitAsync sends Commit record to log & secondary replicas  
        // After quorum responds, all locks released  
        await tx.CommitAsync();  
    }  
    // If CommitAsync not called, Dispose sends Abort  
    // record to log & all locks released  
}  
catch (TimeoutException) {  
    await Task.Delay(100, cancellationToken); goto retry;  
}
```

All operations on reliable dictionary objects (except for ClearAsync which is not undoable), require an ITransaction object. This object has associated with it any and all changes you're attempting to make to any reliable dictionary and/or reliable queue objects within a single partition. You acquire an ITransaction object by calling the partition's StateManager's CreateTransaction method.

In the code above, the ITransaction object is passed to a reliable dictionary's AddAsync method. Internally, dictionary methods that accept a key take a reader/writer lock associated with the key. If the method modifies the key's value, the method takes a write lock on the key and if the method only reads from the key's value, then a read lock is taken on the key. Since AddAsync modifies the key's value to the new, passed-in value, the key's write lock is taken. So, if 2 (or more) threads attempt to add values with the same key at the same time, one thread will acquire the write lock and the other threads will block. By default, methods block for up to 4 seconds to acquire the lock; after 4 seconds, the methods throw a TimeoutException. Method overloads exist allowing you to pass an explicit timeout value if you'd prefer.

Usually, you write your code to react to a TimeoutException by catching it and retrying the entire operation (as shown in the code above). In my simple code, I'm just calling Task.Delay passing 100 milliseconds each time. But, in reality, you might be better off using some kind of exponential back-off delay instead.

Once the lock is acquired, AddAsync adds the key and value object references to an internal temporary dictionary associated with the ITransaction object. This is done to provide you with read-your-own-writes semantics. That is, after you call AddAsync, a later call to TryGetValueAsync (using the same ITransaction object) will return the value even if you have not yet committed the transaction. Next, AddAsync serializes your key and value objects to byte arrays and appends these byte arrays to a log file on the local node. Finally, AddAsync sends the byte arrays to all

the secondary replicas so they have the same key/value information. Even though the key/value information has been written to a log file, the information is not considered part of the dictionary until the transaction that they are associated with has been committed.

In the code above, the call to CommitAsync commits all of the transaction's operations. Specifically, it appends commit information to the log file on the local node and also sends the commit record to all the secondary replicas. Once a quorum (majority) of the replicas has replied, all data changes are considered permanent and any locks associated with keys that were manipulated via the ITransaction object are released so other threads/transactions can manipulate the same keys and their values.

If CommitAsync is not called (usually due to an exception being thrown), then the ITransaction object gets disposed. When disposing an uncommitted ITransaction object, Service Fabric appends abort information to the local node's log file and nothing needs to be sent to any of the secondary replicas. And then, any locks associated with keys that were manipulated via the transaction are released.

## Common pitfalls and how to avoid them

Now that you understand how the reliable collections work internally, let's take a look at some common misuses of them. See the code below:

```
using (ITransaction tx = StateManager.CreateTransaction()) {
    // AddAsync serializes the name/user, logs the bytes,
    // & sends the bytes to the secondary replicas.
    await m_dic.AddAsync(tx, name, user);

    // The line below updates the property's value in memory only; the
    // new value is NOT serialized, logged, & sent to secondary replicas.
    user.LastLogin = DateTime.UtcNow; // Corruption!

    await tx.CommitAsync();
}
```

When working with a regular .NET dictionary, you can add a key/value to the dictionary and then change the value of a property (such as LastLogin). However, this code will not work correctly with a reliable dictionary. Remember from the earlier discussion, the call to AddAsync serializes the key/value objects to byte arrays and then saves the arrays to a local file and also sends them to the secondary replicas. If you later change a property, this changes the property's value in memory only; it does not impact the local file or the data sent to the replicas. If the process crashes, what's in memory is thrown away. When a new process starts or if another replica becomes primary, then the old property value is what is available.

I cannot stress enough how easy it is to make the kind of mistake shown above. And, you will only learn about the mistake if/when the process goes down. The correct way to write the code is simply to reverse the two lines:

```
using (ITransaction tx = StateManager.CreateTransaction()) {
    user.LastLogin = DateTime.UtcNow; // Do this BEFORE calling AddAsync
    await m_dic.AddAsync(tx, name, user);
    await tx.CommitAsync();
}
```

Here is another example showing a common mistake:

```

using (ITransaction tx = StateManager.CreateTransaction()) {
    // Use the user's name to look up their data
    ConditionalValue<User> user =
        await m_dic.TryGetValueAsync(tx, name);

    // The user exists in the dictionary, update one of their properties.
    if (user.HasValue) {
        // The line below updates the property's value in memory only; the
        // new value is NOT serialized, logged, & sent to secondary replicas.
        user.Value.LastLogin = DateTime.UtcNow; // Corruption!
        await tx.CommitAsync();
    }
}

```

Again, with regular .NET dictionaries, the code above works fine and is a common pattern: the developer uses a key to look up a value. If the value exists, the developer changes a property's value. However, with reliable collections, this code exhibits the same problem as already discussed: **you MUST not modify an object once you have given it to a reliable collection.**

The correct way to update a value in a reliable collection, is to get a reference to the existing value and consider the object referred to by this reference immutable. Then, create a new object which is an exact copy of the original object. Now, you can modify the state of this new object and write the new object into the collection so that it gets serialized to byte arrays, appended to the local file and sent to the replicas. After committing the change(s), the in-memory objects, the local file, and all the replicas have the same exact state. All is good!

The code below shows the correct way to update a value in a reliable collection:

```

using (ITransaction tx = StateManager.CreateTransaction()) {
    // Use the user's name to look up their data
    ConditionalValue<User> currentUser =
        await m_dic.TryGetValueAsync(tx, name);

    // The user exists in the dictionary, update one of their properties.
    if (currentUser.HasValue) {
        // Create new user object with the same state as the current user object.
        // NOTE: This must be a deep copy; not a shallow copy. Specifically, only
        // immutable state can be shared by currentUser & updatedUser object graphs.
        User updatedUser = new User(currentUser);

        // In the new object, modify any properties you desire
        updatedUser.LastLogin = DateTime.UtcNow;

        // Update the key's value to the updateUser info
        await m_dic.SetValue(tx, name, updatedUser);

        await tx.CommitAsync();
    }
}

```

## Define immutable data types to prevent programmer error

Ideally, we'd like the compiler to report errors when you accidentally produce code that mutates state of an object that you are supposed to consider immutable. But, the C# compiler does not have the ability to do this. So, to avoid potential programmer bugs, we highly recommend that you define the types you use with reliable collections to be immutable types. Specifically, this means that you stick to core value types (such as numbers [Int32, UInt64, etc.], DateTime, Guid, TimeSpan, and the like). And, of course, you can also use String. It is best to avoid collection properties as serializing and deserializing them can frequently hurt performance. However, if you want to use

collection properties, we highly recommend the use of .NET's immutable collections library ([System.Collections.Immutable](#)). This library is available for download from <http://nuget.org>. We also recommend sealing your classes and making fields read-only whenever possible.

The UserInfo type below demonstrates how to define an immutable type taking advantage of aforementioned recommendations.

```
[DataContract]
// If you don't seal, you must ensure that any derived classes are also immutable
public sealed class UserInfo {
    private static readonly IEnumerable<ItemId> NoBids = ImmutableList<ItemId>.Empty;

    public UserInfo(String email, IEnumerable<ItemId> itemsBidding = null) {
        Email = email;
        ItemsBidding = (itemsBidding == null) ? NoBids : itemsBidding.ToImmutableList();
    }

    [OnDeserialized]
    private void OnDeserialized(StreamingContext context) {
        // Convert the serialized collection to an immutable collection
        ItemsBidding = ItemsBidding.ToImmutableList();
    }

    [DataMember]
    public readonly String Email;

    // Ideally, this would be a readonly field but it can't be because OnDeserialized
    // has to set it. So instead, the getter is public and the setter is private.
    [DataMember]
    public IEnumerable<ItemId> ItemsBidding { get; private set; }

    // Since each UserInfo object is immutable, we add a new ItemId to the ItemsBidding
    // collection by creating a new immutable UserInfo object with the added ItemId.
    public UserInfo AddItemBidding(ItemId itemId) {
        return new UserInfo(Email, ((ImmutableList<ItemId>)ItemsBidding).Add(itemId));
    }
}
```

The ItemId type is also an immutable type as shown here:

```
[DataContract]
public struct ItemId {

    [DataMember] public readonly String Seller;
    [DataMember] public readonly String ItemName;
    public ItemId(String seller, String itemName) {
        Seller = seller;
        ItemName = itemName;
    }
}
```

## Schema versioning (upgrades)

Internally, Reliable Collections serialize your objects using .NET's DataContractSerializer. The serialized objects are persisted to the primary replica's local disk and are also transmitted to the secondary replicas. As your service matures, it's likely you'll want to change the kind of data (schema) your service requires. You must approach versioning of your data with great care. First and foremost, you must always be able to deserialize old data. Specifically, this means your deserialization code must be infinitely backward compatible: Version 333 of your service code must be able to operate on data placed in a reliable collection by version 1 of your service code 5 years

ago.

Furthermore, service code is upgraded one upgrade domain at a time. So, during an upgrade, you have two different versions of your service code running simultaneously. You must avoid having the new version of your service code use the new schema as old versions of your service code might not be able to handle the new schema. When possible, you should design each version of your service to be forward compatible by 1 version. Specifically, this means that V1 of your service code should be able to simply ignore any schema elements it does not explicitly handle. However, it must be able to save any data it doesn't explicitly know about and simply write it back out when updating a dictionary key or value.

#### **WARNING**

While you can modify the schema of a key, you must ensure that your key's hash code and equals algorithms are stable. If you change how either of these algorithms operate, you will not be able to look up the key within the reliable dictionary ever again.

Alternatively, you can perform what is typically referred to as a 2-phase upgrade. With a 2-phase upgrade, you upgrade your service from V1 to V2: V2 contains the code that knows how to deal with the new schema change but this code doesn't execute. When the V2 code reads V1 data, it operates on it and writes V1 data. Then, after the upgrade is complete across all upgrade domains, you can somehow signal to the running V2 instances that the upgrade is complete. (One way to signal this is to roll out a configuration upgrade; this is what makes this a 2-phase upgrade.) Now, the V2 instances can read V1 data, convert it to V2 data, operate on it, and write it out as V2 data. When other instances read V2 data, they do not need to convert it, they just operate on it, and write out V2 data.

## Next Steps

To learn about creating forward compatible data contracts, see [Forward-Compatible Data Contracts](#).

To learn best practices on versioning data contracts, see [Data Contract Versioning](#).

To learn how to implement version tolerant data contracts, see [Version-Tolerant Serialization Callbacks](#).

To learn how to provide a data structure that can interoperate across multiple versions, see [IExtensibleDataObject](#).

# Transactions and lock modes in Azure Service Fabric Reliable Collections

6/27/2017 • 3 min to read • [Edit Online](#)

## Transaction

A transaction is a sequence of operations performed as a single logical unit of work. A transaction must exhibit the following ACID properties. (see: <https://technet.microsoft.com/en-us/library/ms190612>)

- **Atomicity:** A transaction must be an atomic unit of work. In other words, either all its data modifications are performed, or none of them is performed.
- **Consistency:** When completed, a transaction must leave all data in a consistent state. All internal data structures must be correct at the end of the transaction.
- **Isolation:** Modifications made by concurrent transactions must be isolated from the modifications made by any other concurrent transactions. The isolation level used for an operation within an `ITransaction` is determined by the `IReliableState` performing the operation.
- **Durability:** After a transaction has completed, its effects are permanently in place in the system. The modifications persist even in the event of a system failure.

### Isolation levels

Isolation level defines the degree to which the transaction must be isolated from modifications made by other transactions. There are two isolation levels that are supported in Reliable Collections:

- **Repeatable Read:** Specifies that statements cannot read data that has been modified but not yet committed by other transactions and that no other transactions can modify data that has been read by the current transaction until the current transaction finishes. For more details, see <https://msdn.microsoft.com/library/ms173763.aspx>.
  - **Snapshot:** Specifies that data read by any statement in a transaction is the transactionally consistent version of the data that existed at the start of the transaction. The transaction can recognize only data modifications that were committed before the start of the transaction. Data modifications made by other transactions after the start of the current transaction are not visible to statements executing in the current transaction. The effect is as if the statements in a transaction get a snapshot of the committed data as it existed at the start of the transaction.
- Snapshots are consistent across Reliable Collections. For more details, see <https://msdn.microsoft.com/library/ms173763.aspx>.

Reliable Collections automatically choose the isolation level to use for a given read operation depending on the operation and the role of the replica at the time of transaction's creation. Following is the table that depicts isolation level defaults for Reliable Dictionary and Queue operations.

OPERATION \ ROLE	PRIMARY	SECONDARY
Single Entity Read	Repeatable Read	Snapshot
Enumeration, Count	Snapshot	Snapshot

### NOTE

Common examples for Single Entity Operations are `IReliableDictionary.TryGetValueAsync`, `IReliableQueue.TryPeekAsync`.

Both the Reliable Dictionary and the Reliable Queue support Read Your Writes. In other words, any write within a transaction will be visible to a following read that belongs to the same transaction.

## Locks

In Reliable Collections, all transactions implement rigorous two phase locking: a transaction does not release the locks it has acquired until the transaction terminates with either an abort or a commit.

Reliable Dictionary uses row level locking for all single entity operations. Reliable Queue trades off concurrency for strict transactional FIFO property. Reliable Queue uses operation level locks allowing one transaction with `TryPeekAsync` and/or `TryDequeueAsync` and one transaction with `EnqueueAsync` at a time. Note that to preserve FIFO, if a `TryPeekAsync` or `TryDequeueAsync` ever observes that the Reliable Queue is empty, they will also lock `EnqueueAsync`.

Write operations always take Exclusive locks. For read operations, the locking depends on a couple of factors. Any read operation done using Snapshot isolation is lock free. Any Repeatable Read operation by default takes Shared locks. However, for any read operation that supports Repeatable Read, the user can ask for an Update lock instead of the Shared lock. An Update lock is an asymmetric lock used to prevent a common form of deadlock that occurs when multiple transactions lock resources for potential updates at a later time.

The lock compatibility matrix can be found in the following table:

REQUEST \ GRANTED	NONE	SHARED	UPDATE	EXCLUSIVE
Shared	No conflict	No conflict	Conflict	Conflict
Update	No conflict	No conflict	Conflict	Conflict
Exclusive	No conflict	Conflict	Conflict	Conflict

Time-out argument in the Reliable Collections APIs is used for deadlock detection. For example, two transactions (T1 and T2) are trying to read and update K1. It is possible for them to deadlock, because they both end up having the Shared lock. In this case, one or both of the operations will time out.

This deadlock scenario is a great example of how an Update lock can prevent deadlocks.

## Next steps

- [Working with Reliable Collections](#)
- [Reliable Services notifications](#)
- [Reliable Services backup and restore \(disaster recovery\)](#)
- [Reliable State Manager configuration](#)
- [Developer reference for Reliable Collections](#)

# Introduction to ReliableConcurrentQueue in Azure Service Fabric

6/27/2017 • 8 min to read • [Edit Online](#)

Reliable Concurrent Queue is an asynchronous, transactional, and replicated queue which features high concurrency for enqueue and dequeue operations. It is designed to deliver high throughput and low latency by relaxing the strict FIFO ordering provided by [Reliable Queue](#) and instead provides a best-effort ordering.

## APIs

CONCURRENT QUEUE	RELIABLE CONCURRENT QUEUE
void Enqueue(T item)	Task EnqueueAsync(ITransaction tx, T item)
bool TryDequeue(out T result)	Task< ConditionalValue < T > > TryDequeueAsync(ITransaction tx)
int Count()	long Count()

## Comparison with Reliable Queue

Reliable Concurrent Queue is offered as an alternative to [Reliable Queue](#). It should be used in cases where strict FIFO ordering is not required, as guaranteeing FIFO requires a tradeoff with concurrency. [Reliable Queue](#) uses locks to enforce FIFO ordering, with at most one transaction allowed to enqueue and at most one transaction allowed to dequeue at a time. In comparison, Reliable Concurrent Queue relaxes the ordering constraint and allows any number concurrent transactions to interleave their enqueue and dequeue operations. Best-effort ordering is provided, however the relative ordering of two values in a Reliable Concurrent Queue can never be guaranteed.

Reliable Concurrent Queue provides higher throughput and lower latency than [Reliable Queue](#) whenever there are multiple concurrent transactions performing enqueues and/or dequeues.

A sample use case for the ReliableConcurrentQueue is the [Message Queue](#) scenario. In this scenario, one or more message producers create and add items to the queue, and one or more message consumers pull messages from the queue and process them. Multiple producers and consumers can work independently, using concurrent transactions in order to process the queue.

## Usage Guidelines

- The queue expects that the items in the queue have a low retention period. That is, the items would not stay in the queue for a long time.
- The queue does not guarantee strict FIFO ordering.
- The queue does not read its own writes. If an item is enqueued within a transaction, it will not be visible to a dequeuer within the same transaction.
- Dequeues are not isolated from each other. If item A is dequeued in transaction  $txnA$ , even though  $txnA$  is not committed, item A would not be visible to a concurrent transaction  $txnB$ . If  $txnA$  aborts, A will become visible to  $txnB$  immediately.
- *TryPeekAsync* behavior can be implemented by using a *TryDequeueAsync* and then aborting the transaction. An example of this can be found in the Programming Patterns section.

- Count is non-transactional. It can be used to get an idea of the number of elements in the queue, but represents a point-in-time and cannot be relied upon.
- Expensive processing on the dequeued items should not be performed while the transaction is active, to avoid long-running transactions which may have a performance impact on the system.

## Code Snippets

Let us look at a few code snippets and their expected outputs. Exception handling is ignored in this section.

### EnqueueAsync

Here are a few code snippets for using EnqueueAsync followed by their expected outputs.

- *Case 1: Single Enqueue Task*

```
using (var txn = this.StateManager.CreateTransaction())
{
    await this.Queue.EnqueueAsync(txn, 10, cancellationToken);
    await this.Queue.EnqueueAsync(txn, 20, cancellationToken);

    await txn.CommitAsync();
}
```

Assume that the task completed successfully, and that there were no concurrent transactions modifying the queue. The user can expect the queue to contain the items in any of the following orders:

10, 20  
20, 10

- *Case 2: Parallel Enqueue Task*

```
// Parallel Task 1
using (var txn = this.StateManager.CreateTransaction())
{
    await this.Queue.EnqueueAsync(txn, 10, cancellationToken);
    await this.Queue.EnqueueAsync(txn, 20, cancellationToken);

    await txn.CommitAsync();
}

// Parallel Task 2
using (var txn = this.StateManager.CreateTransaction())
{
    await this.Queue.EnqueueAsync(txn, 30, cancellationToken);
    await this.Queue.EnqueueAsync(txn, 40, cancellationToken);

    await txn.CommitAsync();
}
```

Assume that the tasks completed successfully, that the tasks ran in parallel, and that there were no other concurrent transactions modifying the queue. No inference can be made about the order of items in the queue. For this code snippet, the items may appear in any of the  $4!$  possible orderings. The queue will attempt to keep the items in the original (enqueued) order, but may be forced to reorder them due to concurrent operations or faults.

### DequeueAsync

Here are a few code snippets for using TryDequeueAsync followed by the expected outputs. Assume that the queue is already populated with the following items in the queue:

- Case 1: Single Dequeue Task

```
using (var txn = this.StateManager.CreateTransaction())
{
    await this.Queue.TryDequeueAsync(txn, cancellationToken);
    await this.Queue.TryDequeueAsync(txn, cancellationToken);
    await this.Queue.TryDequeueAsync(txn, cancellationToken);

    await txn.CommitAsync();
}
```

Assume that the task completed successfully, and that there were no concurrent transactions modifying the queue. Since no inference can be made about the order of the items in the queue, any three of the items may be dequeued, in any order. The queue will attempt to keep the items in the original (enqueued) order, but may be forced to reorder them due to concurrent operations or faults.

- Case 2: Parallel Dequeue Task

```
// Parallel Task 1
List<int> dequeue1;
using (var txn = this.StateManager.CreateTransaction())
{
    dequeue1.Add(await this.Queue.TryDequeueAsync(txn, cancellationToken)).val;
    dequeue1.Add(await this.Queue.TryDequeueAsync(txn, cancellationToken)).val;

    await txn.CommitAsync();
}

// Parallel Task 2
List<int> dequeue2;
using (var txn = this.StateManager.CreateTransaction())
{
    dequeue2.Add(await this.Queue.TryDequeueAsync(txn, cancellationToken)).val;
    dequeue2.Add(await this.Queue.TryDequeueAsync(txn, cancellationToken)).val;

    await txn.CommitAsync();
}
```

Assume that the tasks completed successfully, that the tasks ran in parallel, and that there were no other concurrent transactions modifying the queue. Since no inference can be made about the order of the items in the queue, the lists `dequeue1` and `dequeue2` will each contain any two items, in any order.

The same item will *not* appear in both lists. Hence, if `dequeue1` has 10, 30, then `dequeue2` would have 20, 40.

- Case 3: Dequeue Ordering With Transaction Abort

Aborting a transaction with in-flight dequeues puts the items back on the head of the queue. The order in which the items are put back on the head of the queue is not guaranteed. Let us look at the following code:

```
using (var txn = this.StateManager.CreateTransaction())
{
    await this.Queue.TryDequeueAsync(txn, cancellationToken);
    await this.Queue.TryDequeueAsync(txn, cancellationToken);

    // Abort the transaction
    await txn.AbortAsync();
}
```

Assume that the items were dequeued in the following order:

```
10, 20
```

When we abort the transaction, the items would be added back to the head of the queue in any of the following orders:

```
10, 20
```

```
20, 10
```

The same is true for all cases where the transaction was not successfully *Committed*.

## Programming Patterns

In this section, let us look at a few programming patterns that might be helpful in using ReliableConcurrentQueue.

### Batch Dequeues

A recommended programming pattern is for the consumer task to batch its dequeues instead of performing one dequeue at a time. The user can choose to throttle delays between every batch or the batch size. The following code snippet shows this programming model. Note that in this example, the processing is done after the transaction is committed, so if a fault were to occur while processing, the unprocessed items will be lost without having been processed. Alternatively, the processing can be done within the transaction's scope, however this may have a negative impact on performance and requires handling of the items already processed.

```

int batchSize = 5;
long delayMs = 100;

while(!cancellationToken.IsCancellationRequested)
{
    // Buffer for dequeued items
    List<int> processItems = new List<int>();

    using (var txn = this.StateManager.CreateTransaction())
    {
        ConditionalValue<int> ret;

        for(int i = 0; i < batchSize; ++i)
        {
            ret = await this.Queue.TryDequeueAsync(txn, cancellationToken);

            if (ret.HasValue)
            {
                // If an item was dequeued, add to the buffer for processing
                processItems.Add(ret.Value);
            }
            else
            {
                // else break the for loop
                break;
            }
        }

        await txn.CommitAsync();
    }

    // Process the dequeues
    for (int i = 0; i < processItems.Count; ++i)
    {
        Console.WriteLine("Value : " + processItems[i]);
    }

    int delayFactor = batchSize - processItems.Count;
    await Task.Delay(TimeSpan.FromMilliseconds(delayMs * delayFactor), cancellationToken);
}

```

## Best-Effort Notification-Based Processing

Another interesting programming pattern uses the Count API. Here, we can implement best-effort notification-based processing for the queue. The queue Count can be used to throttle an enqueue or a dequeue task. Note that as in the previous example, since the processing occurs outside the transaction, unprocessed items may be lost if a fault occurs during processing.

```

int threshold = 5;
long delayMs = 1000;

while(!cancellationToken.IsCancellationRequested)
{
    while (this.Queue.Count < threshold)
    {
        cancellationToken.ThrowIfCancellationRequested();

        // If the queue does not have the threshold number of items, delay the task and check again
        await Task.Delay(TimeSpan.FromMilliseconds(delayMs), cancellationToken);
    }

    // If there are approximately threshold number of items, try and process the queue

    // Buffer for dequeued items
    List<int> processItems = new List<int>();

    using (var txn = this.StateManager.CreateTransaction())
    {
        ConditionalValue<int> ret;

        do
        {
            ret = await this.Queue.TryDequeueAsync(txn, cancellationToken);

            if (ret.HasValue)
            {
                // If an item was dequeued, add to the buffer for processing
                processItems.Add(ret.Value);
            }
        } while (processItems.Count < threshold && ret.HasValue);

        await txn.CommitAsync();
    }

    // Process the dequeues
    for (int i = 0; i < processItems.Count; ++i)
    {
        Console.WriteLine("Value : " + processItems[i]);
    }
}

```

## Best-Effort Drain

A drain of the queue cannot be guaranteed due to the concurrent nature of the data structure. It is possible that, even if no user operations on the queue are in-flight, a particular call to TryDequeueAsync may not return an item which was previously enqueued and committed. The enqueued item is guaranteed to *eventually* become visible to dequeue, however without an out-of-band communication mechanism, an independent consumer cannot know that the queue has reached a steady-state even if all producers have been stopped and no new enqueue operations are allowed. Thus, the drain operation is best-effort as implemented below.

The user should stop all further producer and consumer tasks, and wait for any in-flight transactions to commit or abort, before attempting to drain the queue. If the user knows the expected number of items in the queue, they can set up a notification which signals that all items have been dequeued.

```

int numItemsDequeued;
int batchSize = 5;

ConditionalValue ret;

do
{
    List<int> processItems = new List<int>();

    using (var txn = this.StateManager.CreateTransaction())
    {
        do
        {
            ret = await this.Queue.TryDequeueAsync(txn, cancellationToken);

            if(ret.HasValue)
            {
                // Buffer the dequeues
                processItems.Add(ret.Value);
            }
        } while (ret.HasValue && processItems.Count < batchSize);

        await txn.CommitAsync();
    }

    // Process the dequeues
    for (int i = 0; i < processItems.Count; ++i)
    {
        Console.WriteLine("Value : " + processItems[i]);
    }
} while (ret.HasValue);

```

## Peek

ReliableConcurrentQueue does not provide the *TryPeekAsync* api. Users can get the peek semantic by using a *TryDequeueAsync* and then aborting the transaction. In this example, dequeues are processed only if the item's value is greater than 10.

```

using (var txn = this.StateManager.CreateTransaction())
{
    ConditionalValue ret = await this.Queue.TryDequeueAsync(txn, cancellationToken);
    bool valueProcessed = false;

    if (ret.HasValue)
    {
        if (ret.Value > 10)
        {
            // Process the item
            Console.WriteLine("Value : " + ret.Value);
            valueProcessed = true;
        }
    }

    if (valueProcessed)
    {
        await txn.CommitAsync();
    }
    else
    {
        await txn.AbortAsync();
    }
}

```

## Must Read

- [Reliable Services Quick Start](#)
- [Working with Reliable Collections](#)
- [Reliable Services notifications](#)
- [Reliable Services Backup and Restore \(Disaster Recovery\)](#)
- [Reliable State Manager Configuration](#)
- [Getting Started with Service Fabric Web API Services](#)
- [Advanced Usage of the Reliable Services Programming Model](#)
- [Developer Reference for Reliable Collections](#)

# Reliable Collection object serialization in Azure Service Fabric

6/27/2017 • 4 min to read • [Edit Online](#)

Reliable Collections' replicate and persist their items to make sure they are durable across machine failures and power outages. Both to replicate and to persist items, Reliable Collections' need to serialize them.

Reliable Collections' get the appropriate serializer for a given type from Reliable State Manager. Reliable State Manager contains built-in serializers and allows custom serializers to be registered for a given type.

## Built-in Serializers

Reliable State Manager includes built-in serializer for some common types, so that they can be serialized efficiently by default. For other types, Reliable State Manager falls back to use the [DataContractSerializer](#). Built-in serializers are more efficient since they know their types cannot change and they do not need to include information about the type like its type name.

Reliable State Manager has built-in serializer for following types:

- Guid
- bool
- byte
- sbyte
- byte[]
- char
- string
- decimal
- double
- float
- int
- uint
- long
- ulong
- short
- ushort

## Custom Serialization

Custom serializers are commonly used to increase performance or to encrypt the data over the wire and on disk. Among other reasons, custom serializers are commonly more efficient than generic serializer since they don't need to serialize information about the type.

[IReliableStateManager.TryAddStateSerializer](#) is used to register a custom serializer for the given type T. This registration should happen in the construction of the StatefulServiceBase to ensure that before recovery starts, all Reliable Collections have access to the relevant serializer to read their persisted data.

```
public StatefulBackendService(StatefulServiceContext context)
    : base(context)
{
    if (!this.StateManager.TryAddStateSerializer(new OrderKeySerializer()))
    {
        throw new InvalidOperationException("Failed to set OrderKey custom serializer");
    }
}
```

#### NOTE

Custom serializers are given precedence over built-in serializers. For example, when a custom serializer for int is registered, it is used to serialize integers instead of the built-in serializer for int.

### How to implement a custom serializer

A custom serializer needs to implement the [IStateSerializer](#) interface.

#### NOTE

[IStateSerializer](#) includes an overload for Write and Read that takes in an additional T called base value. This API is for differential serialization. Currently differential serialization feature is not exposed. Hence, these two overloads are not called until differential serialization is exposed and enabled.

Following is an example custom type called OrderKey that contains four properties

```
public class OrderKey : IComparable<OrderKey>, IEquatable<OrderKey>
{
    public byte Warehouse { get; set; }

    public short District { get; set; }

    public int Customer { get; set; }

    public long Order { get; set; }

    #region Object Overrides for GetHashCode, CompareTo and Equals
    #endregion
}
```

Following is an example implementation of [IStateSerializer](#). Note that Read and Write overloads that take in `baseValue`, call their respective overload for forwards compatibility.

```

public class OrderKeySerializer : IStateSerializer<OrderKey>
{
    OrderKey IStateSerializer<OrderKey>.Read(BinaryReader reader)
    {
        var value = new OrderKey();
        value.Warehouse = reader.ReadByte();
        value.District = reader.ReadInt16();
        value.Customer = reader.ReadInt32();
        value.Order = reader.ReadInt64();

        return value;
    }

    void IStateSerializer<OrderKey>.Write(OrderKey value, BinaryWriter writer)
    {
        writer.Write(value.Warehouse);
        writer.Write(value.District);
        writer.Write(value.Customer);
        writer.Write(value.Order);
    }

    // Read overload for differential de-serialization
    OrderKey IStateSerializer<OrderKey>.Read(OrderKey baseValue, BinaryReader reader)
    {
        return ((IStateSerializer<OrderKey>)this).Read(reader);
    }

    // Write overload for differential serialization
    void IStateSerializer<OrderKey>.Write(OrderKey baseValue, OrderKey newValue, BinaryWriter writer)
    {
        ((IStateSerializer<OrderKey>)this).Write(newValue, writer);
    }
}

```

## Upgradability

In a [rolling application upgrade](#), the upgrade is applied to a subset of nodes, one upgrade domain at a time. During this process, some upgrade domains will be on the newer version of your application, and some upgrade domains will be on the older version of your application. During the rollout, the new version of your application must be able to read the old version of your data, and the old version of your application must be able to read the new version of your data. If the data format is not forward and backward compatible, the upgrade may fail, or worse, data may be lost or corrupted.

If you are using built-in serializer, you do not have to worry about compatibility. However, if you are using a custom serializer or the `DataContractSerializer`, the data have to be infinitely backwards and forwards compatible. In other words, each version of serializer needs to be able to serialize and de-serialize any version of the type.

Data Contract users should follow the well-defined versioning rules for adding, removing, and changing fields. Data Contract also has support for dealing with unknown fields, hooking into the serialization and deserialization process, and dealing with class inheritance. For more information, see [Using Data Contract](#).

Custom serializer users should adhere to the guidelines of the serializer they are using to make sure it is backwards and forwards compatible. Common way of supporting all versions is adding size information at the beginning and only adding optional properties. This way each version can read as much it can and jump over the remaining part of the stream.

## Next steps

- [Serialization and upgrade](#)
- [Developer reference for Reliable Collections](#)

- [Upgrading your Application Using Visual Studio](#) walks you through an application upgrade using Visual Studio.
- [Upgrading your Application Using Powershell](#) walks you through an application upgrade using PowerShell.
- Control how your application upgrades by using [Upgrade Parameters](#).
- Learn how to use advanced functionality while upgrading your application by referring to [Advanced Topics](#).
- Fix common problems in application upgrades by referring to the steps in [Troubleshooting Application Upgrades](#).

# Azure Service Fabric Reliable State Manager and Reliable Collection internals

6/27/2017 • 1 min to read • [Edit Online](#)

This document delves inside Reliable State Manager and Reliable Collections to see how core components work behind the scenes.

## NOTE

This document is work in-progress. Add comments to this article to tell us what topic you would like to learn more about.

## Local persistence model: log and checkpoint

The Reliable State Manager and Reliable Collections follow a persistence model that is called Log and Checkpoint. In this model, each state change is logged on disk first and then applied in memory. The complete state itself is persisted only occasionally (a.k.a. Checkpoint). The benefit is that deltas are turned into sequential append-only writes on disk for improved performance.

To better understand the Log and Checkpoint model, let's first look at the infinite disk scenario. The Reliable State Manager logs every operation before it is replicated. Logging allows the Reliable Collections to apply the operation only in memory. Since logs are persisted, even when the replica fails and needs to be restarted, the Reliable State Manager has enough information in its log to replay all the operations the replica has lost. As the disk is infinite, log records never need to be removed and the Reliable Collection needs to manage only the in-memory state.

Now let's look at the finite disk scenario. As log records accumulate, the Reliable State Manager will run out of disk space. Before that happens, the Reliable State Manager needs to truncate its log to make room for the newer records. Reliable State Manager requests the Reliable Collections to checkpoint their in-memory state to disk. At this point, the Reliable Collections' would persist its in-memory state. Once the Reliable Collections complete their checkpoints, the Reliable State Manager can truncate the log to free up disk space. When the replica needs to be restarted, Reliable Collections recover their checkpointed state, and the Reliable State Manager recovers and plays back all the state changes that occurred since the last checkpoint.

Another value add of checkpointing is that it improves recovery times in common scenarios. Log contains all operations that have happened since the last checkpoint. So it may include multiple versions of an item like multiple values for a given row in Reliable Dictionary. In contrast, a Reliable Collection checkpoints only the latest version of each value for a key.

## Next steps

- [Transactions and Locks](#)

# Advanced usage of the Reliable Services programming model

6/30/2017 • 3 min to read • [Edit Online](#)

Azure Service Fabric simplifies writing and managing reliable stateless and stateful services. This guide talks about advanced usages of Reliable Services to gain more control and flexibility over your services. Prior to reading this guide, familiarize yourself with [the Reliable Services programming model](#).

Both stateful and stateless services have two primary entry points for user code:

- `RunAsync(C#) / runAsync(Java)` is a general-purpose entry point for your service code.
- `CreateServiceReplicaListeners(C#)` and `CreateServiceInstanceListeners(C#) / createServiceInstanceListeners(Java)` is for opening communication listeners for client requests.

For most services, these two entry points are sufficient. In rare cases when more control over a service's lifecycle is required, additional lifecycle events are available.

## Stateless service instance lifecycle

A stateless service's lifecycle is very simple. A stateless service can only be opened, closed, or aborted. `RunAsync` in a stateless service is executed when a service instance is opened, and canceled when a service instance is closed or aborted.

Although `RunAsync` should be sufficient in almost all cases, the open, close, and abort events in a stateless service are also available:

- `Task OnOpenAsync(IStatelessServicePartition, CancellationToken) - C# / CompletableFuture<String>`  
`onOpenAsync(CancellationToken) - Java`  
OnOpenAsync is called when the stateless service instance is about to be used. Extended service initialization tasks can be started at this time.
- `Task OnCloseAsync(CancellationToken) - C# / CompletableFuture onCloseAsync(CancellationToken) - Java`  
OnCloseAsync is called when the stateless service instance is going to be gracefully shut down. This can occur when the service's code is being upgraded, the service instance is being moved due to load balancing, or a transient fault is detected. OnCloseAsync can be used to safely close any resources, stop any background processing, finish saving external state, or close down existing connections.
- `void OnAbort() - C# / void onAbort() - Java` OnAbort is called when the stateless service instance is being forcefully shut down. This is generally called when a permanent fault is detected on the node, or when Service Fabric cannot reliably manage the service instance's lifecycle due to internal failures.

## Stateful service replica lifecycle

### NOTE

Stateful reliable services are not supported in Java yet.

A stateful service replica's lifecycle is much more complex than a stateless service instance. In addition to open, close, and abort events, a stateful service replica undergoes role changes during its lifetime. When a stateful service replica changes role, the `OnChangeRoleAsync` event is triggered:

- Task `OnChangeRoleAsync(ReplicaRole, CancellationToken)` `OnChangeRoleAsync` is called when the stateful service replica is changing role, for example to primary or secondary. Primary replicas are given write status (are allowed to create and write to Reliable Collections). Secondary replicas are given read status (can only read from existing Reliable Collections). Most work in a stateful service is performed at the primary replica. Secondary replicas can perform read-only validation, report generation, data mining, or other read-only jobs.

In a stateful service, only the primary replica has write access to state and thus is generally when the service is performing actual work. The `RunAsync` method in a stateful service is executed only when the stateful service replica is primary. The `RunAsync` method is canceled when a primary replica's role changes away from primary, as well as during the close and abort events.

Using the `OnChangeRoleAsync` event allows you to perform work depending on replica role as well as in response to role change.

A stateful service also provides the same four lifecycle events as a stateless service, with the same semantics and use cases:

```
* Task OnOpenAsync(IStatefulServicePartition, CancellationToken)
* Task OnCloseAsync(CancellationToken)
* void OnAbort()
```

## Next steps

For more advanced topics related to Service Fabric, see the following articles:

- [Configuring stateful Reliable Services](#)
- [Service Fabric health introduction](#)
- [Using system health reports for troubleshooting](#)
- [Configuring Services with the Service Fabric Cluster Resource Manager](#)

# Introduction to Service Fabric Reliable Actors

6/30/2017 • 11 min to read • [Edit Online](#)

Reliable Actors is a Service Fabric application framework based on the [Virtual Actor](#) pattern. The Reliable Actors API provides a single-threaded programming model built on the scalability and reliability guarantees provided by Service Fabric.

## What are Actors?

An actor is an isolated, independent unit of compute and state with single-threaded execution. The [actor pattern](#) is a computational model for concurrent or distributed systems in which a large number of these actors can execute simultaneously and independently of each other. Actors can communicate with each other and they can create more actors.

### When to use Reliable Actors

Service Fabric Reliable Actors is an implementation of the actor design pattern. As with any software design pattern, the decision whether to use a specific pattern is made based on whether or not a software design problem fits the pattern.

Although the actor design pattern can be a good fit to a number of distributed systems problems and scenarios, careful consideration of the constraints of the pattern and the framework implementing it must be made. As general guidance, consider the actor pattern to model your problem or scenario if:

- Your problem space involves a large number (thousands or more) of small, independent, and isolated units of state and logic.
- You want to work with single-threaded objects that do not require significant interaction from external components, including querying state across a set of actors.
- Your actor instances won't block callers with unpredictable delays by issuing I/O operations.

## Actors in Service Fabric

In Service Fabric, actors are implemented in the Reliable Actors framework: An actor-pattern-based application framework built on top of [Service Fabric Reliable Services](#). Each Reliable Actor service you write is actually a partitioned, stateful Reliable Service.

Every actor is defined as an instance of an actor type, identical to the way a .NET object is an instance of a .NET type. For example, there may be an actor type that implements the functionality of a calculator and there could be many actors of that type that are distributed on various nodes across a cluster. Each such actor is uniquely identified by an actor ID.

### Actor Lifetime

Service Fabric actors are virtual, meaning that their lifetime is not tied to their in-memory representation. As a result, they do not need to be explicitly created or destroyed. The Reliable Actors runtime automatically activates an actor the first time it receives a request for that actor ID. If an actor is not used for a period of time, the Reliable Actors runtime garbage-collects the in-memory object. It will also maintain knowledge of the actor's existence should it need to be reactivated later. For more details, see [Actor lifecycle and garbage collection](#).

This virtual actor lifetime abstraction carries some caveats as a result of the virtual actor model, and in fact the Reliable Actors implementation deviates at times from this model.

- An actor is automatically activated (causing an actor object to be constructed) the first time a message is sent

to its actor ID. After some period of time, the actor object is garbage collected. In the future, using the actor ID again, causes a new actor object to be constructed. An actor's state outlives the object's lifetime when stored in the state manager.

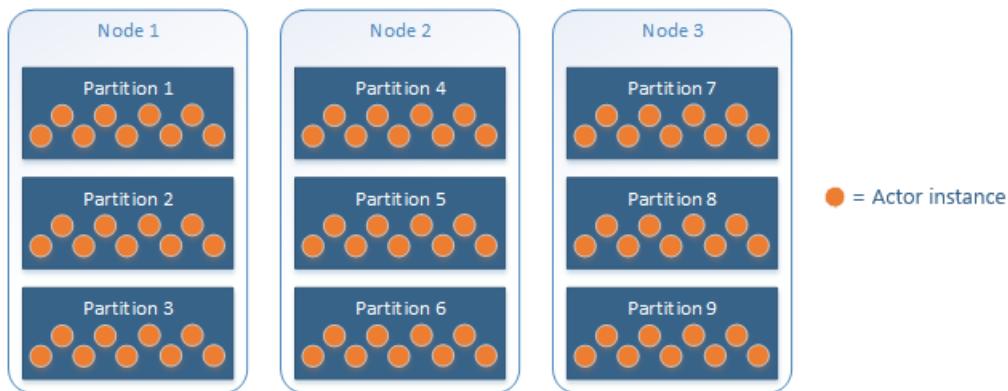
- Calling any actor method for an actor ID activates that actor. For this reason, actor types have their constructor called implicitly by the runtime. Therefore, client code cannot pass parameters to the actor type's constructor, although parameters may be passed to the actor's constructor by the service itself. The result is that actors may be constructed in a partially-initialized state by the time other methods are called on it, if the actor requires initialization parameters from the client. There is no single entry point for the activation of an actor from the client.
- Although Reliable Actors implicitly create actor objects; you do have the ability to explicitly delete an actor and its state.

## Distribution and failover

To provide scalability and reliability, Service Fabric distributes actors throughout the cluster and automatically migrates them from failed nodes to healthy ones as required. This is an abstraction over a [partitioned, stateful Reliable Service](#). Distribution, scalability, reliability, and automatic failover are all provided by virtue of the fact that actors are running inside a stateful Reliable Service called the *Actor Service*.

Actors are distributed across the partitions of the Actor Service, and those partitions are distributed across the nodes in a Service Fabric cluster. Each service partition contains a set of actors. Service Fabric manages distribution and failover of the service partitions.

For example, an actor service with nine partitions deployed to three nodes using the default actor partition placement would be distributed thusly:



The Actor Framework manages partition scheme and key range settings for you. This simplifies some choices but also carries some consideration:

- Reliable Services allows you to choose a partitioning scheme, key range (when using a range partitioning scheme), and partition count. Reliable Actors is restricted to the range partitioning scheme (the uniform Int64 scheme) and requires you use the full Int64 key range.
- By default, actors are randomly placed into partitions resulting in uniform distribution.
- Because actors are randomly placed, it should be expected that actor operations will always require network communication, including serialization and deserialization of method call data, incurring latency and overhead.
- In advanced scenarios, it is possible to control actor partition placement by using Int64 actor IDs that map to specific partitions. However, doing so can result in an unbalanced distribution of actors across partitions.

For more information on how actor services are partitioned, refer to [partitioning concepts for actors](#).

## Actor communication

Actor interactions are defined in an interface that is shared by the actor that implements the interface, and the client that gets a proxy to an actor via the same interface. Because this interface is used to invoke actor methods asynchronously, every method on the interface must be Task-returning.

Method invocations and their responses ultimately result in network requests across the cluster, so the arguments and the result types of the tasks that they return must be serializable by the platform. In particular, they must be [data contract serializable](#).

### The actor proxy

The Reliable Actors client API provides communication between an actor instance and an actor client. To communicate with an actor, a client creates an actor proxy object that implements the actor interface. The client interacts with the actor by invoking methods on the proxy object. The actor proxy can be used for client-to-actor and actor-to-actor communication.

```
// Create a randomly distributed actor ID
ActorId actorId = ActorId.CreateRandom();

// This only creates a proxy object, it does not activate an actor or invoke any methods yet.
IMyActor myActor = ActorProxy.Create<IMyActor>(actorId, new Uri("fabric:/MyApp/MyActorService"));

// This will invoke a method on the actor. If an actor with the given ID does not exist, it will be
// activated by this method call.
await myActor.DoWorkAsync();
```

```
// Create actor ID with some name
ActorId actorId = new ActorId("Actor1");

// This only creates a proxy object, it does not activate an actor or invoke any methods yet.
MyActor myActor = ActorProxyBase.create(actorId, new URI("fabric:/MyApp/MyActorService"), MyActor.class);

// This will invoke a method on the actor. If an actor with the given ID does not exist, it will be
// activated by this method call.
myActor.DoWorkAsync().get();
```

Note that the two pieces of information used to create the actor proxy object are the actor ID and the application name. The actor ID uniquely identifies the actor, while the application name identifies the [Service Fabric application](#) where the actor is deployed.

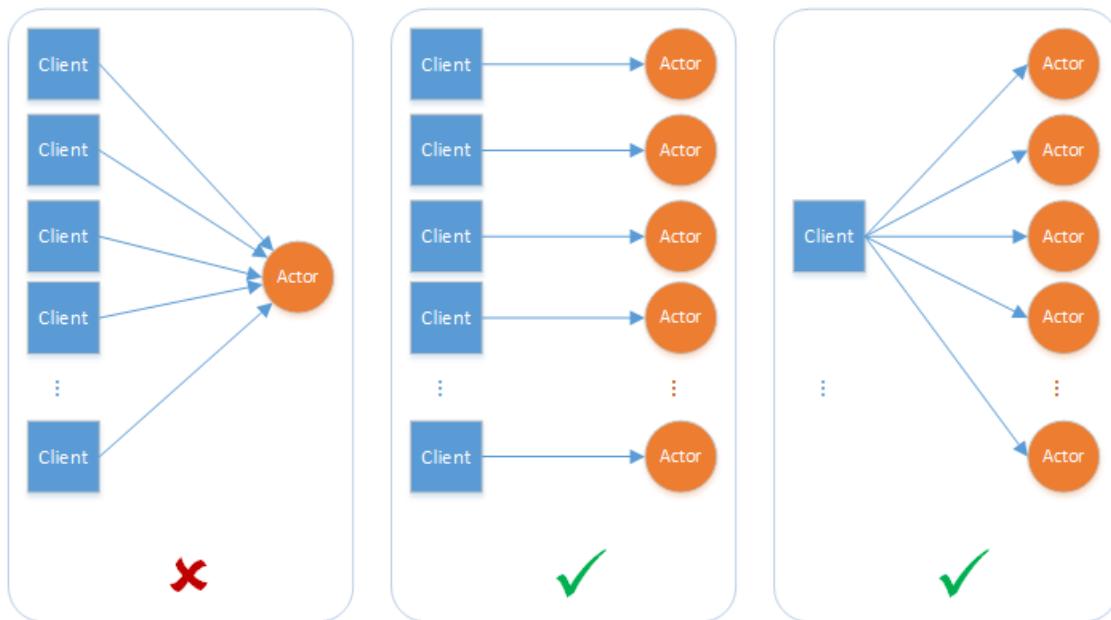
The [ActorProxy](#) (C#) / [ActorProxyBase](#) (Java) class on the client side performs the necessary resolution to locate the actor by ID and open a communication channel with it. It also retries to locate the actor in the cases of communication failures and failovers. As a result, message delivery has the following characteristics:

- Message delivery is best effort.
- Actors may receive duplicate messages from the same client.

### Concurrency

The Reliable Actors runtime provides a simple turn-based access model for accessing actor methods. This means that no more than one thread can be active inside an actor object's code at any time. Turn-based access greatly simplifies concurrent systems as there is no need for synchronization mechanisms for data access. It also means systems must be designed with special considerations for the single-threaded access nature of each actor instance.

- A single actor instance cannot process more than one request at a time. An actor instance can cause a throughput bottleneck if it is expected to handle concurrent requests.
- Actors can deadlock on each other if there is a circular request between two actors while an external request is made to one of the actors simultaneously. The actor runtime will automatically time out on actor calls and throw an exception to the caller to interrupt possible deadlock situations.

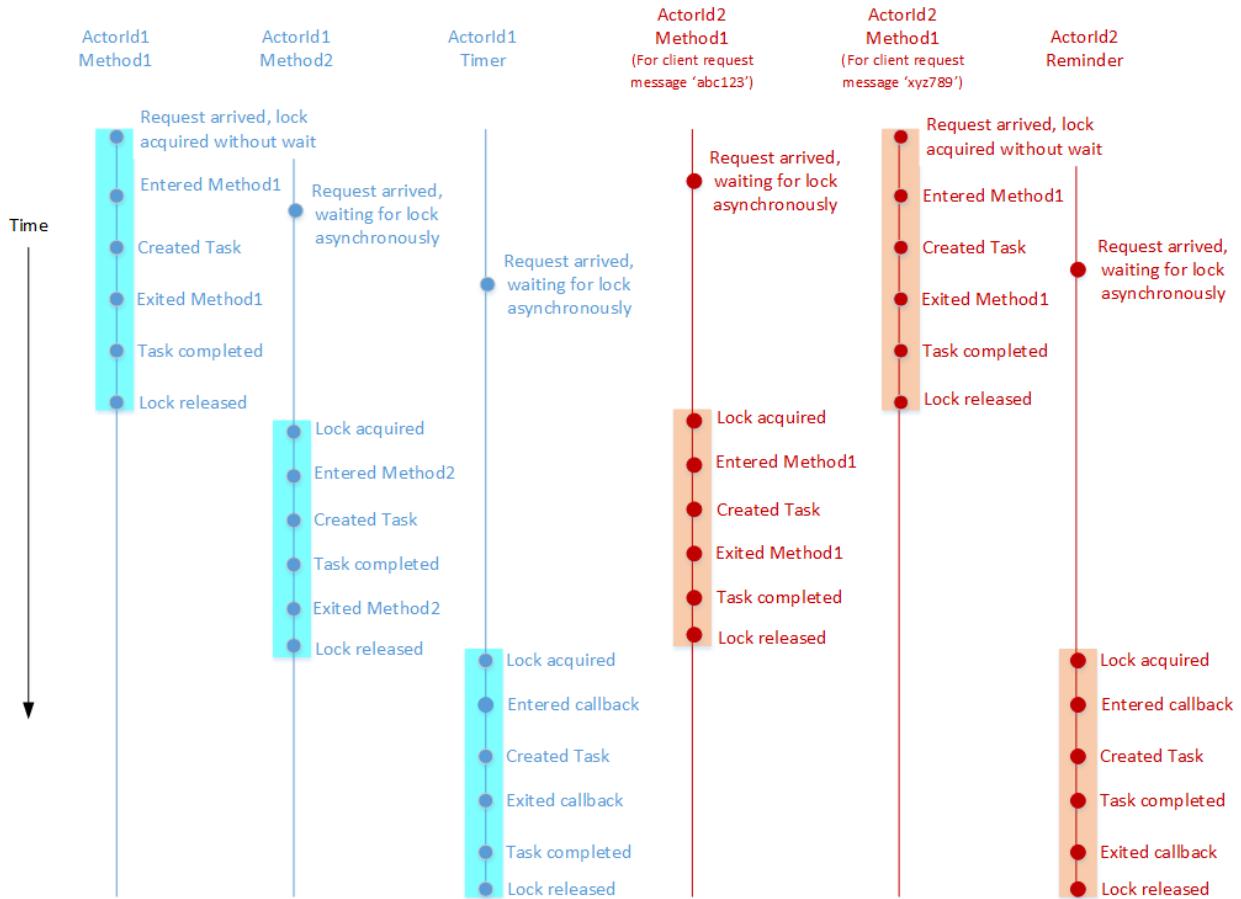


#### Turn-based access

A turn consists of the complete execution of an actor method in response to a request from other actors or clients, or the complete execution of a [timer/reminder](#) callback. Even though these methods and callbacks are asynchronous, the Actors runtime does not interleave them. A turn must be fully finished before a new turn is allowed. In other words, an actor method or timer/reminder callback that is currently executing must be fully finished before a new call to a method or callback is allowed. A method or callback is considered to have finished if the execution has returned from the method or callback and the task returned by the method or callback has finished. It is worth emphasizing that turn-based concurrency is respected even across different methods, timers, and callbacks.

The Actors runtime enforces turn-based concurrency by acquiring a per-actor lock at the beginning of a turn and releasing the lock at the end of the turn. Thus, turn-based concurrency is enforced on a per-actor basis and not across actors. Actor methods and timer/reminder callbacks can execute simultaneously on behalf of different actors.

The following example illustrates the above concepts. Consider an actor type that implements two asynchronous methods (say, *Method1* and *Method2*), a timer, and a reminder. The diagram below shows an example of a timeline for the execution of these methods and callbacks on behalf of two actors (*ActorId1* and *ActorId2*) that belong to this actor type.



This diagram follows these conventions:

- Each vertical line shows the logical flow of execution of a method or a callback on behalf of a particular actor.
- The events marked on each vertical line occur in chronological order, with newer events occurring below older ones.
- Different colors are used for timelines corresponding to different actors.
- Highlighting is used to indicate the duration for which the per-actor lock is held on behalf of a method or callback.

Some important points to consider:

- While *Method1* is executing on behalf of *ActorId2* in response to client request *xyz789*, another client request (*abc123*) arrives that also requires *Method1* to be executed by *ActorId2*. However, the second execution of *Method1* does not begin until the prior execution has finished. Similarly, a reminder registered by *ActorId2* fires while *Method1* is being executed in response to client request *xyz789*. The reminder callback is executed only after both executions of *Method1* are complete. All of this is due to turn-based concurrency being enforced for *ActorId2*.
- Similarly, turn-based concurrency is also enforced for *ActorId1*, as demonstrated by the execution of *Method1*, *Method2*, and the timer callback on behalf of *ActorId1* happening in a serial fashion.
- Execution of *Method1* on behalf of *ActorId1* overlaps with its execution on behalf of *ActorId2*. This is because turn-based concurrency is enforced only within an actor and not across actors.
- In some of the method/callback executions, the `Task (C#)` / `CompletableFuture (Java)` returned by the method/callback finishes after the method returns. In some others, the asynchronous operation has already finished by the time the method/callback returns. In both cases, the per-actor lock is released only after both the method/callback returns and the asynchronous operation finishes.

### Reentrancy

The Actors runtime allows reentrancy by default. This means that if an actor method of *Actor A* calls a method on *Actor B*, which in turn calls another method on *Actor A*, that method is allowed to run. This is because it is part of

the same logical call-chain context. All timer and reminder calls start with the new logical call context. See the [Reliable Actors reentrancy](#) for more details.

#### **Scope of concurrency guarantees**

The Actors runtime provides these concurrency guarantees in situations where it controls the invocation of these methods. For example, it provides these guarantees for the method invocations that are done in response to a client request, as well as for timer and reminder callbacks. However, if the actor code directly invokes these methods outside of the mechanisms provided by the Actors runtime, then the runtime cannot provide any concurrency guarantees. For example, if the method is invoked in the context of some task that is not associated with the task returned by the actor methods, then the runtime cannot provide concurrency guarantees. If the method is invoked from a thread that the actor creates on its own, then the runtime also cannot provide concurrency guarantees. Therefore, to perform background operations, actors should use [actor timers and actor reminders](#) that respect turn-based concurrency.

## Next steps

- Get started by building your first Reliable Actors service:
  - [Getting started with Reliable Actors on .NET](#)
  - [Getting started with Reliable Actors on Java](#)

# How Reliable Actors use the Service Fabric platform

9/27/2017 • 8 min to read • [Edit Online](#)

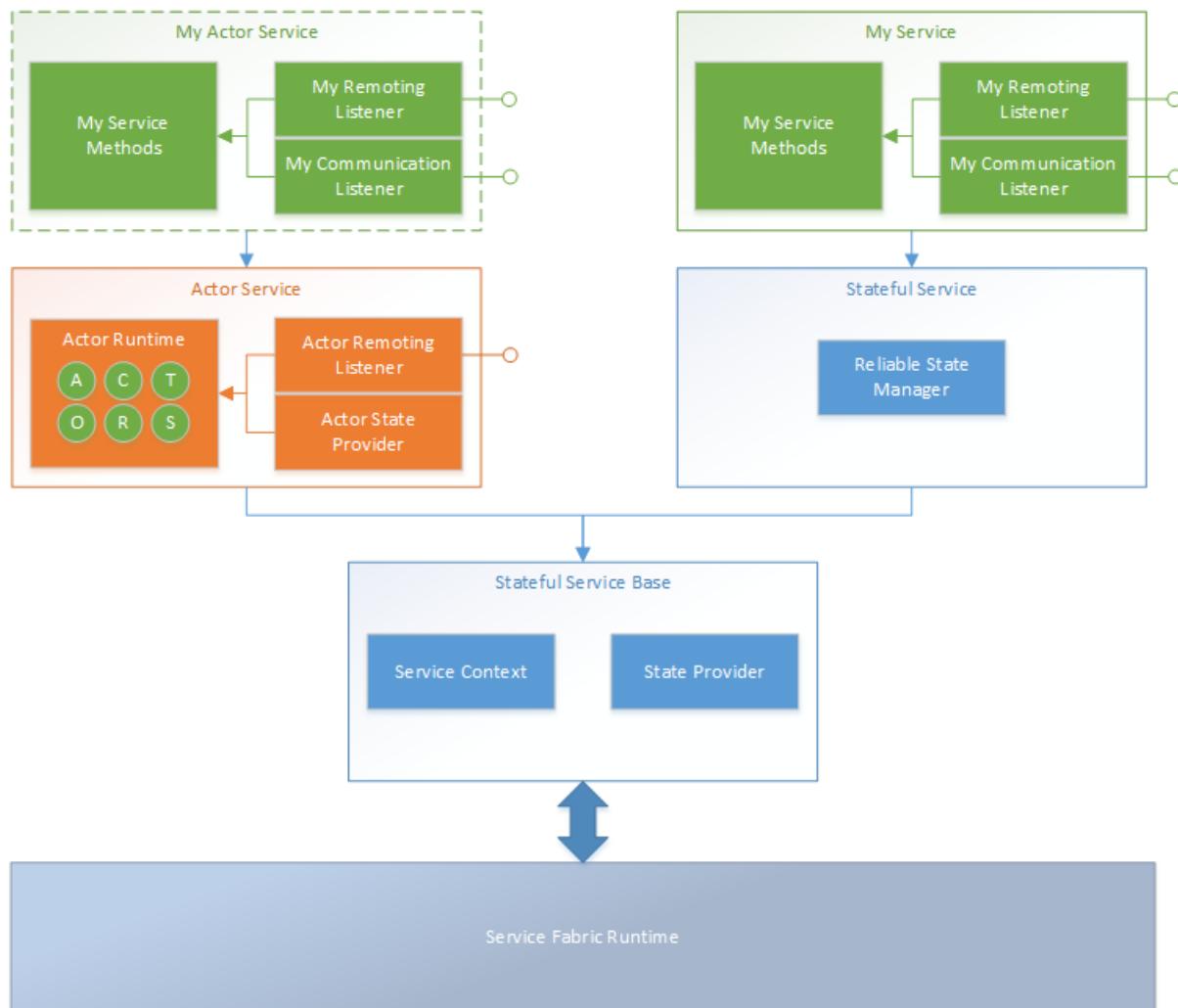
This article explains how Reliable Actors work on the Azure Service Fabric platform. Reliable Actors run in a framework that is hosted in an implementation of a stateful reliable service called the *actor service*. The actor service contains all the components necessary to manage the lifecycle and message dispatching for your actors:

- The Actor Runtime manages lifecycle, garbage collection, and enforces single-threaded access.
- An actor service remoting listener accepts remote access calls to actors and sends them to a dispatcher to route to the appropriate actor instance.
- The Actor State Provider wraps state providers (such as the Reliable Collections state provider) and provides an adapter for actor state management.

These components together form the Reliable Actor framework.

## Service layering

Because the actor service itself is a reliable service, all the [application model](#), lifecycle, [packaging](#), deployment, upgrade, and scaling concepts of Reliable Services apply the same way to actor services.



The preceding diagram shows the relationship between the Service Fabric application frameworks and user code. Blue elements represent the Reliable Services application framework, orange represents the Reliable Actor framework, and green represents user code.

In Reliable Services, your service inherits the `StatefulService` class. This class is itself derived from `StatefulServiceBase` (or `StatelessService` for stateless services). In Reliable Actors, you use the actor service. The actor service is a different implementation of the `StatefulServiceBase` class that implements the actor pattern where your actors run. Because the actor service itself is just an implementation of `StatefulServiceBase`, you can write your own service that derives from `ActorService` and implement service-level features the same way you would when inheriting `StatefulService`, such as:

- Service backup and restore.
- Shared functionality for all actors, for example, a circuit breaker.
- Remote procedure calls on the actor service itself and on each individual actor.

#### NOTE

Stateful services are not currently supported in Java/Linux.

## Using the actor service

Actor instances have access to the actor service in which they are running. Through the actor service, actor instances can programmatically obtain the service context. The service context has the partition ID, service name, application name, and other Service Fabric platform-specific information:

```
Task MyActorMethod()
{
    Guid partitionId = this.ActorService.Context.PartitionId;
    string serviceTypeName = this.ActorService.Context.ServiceTypeName;
    Uri serviceInstanceName = this.ActorService.Context.ServiceName;
    string applicationInstanceName = this.ActorService.Context.CodePackageActivationContext.ApplicationName;
}
```

```
CompletableFuture<?> MyActorMethod()
{
    UUID partitionId = this.getActorService().getServiceContext().getPartitionId();
    String serviceTypeName = this.getActorService().getServiceContext().getServiceTypeName();
    URI serviceInstanceName = this.getActorService().getServiceContext().getServiceName();
    String applicationInstanceName =
    this.getActorService().getServiceContext().getCodePackageActivationContext().getApplicationName();
}
```

Like all Reliable Services, the actor service must be registered with a service type in the Service Fabric runtime. For the actor service to run your actor instances, your actor type must also be registered with the actor service. The `ActorRuntime` registration method performs this work for actors. In the simplest case, you can just register your actor type, and the actor service with default settings will implicitly be used:

```
static class Program
{
    private static void Main()
    {
        ActorRuntime.RegisterActorAsync<MyActor>().GetAwaiter().GetResult();

        Thread.Sleep(Timeout.Infinite);
    }
}
```

Alternatively, you can use a lambda provided by the registration method to construct the actor service yourself. You can then configure the actor service and explicitly construct your actor instances, where you can inject dependencies to your actor through its constructor:

```

static class Program
{
    private static void Main()
    {
        ActorRuntime.RegisterActorAsync<MyActor>(
            (context, actorType) => new ActorService(context, actorType, () => new MyActor())
            .GetAwaiter().GetResult();

        Thread.Sleep(Timeout.Infinite);
    }
}

```

```

static class Program
{
    private static void Main()
    {
        ActorRuntime.registerActorAsync(
            MyActor.class,
            (context, actorTypeInfo) -> new FabricActorService(context, actorTypeInfo),
            timeout);

        Thread.sleep(Long.MAX_VALUE);
    }
}

```

## Actor service methods

The Actor service implements `IActorService` (C#) or `ActorService` (Java), which in turn implements `IService` (C#) or `Service` (Java). This is the interface used by Reliable Services remoting, which allows remote procedure calls on service-level methods that can be called remotely via service remoting.

### Enumerating actors

The actor service allows a client to enumerate metadata about the actors that the service is hosting. Because the actor service is a partitioned stateful service, enumeration is performed per partition. Because each partition might contain many actors, the enumeration is returned as a set of paged results. The pages are looped over until all pages are read. The following example shows how to create a list of all active actors in one partition of an actor service:

```

IActorService actorServiceProxy = ActorServiceProxy.Create(
    new Uri("fabric:/MyApp/MyService"), partitionKey);

ContinuationToken continuationToken = null;
List<ActorInformation> activeActors = new List<ActorInformation>();

do
{
    PagedResult<ActorInformation> page = await actorServiceProxy.GetActorsAsync(continuationToken,
        cancellationToken);

    activeActors.AddRange(page.Items.Where(x => x.IsActive));

    continuationToken = page.ContinuationToken;
}
while (continuationToken != null);

```

```

ActorService actorServiceProxy = ActorServiceProxy.create(
    new URI("fabric:/MyApp/MyService"), partitionKey);

ContinuationToken continuationToken = null;
List<ActorInformation> activeActors = new ArrayList<ActorInformation>();

do
{
    PagedResult<ActorInformation> page = actorServiceProxy.getActorsAsync(continuationToken);

    while(ActorInformation x: page.getItems())
    {
        if(x.isActive()){
            activeActors.add(x);
        }
    }

    continuationToken = page.getContinuationToken();
}
while (continuationToken != null);

```

### Deleting actors

The actor service also provides a function for deleting actors:

```

ActorId actorToDelete = new ActorId(id);

IActorService myActorServiceProxy = ActorServiceProxy.Create(
    new Uri("fabric:/MyApp/MyService"), actorToDelete);

await myActorServiceProxy.DeleteActorAsync(actorToDelete, cancellationToken)

```

```

ActorId actorToDelete = new ActorId(id);

ActorService myActorServiceProxy = ActorServiceProxy.create(
    new URI("fabric:/MyApp/MyService"), actorToDelete);

myActorServiceProxy.deleteActorAsync(actorToDelete);

```

For more information on deleting actors and their state, see the [actor lifecycle documentation](#).

### Custom actor service

By using the actor registration lambda, you can register your own custom actor service that derives from `ActorService` (C#) and `FabricActorService` (Java). In this custom actor service, you can implement your own service-level functionality by writing a service class that inherits `ActorService` (C#) or `FabricActorService` (Java). A custom actor service inherits all the actor runtime functionality from `ActorService` (C#) or `FabricActorService` (Java) and can be used to implement your own service methods.

```

class MyActorService : ActorService
{
    public MyActorService(StatefulServiceContext context, ActorTypeInformation typeInfo, Func<ActorBase>
newActor)
        : base(context, typeInfo, newActor)
    { }

}

```

```
class MyActorService extends FabricActorService
{
    public MyActorService(StatefulServiceContext context, ActorTypeInformation typeInfo,
BiFunction<FabricActorService, ActorId, ActorBase> newActor)
    {
        super(context, typeInfo, newActor);
    }
}
```

```
static class Program
{
    private static void Main()
    {
        ActorRuntime.RegisterActorAsync<MyActor>(
            (context, actorType) => new MyActorService(context, actorType, () => new MyActor()))
            .GetAwaiter().GetResult();

        Thread.Sleep(Timeout.Infinite);
    }
}
```

```
public class Program
{
    public static void main(String[] args)
    {
        ActorRuntime.registerActorAsync(
            MyActor.class,
            (context, actorTypeInfo) -> new FabricActorService(context, actorTypeInfo),
            timeout);
        Thread.sleep(Long.MAX_VALUE);
    }
}
```

### Implementing actor backup and restore

In the following example, the custom actor service exposes a method to back up actor data by taking advantage of the remoting listener already present in `ActorService`:

```
public interface IMyActorService : IService
{
    Task BackupActorsAsync();
}

class MyActorService : ActorService, IMyActorService
{
    public MyActorService(StatefulServiceContext context, ActorTypeInformation typeInfo, Func<ActorBase>
newActor)
        : base(context, typeInfo, newActor)
    { }

    public Task BackupActorsAsync()
    {
        return this.BackupAsync(new BackupDescription(PerformBackupAsync));
    }

    private async Task<bool> PerformBackupAsync(BackupInfo backupInfo, CancellationToken cancellationToken)
    {
        try
        {
            // store the contents of backupInfo.Directory
            return true;
        }
        finally
        {
            Directory.Delete(backupInfo.Directory, recursive: true);
        }
    }
}
```

```

public interface MyActorService extends Service
{
    CompletableFuture<?> backupActorsAsync();
}

class MyActorServiceImpl extends ActorService implements MyActorService
{
    public MyActorService(StatefulServiceContext context, ActorTypeInformation typeInfo,
    Func<FabricActorService, ActorId, ActorBase> newActor)
    {
        super(context, typeInfo, newActor);
    }

    public CompletableFuture backupActorsAsync()
    {
        return this.backupAsync(new BackupDescription((backupInfo, cancellationToken) ->
performBackupAsync(backupInfo, cancellationToken)));
    }

    private CompletableFuture<Boolean> performBackupAsync(BackupInfo backupInfo, CancellationToken
cancellationToken)
    {
        try
        {
            // store the contents of backupInfo.Directory
            return true;
        }
        finally
        {
            deleteDirectory(backupInfo.Directory)
        }
    }

    void deleteDirectory(File file) {
        File[] contents = file.listFiles();
        if (contents != null) {
            for (File f : contents) {
                deleteDirectory(f);
            }
        }
        file.delete();
    }
}

```

In this example, `IMyActorService` is a remoting contract that implements `IService` (C#) and `Service` (Java), and is then implemented by `MyActorService`. By adding this remoting contract, methods on `IMyActorService` are now also available to a client by creating a remoting proxy via `ActorServiceProxy`:

```

IMyActorService myActorServiceProxy = ActorServiceProxy.Create<IMyActorService>(
    new Uri("fabric:/MyApp/MyService"), ActorId.CreateRandom());

await myActorServiceProxy.BackupActorsAsync();

```

```

MyActorService myActorServiceProxy = ActorServiceProxy.create(MyActorService.class,
    new URI("fabric:/MyApp/MyService"), actorId);

myActorServiceProxy.backupActorsAsync();

```

## Application model

Actor services are Reliable Services, so the application model is the same. However, the actor framework build

tools generate some of the application model files for you.

## Service manifest

The actor framework build tools automatically generate the contents of your actor service's ServiceManifest.xml file. This file includes:

- Actor service type. The type name is generated based on your actor's project name. Based on the persistence attribute on your actor, the HasPersistedState flag is also set accordingly.
- Code package.
- Config package.
- Resources and endpoints.

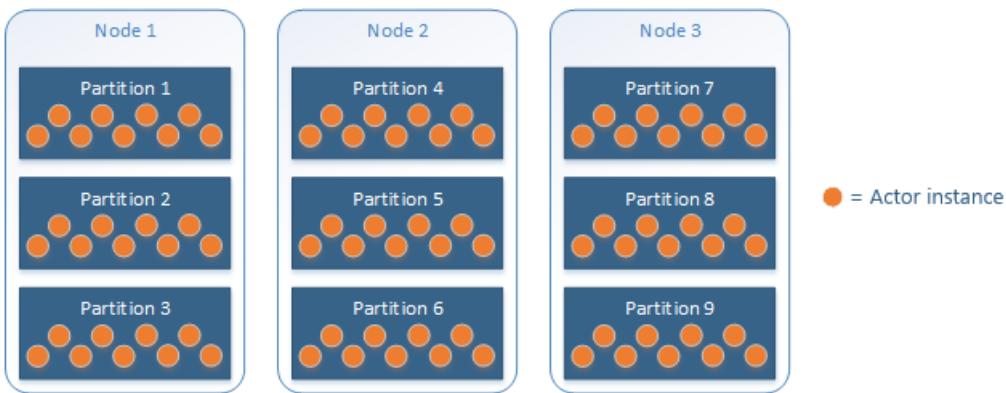
## Application manifest

The actor framework build tools automatically create a default service definition for your actor service. The build tools populate the default service properties:

- Replica set count is determined by the persistence attribute on your actor. Each time the persistence attribute on your actor is changed, the replica set count in the default service definition is reset accordingly.
- Partition scheme and range are set to Uniform Int64 with the full Int64 key range.

# Service Fabric partition concepts for actors

Actor services are partitioned stateful services. Each partition of an actor service contains a set of actors. Service partitions are automatically distributed over multiple nodes in Service Fabric. Actor instances are distributed as a result.



Reliable Services can be created with different partition schemes and partition key ranges. The actor service uses the Int64 partitioning scheme with the full Int64 key range to map actors to partitions.

## Actor ID

Each actor that's created in the service has a unique ID associated with it, represented by the `ActorId` class.

`ActorId` is an opaque ID value that can be used for uniform distribution of actors across the service partitions by generating random IDs:

```
ActorProxy.Create<IMyActor>(ActorId.CreateRandom());
```

```
ActorProxyBase.create<MyActor>(MyActor.class, ActorId.newId());
```

Every `ActorId` is hashed to an Int64. This is why the actor service must use an Int64 partitioning scheme with the full Int64 key range. However, custom ID values can be used for an `ActorID`, including GUIDs/UUIDs, strings, and Int64s.

```
ActorProxy.Create<IMyActor>(new ActorId(Guid.NewGuid()));
ActorProxy.Create<IMyActor>(new ActorId("myActorId"));
ActorProxy.Create<IMyActor>(new ActorId(1234));
```

```
ActorProxyBase.create(MyActor.class, new ActorId(UUID.randomUUID()));
ActorProxyBase.create(MyActor.class, new ActorId("myActorId"));
ActorProxyBase.create(MyActor.class, new ActorId(1234));
```

When you're using GUIDs/UUIDs and strings, the values are hashed to an Int64. However, when you're explicitly providing an Int64 to an `ActorId`, the Int64 will map directly to a partition without further hashing. You can use this technique to control which partition the actors are placed in.

## Actor using Remoting V2 Stack

With 2.8 nuget package, users can now use Remoting V2 stack, which is more performant and provides features like custom Serialization. Remoting V2 is not backward compatible with existing Remoting stack (we are calling now it as V1 Remoting stack).

Following changes are required to use the Remoting V2 Stack.

1. Add the following assembly attribute on Actor Interfaces.

```
[assembly:FabricTransportActorRemotingProvider(RemotingListener =
RemotingListener.V2Listener,RemotingClient = RemotingClient.V2Client)]
```

2. Build and Upgrade ActorService And Actor Client projects to start using V2 Stack.

### **Actor Service Upgrade to Remoting V2 Stack without impacting Service Availability.**

This change will be a 2-step upgrade. Follow the steps in the same sequence as listed.

1. Add the following assembly attribute on Actor Interfaces. This attribute will start two listeners for ActorService, V1 (existing) and V2 Listener. Upgrade ActorService with this change.

```
[assembly:FabricTransportActorRemotingProvider(RemotingListener =
RemotingListener.CompatListener,RemotingClient = RemotingClient.V2Client)]
```

2. Upgrade ActorClients after completing the above upgrade. This step makes sure Actor Proxy is using Remoting V2 Stack.

3. This step is optional. Change the above attribute to remove V1 Listener.

```
[assembly:FabricTransportActorRemotingProvider(RemotingListener =
RemotingListener.V2Listener,RemotingClient = RemotingClient.V2Client)]
```

## Next steps

- [Actor state management](#)
- [Actor lifecycle and garbage collection](#)
- [Actors API reference documentation](#)
- [.NET sample code](#)
- [Java sample code](#)

# Actor lifecycle, automatic garbage collection, and manual delete

10/12/2017 • 5 min to read • [Edit Online](#)

An actor is activated the first time a call is made to any of its methods. An actor is deactivated (garbage collected by the Actors runtime) if it is not used for a configurable period of time. An actor and its state can also be deleted manually at any time.

## Actor activation

When an actor is activated, the following occurs:

- When a call comes for an actor and one is not already active, a new actor is created.
- The actor's state is loaded if it's maintaining state.
- The `OnActivateAsync` (C#) or `onActivateAsync` (Java) method (which can be overridden in the actor implementation) is called.
- The actor is now considered active.

## Actor deactivation

When an actor is deactivated, the following occurs:

- When an actor is not used for some period of time, it is removed from the Active Actors table.
- The `OnDeactivateAsync` (C#) or `onDeactivateAsync` (Java) method (which can be overridden in the actor implementation) is called. This clears all the timers for the actor. Actor operations like state changes should not be called from this method.

### TIP

The Fabric Actors runtime emits some [events related to actor activation and deactivation](#). They are useful in diagnostics and performance monitoring.

## Actor garbage collection

When an actor is deactivated, references to the actor object are released and it can be garbage collected normally by the common language runtime (CLR) or java virtual machine (JVM) garbage collector. Garbage collection only cleans up the actor object; it does **not** remove state stored in the actor's State Manager. The next time the actor is activated, a new actor object is created and its state is restored.

What counts as "being used" for the purpose of deactivation and garbage collection?

- Receiving a call
- `IRemindable.ReceiveReminderAsync` method being invoked (applicable only if the actor uses reminders)

### NOTE

If the actor uses timers and its timer callback is invoked, it does **not** count as "being used".

Before we go into the details of deactivation, it is important to define the following terms:

- *Scan interval*. This is the interval at which the Actors runtime scans its Active Actors table for actors that can be deactivated and garbage collected. The default value for this is 1 minute.
- *Idle timeout*. This is the amount of time that an actor needs to remain unused (idle) before it can be deactivated and garbage collected. The default value for this is 60 minutes.

Typically, you do not need to change these defaults. However, if necessary, these intervals can be changed through `ActorServiceSettings` when registering your [Actor Service](#):

```
public class Program
{
    public static void Main(string[] args)
    {
        ActorRuntime.RegisterActorAsync<MyActor>((context, actorType) =>
            new ActorService(context, actorType,
                settings:
                    new ActorServiceSettings()
                    {
                        ActorGarbageCollectionSettings =
                            new ActorGarbageCollectionSettings(10, 2)
                    })
            .GetAwaiter()
            .GetResult();
    }
}
```

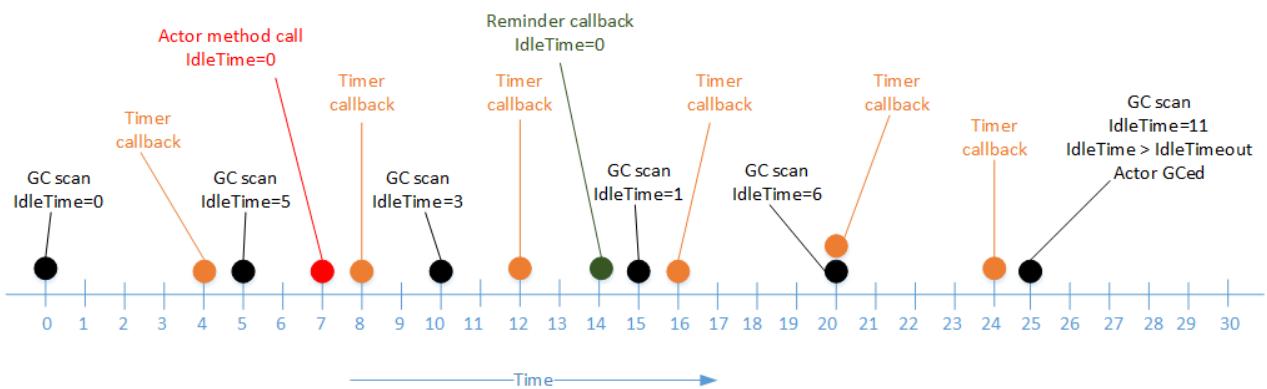
```
public class Program
{
    public static void main(String[] args)
    {
        ActorRuntime.registerActorAsync(
            MyActor.class,
            (context, actorTypeInfo) -> new FabricActorService(context, actorTypeInfo),
            timeout);
    }
}
```

For each active actor, the actor runtime keeps track of the amount of time that it has been idle (i.e. not used). The actor runtime checks each of the actors every `ScanIntervalInSeconds` to see if it can be garbage collected and collects it if it has been idle for `IdleTimeoutInSeconds`.

Anytime an actor is used, its idle time is reset to 0. After this, the actor can be garbage collected only if it again remains idle for `IdleTimeoutInSeconds`. Recall that an actor is considered to have been used if either an actor interface method or an actor reminder callback is executed. An actor is **not** considered to have been used if its timer callback is executed.

The following diagram shows the lifecycle of a single actor to illustrate these concepts.

## ScanInterval = 5, IdleTimeout = 10



The example shows the impact of actor method calls, reminders, and timers on the lifetime of this actor. The following points about the example are worth mentioning:

- ScanInterval and IdleTimeout are set to 5 and 10 respectively. (Units do not matter here, since our purpose is only to illustrate the concept.)
- The scan for actors to be garbage collected happens at T=0,5,10,15,20,25, as defined by the scan interval of 5.
- A periodic timer fires at T=4,8,12,16,20,24, and its callback executes. It does not impact the idle time of the actor.
- An actor method call at T=7 resets the idle time to 0 and delays the garbage collection of the actor.
- An actor reminder callback executes at T=14 and further delays the garbage collection of the actor.
- During the garbage collection scan at T=25, the actor's idle time finally exceeds the idle timeout of 10, and the actor is garbage collected.

An actor will never be garbage collected while it is executing one of its methods, no matter how much time is spent in executing that method. As mentioned earlier, the execution of actor interface methods and reminder callbacks prevents garbage collection by resetting the actor's idle time to 0. The execution of timer callbacks does not reset the idle time to 0. However, the garbage collection of the actor is deferred until the timer callback has completed execution.

## Deleting actors and their state

Garbage collection of deactivated actors only cleans up the actor object, but it does not remove data that is stored in an actor's State Manager. When an actor is re-activated, its data is again made available to it through the State Manager. In cases where actors store data in State Manager and are deactivated but never re-activated, it may be necessary to clean up their data.

The [Actor Service](#) provides a function for deleting actors from a remote caller:

```
ActorId actorToDelete = new ActorId(id);

IActorService myActorServiceProxy = ActorServiceProxy.Create(
    new Uri("Fabric:/MyApp/MyService"), actorToDelete);

await myActorServiceProxy.DeleteActorAsync(actorToDelete, cancellationToken)
```

```
ActorId actorToDelete = new ActorId(id);

ActorService myActorServiceProxy = ActorServiceProxy.create(
    new Uri("Fabric:/MyApp/MyService"), actorToDelete);

myActorServiceProxy.deleteActorAsync(actorToDelete);
```

Deleting an actor has the following effects depending on whether or not the actor is currently active:

- **Active Actor**

- Actor is removed from active actors list and is deactivated.
- Its state is deleted permanently.

- **Inactive Actor**

- Its state is deleted permanently.

Note that an actor cannot call `delete` on itself from one of its actor methods because the actor cannot be deleted while executing within an actor call context, in which the runtime has obtained a lock around the actor call to enforce single-threaded access.

## Next steps

- [Actor timers and reminders](#)
- [Actor events](#)
- [Actor reentrancy](#)
- [Actor diagnostics and performance monitoring](#)
- [Actor API reference documentation](#)
- [C# Sample code](#)
- [Java Sample code](#)

# Reliable Actors state management

6/30/2017 • 8 min to read • [Edit Online](#)

Reliable Actors are single-threaded objects that can encapsulate both logic and state. Because actors run on Reliable Services, they can maintain state reliably by using the same persistence and replication mechanisms that Reliable Services uses. This way, actors don't lose their state after failures, upon reactivation after garbage collection, or when they are moved around between nodes in a cluster due to resource balancing or upgrades.

## State persistence and replication

All Reliable Actors are considered *stateful* because each actor instance maps to a unique ID. This means that repeated calls to the same actor ID are routed to the same actor instance. In a stateless system, by contrast, client calls are not guaranteed to be routed to the same server every time. For this reason, actor services are always stateful services.

Even though actors are considered stateful, that does not mean they must store state reliably. Actors can choose the level of state persistence and replication based on their data storage requirements:

- **Persisted state:** State is persisted to disk and is replicated to 3 or more replicas. This is the most durable state storage option, where state can persist through complete cluster outage.
- **Volatile state:** State is replicated to 3 or more replicas and only kept in memory. This provides resilience against node failure and actor failure, and during upgrades and resource balancing. However, state is not persisted to disk. So if all replicas are lost at once, the state is lost as well.
- **No persisted state:** State is not replicated or written to disk. This level is for actors that simply don't need to maintain state reliably.

Each level of persistence is simply a different *state provider* and *replication* configuration of your service. Whether or not state is written to disk depends on the state provider--the component in a reliable service that stores state. Replication depends on how many replicas a service is deployed with. As with Reliable Services, both the state provider and replica count can easily be set manually. The actor framework provides an attribute that, when used on an actor, automatically selects a default state provider and automatically generates settings for replica count to achieve one of these three persistence settings. The `StatePersistence` attribute is not inherited by derived class, each Actor type must provide its `StatePersistence` level.

### Persisted state

```
[StatePersistence(StatePersistence.Persisted)]
class MyActor : Actor, IMyActor
{}
```

```
@StatePersistenceAttribute(statePersistence = StatePersistence.Persisted)
class MyActorImpl extends FabricActor implements MyActor
{}
```

This setting uses a state provider that stores data on disk and automatically sets the service replica count to 3.

### Volatile state

```
[StatePersistence(StatePersistence.Volatile)]
class MyActor : Actor, IMyActor
{}
```

```
@StatePersistenceAttribute(statePersistence = StatePersistence.Volatile)
class MyActorImpl extends FabricActor implements MyActor
{}
```

This setting uses an in-memory-only state provider and sets the replica count to 3.

## No persisted state

```
[StatePersistence(StatePersistence.None)]
class MyActor : Actor, IMyActor
{}
```

```
@StatePersistenceAttribute(statePersistence = StatePersistence.None)
class MyActorImpl extends FabricActor implements MyActor
{}
```

This setting uses an in-memory-only state provider and sets the replica count to 1.

## Defaults and generated settings

When you're using the `StatePersistence` attribute, a state provider is automatically selected for you at runtime when the actor service starts. The replica count, however, is set at compile time by the Visual Studio actor build tools. The build tools automatically generate a *default service* for the actor service in ApplicationManifest.xml.

Parameters are created for **min replica set size** and **target replica set size**.

You can change these parameters manually. But each time the `StatePersistence` attribute is changed, the parameters are set to the default replica set size values for the selected `StatePersistence` attribute, overriding any previous values. In other words, the values that you set in ServiceManifest.xml are *only* overridden at build time when you change the `StatePersistence` attribute value.

```

<ApplicationManifest xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ApplicationTypeName="Application12Type" ApplicationTypeVersion="1.0.0"
xmlns="http://schemas.microsoft.com/2011/01/fabric">
    <Parameters>
        <Parameter Name="MyActorService_PartitionCount" DefaultValue="10" />
        <Parameter Name="MyActorService_MinReplicaSetSize" DefaultValue="3" />
        <Parameter Name="MyActorService_TargetReplicaSetSize" DefaultValue="3" />
    </Parameters>
    <ServiceManifestImport>
        <ServiceManifestRef ServiceManifestName="MyActorPkg" ServiceManifestVersion="1.0.0" />
    </ServiceManifestImport>
    <DefaultServices>
        <Service Name="MyActorService" GeneratedIdRef="77d965dc-85fb-488c-bd06-c6c1fe29d593|Persisted">
            <StatefulService ServiceTypeName="MyActorServiceType" TargetReplicaSetSize="
[MyActorService_TargetReplicaSetSize]" MinReplicaSetSize="[MyActorService_MinReplicaSetSize]">
                <UniformInt64Partition PartitionCount="[MyActorService_PartitionCount]" LowKey="-
9223372036854775808" HighKey="9223372036854775807" />
            </StatefulService>
        </Service>
    </DefaultServices>
</ApplicationManifest>

```

## State manager

Every actor instance has its own state manager: a dictionary-like data structure that reliably stores key/value pairs. The state manager is a wrapper around a state provider. You can use it to store data regardless of which persistence setting is used. It does not provide any guarantees that a running actor service can be changed from a volatile (in-memory-only) state setting to a persisted state setting through a rolling upgrade while preserving data. However, it is possible to change replica count for a running service.

State manager keys must be strings. Values are generic and can be any type, including custom types. Values stored in the state manager must be data contract serializable because they might be transmitted over the network to other nodes during replication and might be written to disk, depending on an actor's state persistence setting.

The state manager exposes common dictionary methods for managing state, similar to those found in Reliable Dictionary.

### Accessing state

State can be accessed through the state manager by key. State manager methods are all asynchronous because they might require disk I/O when actors have persisted state. Upon first access, state objects are cached in memory. Repeat access operations access objects directly from memory and return synchronously without incurring disk I/O or asynchronous context-switching overhead. A state object is removed from the cache in the following cases:

- An actor method throws an unhandled exception after it retrieves an object from the state manager.
- An actor is reactivated, either after being deactivated or after failure.
- The state provider pages state to disk. This behavior depends on the state provider implementation. The default state provider for the `Persisted` setting has this behavior.

You can retrieve state by using a standard `Get` operation that throws `KeyNotFoundException` (C#) or `NoSuchElementException` (Java) if an entry does not exist for the key:

```
[StatePersistence(StatePersistence.Persisted)]
class MyActor : Actor, IMyActor
{
    public MyActor(ActorService actorService, ActorId actorId)
        : base(actorService, actorId)
    {
    }

    public Task<int> GetCountAsync()
    {
        return this.StateManager.GetStateAsync<int>("MyState");
    }
}
```

```
@StatePersistenceAttribute(statePersistence = StatePersistence.Persisted)
class MyActorImpl extends FabricActor implements MyActor
{
    public MyActorImpl(ActorService actorService, ActorId actorId)
    {
        super(actorService, actorId);
    }

    public CompletableFuture<Integer> getCountAsync()
    {
        return this.stateManager().getStateAsync("MyState");
    }
}
```

You can also retrieve state by using a *TryGet* method that does not throw if an entry does not exist for a key:

```
class MyActor : Actor, IMyActor
{
    public MyActor(ActorService actorService, ActorId actorId)
        : base(actorService, actorId)
    {
    }

    public async Task<int> GetCountAsync()
    {
        ConditionalValue<int> result = await this.StateManager.TryGetStateAsync<int>("MyState");
        if (result.HasValue)
        {
            return result.Value;
        }

        return 0;
    }
}
```

```

class MyActorImpl extends FabricActor implements MyActor
{
    public MyActorImpl(ActorService actorService, ActorId actorId)
    {
        super(actorService, actorId);
    }

    public CompletableFuture<Integer> getCountAsync()
    {
        return this.stateManager().<Integer>tryGetStateAsync("MyState").thenApply(result -> {
            if (result.HasValue())
                return result.getValue();
            else
                return 0;
        });
    }
}

```

## Saving state

The state manager retrieval methods return a reference to an object in local memory. Modifying this object in local memory alone does not cause it to be saved durably. When an object is retrieved from the state manager and modified, it must be reinserted into the state manager to be saved durably.

You can insert state by using an unconditional *Set*, which is the equivalent of the `dictionary["key"] = value` syntax:

```

[StatePersistence(StatePersistence.Persisted)]
class MyActor : Actor, IMyActor
{
    public MyActor(ActorService actorService, ActorId actorId)
        : base(actorService, actorId)
    {
    }

    public Task SetCountAsync(int value)
    {
        return this.StateManager.SetStateAsync<int>("MyState", value);
    }
}

```

```

@StatePersistenceAttribute(statePersistence = StatePersistence.Persisted)
class MyActorImpl extends FabricActor implements MyActor
{
    public MyActorImpl(ActorService actorService, ActorId actorId)
    {
        super(actorService, actorId);
    }

    public CompletableFuture setCountAsync(int value)
    {
        return this.stateManager().setStateAsync("MyState", value);
    }
}

```

You can add state by using an *Add* method. This method throws `InvalidOperationException` (C#) or `IllegalStateException` (Java) when it tries to add a key that already exists.

```
[StatePersistence(StatePersistence.Persisted)]
class MyActor : Actor, IMyActor
{
    public MyActor(ActorService actorService, ActorId actorId)
        : base(actorService, actorId)
    {
    }

    public Task AddCountAsync(int value)
    {
        return this.StateManager.AddStateAsync<int>("MyState", value);
    }
}
```

```
@StatePersistenceAttribute(statePersistence = StatePersistence.Persisted)
class MyActorImpl extends FabricActor implements MyActor
{
    public MyActorImpl(ActorService actorService, ActorId actorId)
    {
        super(actorService, actorId);
    }

    public CompletableFuture addCountAsync(int value)
    {
        return this.stateManager().addOrUpdateStateAsync("MyState", value, (key, old_value) -> old_value +
value);
    }
}
```

You can also add state by using a *TryAdd* method. This method does not throw when it tries to add a key that already exists.

```
[StatePersistence(StatePersistence.Persisted)]
class MyActor : Actor, IMyActor
{
    public MyActor(ActorService actorService, ActorId actorId)
        : base(actorService, actorId)
    {
    }

    public async Task AddCountAsync(int value)
    {
        bool result = await this.StateManager.TryAddStateAsync<int>("MyState", value);

        if (result)
        {
            // Added successfully!
        }
    }
}
```

```

@StatePersistenceAttribute(statePersistence = StatePersistence.Persisted)
class MyActorImpl extends FabricActor implements MyActor
{
    public MyActorImpl(ActorService actorService, ActorId actorId)
    {
        super(actorService, actorId);
    }

    public CompletableFuture addCountAsync(int value)
    {
        return this.stateManager().tryAddStateAsync("MyState", value).thenApply((result)->{
            if(result)
            {
                // Added successfully!
            }
        });
    }
}

```

At the end of an actor method, the state manager automatically saves any values that have been added or modified by an insert or update operation. A "save" can include persisting to disk and replication, depending on the settings used. Values that have not been modified are not persisted or replicated. If no values have been modified, the save operation does nothing. If saving fails, the modified state is discarded and the original state is reloaded.

You can also save state manually by calling the `SaveStateAsync` method on the actor base:

```

async Task IMyActor.SetCountAsync(int count)
{
    await this.StateManager.AddOrUpdateStateAsync("count", count, (key, value) => count > value ? count : value);

    await this.SaveStateAsync();
}

```

```

interface MyActor {
    CompletableFuture setCountAsync(int count)
    {
        this.stateManager().addOrUpdateStateAsync("count", count, (key, value) -> count > value ? count : value).thenApply();

        this.stateManager().saveStateAsync().thenApply();
    }
}

```

## Removing state

You can remove state permanently from an actor's state manager by calling the `Remove` method. This method throws `KeyNotFoundException` (C#) or `NoSuchElementException` (Java) when it tries to remove a key that doesn't exist.

```
[StatePersistence(StatePersistence.Persisted)]
class MyActor : Actor, IMyActor
{
    public MyActor(ActorService actorService, ActorId actorId)
        : base(actorService, actorId)
    {
    }

    public Task RemoveCountAsync()
    {
        return this.StateManager.RemoveStateAsync("MyState");
    }
}
```

```
@StatePersistenceAttribute(statePersistence = StatePersistence.Persisted)
class MyActorImpl extends FabricActor implements MyActor
{
    public MyActorImpl(ActorService actorService, ActorId actorId)
    {
        super(actorService, actorId);
    }

    public CompletableFuture removeCountAsync()
    {
        return this.stateManager().removeStateAsync("MyState");
    }
}
```

You can also remove state permanently by using the *TryRemove* method. This method does not throw when it tries to remove a key that does not exist.

```
[StatePersistence(StatePersistence.Persisted)]
class MyActor : Actor, IMyActor
{
    public MyActor(ActorService actorService, ActorId actorId)
        : base(actorService, actorId)
    {
    }

    public async Task RemoveCountAsync()
    {
        bool result = await this.StateManager.TryRemoveStateAsync("MyState");

        if (result)
        {
            // State removed!
        }
    }
}
```

```
@StatePersistenceAttribute(statePersistence = StatePersistence.Persisted)
class MyActorImpl extends FabricActor implements MyActor
{
    public MyActorImpl(ActorService actorService, ActorId actorId)
    {
        super(actorService, actorId);
    }

    public CompletableFuture removeCountAsync()
    {
        return this.stateManager().tryRemoveStateAsync("MyState").thenApply((result)->{
            if(result)
            {
                // State removed!
            }
        });
    }
}
```

## Next steps

State that's stored in Reliable Actors must be serialized before its written to disk and replicated for high availability.

Learn more about [Actor type serialization](#).

Next, learn more about [Actor diagnostics and performance monitoring](#).

# Polymorphism in the Reliable Actors framework

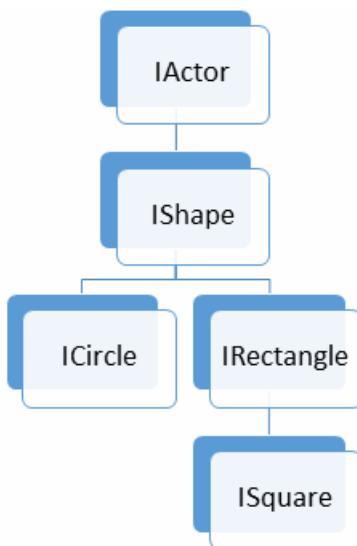
7/6/2017 • 2 min to read • [Edit Online](#)

The Reliable Actors framework allows you to build actors using many of the same techniques that you would use in object-oriented design. One of those techniques is polymorphism, which allows types and interfaces to inherit from more generalized parents. Inheritance in the Reliable Actors framework generally follows the .NET model with a few additional constraints. In case of Java/Linux, it follows the Java model.

## Interfaces

The Reliable Actors framework requires you to define at least one interface to be implemented by your actor type. This interface is used to generate a proxy class that can be used by clients to communicate with your actors.

Interfaces can inherit from other interfaces as long as every interface that is implemented by an actor type and all of its parents ultimately derive from `IActor`(C#) or `Actor`(Java). `IActor`(C#) and `Actor`(Java) are the platform-defined base interfaces for actors in the frameworks .NET and Java respectively. Thus, the classic polymorphism example using shapes might look something like this:



## Types

You can also create a hierarchy of actor types, which are derived from the base `Actor` class that is provided by the platform. In the case of shapes, you might have a base `Shape` (C#) or `ShapeImpl` (Java) type:

```
public abstract class Shape : Actor, IShape
{
    public abstract Task<int> GetVerticeCount();

    public abstract Task<double> GetAreaAsync();
}
```

```
public abstract class ShapeImpl extends FabricActor implements Shape
{
    public abstract CompletableFuture<int> getVerticeCount();

    public abstract CompletableFuture<double> getAreaAsync();
}
```

Subtypes of `Shape` (C#) or `ShapeImpl` (Java) can override methods from the base.

```
[ActorService(Name = "Circle")]
[StatePersistence(StatePersistence.Persisted)]
public class Circle : Shape, ICircle
{
    public override Task<int> GetVerticeCount()
    {
        return Task.FromResult(0);
    }

    public override async Task<double> GetAreaAsync()
    {
        CircleState state = await this.StateManager.GetStateAsync<CircleState>("circle");

        return Math.PI *
            state.Radius *
            state.Radius;
    }
}
```

```
@ActorServiceAttribute(name = "Circle")
@StatePersistenceAttribute(statePersistence = StatePersistence.Persisted)
public class Circle extends ShapeImpl implements Circle
{
    @Override
    public CompletableFuture<Integer> getVerticeCount()
    {
        return CompletableFuture.completedFuture(0);
    }

    @Override
    public CompletableFuture<Double> getAreaAsync()
    {
        return (this.stateManager().getStateAsync<CircleState>("circle").thenApply(state->{
            return Math.PI * state.radius * state.radius;
        }));
    }
}
```

Note the `ActorService` attribute on the actor type. This attribute tells the Reliable Actor framework that it should automatically create a service for hosting actors of this type. In some cases, you may wish to create a base type that is solely intended for sharing functionality with subtypes and will never be used to instantiate concrete actors. In those cases, you should use the `abstract` keyword to indicate that you will never create an actor based on that type.

## Next steps

- See [how the Reliable Actors framework leverages the Service Fabric platform](#) to provide reliability, scalability, and consistent state.
- Learn about the [actor lifecycle](#).

# Reliable Actors reentrancy

6/30/2017 • 1 min to read • [Edit Online](#)

The Reliable Actors runtime, by default, allows logical call context-based reentrancy. This allows for actors to be reentrant if they are in the same call context chain. For example, Actor A sends a message to Actor B, who sends a message to Actor C. As part of the message processing, if Actor C calls Actor A, the message is reentrant, so it will be allowed. Any other messages that are part of a different call context will be blocked on Actor A until it finishes processing.

There are two options available for actor reentrancy defined in the `ActorReentrancyMode` enum:

- `LogicalCallContext` (default behavior)
- `Disallowed` - disables reentrancy

```
public enum ActorReentrancyMode
{
    LogicalCallContext = 1,
    Disallowed = 2
}
```

```
public enum ActorReentrancyMode
{
    LogicalCallContext(1),
    Disallowed(2)
}
```

Reentrancy can be configured in an `ActorService`'s settings during registration. The setting applies to all actor instances created in the actor service.

The following example shows an actor service that sets the reentrancy mode to `ActorReentrancyMode.Disallowed`. In this case, if an actor sends a reentrant message to another actor, an exception of type `FabricException` will be thrown.

```

static class Program
{
    static void Main()
    {
        try
        {
            ActorRuntime.RegisterActorAsync<Actor1>(
                (context, actorType) => new ActorService(
                    context,
                    actorType, () => new Actor1(),
                    settings: new ActorServiceSettings()
                {
                    ActorConcurrencySettings = new ActorConcurrencySettings()
                    {
                        ReentrancyMode = ActorReentrancyMode.Disallowed
                    }
                }))
                .GetAwaiter().GetResult();

            Thread.Sleep(Timeout.Infinite);
        }
        catch (Exception e)
        {
            ActorEventSource.Current.ActorHostInitializationFailed(e.ToString());
            throw;
        }
    }
}

```

```

static class Program
{
    static void Main()
    {
        try
        {
            ActorConcurrencySettings actorConcurrencySettings = new ActorConcurrencySettings();
            actorConcurrencySettings.setReentrancyMode(ActorReentrancyMode.Disallowed);

            ActorServiceSettings actorServiceSettings = new ActorServiceSettings();
            actorServiceSettings.setActorConcurrencySettings(actorConcurrencySettings);

            ActorRuntime.registerActorAsync(
                Actor1.getClass(),
                (context, actorType) -> new FabricActorService(
                    context,
                    actorType, () -> new Actor1(),
                    null,
                    stateProvider,
                    actorServiceSettings, timeout);

            Thread.sleep(Long.MAX_VALUE);
        }
        catch (Exception e)
        {
            throw e;
        }
    }
}

```

## Next steps

- Learn more about reentrancy in the [Actor API reference documentation](#)

# Notes on Service Fabric Reliable Actors type serialization

6/30/2017 • 1 min to read • [Edit Online](#)

The arguments of all methods, result types of the tasks returned by each method in an actor interface, and objects stored in an actor's state manager must be [data contract serializable](#). This also applies to the arguments of the methods defined in [actor event interfaces](#). (Actor event interface methods always return void.)

## Custom data types

In this example, the following actor interface defines a method that returns a custom data type called `VoicemailBox`:

```
public interface IVoiceMailBoxActor : IActor
{
    Task<VoicemailBox> GetMailBoxAsync();
}
```

```
public interface VoiceMailBoxActor extends Actor
{
    CompletableFuture<VoicemailBox> getMailBoxAsync();
}
```

The interface is implemented by an actor that uses the state manager to store a `VoicemailBox` object:

```
[StatePersistence(StatePersistence.Persisted)]
public class VoiceMailBoxActor : Actor, IVoicemailBoxActor
{
    public VoiceMailBoxActor(ActorService actorService, ActorId actorId)
        : base(actorService, actorId)
    {
    }

    public Task<VoicemailBox> GetMailboxAsync()
    {
        return this.StateManager.GetStateAsync<VoicemailBox>("Mailbox");
    }
}
```

```
@StatePersistenceAttribute(statePersistence = StatePersistence.Persisted)
public class VoiceMailBoxActorImpl extends FabricActor implements VoicemailBoxActor
{
    public VoiceMailBoxActorImpl(ActorService actorService, ActorId actorId)
    {
        super(actorService, actorId);
    }

    public CompletableFuture<VoicemailBox> getMailBoxAsync()
    {
        return this.stateManager().getstateAsync("Mailbox");
    }
}
```

In this example, the `VoicemailBox` object is serialized when:

- The object is transmitted between an actor instance and a caller.
- The object is saved in the state manager where it is persisted to disk and replicated to other nodes.

The Reliable Actor framework usesDataContract serialization. Therefore, the custom data objects and their members must be annotated with the **DataContract** and **DataMember** attributes, respectively.

```
[DataContract]
public class Voicemail
{
    [DataMember]
    public Guid Id { get; set; }

    [DataMember]
    public string Message { get; set; }

    [DataMember]
    public DateTime ReceivedAt { get; set; }
}
```

```
public class Voicemail implements Serializable
{
    private static final long serialVersionUID = 42L;

    private UUID id;                      //getUUID() and setUUID()

    private String message;                //getMessage() and setMessage()

    private GregorianCalendar receivedAt; //getReceivedAt() and setReceivedAt()
}
```

```
[DataContract]
public class VoicemailBox
{
    public VoicemailBox()
    {
        this.MessageList = new List<Voicemail>();
    }

    [DataMember]
    public List<Voicemail> MessageList { get; set; }

    [DataMember]
    public string Greeting { get; set; }
}
```

```
public class VoicemailBox implements Serializable
{
    static final long serialVersionUID = 42L;

    public VoicemailBox()
    {
        this.messageList = new ArrayList<Voicemail>();
    }

    private List<Voicemail> messageList;   //getMessageList() and setMessageList()

    private String greeting;                 //getGreeting() and setGreeting()
}
```

## Next steps

- Actor lifecycle and garbage collection
- Actor timers and reminders
- Actor events
- Actor reentrancy
- Actor polymorphism and object-oriented design patterns
- Actor diagnostics and performance monitoring

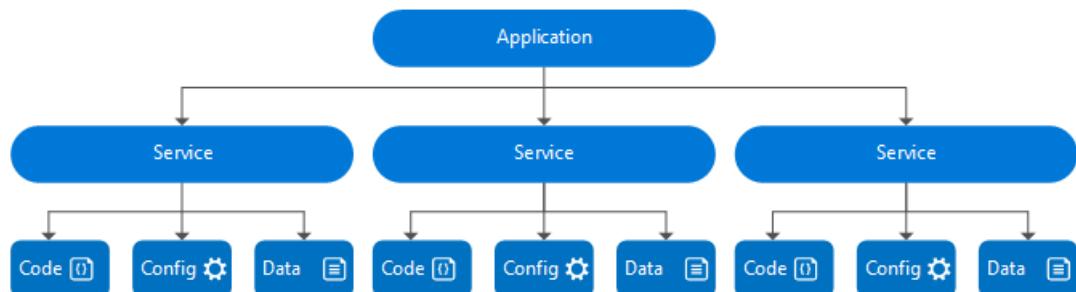
# Model an application in Service Fabric

8/15/2017 • 7 min to read • [Edit Online](#)

This article provides an overview of the Azure Service Fabric application model and how to define an application and service via manifest files.

## Understand the application model

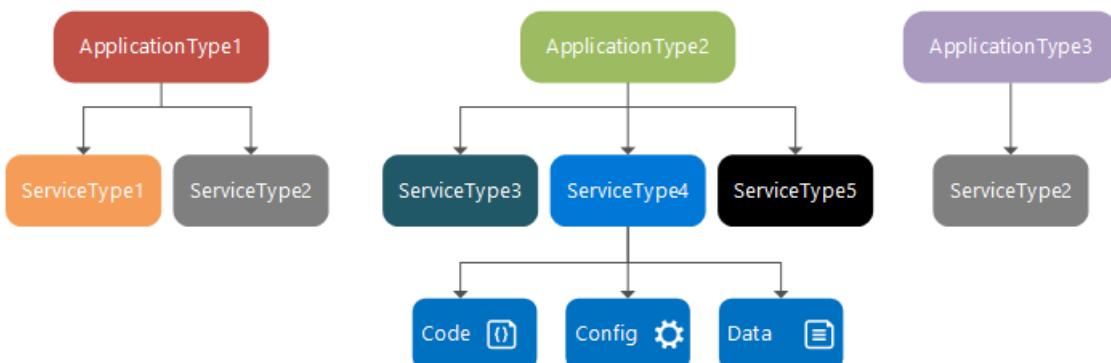
An application is a collection of constituent services that perform a certain function or functions. A service performs a complete and standalone function and can start and run independently of other services. A service is composed of code, configuration, and data. For each service, code consists of the executable binaries, configuration consists of service settings that can be loaded at run time, and data consists of arbitrary static data to be consumed by the service. Each component in this hierarchical application model can be versioned and upgraded independently.



An application type is a categorization of an application and consists of a bundle of service types. A service type is a categorization of a service. The categorization can have different settings and configurations, but the core functionality remains the same. The instances of a service are the different service configuration variations of the same service type.

Classes (or "types") of applications and services are described through XML files (application manifests and service manifests). The manifests are the templates against which applications can be instantiated from the cluster's image store. The schema definition for the ServiceManifest.xml and ApplicationManifest.xml file is installed with the Service Fabric SDK and tools to `C:\Program Files\Microsoft SDKs\Service Fabric\schemas\ServiceFabricServiceModel.xsd`.

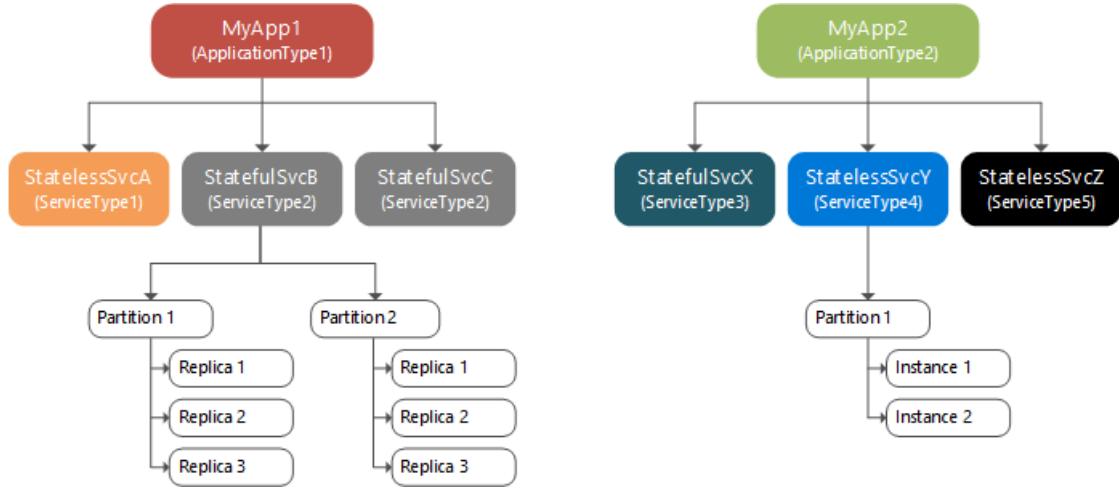
The code for different application instances run as separate processes even when hosted by the same Service Fabric node. Furthermore, the lifecycle of each application instance can be managed (for example, upgraded) independently. The following diagram shows how application types are composed of service types, which in turn are composed of code, configuration, and data packages. To simplify the diagram, only the code/config/data packages for `ServiceType4` are shown, though each service type would include some or all those package types.



Two different manifest files are used to describe applications and services: the service manifest and application manifest. Manifests are covered in detail in the following sections.

There can be one or more instances of a service type active in the cluster. For example, stateful service instances, or replicas, achieve high reliability by replicating state between replicas located on different nodes in the cluster. Replication essentially provides redundancy for the service to be available even if one node in a cluster fails. A [partitioned service](#) further divides its state (and access patterns to that state) across nodes in the cluster.

The following diagram shows the relationship between applications and service instances, partitions, and replicas.



**TIP**

You can view the layout of applications in a cluster using the Service Fabric Explorer tool available at <http://<yourclusteraddress>:19080/Explorer>. For more information, see [Visualizing your cluster with Service Fabric Explorer](#).

## Describe a service

The service manifest declaratively defines the service type and version. It specifies service metadata such as service type, health properties, load-balancing metrics, service binaries, and configuration files. Put another way, it describes the code, configuration, and data packages that compose a service package to support one or more service types. Here is a simple example service manifest:

```

<?xml version="1.0" encoding="utf-8" ?>
<ServiceManifest Name="MyServiceManifest" Version="SvcManifestVersion1"
xmlns="http://schemas.microsoft.com/2011/01/fabric" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <Description>An example service manifest</Description>
  <ServiceTypes>
    <StatelessServiceType ServiceTypeName="MyServiceType" />
  </ServiceTypes>
  <CodePackage Name="MyCode" Version="CodeVersion1">
    <SetupEntryPoint>
      <ExeHost>
        <Program>MySetup.bat</Program>
      </ExeHost>
    </SetupEntryPoint>
    <EntryPoint>
      <ExeHost>
        <Program>MyServiceHost.exe</Program>
      </ExeHost>
    </EntryPoint>
    <EnvironmentVariables>
      <EnvironmentVariable Name="MyEnvVariable" Value="" />
      <EnvironmentVariable Name="HttpGatewayPort" Value="19080" />
    </EnvironmentVariables>
  </CodePackage>
  <ConfigPackage Name="MyConfig" Version="ConfigVersion1" />
  <DataPackage Name="MyData" Version="DataVersion1" />
</ServiceManifest>

```

**Version** attributes are unstructured strings and not parsed by the system. Version attributes are used to version each component for upgrades.

**ServiceTypes** declares what service types are supported by **CodePackages** in this manifest. When a service is instantiated against one of these service types, all code packages declared in this manifest are activated by running their entry points. The resulting processes are expected to register the supported service types at run time. Service types are declared at the manifest level and not the code package level. So when there are multiple code packages, they are all activated whenever the system looks for any one of the declared service types.

**SetupEntryPoint** is a privileged entry point that runs with the same credentials as Service Fabric (typically the *LocalSystem* account) before any other entry point. The executable specified by **EntryPoint** is typically the long-running service host. The presence of a separate setup entry point avoids having to run the service host with high privileges for extended periods of time. The executable specified by **EntryPoint** is run after **SetupEntryPoint** exits successfully. If the process ever terminates or crashes, the resulting process is monitored and restarted (beginning again with **SetupEntryPoint**).

Typical scenarios for using **SetupEntryPoint** are when you run an executable before the service starts or you perform an operation with elevated privileges. For example:

- Setting up and initializing environment variables that the service executable needs. This is not limited to only executables written via the Service Fabric programming models. For example, npm.exe needs some environment variables configured for deploying a node.js application.
- Setting up access control by installing security certificates.

For more details on how to configure the **SetupEntryPoint** see [Configure the policy for a service setup entry point](#)

**EnvironmentVariables** provides a list of environment variables that are set for this code package.

Environment variables can be overridden in the `ApplicationManifest.xml` to provide different values for different service instances.

**DataPackage** declares a folder, named by the **Name** attribute, that contains arbitrary static data to be consumed by the process at run time.

**ConfigPackage** declares a folder, named by the **Name** attribute, that contains a *Settings.xml* file. The settings file contains sections of user-defined, key-value pair settings that the process reads back at run time. During an upgrade, if only the **ConfigPackage version** has changed, then the running process is not restarted. Instead, a callback notifies the process that configuration settings have changed so they can be reloaded dynamically. Here is an example *Settings.xml* file:

```
<Settings xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.microsoft.com/2011/01/fabric">
  <Section Name="MyConfigurationSection">
    <Parameter Name="MySettingA" Value="Example1" />
    <Parameter Name="MySettingB" Value="Example2" />
  </Section>
</Settings>
```

#### NOTE

A service manifest can contain multiple code, configuration, and data packages. Each of those can be versioned independently.

## Describe an application

The application manifest declaratively describes the application type and version. It specifies service composition metadata such as stable names, partitioning scheme, instance count/replication factor, security/isolation policy, placement constraints, configuration overrides, and constituent service types. The load-balancing domains into which the application is placed are also described.

Thus, an application manifest describes elements at the application level and references one or more service manifests to compose an application type. Here is a simple example application manifest:

```
<?xml version="1.0" encoding="utf-8" ?>
<ApplicationManifest
  ApplicationTypeName="MyApplicationType"
  ApplicationTypeVersion="AppManifestVersion1"
  xmlns="http://schemas.microsoft.com/2011/01/fabric"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Description>An example application manifest</Description>
  <ServiceManifestImport>
    <ServiceManifestRef ServiceManifestName="MyServiceManifest"
      ServiceManifestVersion="SvcManifestVersion1"/>
    <ConfigOverrides/>
    <EnvironmentOverrides CodePackageRef="MyCode"/>
  </ServiceManifestImport>
  <DefaultServices>
    <Service Name="MyService">
      <StatelessService ServiceTypeName="MyServiceType" InstanceCount="1">
        <SingletonPartition/>
      </StatelessService>
    </Service>
  </DefaultServices>
</ApplicationManifest>
```

Like service manifests, **Version** attributes are unstructured strings and are not parsed by the system. Version attributes are also used to version each component for upgrades.

**ServiceManifestImport** contains references to service manifests that compose this application type.

Imported service manifests determine what service types are valid within this application type. Within the ServiceManifestImport, you override configuration values in Settings.xml and environment variables in ServiceManifest.xml files.

**DefaultServices** declares service instances that are automatically created whenever an application is instantiated against this application type. Default services are just a convenience and behave like normal services in every respect after they have been created. They are upgraded along with any other services in the application instance and can be removed as well.

**NOTE**

An application manifest can contain multiple service manifest imports and default services. Each service manifest import can be versioned independently.

To learn how to maintain different application and service parameters for individual environments, see [Managing application parameters for multiple environments](#).

## Next steps

[Package an application](#) and make it ready to deploy.

[Deploy and remove applications](#) describes how to use PowerShell to manage application instances.

[Managing application parameters for multiple environments](#) describes how to configure parameters and environment variables for different application instances.

[Configure security policies for your application](#) describes how to run services under security policies to restrict access.

[Application hosting models](#) describe relationship between replicas (or instances) of a deployed service and service-host process.

# Service Fabric hosting model

10/24/2017 • 10 min to read • [Edit Online](#)

This article provides an overview of application hosting models provided by Service Fabric, and describes the differences between the **Shared Process** and **Exclusive Process** models. It describes how a deployed application looks on a Service Fabric node and relationship between replicas (or instances) of the service and the service-host process.

Before proceeding further, make sure that you are familiar with [Service Fabric Application Model](#) and understand various concepts and relation among them.

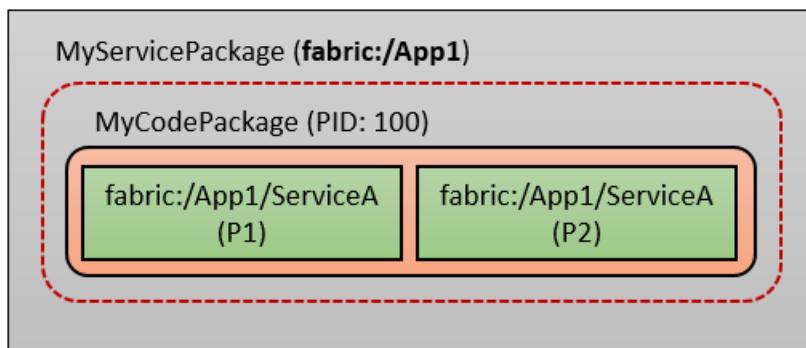
## NOTE

In this article, for simplicity, unless explicitly mentioned:

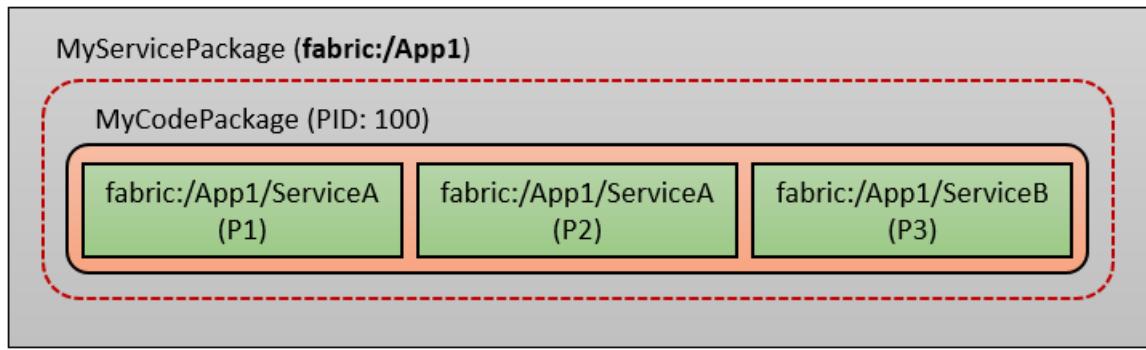
- All uses of word *replica* refers to both a replica of a stateful service or an instance of a stateless service.
- *CodePackage* is treated equivalent to *ServiceHost* process that registers a *ServiceType* and hosts replicas of services of that *ServiceType*.

To understand the hosting model, let us walk through an example. Let us say, we have an *ApplicationType* 'MyAppType' which has a *ServiceType* 'MyServiceType' which is provided by *ServicePackage* 'MyServicePackage' which has a *CodePackage* 'MyCodePackage' which registers *ServiceType* 'MyServiceType' when it runs.

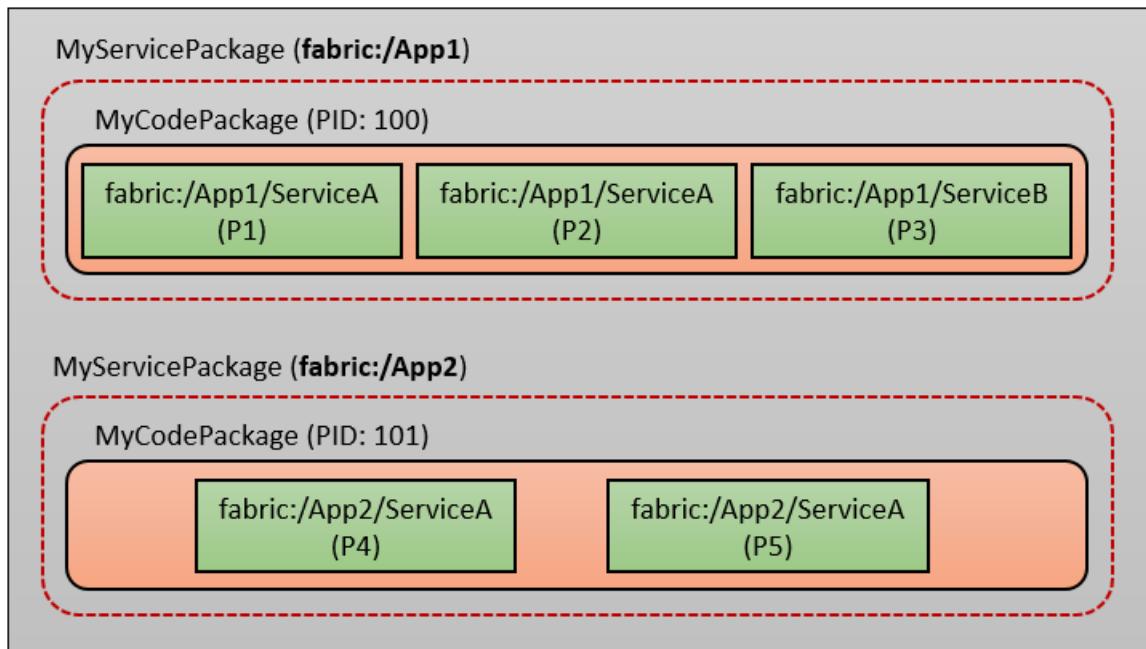
Let's say we have a 3 node cluster and we create an *application fabric:/App1* of type 'MyAppType'. Inside this *application fabric:/App1* we create a service **fabric:/App1/ServiceA** of type 'MyServiceType' which has 2 partitions (say **P1 & P2**) and 3 replicas per partition. The following diagram shows the view of this application as it ends up deployed on a node.



Service Fabric activated 'MyServicePackage' which started 'MyCodePackage' which is hosting replicas from both the partitions i.e. **P1 & P2**. Note that all the nodes in the cluster will have same view since we chose number of replicas per partition equal to number of nodes in the cluster. Let's create another service **fabric:/App1/ServiceB** in application **fabric:/App1** which has 1 partition (say **P3**) and 3 replicas per partition. Following diagram shows the new view on the node:



As we can see Service Fabric placed the new replica for partition **P3** of service **fabric:/App1/ServiceB** in the existing activation of 'MyServicePackage'. Now lets create another *application* **fabric:/App2** of type 'MyAppType' and inside **fabric:/App2** create service **fabric:/App2/ServiceA** which has 2 partitions (say **P4** & **P5**) and 3 replicas per partition. Following diagrams shows the new node view:



This time Service Fabric has activated a new copy of 'MyServicePackage' which starts a new copy of 'MyCodePackage' and replicas from both partitions of service **fabric:/App2/ServiceA** (i.e. **P4** & **P5**) are placed in this new copy 'MyCodePackage'.

## Shared process model

What we saw above is the default hosting model provided by Service Fabric and is referred to as **Shared Process** model. In this model, for a given *application*, only one copy of a given *ServicePackage* is activated on a *Node* (which starts all the *CodePackages* contained in it) and all the replicas of all services of a given *ServiceType* are placed in the *CodePackage* that registers that *ServiceType*. In other words, all the replicas of all services on a node of a given *ServiceType* share the same process.

## Exclusive process model

The other hosting model provided by Service Fabric is **Exclusive Process** model. In this model, on a given *Node*, for placing each replica, Service Fabric activates a new copy of *ServicePackage* (which starts all the *CodePackages* contained in it) and replica is placed in the *CodePackage* that registered the *ServiceType* of the service to which replica belongs. In other words, each replica lives in its own dedicated process.

This model is supported starting version 5.6 of Service Fabric. **Exclusive Process** model can be chosen at the time of creating the service (using [PowerShell](#), [REST](#) or [FabricClient](#)) by specifying **ServicePackageActivationMode** as 'ExclusiveProcess'.

```
PS C:\>New-ServiceFabricService -ApplicationName "fabric:/App1" -ServiceName "fabric:/App1/ServiceA" -ServiceTypeName "MyServiceType" -Stateless -PartitionSchemeSingleton -InstanceCount -1 -ServicePackageActivationMode "ExclusiveProcess"
```

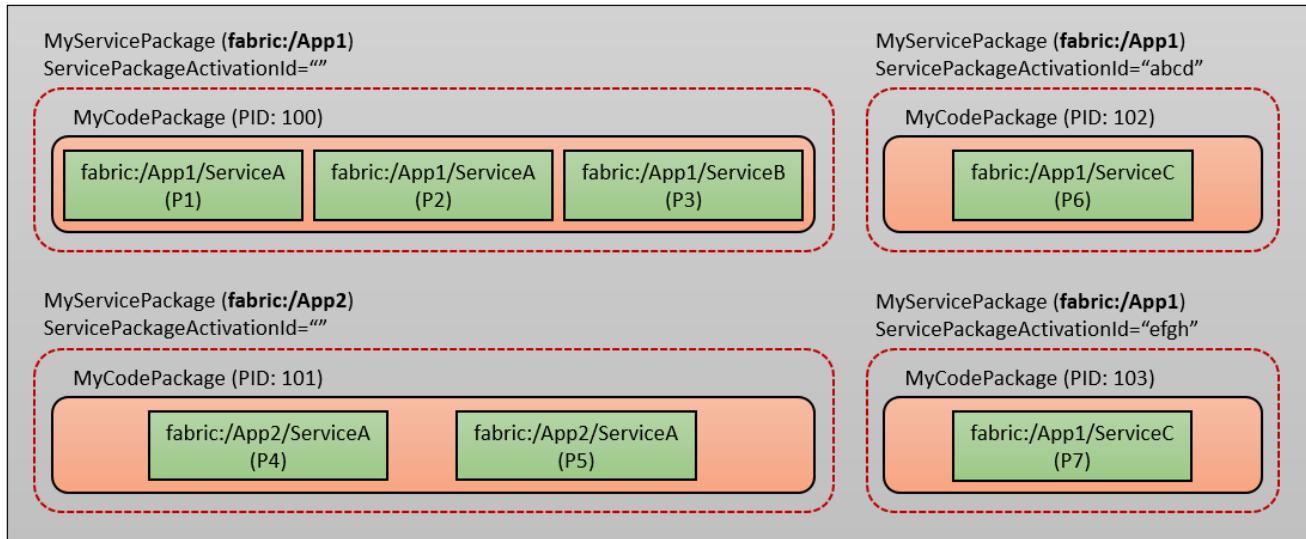
```
var serviceDescription = new StatelessServiceDescription
{
    ApplicationName = new Uri("fabric:/App1"),
    ServiceName = new Uri("fabric:/App1/ServiceA"),
    ServiceTypeName = "MyServiceType",
    PartitionSchemeDescription = new SingletonPartitionSchemeDescription(),
    InstanceCount = -1,
    ServicePackageActivationMode = ServicePackageActivationMode.ExclusiveProcess
};

var fabricClient = new FabricClient(clusterEndpoints);
await fabricClient.ServiceManager.CreateServiceAsync(serviceDescription);
```

If you have a default service in your application manifest, you can choose **Exclusive Process** model by specifying **ServicePackageActivationMode** attribute as shown below:

```
<DefaultServices>
  <Service Name="MyService" ServicePackageActivationMode="ExclusiveProcess">
    <StatelessService ServiceTypeName="MyServiceType" InstanceCount="1">
      <SingletonPartition/>
    </StatelessService>
  </Service>
</DefaultServices>
```

Continuing with our example above, lets create another service **fabric:/App1/ServiceC** in application **fabric:/App1** which has 2 partitions (say **P6** & **P7**) and 3 replicas per partition with **ServicePackageActivationMode** set to 'ExclusiveProcess'. Following diagram shows new view on the node:



As you can see, Service Fabric activated two new copies of 'MyServicePackage' (one for each replica from partition **P6** & **P7**) and placed each replica in its dedicated copy of *CodePackage*. Another thing to note here is, when **Exclusive Process** model is used, for a given *application*, multiple copies of a given *ServicePackage* can be active on a *Node*. In above example, we see that three copies of 'MyServicePackage' are active for **fabric:/App1**. Each of these active copies of 'MyServicePackage' has a **ServicePackageActivationId** associated with it which identifies that copy within *application* **fabric:/App1**.

When only **Shared Process** model is used for an *application*, like **fabric:/App2** in above example, there is only one active copy of *ServicePackage* on a *Node* and **ServicePackageActivationId** for this activation of

*ServicePackage* is 'empty string'.

#### NOTE

- **Shared Process** hosting model corresponds to **ServicePackageActivationMode** equal **SharedProcess**. This is the default hosting model and **ServicePackageActivationMode** need not be specified at the time of creating the service.
- **Exclusive Process** hosting model corresponds to **ServicePackageActivationMode** equal **ExclusiveProcess** and need to be explicitly specified at the time of creating the service.
- Hosting model of a service can be known by querying the [service description](#) and looking at value of **ServicePackageActivationMode**.

## Working with deployed service package

An active copy of a *ServicePackage* on a node is referred as [deployed service package](#). As previously mentioned above, when **Exclusive Process** model is used for creating services, for a given *application*, there could be multiple deployed service packages for the same *ServicePackage*. While performing operations specific to deployed service package like [reporting health of a deployed service package](#) or [restarting code package of a deployed service package](#) etc., **ServicePackageActivationId** needs to be provided to identify a specific deployed service package.

**ServicePackageActivationId** of a deployed service package can be obtained by querying the list of [deployed service packages](#) on a node. When querying [deployed service types](#), [deployed replicas](#) and [deployed code packages](#) on a node, the query result also contains the **ServicePackageActivationId** of parent deployed service package.

#### NOTE

- Under **Shared Process** hosting model, on a given *node*, for a given *application*, only one copy of a *ServicePackage* is activated. It has **ServicePackageActivationId** equal to *empty string* and need not be specified while performing deployed service package related operations.
- Under **Exclusive Process** hosting model, on a given *node*, for a given *application*, one or more copies of a *ServicePackage* can be active. Each activation has a *non-empty* **ServicePackageActivationId** and needs to be specified while performing deployed service package related operations.
- If **ServicePackageActivationId** is omitted it defaults to 'empty string'. If a deployed service package that was activated under **Shared Process** model is present, then operation will be performed on it, otherwise the operation will fail.
- It is not recommended to query once and cache **ServicePackageActivationId** as it is dynamically generated and can change for various reasons. Before performing an operation that needs **ServicePackageActivationId**, you should first query the list of [deployed service packages](#) on a node and then use **ServicePackageActivationId** from query result to perform the original operation.

## Guest executable and container applications

Service Fabric treats [Guest executable](#) and [container](#) applications as stateless services which are self-contained i.e. there is no Service Fabric runtime in *ServiceHost* (a process or container). Since these services are self-contained, number of replicas per *ServiceHost* is not applicable for these services. The most common configuration used with these services is single-partition with [InstanceCount](#) equal to -1 (i.e. one copy of the service code running on each node of cluster).

The default **ServicePackageActivationMode** for these services is **SharedProcess** in which case Service Fabric

only activates one copy of *ServicePackage* on a *Node* for a given *application* which means only one copy of service code will run a *Node*. If you want multiple copies of your service code to run on a *Node* when you create multiple services (*Service1* to *ServiceN*) of *ServiceType* (specified in *ServiceManifest*) or when your service is multi-partitioned, you should specify **ServicePackageActivationMode** as **ExclusiveProcess** at the time of creating the service.

## Changing hosting model of an existing service

Changing hosting model of an existing service from **Shared Process** to **Exclusive Process** and vice-versa through upgrade or update mechanism (or in default service specification in application manifest) is currently not supported. Support for this feature will come in future versions.

## Choosing between shared process and exclusive process model

Both these hosting models have its pros and cons and user needs to evaluate which one fits their requirements best. **Shared Process** model enables better utilization of OS resources because fewer processes are spawned, multiple replicas in the same process can share ports, etc. However, if one of the replicas hits an error where it needs to bring down the service host, it will impact all other replicas in same process.

**Exclusive Process** model provides better isolation with every replica in its own process and a misbehaving replica will not impact other replicas. It comes in handy for cases where port sharing is not supported by the communication protocol. It facilitates the ability to apply resource governance at replica level. On the other hand, **Exclusive Process** will consume more OS resources as it spawns one process for each replica on the node.

## Exclusive process model and application model considerations

The recommended way to model your application in Service Fabric is to keep one *ServiceType* per *ServicePackage* and this model works well for most of the applications.

Intended for certain use cases, Service Fabric also allows more than one *ServiceType* per *ServicePackage* (and one *CodePackage* can register more than one *ServiceType*). The following are some of the scenarios where these configurations can be useful:

- You want to optimize OS resource utilization by spawning fewer processes and having higher replica density per process.
- Replicas from different *ServiceTypes* need to share some common data that has a high initialization or memory cost.
- You have a free service offering and you want to put a limit on resource utilization by putting all replicas of the service in same process.

**Exclusive Process** hosting model is not coherent with application model having multiple *ServiceTypes* per *ServicePackage*. This is because multiple *ServiceTypes* per *ServicePackage* is designed to achieve higher resource sharing among replicas and enables higher replica density per process. This is contrary to what **Exclusive Process** model is designed to achieve.

Consider the case of multiple *ServiceTypes* per *ServicePackage* with different *CodePackage* registering each *ServiceType*. Let's say we have a *ServicePackage* 'MultiTypeServicePackge' which has two *CodePackages*:

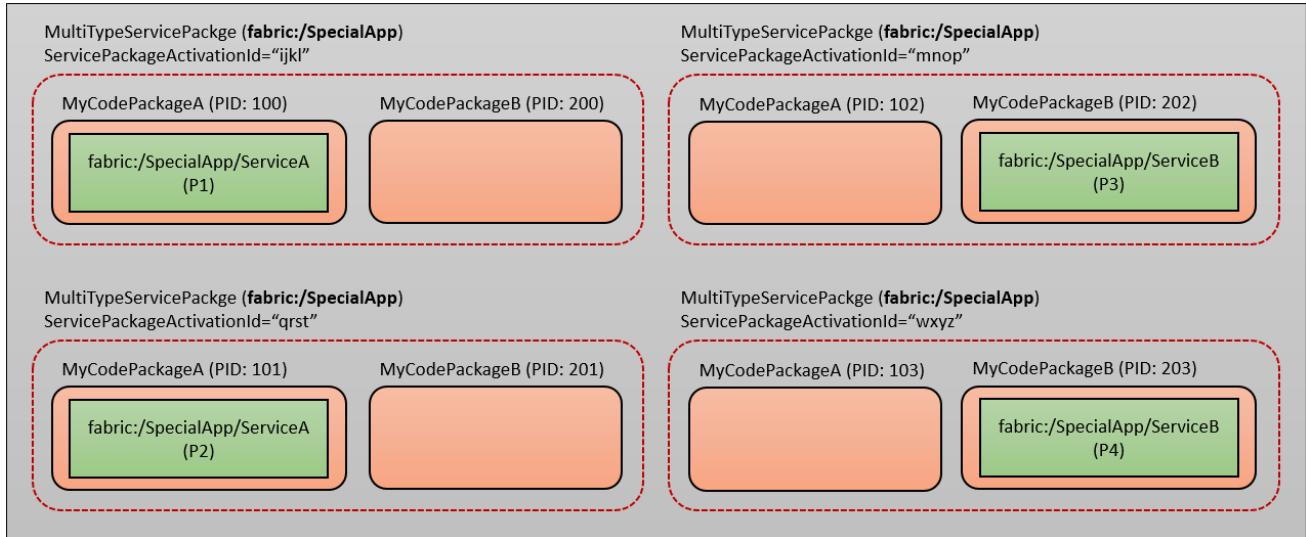
- 'MyCodePackageA' which registers *ServiceType* 'MyServiceTypeA'.
- 'MyCodePackageB' which registers *ServiceType* 'MyServiceTypeB'.

Now, lets say, we create an *application* **fabric:/SpecialApp** and inside **fabric:/SpecialApp** we create following two services with **Exclusive Process** model:

- Service **fabric:/SpecialApp/ServiceA** of type 'MyServiceTypeA' with two partitions (say **P1** and **P2**) and 3 replicas per partition.

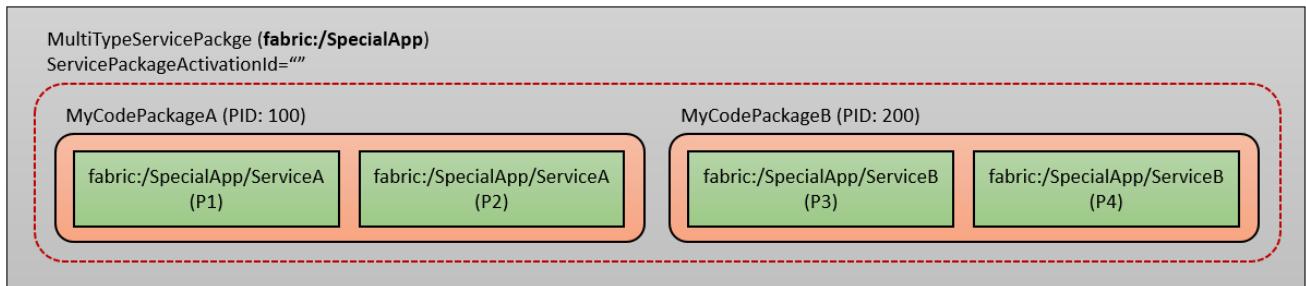
- Service **fabric:/SpecialApp/ServiceB** of type 'MyServiceTypeB' with two partitions (say **P3** and **P4**) and 3 replicas per partition.

On a given node, both the services will have two replicas each. Since we used **Exclusive Process** model to create the services, Service Fabric will activate a new copy of 'MultiTypeServicePackge' for each replica. Each activation of 'MultiTypeServicePackge' will start a copy of 'MyCodePackageA' and 'MyCodePackageB'. However, only one of 'MyCodePackageA' or 'MyCodePackageB' will host the replica for which 'MultiTypeServicePackge' was activated. Following diagram shows the node view:



As we can see, in the activation of 'MultiTypeServicePackge' for replica of partition **P1** of service **fabric:/SpecialApp/ServiceA**, 'MyCodePackageA' is hosting the replica and 'MyCodePackageB' is just up and running. Similarly, in activation of 'MultiTypeServicePackge' for replica of partition **P3** of service **fabric:/SpecialApp/ServiceB**, 'MyCodePackageB' is hosting the replica and 'MyCodePackageA' is just up and running and so on. Hence, more the number of *CodePackages* (registering different *ServiceTypes*) per *ServicePackage*, higher will be redundant resource usage.

On the other hand if we create services **fabric:/SpecialApp/ServiceA** and **fabric:/SpecialApp/ServiceB** with **Shared Process** model, Service Fabric will activate only one copy of 'MultiTypeServicePackge' for *application fabric:/SpecialApp* (as we saw previously). 'MyCodePackageA' will host all replicas for service **fabric:/SpecialApp/ServiceA** (or of any service of type 'MyServiceTypeA' to be more precise) and 'MyCodePackageB' will host all replicas for service **fabric:/SpecialApp/ServiceB** (or of any service of type 'MyServiceTypeB' to be more precise). Following diagram shows the node view in this setting:



In the above example, you might think if 'MyCodePackageA' registers both 'MyServiceTypeA' and 'MyServiceTypeB' and there is no 'MyCodePackageB', then there will be no redundant *CodePackage* running. This is correct, however, as mentioned previously, this application model does not align with **Exclusive Process** hosting model. If goal is to put each replica in its own dedicated process then registering both *ServiceTypes* from same *CodePackage* is not needed and putting each *ServiceType* in its own *ServicePacakge* is a more natural choice.

## Next steps

[Package an application](#) and get it ready to deploy.

[Deploy and remove applications](#) describes how to use PowerShell to manage application instances.

# Specify resources in a service manifest

9/22/2017 • 4 min to read • [Edit Online](#)

## Overview

The service manifest allows resources that are used by the service to be declared/changed without changing the compiled code. Azure Service Fabric supports configuration of endpoint resources for the service. The access to the resources that are specified in the service manifest can be controlled via the `SecurityGroup` in the application manifest. The declaration of resources allows these resources to be changed at deployment time, meaning the service doesn't need to introduce a new configuration mechanism. The schema definition for the `ServiceManifest.xml` file is installed with the Service Fabric SDK and tools to `C:\Program Files\Microsoft SDKs\Service Fabric\schemas\ServiceFabricServiceModel.xsd`.

## Endpoints

When an endpoint resource is defined in the service manifest, Service Fabric assigns ports from the reserved application port range when a port isn't specified explicitly. For example, look at the endpoint `ServiceEndpoint1` specified in the manifest snippet provided after this paragraph. Additionally, services can also request a specific port in a resource. Service replicas running on different cluster nodes can be assigned different port numbers, while replicas of a service running on the same node share the port. The service replicas can then use these ports as needed for replication and listening for client requests.

```
<Resources>
  <Endpoints>
    <Endpoint Name="ServiceEndpoint1" Protocol="http"/>
    <Endpoint Name="ServiceEndpoint2" Protocol="http" Port="80"/>
    <Endpoint Name="ServiceEndpoint3" Protocol="https"/>
  </Endpoints>
</Resources>
```

Refer to [Configuring stateful Reliable Services](#) to read more about referencing endpoints from the config package settings file (`settings.xml`).

## Example: specifying an HTTP endpoint for your service

The following service manifest defines one TCP endpoint resource and two HTTP endpoint resources in the `<Resources>` element.

HTTP endpoints are automatically ACL'd by Service Fabric.

```

<?xml version="1.0" encoding="utf-8"?>
<ServiceManifest Name="Stateful1Pkg"
    Version="1.0.0"
    xmlns="http://schemas.microsoft.com/2011/01/fabric"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<ServiceTypes>
    <!-- This is the name of your ServiceType.
        This name must match the string used in the RegisterServiceType call in Program.cs. -->
    <StatefulServiceType ServiceTypeName="Stateful1Type" HasPersistedState="true" />
</ServiceTypes>

<!-- Code package is your service executable. -->
<CodePackage Name="Code" Version="1.0.0">
    <EntryPoint>
        <ExeHost>
            <Program>Stateful1.exe</Program>
        </ExeHost>
    </EntryPoint>
</CodePackage>

<!-- Config package is the contents of the Config directory under PackageRoot that contains an
    independently updateable and versioned set of custom configuration settings for your service. -->
<ConfigPackage Name="Config" Version="1.0.0" />

<Resources>
    <Endpoints>
        <!-- This endpoint is used by the communication listener to obtain the port number on which to
            listen. Note that if your service is partitioned, this port is shared with
            replicas of different partitions that are placed in your code. -->
        <Endpoint Name="ServiceEndpoint1" Protocol="http"/>
        <Endpoint Name="ServiceEndpoint2" Protocol="http" Port="80"/>
        <Endpoint Name="ServiceEndpoint3" Protocol="https"/>

        <!-- This endpoint is used by the replicator for replicating the state of your service.
            This endpoint is configured through the ReplicatorSettings config section in the Settings.xml
            file under the ConfigPackage. -->
        <Endpoint Name="ReplicatorEndpoint" />
    </Endpoints>
</Resources>
</ServiceManifest>

```

## Example: specifying an HTTPS endpoint for your service

The HTTPS protocol provides server authentication and is also used for encrypting client-server communication. To enable HTTPS on your Service Fabric service, specify the protocol in the *Resources -> Endpoints -> Endpoint* section of the service manifest, as shown earlier for the endpoint *ServiceEndpoint3*.

### NOTE

A service's protocol cannot be changed during application upgrade. If it is changed during upgrade, it is a breaking change.

Here is an example ApplicationManifest that you need to set for HTTPS. The thumbprint for your certificate must be provided. The EndpointRef is a reference to EndpointResource in ServiceManifest, for which you set the HTTPS protocol. You can add more than one EndpointCertificate.

For Linux clusters, the **MY** store defaults to the folder **/var/lib/sfcerts**.

## Overriding Endpoints in ServiceManifest.xml

In the ApplicationManifest add a ResourceOverrides section which will be a sibling to ConfigOverrides section. In this section you can specify the override for the Endpoints section in the resources section specified in the Service manifest.

In order to override EndPoint in ServiceManifest using ApplicationParameters change the ApplicationManifest as following:

In the ServiceManifestImport section add a new section "ResourceOverrides"

```

<ServiceManifestImport>
    <ServiceManifestRef ServiceManifestName="Stateless1Pkg" ServiceManifestVersion="1.0.0" />
    <ConfigOverrides />
    <ResourceOverrides>
        <Endpoints>
            <Endpoint Name="ServiceEndpoint" Port="[Port]" Protocol="[Protocol]" Type="[Type]" />
            <Endpoint Name="ServiceEndpoint1" Port="[Port1]" Protocol="[Protocol1]" />
        </Endpoints>
    </ResourceOverrides>
    <Policies>
        <EndpointBindingPolicy CertificateRef="TestCert1" EndpointRef="ServiceEndpoint"/>
    </Policies>
</ServiceManifestImport>

```

In the Parameters add below:

```

<Parameters>
    <Parameter Name="Port" DefaultValue="" />
    <Parameter Name="Protocol" DefaultValue="" />
    <Parameter Name="Type" DefaultValue="" />
    <Parameter Name="Port1" DefaultValue="" />
    <Parameter Name="Protocol1" DefaultValue="" />
</Parameters>

```

While deploying the application now you can pass in these values as ApplicationParameters for example:

```

PS C:\> New-ServiceFabricApplication -ApplicationName fabric:/myapp -ApplicationTypeName "AppType" -ApplicationTypeVersion "1.0.0" -ApplicationParameter @{Port='1001'; Protocol='https'; Type='Input'; Port1='2001'; Protocol='http'}

```

Note: If the values provided for the ApplicationParameters is empty we go back to the default value provided in the ServiceManifest for the corresponding EndPointName.

For example:

If in the ServiceManifest you specified

```

<Resources>
    <Endpoints>
        <Endpoint Name="ServiceEndpoint1" Protocol="tcp"/>
    </Endpoints>
</Resources>

```

And the Port1 and Protocol1 value for Application parameters is null or empty. The port is still decided by ServiceFabric. And the Protocol will be tcp.

Suppose you specify a wrong value. Like for Port you specified a string value "Foo" instead of an int. New-ServiceFabricApplication command will fail with an error : The override parameter with name 'ServiceEndpoint1' attribute 'Port1' in section 'ResourceOverrides' is invalid. The value specified is 'Foo' and required is 'int'.

# Service state

8/18/2017 • 1 min to read • [Edit Online](#)

**Service state** refers to the in-memory or on disk data that a service requires to function. It includes, for example, the data structures and member variables that the service reads and writes to do work. Depending on how the service is architected, it could also include files or other resources that are stored on disk. For example, the files a database would use to store data and transaction logs.

As an example service, let's consider a calculator. A basic calculator service takes two numbers and returns their sum. Performing this calculation involves no member variables or other information.

Now consider the same calculator, but with an additional method for storing and returning the last sum it has computed. This service is now stateful. Stateful means it contains some state that it writes to when it computes a new sum and reads from when you ask it to return the last computed sum.

In Azure Service Fabric, the first service is called a stateless service. The second service is called a stateful service.

## Storing service state

State can be either externalized or co-located with the code that is manipulating the state. Externalization of state is typically done by using an external database or other data store that runs on different machines over the network or out of process on the same machine. In our calculator example, the data store could be a SQL database or instance of Azure Table Store. Every request to compute the sum performs an update on this data, and requests to the service to return the value result in the current value being fetched from the store.

State can also be co-located with the code that manipulates the state. Stateful services in Service Fabric are typically built using this model. Service Fabric provides the infrastructure to ensure that this state is highly available, consistent, and durable, and that the services built this way can easily scale.

## Next steps

For more information on Service Fabric concepts, see the following articles:

- [Availability of Service Fabric services](#)
- [Scalability of Service Fabric services](#)
- [Partitioning Service Fabric services](#)
- [Service Fabric Reliable Services](#)

# Partition Service Fabric reliable services

7/5/2017 • 15 min to read • [Edit Online](#)

This article provides an introduction to the basic concepts of partitioning Azure Service Fabric reliable services. The source code used in the article is also available on [GitHub](#).

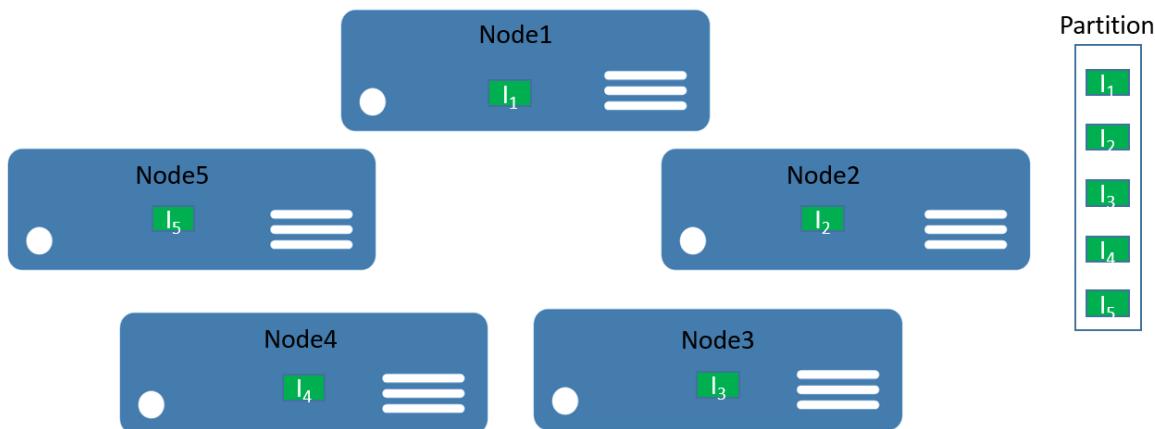
## Partitioning

Partitioning is not unique to Service Fabric. In fact, it is a core pattern of building scalable services. In a broader sense, we can think about partitioning as a concept of dividing state (data) and compute into smaller accessible units to improve scalability and performance. A well-known form of partitioning is [data partitioning](#), also known as sharding.

### Partition Service Fabric stateless services

For stateless services, you can think about a partition being a logical unit that contains one or more instances of a service. Figure 1 shows a stateless service with five instances distributed across a cluster using one partition.

### Five instances of a stateless service across a cluster



There are really two types of stateless service solutions. The first one is a service that persists its state externally, for example in an Azure SQL database (like a website that stores the session information and data). The second one is computation-only services (like a calculator or image thumbnailing) that do not manage any persistent state.

In either case, partitioning a stateless service is a very rare scenario--scalability and availability are normally achieved by adding more instances. The only time you want to consider multiple partitions for stateless service instances is when you need to meet special routing requests.

As an example, consider a case where users with IDs in a certain range should only be served by a particular service instance. Another example of when you could partition a stateless service is when you have a truly partitioned backend (e.g. a sharded SQL database) and you want to control which service instance should write to the database shard--or perform other preparation work within the stateless service that requires the same partitioning information as is used in the backend. Those types of scenarios can also be solved in different ways and do not necessarily require service partitioning.

The remainder of this walkthrough focuses on stateful services.

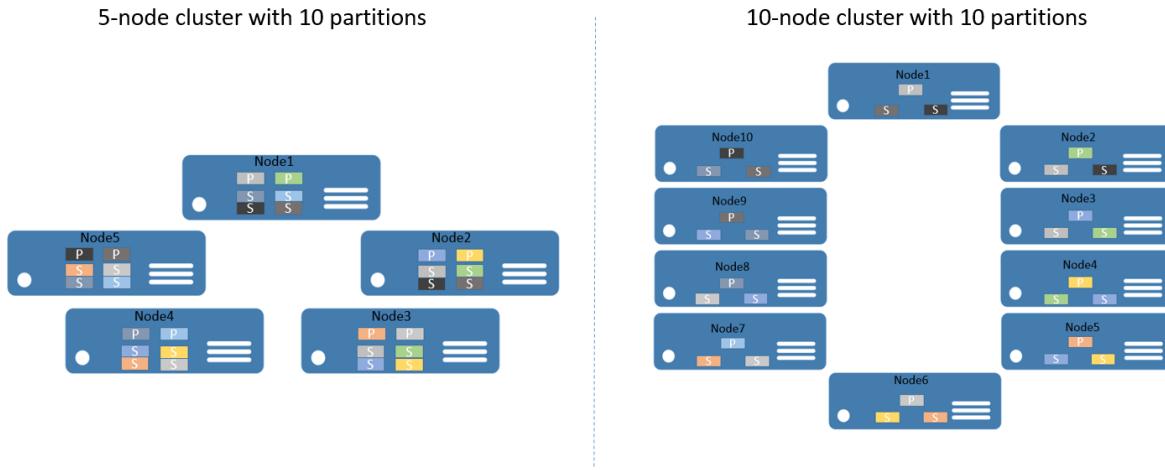
## Partition Service Fabric stateful services

Service Fabric makes it easy to develop scalable stateful services by offering a first-class way to partition state (data). Conceptually, you can think about a partition of a stateful service as a scale unit that is highly reliable through [replicas](#) that are distributed and balanced across the nodes in a cluster.

Partitioning in the context of Service Fabric stateful services refers to the process of determining that a particular service partition is responsible for a portion of the complete state of the service. (As mentioned before, a partition is a set of [replicas](#)). A great thing about Service Fabric is that it places the partitions on different nodes. This allows them to grow to a node's resource limit. As the data needs grow, partitions grow, and Service Fabric rebalances partitions across nodes. This ensures the continued efficient use of hardware resources.

To give you an example, say you start with a 5-node cluster and a service that is configured to have 10 partitions and a target of three replicas. In this case, Service Fabric would balance and distribute the replicas across the cluster--and you would end up with two primary [replicas](#) per node. If you now need to scale out the cluster to 10 nodes, Service Fabric would rebalance the primary [replicas](#) across all 10 nodes. Likewise, if you scaled back to 5 nodes, Service Fabric would rebalance all the replicas across the 5 nodes.

Figure 2 shows the distribution of 10 partitions before and after scaling the cluster.



As a result, the scale-out is achieved since requests from clients are distributed across computers, overall performance of the application is improved, and contention on access to chunks of data is reduced.

## Plan for partitioning

Before implementing a service, you should always consider the partitioning strategy that is required to scale out. There are different ways, but all of them focus on what the application needs to achieve. For the context of this article, let's consider some of the more important aspects.

A good approach is to think about the structure of the state that needs to be partitioned, as the first step.

Let's take a simple example. If you were to build a service for a countywide poll, you could create a partition for each city in the county. Then, you could store the votes for every person in the city in the partition that corresponds to that city. Figure 3 illustrates a set of people and the city in which they reside.

Data:

UserID	FirstName	LastName	City	Vote
bscholl	Boris	Scholl	Kirkland	Y
chackdan	Chacko	Daniel	Redmond	N
jeffreyr	Jeffrey	Richter	Redmond	Y
kunalds	Kunal	Deep Singh	Seattle	Y
mfussell	Mark	Fussell	Sammamish	N
masnider	Matt	Snider	Seattle	Y
subramar	Mani	Ramaswamy	Redmond	Y
seanmck	Sean	McKenna	Seattle	Y
vturecek	Vaclav	Turecek	Seattle	N

Partitions:



As the population of cities varies widely, you may end up with some partitions that contain a lot of data (e.g. Seattle) and other partitions with very little state (e.g. Kirkland). So what is the impact of having partitions with uneven amounts of state?

If you think about the example again, you can easily see that the partition that holds the votes for Seattle will get more traffic than the Kirkland one. By default, Service Fabric makes sure that there is about the same number of primary and secondary replicas on each node. So you may end up with nodes that hold replicas that serve more traffic and others that serve less traffic. You would preferably want to avoid hot and cold spots like this in a cluster.

In order to avoid this, you should do two things, from a partitioning point of view:

- Try to partition the state so that it is evenly distributed across all partitions.
- Report load from each of the replicas for the service. (For information on how, check out this article on [Metrics and Load](#)). Service Fabric provides the capability to report load consumed by services, such as amount of memory or number of records. Based on the metrics reported, Service Fabric detects that some partitions are serving higher loads than others and rebalances the cluster by moving replicas to more suitable nodes, so that overall no node is overloaded.

Sometimes, you cannot know how much data will be in a given partition. So a general recommendation is to do both--first, by adopting a partitioning strategy that spreads the data evenly across the partitions and second, by reporting load. The first method prevents situations described in the voting example, while the second helps smooth out temporary differences in access or load over time.

Another aspect of partition planning is to choose the correct number of partitions to begin with. From a Service Fabric perspective, there is nothing that prevents you from starting out with a higher number of partitions than anticipated for your scenario. In fact, assuming the maximum number of partitions is a valid approach.

In rare cases, you may end up needing more partitions than you have initially chosen. As you cannot change the partition count after the fact, you would need to apply some advanced partition approaches, such as creating a new service instance of the same service type. You would also need to implement some client-side logic that routes the requests to the correct service instance, based on client-side knowledge that your client code must maintain.

Another consideration for partitioning planning is the available computer resources. As the state needs to be

accessed and stored, you are bound to follow:

- Network bandwidth limits
- System memory limits
- Disk storage limits

So what happens if you run into resource constraints in a running cluster? The answer is that you can simply scale out the cluster to accommodate the new requirements.

[The capacity planning guide](#) offers guidance for how to determine how many nodes your cluster needs.

## Get started with partitioning

This section describes how to get started with partitioning your service.

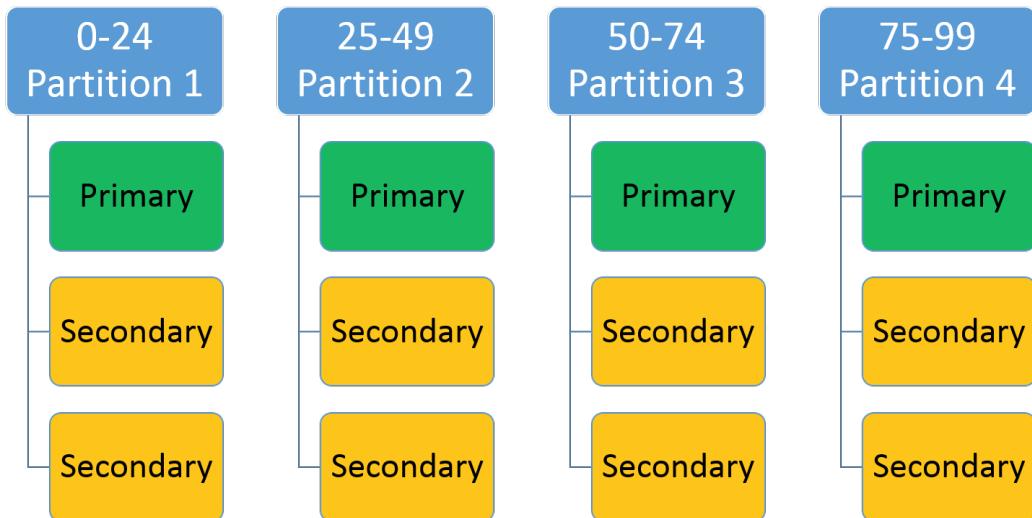
Service Fabric offers a choice of three partition schemes:

- Ranged partitioning (otherwise known as UniformInt64Partition).
- Named partitioning. Applications using this model usually have data that can be bucketed, within a bounded set. Some common examples of data fields used as named partition keys would be regions, postal codes, customer groups, or other business boundaries.
- Singleton partitioning. Singleton partitions are typically used when the service does not require any additional routing. For example, stateless services use this partitioning scheme by default.

Named and Singleton partitioning schemes are special forms of ranged partitions. By default, the Visual Studio templates for Service Fabric use ranged partitioning, as it is the most common and useful one. The remainder of this article focuses on the ranged partitioning scheme.

### Ranged partitioning scheme

This is used to specify an integer range (identified by a low key and high key) and a number of partitions (n). It creates n partitions, each responsible for a non-overlapping subrange of the overall partition key range. For example, a ranged partitioning scheme with a low key of 0, a high key of 99, and a count of 4 would create four partitions, as shown below.



A common approach is to create a hash based on a unique key within the data set. Some common examples of keys would be a vehicle identification number (VIN), an employee ID, or a unique string. By using this unique key, you would then generate a hash code, modulus the key range, to use as your key. You can specify the upper and lower bounds of the allowed key range.

### Select a hash algorithm

An important part of hashing is selecting your hash algorithm. A consideration is whether the goal is to group

similar keys near each other (locality sensitive hashing)--or if activity should be distributed broadly across all partitions (distribution hashing), which is more common.

The characteristics of a good distribution hashing algorithm are that it is easy to compute, it has few collisions, and it distributes the keys evenly. A good example of an efficient hash algorithm is the [FNV-1](#) hash algorithm.

A good resource for general hash code algorithm choices is the [Wikipedia page on hash functions](#).

## Build a stateful service with multiple partitions

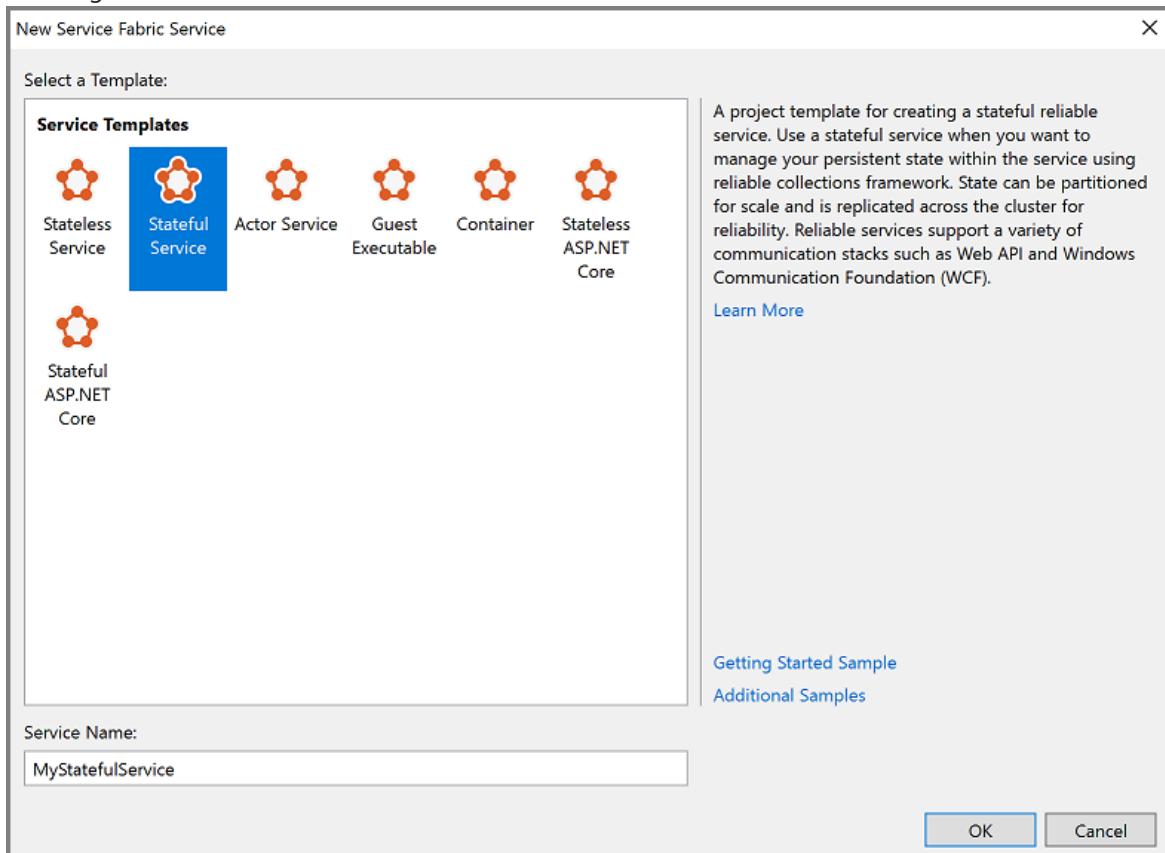
Let's create your first reliable stateful service with multiple partitions. In this example, you will build a very simple application where you want to store all last names that start with the same letter in the same partition.

Before you write any code, you need to think about the partitions and partition keys. You need 26 partitions (one for each letter in the alphabet), but what about the low and high keys? As we literally want to have one partition per letter, we can use 0 as the low key and 25 as the high key, as each letter is its own key.

### NOTE

This is a simplified scenario, as in reality the distribution would be uneven. Last names starting with the letters "S" or "M" are more common than the ones starting with "X" or "Y".

1. Open **Visual Studio > File > New > Project**.
2. In the **New Project** dialog box, choose the Service Fabric application.
3. Call the project "AlphabetPartitions".
4. In the **Create a Service** dialog box, choose **Stateful** service and call it "Alphabet.Processing" as shown in the image below.



5. Set the number of partitions. Open the Applicationmanifest.xml file located in the ApplicationPackageRoot folder of the AlphabetPartitions project and update the parameter Processing\_PartitionCount to 26 as shown below.

```
<Parameter Name="Processing_PartitionCount" DefaultValue="26" />
```

You also need to update the LowKey and HighKey properties of the StatefulService element in the ApplicationManifest.xml as shown below.

```
<Service Name="Processing">
  <StatefulService ServiceTypeName="ProcessingType" TargetReplicaSetSize=""
[Processing_TargetReplicaSetSize]" MinReplicaSetSize="[Processing_MinReplicaSetSize]">
    <UniformInt64Partition PartitionCount="[Processing_PartitionCount]" LowKey="0" HighKey="25" />
  </StatefulService>
</Service>
```

6. For the service to be accessible, open up an endpoint on a port by adding the endpoint element of ServiceManifest.xml (located in the PackageRoot folder) for the Alphabet.Processing service as shown below:

```
<Endpoint Name="ProcessingServiceEndpoint" Port="8089" Protocol="http" Type="Internal" />
```

Now the service is configured to listen to an internal endpoint with 26 partitions.

7. Next, you need to override the `CreateServiceReplicaListeners()` method of the Processing class.

**NOTE**

For this sample, we assume that you are using a simple HttpCommunicationListener. For more information on reliable service communication, see [The Reliable Service communication model](#).

8. A recommended pattern for the URL that a replica listens on is the following format:

`{scheme}://{nodeIp}:{port}/{partitionid}/{replicaId}/{guid}`. So you want to configure your communication listener to listen on the correct endpoints and with this pattern.

Multiple replicas of this service may be hosted on the same computer, so this address needs to be unique to the replica. This is why partition ID + replica ID are in the URL. HttpListener can listen on multiple addresses on the same port as long as the URL prefix is unique.

The extra GUID is there for an advanced case where secondary replicas also listen for read-only requests. When that's the case, you want to make sure that a new unique address is used when transitioning from primary to secondary to force clients to re-resolve the address. '+' is used as the address here so that the replica listens on all available hosts (IP, FQDN, localhost, etc.) The code below shows an example.

```

protected override IEnumerable<ServiceReplicaListener> CreateServiceReplicaListeners()
{
    return new[] { new ServiceReplicaListener(context => this.CreateInternalListener(context))};
}
private ICommunicationListener CreateInternalListener(ServiceContext context)
{

    EndpointResourceDescription internalEndpoint =
context.CodePackageActivationContext.GetEndpoint("ProcessingServiceEndpoint");
    string uriPrefix = String.Format(
        "{0}://+:{1}/{2}/{3}-{4}/",
        internalEndpoint.Protocol,
        internalEndpoint.Port,
        context.PartitionId,
        context.ReplicaOrInstanceId,
        Guid.NewGuid());

    string nodeIP = FabricRuntime.GetNodeContext().IPAddressOrFQDN;

    string uriPublished = uriPrefix.Replace("+", nodeIP);
    return new HttpCommunicationListener(uriPrefix, uriPublished, this.ProcessInternalRequest);
}

```

It's also worth noting that the published URL is slightly different from the listening URL prefix. The listening URL is given to `HttpListener`. The published URL is the URL that is published to the Service Fabric Naming Service, which is used for service discovery. Clients will ask for this address through that discovery service. The address that clients get needs to have the actual IP or FQDN of the node in order to connect. So you need to replace '+' with the node's IP or FQDN as shown above.

9. The last step is to add the processing logic to the service as shown below.

```

private async Task ProcessInternalRequest(HttpListenerContext context, CancellationToken
cancelRequest)
{
    string output = null;
    string user = context.Request.QueryString["lastname"].ToString();

    try
    {
        output = await this.AddUserAsync(user);
    }
    catch (Exception ex)
    {
        output = ex.Message;
    }

    using (HttpListenerResponse response = context.Response)
    {
        if (output != null)
        {
            byte[] outBytes = Encoding.UTF8.GetBytes(output);
            response.OutputStream.Write(outBytes, 0, outBytes.Length);
        }
    }
}

private async Task<string> AddUserAsync(string user)
{
    IReliableDictionary<String, String> dictionary = await
this.StateManager.GetOrAddAsync<IReliableDictionary<String, String>>("dictionary");

    using (ITransaction tx = this.StateManager.CreateTransaction())
    {
        bool addResult = await dictionary.TryAddAsync(tx, user.ToUpperInvariant(), user);

        await tx.CommitAsync();

        return String.Format(
            "User {0} {1}",
            user,
            addResult ? "successfully added" : "already exists");
    }
}

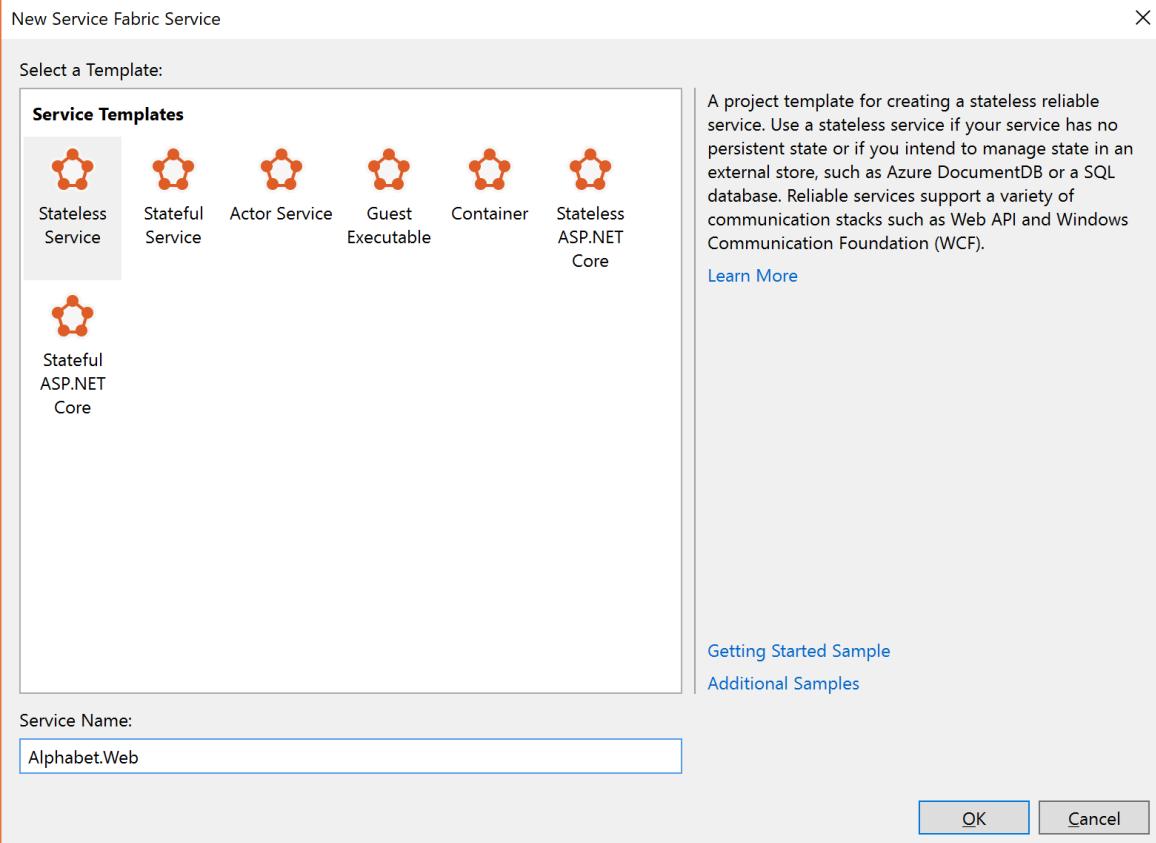
```

`ProcessInternalRequest` reads the values of the query string parameter used to call the partition and calls `AddUserAsync` to add the lastname to the reliable dictionary `dictionary`.

- Let's add a stateless service to the project to see how you can call a particular partition.

This service serves as a simple web interface that accepts the lastname as a query string parameter, determines the partition key, and sends it to the Alphabet.Processing service for processing.

- In the **Create a Service** dialog box, choose **Stateless** service and call it "Alphabet.Web" as shown below.



12. Update the endpoint information in the ServiceManifest.xml of the Alphabet.WebApi service to open up a port as shown below.

```
<Endpoint Name="Web ApiService Endpoint" Protocol="http" Port="8081"/>
```

13. You need to return a collection of ServiceInstanceListeners in the class Web. Again, you can choose to implement a simple HttpCommunicationListener.

```
protected override IEnumerable<ServiceInstanceListener> CreateServiceInstanceListeners()
{
    return new[] {new ServiceInstanceListener(context => this.CreateInputListener(context))};
}
private ICommunicationListener CreateInputListener(ServiceContext context)
{
    // Service instance's URL is the node's IP & desired port
    EndpointResourceDescription inputEndpoint =
context.CodePackageActivationContext.GetEndpoint("Web ApiService Endpoint")
    string uriPrefix = String.Format("{0}://+:{1}/alphabetpartitions/", inputEndpoint.Protocol,
inputEndpoint.Port);
    var uriPublished = uriPrefix.Replace("+", FabricRuntime.GetNodeContext().IPAddressOrFQDN);
    return new HttpCommunicationListener(uriPrefix, uriPublished, this.ProcessInputRequest);
}
```

14. Now you need to implement the processing logic. The HttpCommunicationListener calls `ProcessInputRequest` when a request comes in. So let's go ahead and add the code below.

```

private async Task ProcessInputRequest(HttpContext context, CancellationToken cancelRequest)
{
    String output = null;
    try
    {
        string lastname = context.Request.QueryString["lastname"];
        char firstLetterOfLastName = lastname.First();
        ServicePartitionKey partitionKey = new
ServicePartitionKey(Char.ToUpper(firstLetterOfLastName) - 'A');

        ResolvedServicePartition partition = await
this.servicePartitionResolver.ResolveAsync(alphabetServiceUri, partitionKey, cancelRequest);
        ResolvedServiceEndpoint ep = partition.GetEndpoint();

        JObject addresses = JObject.Parse(ep.Address);
        string primaryReplicaAddress = (string)addresses["Endpoints"].First();

        UriBuilder primaryReplicaUriBuilder = new UriBuilder(primaryReplicaAddress);
        primaryReplicaUriBuilder.Query = "lastname=" + lastname;

        string result = await this.httpClient.GetStringAsync(primaryReplicaUriBuilder.Uri);

        output = String.Format(
            "Result: {0}. <p>Partition key: '{1}' generated from the first letter '{2}' of input
value '{3}'. <br>Processing service partition ID: {4}. <br>Processing service replica address: {5}",
            result,
            partitionKey,
            firstLetterOfLastName,
            lastname,
            partition.Info.Id,
            primaryReplicaAddress);
    }
    catch (Exception ex) { output = ex.Message; }

    using (var response = context.Response)
    {
        if (output != null)
        {
            output = output + "added to Partition: " + primaryReplicaAddress;
            byte[] outBytes = Encoding.UTF8.GetBytes(output);
            response.OutputStream.Write(outBytes, 0, outBytes.Length);
        }
    }
}

```

Let's walk through it step by step. The code reads the first letter of the query string parameter `lastname` into a char. Then, it determines the partition key for this letter by subtracting the hexadecimal value of `A` from the hexadecimal value of the last names' first letter.

```

string lastname = context.Request.QueryString["lastname"];
char firstLetterOfLastName = lastname.First();
ServicePartitionKey partitionKey = new ServicePartitionKey(Char.ToUpper(firstLetterOfLastName) -
'A');

```

Remember, for this example, we are using 26 partitions with one partition key per partition. Next, we obtain the service partition `partition` for this key by using the `ResolveAsync` method on the `servicePartitionResolver` object. `servicePartitionResolver` is defined as

```

private readonly ServicePartitionResolver servicePartitionResolver =
ServicePartitionResolver.GetDefault();

```

The `ResolveAsync` method takes the service URI, the partition key, and a cancellation token as parameters.

The service URI for the processing service is `fabric:/AlphabetPartitions/Processing`. Next, we get the endpoint of the partition.

```
ResolvedServiceEndpoint ep = partition.GetEndpoint()
```

Finally, we build the endpoint URL plus the querystring and call the processing service.

```
JObject addresses = JObject.Parse(ep.Address);
string primaryReplicaAddress = (string)addresses["Endpoints"].First();

UriBuilder primaryReplicaUriBuilder = new UriBuilder(primaryReplicaAddress);
primaryReplicaUriBuilder.Query = "lastname=" + lastname;

string result = await this.httpClient.GetStringAsync(primaryReplicaUriBuilder.Uri);
```

Once the processing is done, we write the output back.

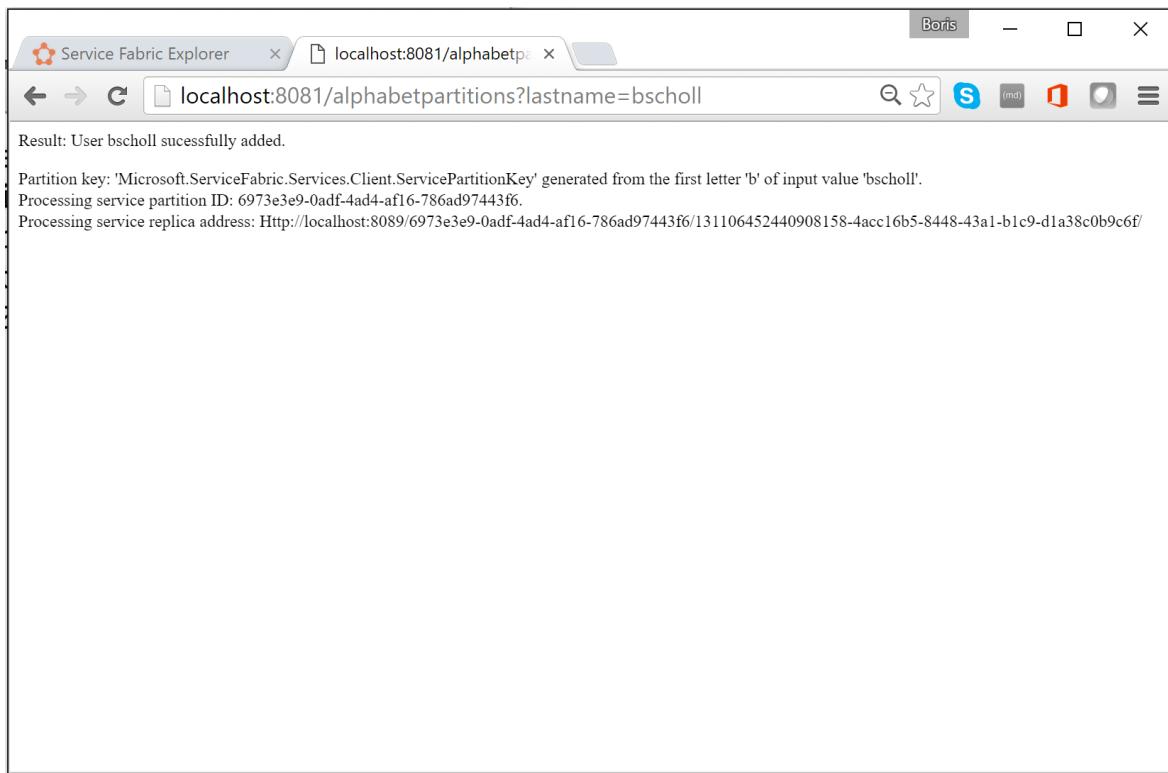
15. The last step is to test the service. Visual Studio uses application parameters for local and cloud deployment. To test the service with 26 partitions locally, you need to update the `Local.xml` file in the ApplicationParameters folder of the AlphabetPartitions project as shown below:

```
<Parameters>
<Parameter Name="Processing_PartitionCount" Value="26" />
<Parameter Name="WebApi_InstanceCount" Value="1" />
</Parameters>
```

16. Once you finish deployment, you can check the service and all of its partitions in the Service Fabric Explorer.

ID	Partition Kind	Health State	Status
09a14df4-3d7b-4eb2-b1db-19361fc7e06d	Int64Range	OK	Ready
0dc42a0c-4ed8-42b5-acd7-94a128e5d0eb	Int64Range	OK	Ready
0e17f3e4-83ae-41b3-8cab-384bde40b80e	Int64Range	OK	Ready
29b54c74-212f-47dd-8f66-9b2729d4baa9	Int64Range	OK	Ready
2bc90c81-2148-45ce-976d-e4cd337d23d0	Int64Range	OK	Ready
396b2632-b32a-44a3-b60f-8f054a2d261a	Int64Range	OK	Ready
3a7bebe54-d006-4e34-a33e-49c49b1c582d	Int64Range	OK	Ready
528fbefb9-beed-4d04-aa2d-48d9e9fb9bch5	Int64Range	OK	Ready
5fb703c2-2bdf-4b50-a86d-e4c9c949628	Int64Range	OK	Ready
63a332b8-5072-4447-a4b8-a0af514031c4	Int64Range	OK	Ready

17. In a browser, you can test the partitioning logic by entering `http://localhost:8081/?lastname=somename`. You will see that each last name that starts with the same letter is being stored in the same partition.



The entire source code of the sample is available on [GitHub](#).

## Next steps

For information on Service Fabric concepts, see the following:

- [Availability of Service Fabric services](#)
- [Scalability of Service Fabric services](#)
- [Capacity planning for Service Fabric applications](#)

# Availability of Service Fabric services

10/16/2017 • 1 min to read • [Edit Online](#)

This article gives an overview of how Azure Service Fabric maintains the availability of a service.

## Availability of Service Fabric stateless services

Service Fabric services can be either stateful or stateless. A stateless service is an application service that does not have a [local state](#) that needs to be highly available or reliable.

Creating a stateless service requires defining an `InstanceCount`. The instance count defines the number of instances of the stateless service's application logic that should be running in the cluster. Increasing the number of instances is the recommended way of scaling out a stateless service.

When an instance of a stateless named-service fails, a new instance is created on an eligible node in the cluster. For example, a stateless service instance might fail on Node1 and be re-created on Node5.

## Availability of Service Fabric stateful services

A stateful service has a state associated with it. In Service Fabric, a stateful service is modeled as a set of replicas. Each replica is a running instance of the code of the service. The replica also has a copy of the state for that service. Read and write operations are performed at one replica, called the *Primary*. Changes to state from write operations are *replicated* to the other replicas in the replica set, called *Active Secondaries*, and applied.

There can be only one Primary replica, but there can be multiple Active Secondary replicas. The number of active Secondary replicas is configurable, and a higher number of replicas can tolerate a greater number of concurrent software and hardware failures.

If the Primary replica goes down, Service Fabric makes one of the Active Secondary replicas the new Primary replica. This Active Secondary replica already has the updated version of the state, via *replication*, and it can continue processing further read/write operations. This process is known as *reconfiguration* and is described further in the [Reconfiguration](#) article.

The concept of a replica being either a Primary or Active Secondary, is known as the *replica role*. These replicas are described further in the [Replicas and instances](#) article.

## Next steps

For more information on Service Fabric concepts, see the following articles:

- [Scaling Service Fabric services](#)
- [Partitioning Service Fabric services](#)
- [Defining and managing state](#)
- [Reliable Services](#)

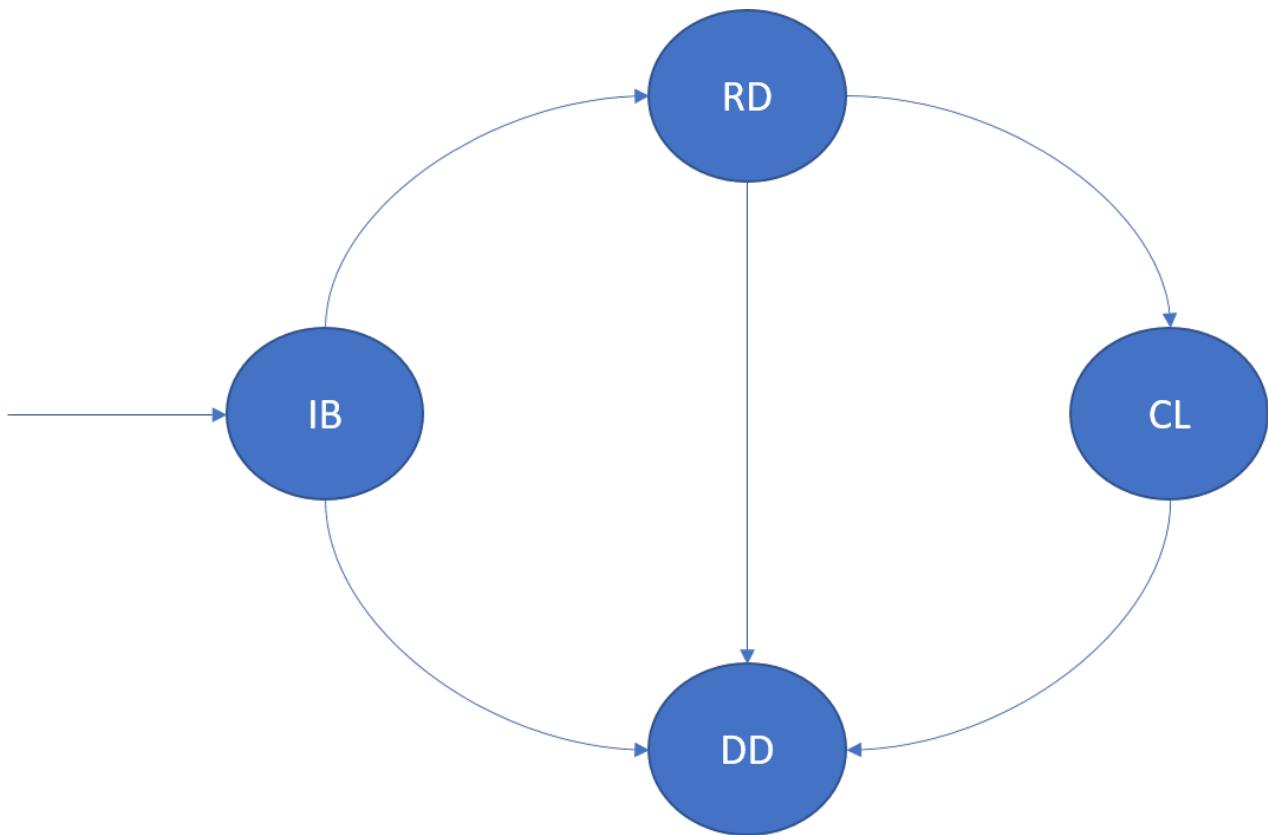
# Replicas and instances

10/9/2017 • 6 min to read • [Edit Online](#)

This article gives an overview of the lifecycle of replicas of stateful services and instances of stateless services.

## Instances of stateless services

An instance of a stateless service is a copy of the service logic that runs on one of the nodes of the cluster. An instance within a partition is uniquely identified by its **InstanceId**. The lifecycle of an instance is modeled in the following diagram:



### InBuild (IB)

After the Cluster Resource Manager determines a placement for the instance, it enters this lifecycle state. The instance is started on the node. The application host is started, the instance is created and then opened. After the startup finishes, the instance transitions to the ready state.

If the application host or node for this instance crashes, it transitions to the dropped state.

### Ready (RD)

In the ready state, the instance is up and running on the node. If this instance is a reliable service, **RunAsync** has been invoked.

If the application host or node for this instance crashes, it transitions to the dropped state.

### Closing (CL)

In the closing state, Azure Service Fabric is in the process of shutting down the instance on this node. This shutdown might be due to many reasons--for example, an application upgrade, load balancing, or the service being deleted. After shutdown finishes, it transitions to the dropped state.

## Dropped (DD)

In the dropped state, the instance is no longer running on the node. At this point, Service Fabric maintains the metadata about this instance, which is eventually deleted as well.

### NOTE

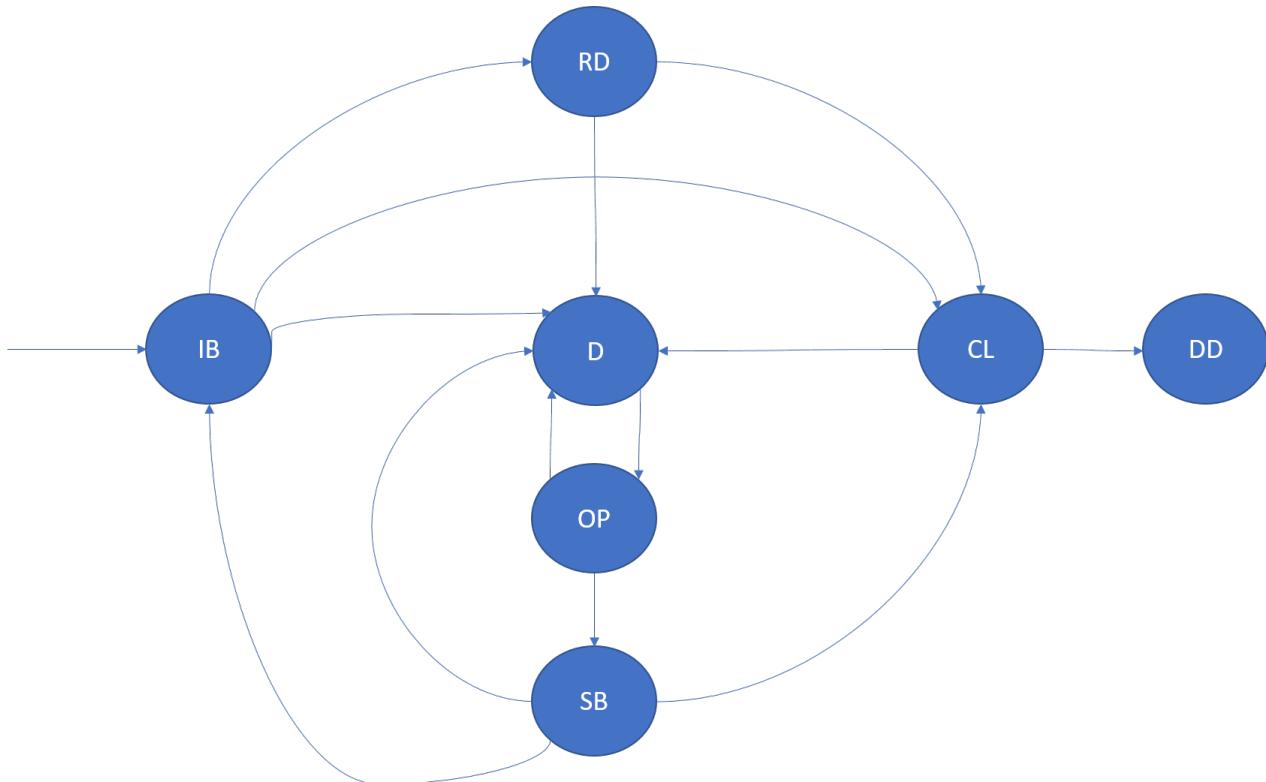
It is possible to transition from any state to the dropped state by using the **ForceRemove** option on `Remove-ServiceFabricReplica`.

## Replicas of stateful services

A replica of a stateful service is a copy of the service logic running on one of the nodes of the cluster. In addition, the replica maintains a copy of the state of that service. Two related concepts describe the lifecycle and behavior of stateful replicas:

- Replica lifecycle
- Replica role

The following discussion describes persisted stateful services. For volatile (or in-memory) stateful services, the down and dropped states are equivalent.



### InBuild (IB)

An InBuild replica is a replica that's created or prepared for joining the replica set. Depending on the replica role, the IB has different semantics.

If the application host or the node for an InBuild replica crashes, it transitions to the down state.

- **Primary InBuild replicas:** Primary InBuild are the first replicas for a partition. This replica usually happens when the partition is being created. Primary InBuild replicas also arise when all the replicas of a partition restart or are dropped.
- **IdleSecondary InBuild replicas:** These are either new replicas that are created by the Cluster Resource Manager, or existing replicas that went down and need to be added back into the set. These replicas are seeded or built by the primary before they can join the replica set as ActiveSecondary and participate in

quorum acknowledgement of operations.

- **ActiveSecondary InBuild replicas:** This state is observed in some queries. It is an optimization where the replica set is not changing, but a replica needs to be built. The replica itself follows the normal state machine transitions (as described in the section on replica roles).

## Ready (RD)

A Ready replica is a replica that's participating in replication and quorum acknowledgement of operations. The ready state is applicable to primary and active secondary replicas.

If the application host or the node for a ready replica crashes, it transitions to the down state.

## Closing (CL)

A replica enters the closing state in the following scenarios:

- **Shutting down the code for the replica:** Service Fabric might need to shut down the running code for a replica. This shutdown might be for many reasons. For example, it can happen because of an application, fabric, or infrastructure upgrade, or because of a fault reported by the replica. When the replica close finishes, the replica transitions to the down state. The persisted state associated with this replica that's stored on disk is not cleaned up.
- **Removing the replica from the cluster:** Service Fabric might need to remove the persisted state and shut down the running code for a replica. This shutdown might be for many reasons, for example, load balancing.

## Dropped (DD)

In the dropped state, the instance is no longer running on the node. There is also no state left on the node. At this point, Service Fabric maintains the metadata about this instance, which is eventually deleted as well.

## Down (D)

In the down state, the replica code is not running, but the persisted state for that replica exists on that node. A replica can be down for many reasons--for example, the node being down, a crash in the replica code, an application upgrade, or replica faults.

A down replica is opened by Service Fabric as required, for example, when the upgrade finishes on the node.

The replica role is not relevant in the down state.

## Opening (OP)

A down replica enters the opening state when Service Fabric needs to bring the replica back up again. For example, this state might be after a code upgrade for the application finishes on a node.

If the application host or the node for an opening replica crashes, it transitions to the down state.

The replica role is not relevant in the opening state.

## StandBy (SB)

A StandBy replica is a replica of a persisted service that went down and was then opened. This replica might be used by Service Fabric if it needs to add another replica to the replica set (because the replica already has some portion of the state and the build process is faster). After the StandByReplicaKeepDuration expires, the standby replica is discarded.

If the application host or the node for a standby replica crashes, it transitions to the down state.

The replica role is not relevant in the standby state.

#### NOTE

Any replica that's not down or dropped is considered to be *up*.

#### NOTE

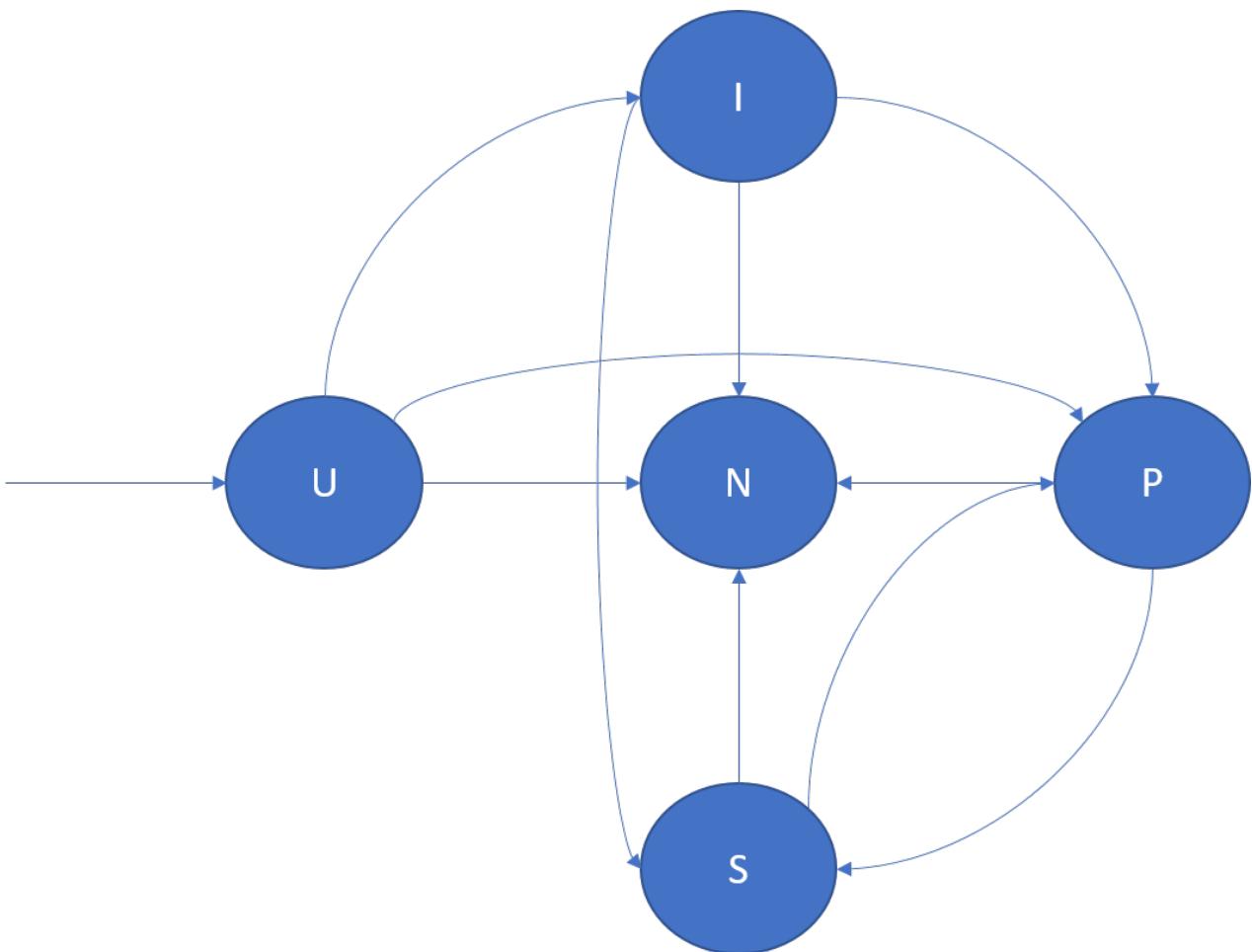
It's possible to transition from any state to the dropped state by using the **ForceRemove** option on `Remove-ServiceFabricReplica`.

## Replica role

The role of the replica determines its function in the replica set:

- **Primary (P)**: There is one primary in the replica set that is responsible for performing read and write operations.
- **ActiveSecondary (S)**: These are replicas that receive state updates from the primary, apply them, and then send back acknowledgements. There are multiple active secondaries in the replica set. The number of these active secondaries determines the number of faults the service can handle.
- **IdleSecondary (I)**: These replicas are being built by the primary. They are receiving state from the primary before they can be promoted to active secondary.
- **None (N)**: These replicas don't have a responsibility in the replica set.
- **Unknown (U)**: This is the initial role of a replica before it receives any **ChangeRole** API call from Service Fabric.

The following diagram illustrates the replica role transitions and some example scenarios in which they can occur:



- U -> P: Creation of a new primary replica.

- U -> I: Creation of a new idle replica.
- U -> N: Deletion of a standby replica.
- I -> S: Promotion of the idle secondary to active secondary so that its acknowledgements contribute toward quorum.
- I -> P: Promotion of the idle secondary to primary. This can happen under special reconfigurations when the idle secondary is the correct candidate to be primary.
- I -> N: Deletion of the idle secondary replica.
- S -> P: Promotion of the active secondary to primary. This can be due to failover of the primary or a primary movement initiated by the Cluster Resource Manager. For example, it might be in response to an application upgrade or load balancing.
- S -> N: Deletion of the active secondary replica.
- P -> S: Demotion of the primary replica. This can be due to a primary movement initiated by the Cluster Resource Manager. For example, it might be in response to an application upgrade or load balancing.
- P -> N: Deletion of the primary replica.

#### NOTE

Higher-level programming models, such as [Reliable Actors](#) and [Reliable Services](#), hide the concept of replica roles from the developer. In Actors, the notion of a role is unnecessary. In Services, it's largely simplified for most scenarios.

## Next steps

For more information on Service Fabric concepts, see the following article:

[Reliable Services lifecycle - C#](#)

# Reconfiguration in Azure Service Fabric

10/9/2017 • 2 min to read • [Edit Online](#)

A *configuration* is defined as the replicas and their roles for a partition of a stateful service.

A *reconfiguration* is the process of moving one configuration to another configuration. It makes a change to the replica set for a partition of a stateful service. The old configuration is called the *previous configuration (PC)*, and the new configuration is called the *current configuration (CC)*. The reconfiguration protocol in Azure Service Fabric preserves consistency and maintains availability during any changes to the replica set.

Failover Manager initiates reconfigurations in response to different events in the system. For instance, if the primary fails then a reconfiguration is initiated to promote an active secondary to a primary. Another example is in response to application upgrades when it might be necessary to move the primary to another node in order to upgrade the node.

## Reconfiguration types

Reconfigurations can be classified into two types:

- Reconfigurations where the primary is changing:
  - **Failover:** Failovers are reconfigurations in response to the failure of a running primary.
  - **SwapPrimary:** Swaps are reconfigurations where Service Fabric needs to move a running primary from one node to another, usually in response to load balancing or an upgrade.
- Reconfigurations where the primary is not changing.

## Reconfiguration phases

A reconfiguration proceeds in several phases:

- **Phase0:** This phase happens in swap-primary reconfigurations where the current primary transfers its state to the new primary and transitions to active secondary.
- **Phase1:** This phase happens during reconfigurations where the primary is changing. During this phase, Service Fabric identifies the correct primary among the current replicas. This phase is not needed during swap-primary reconfigurations because the new primary has already been chosen.
- **Phase2:** During this phase, Service Fabric ensures that all data is available in a majority of the replicas of the current configuration.

There are several other phases that are for internal use only.

## Stuck reconfigurations

Reconfigurations can get *stuck* for a variety of reasons. Some of the common reasons include:

- **Down replicas:** Some reconfiguration phases require a majority of the replicas in the configuration to be up.
- **Network or communication problems:** Reconfigurations require network connectivity between different nodes.
- **API failures:** The reconfiguration protocol requires that service implementations finish certain APIs. For example, not honoring the cancellation token in a reliable service causes SwapPrimary reconfigurations to get stuck.

Use health reports from system components, such as System.FM, System.RA, and System.RAP, to diagnose where a reconfiguration is stuck. The [system health report page](#) describes these health reports.

## Next steps

For more information on Service Fabric concepts, see the following articles:

- [Reliable Services lifecycle - C#](#)
- [System health reports](#)
- [Replicas and instances](#)

# Connect and communicate with services in Service Fabric

6/27/2017 • 8 min to read • [Edit Online](#)

In Service Fabric, a service runs somewhere in a Service Fabric cluster, typically distributed across multiple VMs. It can be moved from one place to another, either by the service owner, or automatically by Service Fabric. Services are not statically tied to a particular machine or address.

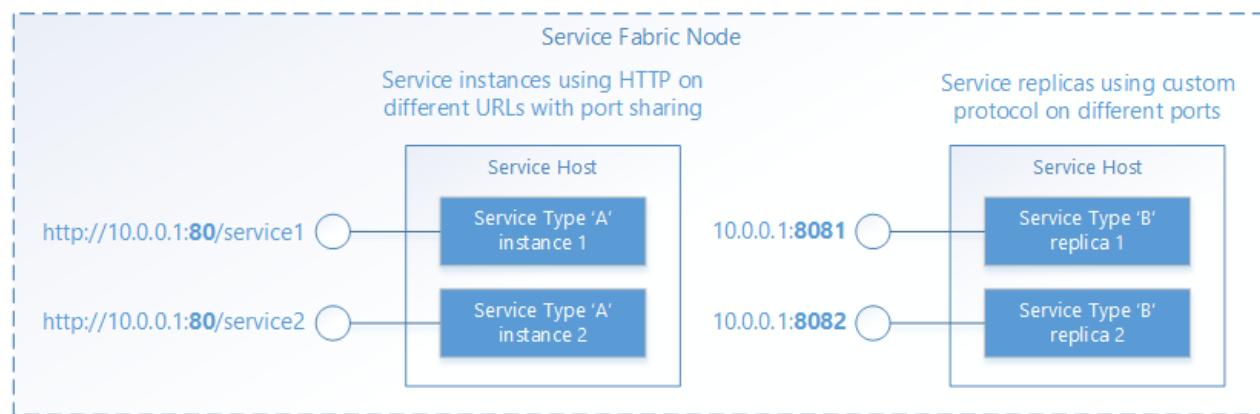
A Service Fabric application is generally composed of many different services, where each service performs a specialized task. These services may communicate with each other to form a complete function, such as rendering different parts of a web application. There are also client applications that connect to and communicate with services. This document discusses how to set up communication with and between your services in Service Fabric.

This Microsoft Virtual Academy video also discusses service communication:



## Bring your own protocol

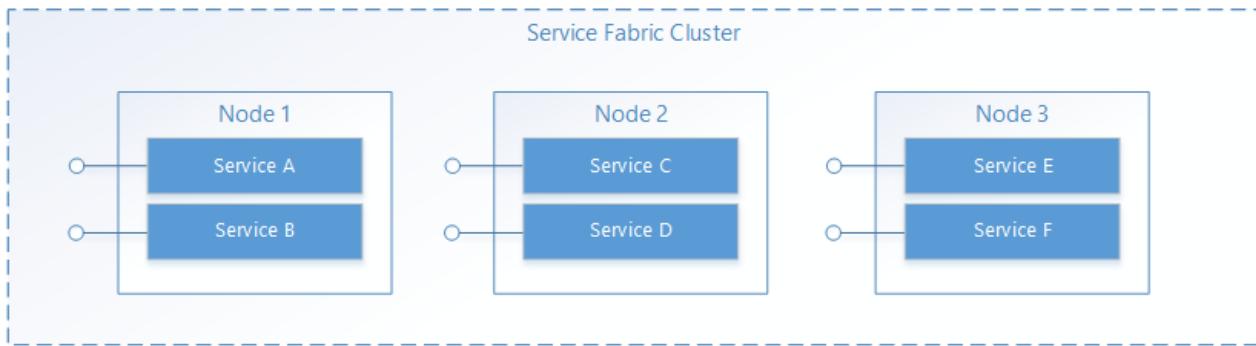
Service Fabric helps manage the lifecycle of your services but it does not make decisions about what your services do. This includes communication. When your service is opened by Service Fabric, that's your service's opportunity to set up an endpoint for incoming requests, using whatever protocol or communication stack you want. Your service will listen on a normal **IP:port** address using any addressing scheme, such as a URI. Multiple service instances or replicas may share a host process, in which case they will either need to use different ports or use a port-sharing mechanism, such as the http.sys kernel driver in Windows. In either case, each service instance or replica in a host process must be uniquely addressable.



## Service discovery and resolution

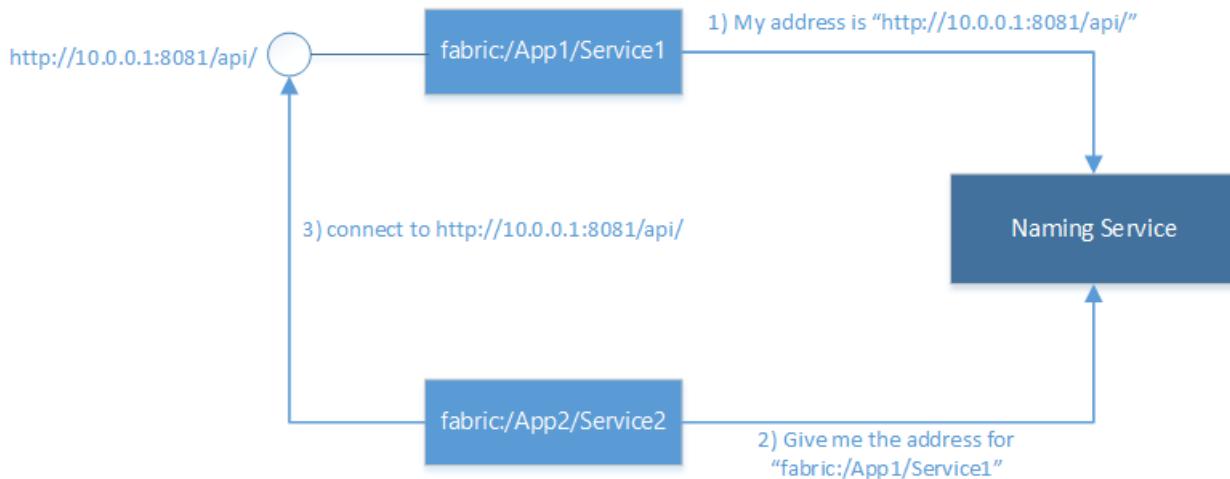
In a distributed system, services may move from one machine to another over time. This can happen for various

reasons, including resource balancing, upgrades, failovers, or scale-out. This means service endpoint addresses change as the service moves to nodes with different IP addresses, and may open on different ports if the service uses a dynamically selected port.



Service Fabric provides a discovery and resolution service called the Naming Service. The Naming Service maintains a table that maps named service instances to the endpoint addresses they listen on. All named service instances in Service Fabric have unique names represented as URLs, for example,

"fabric:/MyApplication/MyService". The name of the service does not change over the lifetime of the service, it's only the endpoint addresses that can change when services move. This is analogous to websites that have constant URLs but where the IP address may change. And similar to DNS on the web, which resolves website URLs to IP addresses, Service Fabric has a registrar that maps service names to their endpoint address.



Resolving and connecting to services involves the following steps run in a loop:

- **Resolve:** Get the endpoint that a service has published from the Naming Service.
- **Connect:** Connect to the service over whatever protocol it uses on that endpoint.
- **Retry:** A connection attempt may fail for any number of reasons, for example if the service has moved since the last time the endpoint address was resolved. In that case, the preceding resolve and connect steps need to be retried, and this cycle is repeated until the connection succeeds.

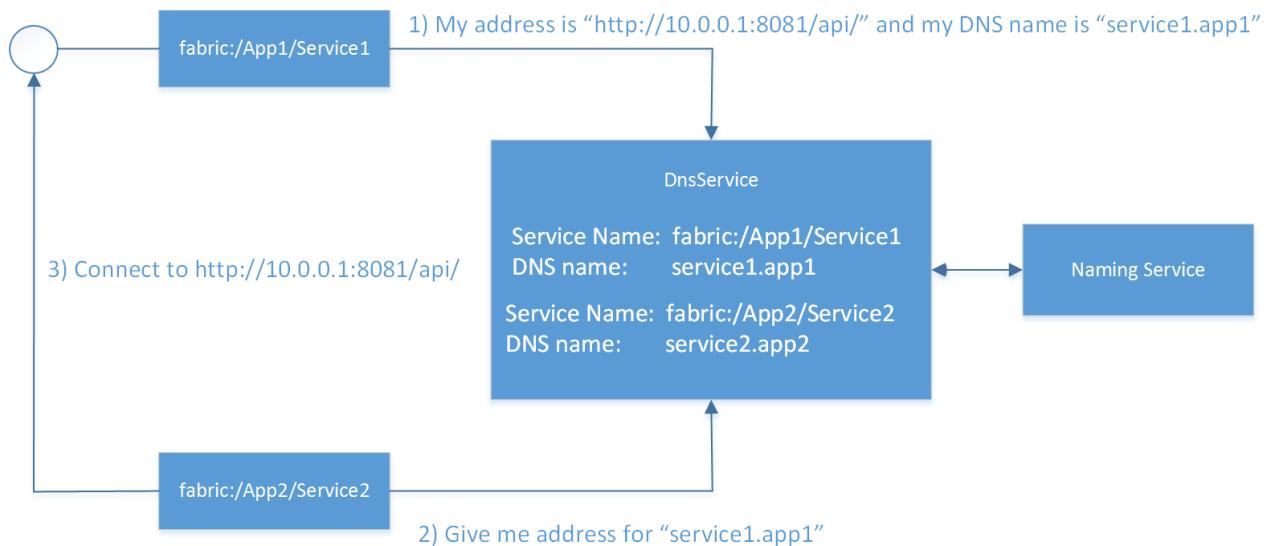
## Connecting to other services

Services connecting to each other inside a cluster generally can directly access the endpoints of other services because the nodes in a cluster are on the same local network. To make it easier to connect between services, Service Fabric provides additional services that use the Naming Service. A DNS service and a reverse proxy service.

### DNS service

Since many services, especially containerized services, can have an existing URL name, being able to resolve these using the standard DNS protocol (rather than the Naming Service protocol) is very convenient, especially in application "lift and shift" scenarios. This is exactly what the DNS service does. It enables you to map DNS names to a service name and hence resolve endpoint IP addresses.

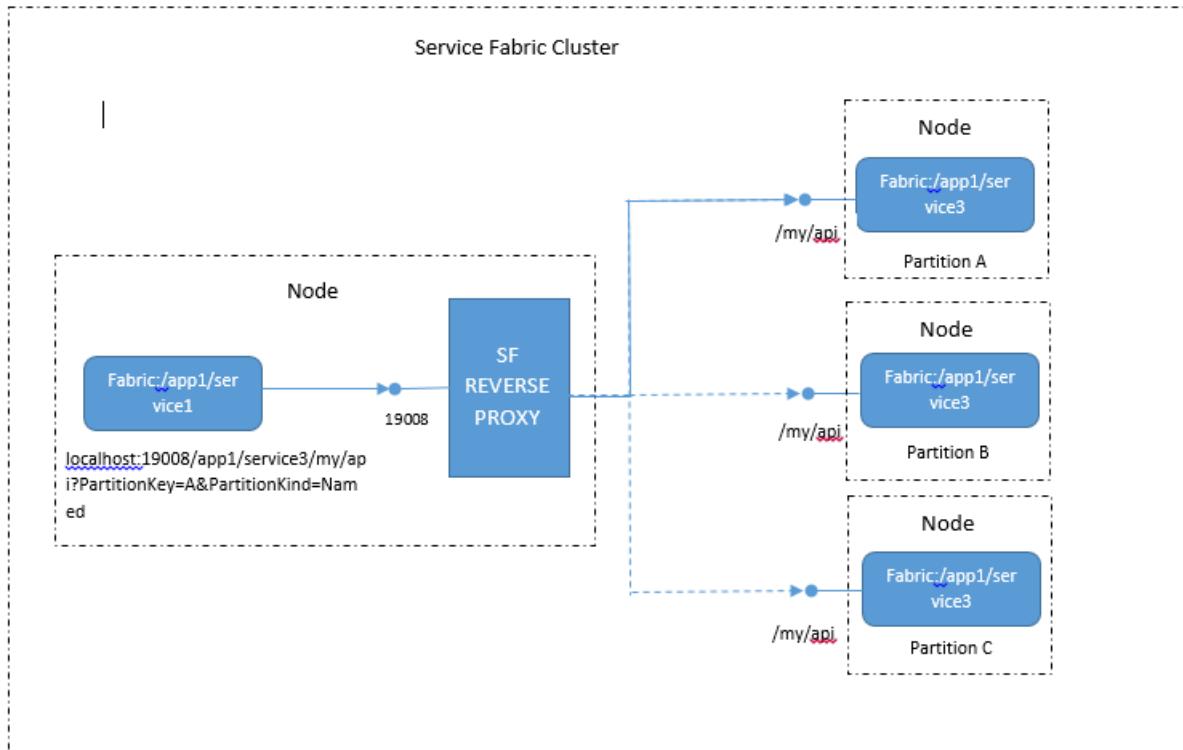
As shown in the following diagram, the DNS service, running in the Service Fabric cluster, maps DNS names to service names which are then resolved by the Naming Service to return the endpoint addresses to connect to. The DNS name for the service is provided at the time of creation.



For more details on how to use the DNS service see [DNS service in Azure Service Fabric](#) article.

### Reverse proxy service

The reverse proxy addresses services in the cluster that exposes HTTP endpoints including HTTPS. The reverse proxy greatly simplifies calling other services and their methods by having a specific URL format and handles the resolve, connect, retry steps required for one service to communicate with another using the Naming Service. In other words, it hides the Naming Service from you when calling other services by making this as simple as calling a URL.



For more details on how to use the reverse proxy service see [Reverse proxy in Azure Service Fabric](#) article.

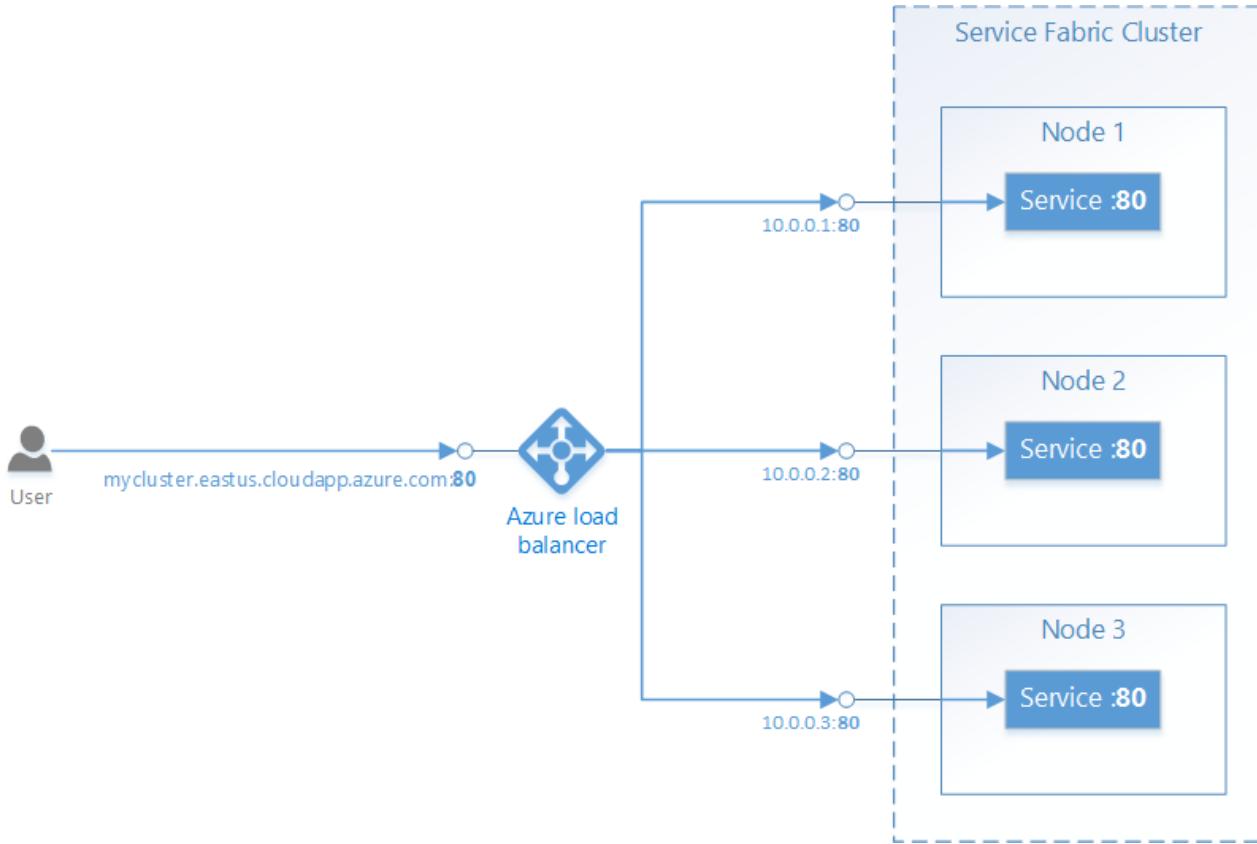
## Connections from external clients

Services connecting to each other inside a cluster generally can directly access the endpoints of other services because the nodes in a cluster are on the same local network. In some environments, however, a cluster may be

behind a load balancer that routes external ingress traffic through a limited set of ports. In these cases, services can still communicate with each other and resolve addresses using the Naming Service, but extra steps must be taken to allow external clients to connect to services.

## Service Fabric in Azure

A Service Fabric cluster in Azure is placed behind an Azure Load Balancer. All external traffic to the cluster must pass through the load balancer. The load balancer will automatically forward traffic inbound on a given port to a random *node* that has the same port open. The Azure Load Balancer only knows about ports open on the *nodes*, it does not know about ports open by individual services.



For example, in order to accept external traffic on port **80**, the following things must be configured:

1. Write a service that listens on port 80. Configure port 80 in the service's ServiceManifest.xml and open a listener in the service, for example, a self-hosted web server.

```
<Resources>
  <Endpoints>
    <Endpoint Name="WebEndpoint" Protocol="http" Port="80" />
  </Endpoints>
</Resources>
```

```
class HttpCommunicationListener : ICommunicationListener
{
    ...
    public Task<string> OpenAsync(CancellationToken cancellationToken)
    {
        EndpointResourceDescription endpoint =
            serviceContext.CodePackageActivationContext.GetEndpoint("WebEndpoint");

        string uriPrefix = $"{endpoint.Protocol}://+:{endpoint.Port}/myapp/";

        this.httpListener = new HttpListener();
        this.httpListener.Prefixes.Add(uriPrefix);
        this.httpListener.Start();

        string publishUri = uriPrefix.Replace("+", FabricRuntime.GetNodeContext().IPAddressOrFQDN);
        return Task.FromResult(publishUri);
    }

    ...
}

class WebService : StatelessService
{
    ...
    protected override IEnumerable<ServiceInstanceListener> CreateServiceInstanceListeners()
    {
        return new[] { new ServiceInstanceListener(context => new
HttpCommunicationListener(context))};
    }

    ...
}
```

```

class HttpCommunicationlistener implements CommunicationListener {
    ...

    @Override
    public CompletableFuture<String> openAsync(CancellationToken arg0) {
        EndpointResourceDescription endpoint =
            this.serviceContext.getCodePackageActivationContext().getEndpoint("WebEndpoint");
        try {
            HttpServer server = com.sun.net.httpserver.HttpServer.create(new
InetSocketAddress(endpoint.getPort()), 0);
            server.start();

            String publishUri = String.format("http://%s:%d/",
                this.serviceContext.getNodeContext().getIpAddressOrFQDN(), endpoint.getPort());
            return CompletableFuture.completedFuture(publishUri);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    ...
}

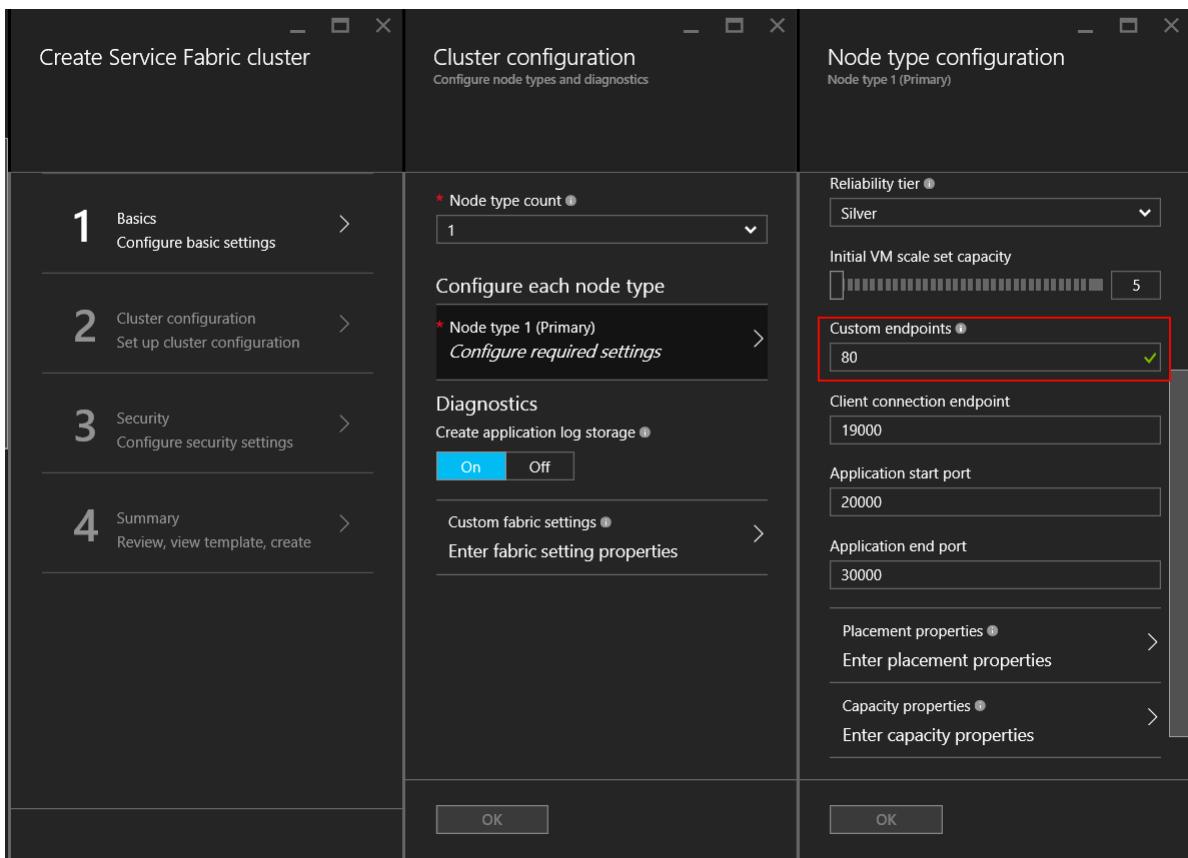
class WebService extends StatelessService {
    ...

    @Override
    protected List<ServiceInstanceListener> createServiceInstanceListeners() {
        <ServiceInstanceListener> listeners = new ArrayList<ServiceInstanceListener>();
        listeners.add(new ServiceInstanceListener((context) -> new
HttpCommunicationlistener(context)));
        return listeners;
    }

    ...
}

```

2. Create a Service Fabric Cluster in Azure and specify port **80** as a custom endpoint port for the node type that will host the service. If you have more than one node type, you can set up a *placement constraint* on the service to ensure it only runs on the node type that has the custom endpoint port opened.



3. Once the cluster has been created, configure the Azure Load Balancer in the cluster's Resource Group to forward traffic on port 80. When creating a cluster through the Azure portal, this is set up automatically for each custom endpoint port that was configured.

4. The Azure Load Balancer uses a probe to determine whether or not to send traffic to a particular node. The probe periodically checks an endpoint on each node to determine whether or not the node is responding. If the probe fails to receive a response after a configured number of times, the load balancer stops sending traffic to that node. When creating a cluster through the Azure portal, a probe is automatically set up for each custom endpoint port that was configured.

The screenshot shows the Azure Service Fabric Explorer interface. On the left, the 'Probes' blade is open, displaying a list of probes. The list includes:

NAME	PROT...	PORT	USED BY
AppPortProbe1	TCP	80	1 rules
AppPortProbe2	TCP	8080	1 rules
FabricGatewayProbe	TCP	19000	1 rules
FabricHttpGatewayProbe	TCP	19080	1 rules

On the right, a detailed view of the 'AppPortProbe1' probe is shown. The configuration fields are:

- Name:** AppPortProbe1
- Protocol:** TCP (selected)
- Port:** 80
- Interval:** 5 seconds
- Unhealthy threshold:** 2 consecutive failures

It's important to remember that the Azure Load Balancer and the probe only know about the *nodes*, not the *services* running on the nodes. The Azure Load Balancer will always send traffic to nodes that respond to the probe, so care must be taken to ensure services are available on the nodes that are able to respond to the probe.

## Reliable Services: Built-in communication API options

The Reliable Services framework ships with several pre-built communication options. The decision about which one will work best for you depends on the choice of the programming model, the communication framework, and the programming language that your services are written in.

- **No specific protocol:** If you don't have a particular choice of communication framework, but you want to get something up and running quickly, then the ideal option for you is [service remoting](#), which allows strongly-typed remote procedure calls for Reliable Services and Reliable Actors. This is the easiest and fastest way to get started with service communication. Service remoting handles resolution of service addresses, connection, retry, and error handling. This is available for both C# and Java applications.
- **HTTP:** For language-agnostic communication, HTTP provides an industry-standard choice with tools and HTTP servers available in many different languages, all supported by Service Fabric. Services can use any HTTP stack available, including [ASP.NET Web API](#) for C# applications. Clients written in C# can leverage the `ICommunicationClient` and `ServicePartitionClient` classes, whereas for Java, use the `CommunicationClient` and `FabricServicePartitionClient` classes, [for service resolution, HTTP connections, and retry loops](#).
- **WCF:** If you have existing code that uses WCF as your communication framework, then you can use the `WcfCommunicationListener` for the server side and `WcfCommunicationClient` and `ServicePartitionClient` classes for the client. This however is only available for C# applications on Windows based clusters. For more details, see this article about [WCF-based implementation of the communication stack](#).

## Using custom protocols and other communication frameworks

Services can use any protocol or framework for communication, whether its a custom binary protocol over TCP sockets, or streaming events through [Azure Event Hubs](#) or [Azure IoT Hub](#). Service Fabric provides communication APIs that you can plug your communication stack into, while all the work to discover and connect is abstracted from you. See this article about the [Reliable Service communication model](#) for more details.

## Next steps

Learn more about the concepts and APIs available in the [Reliable Services communication model](#), then get started

quickly with [service remoting](#) or go in-depth to learn how to write a communication listener using [Web API with OWIN self-host](#).

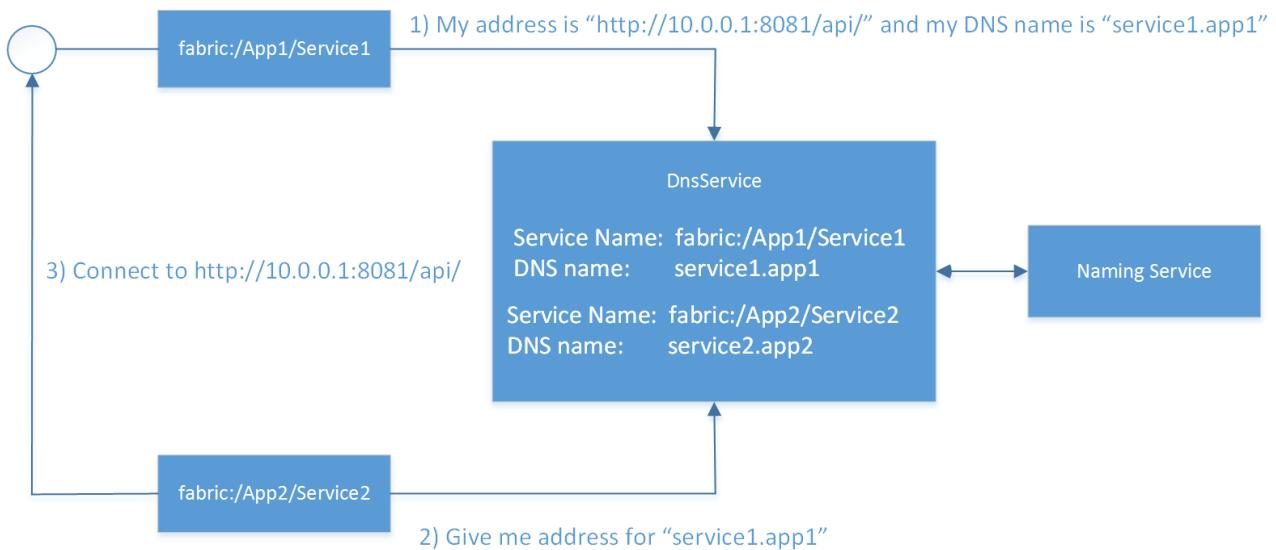
# DNS Service in Azure Service Fabric

8/15/2017 • 3 min to read • [Edit Online](#)

The DNS Service is an optional system service that you can enable in your cluster to discover other services using the DNS protocol.

Many services, especially containerized services, can have an existing URL name, and being able to resolve them using the standard DNS protocol (rather than the Naming Service protocol) is desirable, particularly in "lift and shift" scenarios. The DNS service enables you to map DNS names to a service name and hence resolve endpoint IP addresses.

The DNS service maps DNS names to service names, which in turn are resolved by the Naming Service to return the service endpoint. The DNS name for the service is provided at the time of creation.



## Enabling the DNS service

First you need to enable the DNS service in your cluster. Get the template for the cluster that you want to deploy. You can either use the [sample templates](#) or create a Resource Manager template. You can enable the DNS service with the following steps:

1. Check that the `apiVersion` is set to `2017-07-01-preview` for the `Microsoft.ServiceFabric/clusters` resource, and if not, update it as shown in the following snippet:

```
{  
    "apiVersion": "2017-07-01-preview",  
    "type": "Microsoft.ServiceFabric/clusters",  
    "name": "[parameters('clusterName')]",  
    "location": "[parameters('clusterLocation')]",  
    ...  
}
```

2. Now enable the DNS service by adding the following `addonFeatures` section after the `fabricSettings` section as shown in the following snippet:

```
"fabricSettings": [  
    ...  
,  
    "addonFeatures": [  
        "DnsService"  
    ],
```

- Once you have updated your cluster template with the preceding changes, apply them and let the upgrade complete. Once complete, the DNS system service starts running in your cluster that is called `fabric:/System/DnsService` under system service section in the Service Fabric explorer.

Alternatively, you can enable the DNS service through the portal at the time of cluster creation. The DNS service can be enabled by checking the box for `Include DNS service` in the `Cluster configuration` menu as shown in the following screenshot:

Microsoft Azure New > Create Service Fabric cluster > Cluster configuration

Create Service Fabric cluster Cluster configuration

Cluster configuration  
Configure node types and diagnostics

Node types

Node types define the scale sets that will be used to manage your cluster. You specify between 1 and 3 in the portal. [Learn more about node types](#)

\* Node type count  1  2  3

\* Node type 1 (Primary) [Configure required settings](#)

Diagnostics

Create application log storage  On  Off

Application Insights key (optional)

Add on features

Include DNS service [\(Optional\)](#)

Include repair manager [\(Optional\)](#)

Fabric version

Fabric upgrade mode [\(Optional\)](#)  
 Automatic  Manual

\* Fabric version

Custom fabric settings [\(Optional\)](#) [Enter fabric setting properties](#)

- Hide optional settings

OK

## Setting the DNS name for your service

Once the DNS service is running in your cluster, you can set a DNS name for your services either declaratively for default services in the `ApplicationManifest.xml` or through Powershell commands.

### Setting the DNS name for a default service in the ApplicationManifest.xml

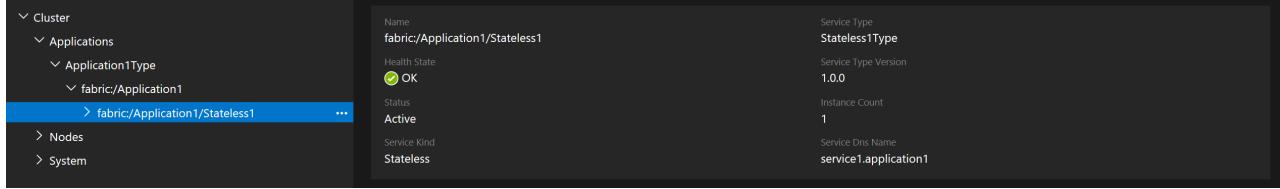
Open your project in Visual Studio, or your favorite editor, and open the `ApplicationManifest.xml` file. Go to the default services section, and for each service add the `ServiceDnsName` attribute. The following example shows how to set the DNS name of the service to `service1.application1`

```

<Service Name="Stateless1" ServiceDnsName="service1.application1">
<StatelessService ServiceTypeName="Stateless1Type" InstanceCount="[Stateless1_InstanceCount]">
    <SingletonPartition />
</StatelessService>
</Service>

```

Once the application is deployed, the service instance in the Service Fabric explorer shows the DNS name for this instance, as shown in the following figure:



### Setting the DNS name for a service using Powershell

You can set the DNS name for a service when creating it using the `New-ServiceFabricService` Powershell. The following example creates a new stateless service with the DNS name `service1.application1`

```

New-ServiceFabricService ` 
-Stateless ` 
-PartitionSchemeSingleton ` 
-ApplicationName `fabric:/application1` 
-ServiceName fabric:/application1/service1` 
-ServiceTypeName Service1Type` 
-InstanceCount 1` 
-ServiceDnsName service1.application1

```

## Using DNS in your services

If you deploy more than one service, you can find the endpoints of other services to communicate with by using a DNS name. The DNS service is only applicable to stateless services, since the DNS protocol cannot communicate with stateful services. For stateful services, you can use the built-in reverse proxy for http calls to call a particular service partition.

The following code shows how to call another service, which is simply a regular http call where you provide the port and any optional path as part of the URL.

```
public class ValuesController : Controller
{
    // GET api
    [HttpGet]
    public async Task<string> Get()
    {
        string result = "";
        try
        {
            Uri uri = new Uri("http://service1.application1:8080/api/values");
            HttpClient client = new HttpClient();
            var response = await client.GetAsync(uri);
            result = await response.Content.ReadAsStringAsync();

        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }

        return result;
    }
}
```

## Next steps

Learn more about service communication within the cluster with [connect and communicate with services](#)

# Reverse proxy in Azure Service Fabric

10/2/2017 • 10 min to read • [Edit Online](#)

Reverse proxy built into Azure Service Fabric helps microservices running in a Service Fabric cluster discover and communicate with other services that have http endpoints.

## Microservices communication model

Microservices in Service Fabric run on a subset of nodes in the cluster and can migrate between the nodes for various reasons. As a result, the endpoints for microservices can change dynamically. To discover and communicate with other services in the cluster, microservice must go through the following steps:

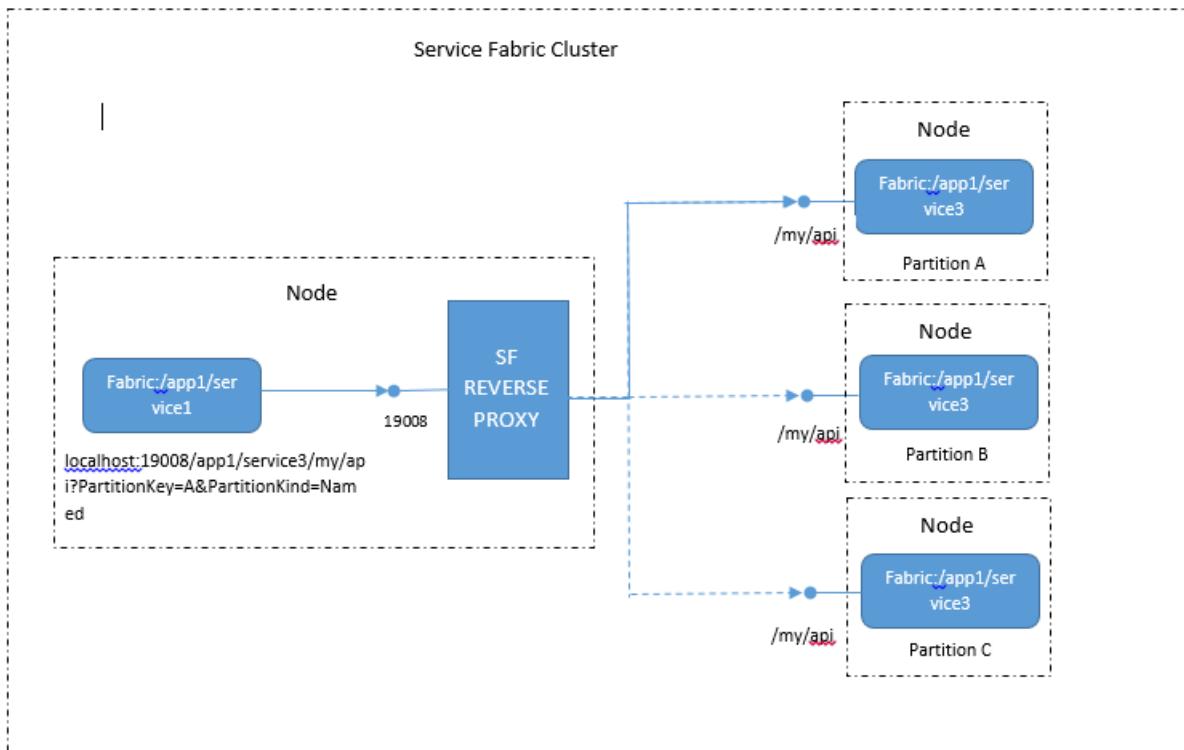
1. Resolve the service location through the naming service.
2. Connect to the service.
3. Wrap the preceding steps in a loop that implements service resolution and retry policies to apply on connection failures

For more information, see [Connect and communicate with services](#).

### Communicating by using the reverse proxy

Reverse proxy is a service that runs on every node and handles endpoint resolution, automatic retry, and other connection failures on behalf of client services. Reverse proxy can be configured to apply various policies as it handles requests from client services. Using a reverse proxy allows the client service to use any client-side HTTP communication libraries and does not require special resolution and retry logic in the service.

Reverse proxy exposes one or more endpoints on local node for client services to use for sending requests to other services.



### Supported Platforms

Reverse proxy in Service Fabric currently supports the following platforms

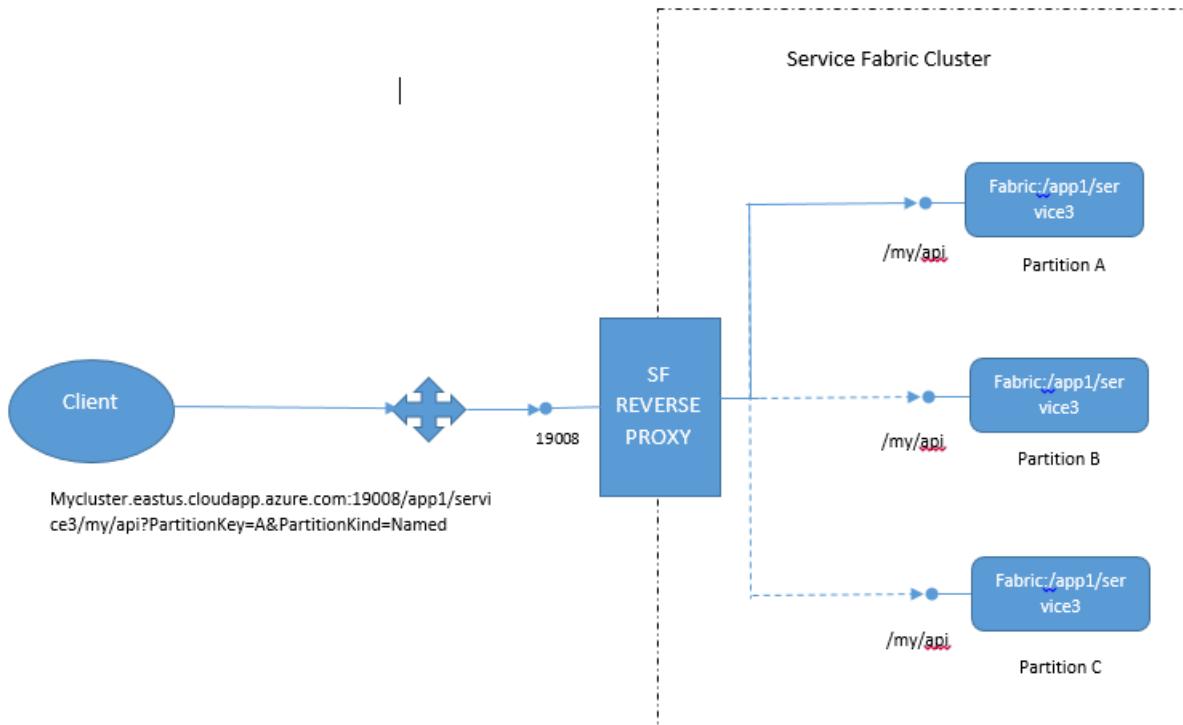
- *Windows Cluster*: Windows 8 and later or Windows Server 2012 and later
- *Linux Cluster*: Reverse Proxy is not currently available for Linux clusters

## Reaching microservices from outside the cluster

The default external communication model for microservices is an opt-in model where each service cannot be accessed directly from external clients. [Azure Load Balancer](#), which is a network boundary between microservices and external clients, performs network address translation and forwards external requests to internal IP:port endpoints. To make a microservice's endpoint directly accessible to external clients, you must first configure Load Balancer to forward traffic to each port that the service uses in the cluster. Furthermore, most microservices, especially stateful microservices, don't live on all nodes of the cluster. The microservices can move between nodes on failover. In such cases, Load Balancer cannot effectively determine the location of the target node of the replicas to which it should forward traffic.

### Reaching microservices via the reverse proxy from outside the cluster

Instead of configuring the port of an individual service in Load Balancer, you can configure just the port of the reverse proxy in Load Balancer. This configuration lets clients outside the cluster reach services inside the cluster by using the reverse proxy without additional configuration.



#### WARNING

When you configure the reverse proxy's port in Load Balancer, all microservices in the cluster that expose an HTTP endpoint are addressable from outside the cluster.

## URI format for addressing services by using the reverse proxy

The reverse proxy uses a specific uniform resource identifier (URI) format to identify the service partition to which the incoming request should be forwarded:

```
http(s)://<Cluster FQDN | internal IP>:Port/<ServiceInstanceName>/<Suffix path>?PartitionKey=<key>&PartitionKind=<partitionkind>&ListenerName=<listenerName>&TargetReplicaSelector=<targetReplicaSelector>&Timeout=<timeout_in_seconds>
```

- **http(s):** The reverse proxy can be configured to accept HTTP or HTTPS traffic. For HTTPS forwarding, refer to [Connect to a secure service with the reverse proxy](#) once you have reverse proxy setup to listen on HTTPS.
- **Cluster fully qualified domain name (FQDN) | internal IP:** For external clients, you can configure the reverse proxy so that it is reachable through the cluster domain, such as mycluster.eastus.cloudapp.azure.com. By default, the reverse proxy runs on every node. For internal traffic, the reverse proxy can be reached on localhost or on any internal node IP, such as 10.0.0.1.
- **Port:** This is the port, such as 19081, that has been specified for the reverse proxy.
- **ServiceInstanceName:** This is the fully-qualified name of the deployed service instance that you are trying to reach without the "fabric://" scheme. For example, to reach the fabric:/myapp/myservice/ service, you would use myapp/myservice.

The service instance name is case-sensitive. Using a different casing for the service instance name in the URL causes the requests to fail with 404 (Not Found).

- **Suffix path:** This is the actual URL path, such as myapi/values/add/3, for the service that you want to connect to.
- **PartitionKey:** For a partitioned service, this is the computed partition key of the partition that you want to reach. Note that this is *not* the partition ID GUID. This parameter is not required for services that use the singleton partition scheme.
- **PartitionKind:** This is the service partition scheme. This can be 'Int64Range' or 'Named'. This parameter is not required for services that use the singleton partition scheme.
- **ListenerName** The endpoints from the service are of the form {"Endpoints": {"Listener1":"Endpoint1","Listener2":"Endpoint2" ...}}. When the service exposes multiple endpoints, this identifies the endpoint that the client request should be forwarded to. This can be omitted if the service has only one listener.
- **TargetReplicaSelector** This specifies how the target replica or instance should be selected.
  - When the target service is stateful, the TargetReplicaSelector can be one of the following: 'PrimaryReplica', 'RandomSecondaryReplica', or 'RandomReplica'. When this parameter is not specified, the default is 'PrimaryReplica'.
  - When the target service is stateless, reverse proxy picks a random instance of the service partition to forward the request to.
- **Timeout:** This specifies the timeout for the HTTP request created by the reverse proxy to the service on behalf of the client request. The default value is 60 seconds. This is an optional parameter.

## Example usage

As an example, let's take the fabric:/MyApp/MyService service that opens an HTTP listener on the following URL:

```
http://10.0.0.5:10592/3f0d39ad-924b-4233-b4a7-02617c6308a6-130834621071472715/
```

Following are the resources for the service:

- /index.html
- /api/users/<userId>

If the service uses the singleton partitioning scheme, the *PartitionKey* and *PartitionKind* query string parameters are not required, and the service can be reached by using the gateway as:

- Externally: http://mycluster.eastus.cloudapp.azure.com:19081/MyApp/MyService

- Internally: `http://localhost:19081/MyApp/MyService`

If the service uses the Uniform Int64 partitioning scheme, the *PartitionKey* and *PartitionKind* query string parameters must be used to reach a partition of the service:

- Externally:

`http://mycluster.eastus.cloudapp.azure.com:19081/MyApp/MyService?PartitionKey=3&PartitionKind=Int64Range`

- Internally: `http://localhost:19081/MyApp/MyService?PartitionKey=3&PartitionKind=Int64Range`

To reach the resources that the service exposes, simply place the resource path after the service name in the URL:

- Externally:

`http://mycluster.eastus.cloudapp.azure.com:19081/MyApp/MyService/index.html?`  
`PartitionKey=3&PartitionKind=Int64Range`

- Internally: `http://localhost:19081/MyApp/MyService/api/users/6?PartitionKey=3&PartitionKind=Int64Range`

The gateway will then forward these requests to the service's URL:

- `http://10.0.0.5:10592/3f0d39ad-924b-4233-b4a7-02617c6308a6-130834621071472715/index.html`
- `http://10.0.0.5:10592/3f0d39ad-924b-4233-b4a7-02617c6308a6-130834621071472715/api/users/6`

## Special handling for port-sharing services

Azure Application Gateway attempts to resolve a service address again and retry the request when a service cannot be reached. This is a major benefit of Application Gateway because client code does not need to implement its own service resolution and resolve loop.

Generally, when a service cannot be reached, the service instance or replica has moved to a different node as part of its normal lifecycle. When this happens, Application Gateway might receive a network connection error indicating that an endpoint is no longer open on the originally resolved address.

However, replicas or service instances can share a host process and might also share a port when hosted by an http.sys-based web server, including:

- [System.Net.HttpListener](#)
- [ASP.NET Core WebListener](#)
- [Katana](#)

In this situation, it is likely that the web server is available in the host process and responding to requests, but the resolved service instance or replica is no longer available on the host. In this case, the gateway will receive an HTTP 404 response from the web server. Thus, an HTTP 404 has two distinct meanings:

- Case #1: The service address is correct, but the resource that the user requested does not exist.
- Case #2: The service address is incorrect, and the resource that the user requested might exist on a different node.

The first case is a normal HTTP 404, which is considered a user error. However, in the second case, the user has requested a resource that does exist. Application Gateway was unable to locate it because the service itself has moved. Application Gateway needs to resolve the address again and retry the request.

Application Gateway thus needs a way to distinguish between these two cases. To make that distinction, a hint from the server is required.

- By default, Application Gateway assumes case #2 and attempts to resolve and issue the request again.
- To indicate case #1 to Application Gateway, the service should return the following HTTP response header:

`X-ServiceFabric : ResourceNotFound`

This HTTP response header indicates a normal HTTP 404 situation in which the requested resource does not exist, and Application Gateway will not attempt to resolve the service address again.

## Setup and configuration

### Enable reverse proxy via Azure portal

Azure portal provides an option to enable Reverse proxy while creating a new Service Fabric cluster. Under **Create Service Fabric cluster**, Step 2: Cluster Configuration, Node type configuration, select the checkbox to "Enable reverse proxy". For configuring secure reverse proxy, SSL certificate can be specified in Step 3: Security, Configure cluster security settings, select the checkbox to "Include a SSL certificate for reverse proxy" and enter the certificate details.

### Enable reverse proxy via Azure Resource Manager templates

You can use the [Azure Resource Manager template](#) to enable the reverse proxy in Service Fabric for the cluster.

Refer to [Configure HTTPS Reverse Proxy in a secure cluster](#) for Azure Resource Manager template samples to configure secure reverse proxy with a certificate and handling certificate rollover.

First, you get the template for the cluster that you want to deploy. You can either use the sample templates or create a custom Resource Manager template. Then, you can enable the reverse proxy by using the following steps:

1. Define a port for the reverse proxy in the [Parameters section](#) of the template.

```
"SFReverseProxyPort": {  
    "type": "int",  
    "defaultValue": 19081,  
    "metadata": {  
        "description": "Endpoint for Service Fabric Reverse proxy"  
    }  
},
```

2. Specify the port for each of the nodetype objects in the [Cluster Resource type section](#).

The port is identified by the parameter name, reverseProxyEndpointPort.

```
{  
    "apiVersion": "2016-09-01",  
    "type": "Microsoft.ServiceFabric/clusters",  
    "name": "[parameters('clusterName')]",  
    "location": "[parameters('clusterLocation')]",  
    ...  
    "nodeTypes": [  
        {  
            ...  
            "reverseProxyEndpointPort": "[parameters('SFReverseProxyPort')]",  
            ...  
        },  
        ...  
    ],  
    ...  
}
```

3. To address the reverse proxy from outside the Azure cluster, set up the Azure Load Balancer rules for the port that you specified in step 1.

```
{
    "apiVersion": "[variables('lbApiVersion')]",
    "type": "Microsoft.Network/loadBalancers",
    ...
    ...
    "loadBalancingRules": [
        ...
        {
            "name": "LBSFReverseProxyRule",
            "properties": {
                "backendAddressPool": {
                    "id": "[variables('lbPoolID0')]"
                },
                "backendPort": "[parameters('SFReverseProxyPort')]",
                "enableFloatingIP": "false",
                "frontendIPConfiguration": {
                    "id": "[variables('lbIPConfig0')]"
                },
                "frontendPort": "[parameters('SFReverseProxyPort')]",
                "idleTimeoutInMinutes": "5",
                "probe": {
                    "id": "[concat(variables('lbID0'), '/probes/SFReverseProxyProbe')]"
                },
                "protocol": "tcp"
            }
        }
    ],
    "probes": [
        ...
        {
            "name": "SFReverseProxyProbe",
            "properties": {
                "intervalInSeconds": 5,
                "numberOfProbes": 2,
                "port": "[parameters('SFReverseProxyPort')]",
                "protocol": "tcp"
            }
        }
    ]
}
```

- To configure SSL certificates on the port for the reverse proxy, add the certificate to the **reverseProxyCertificate** property in the [Cluster Resource type section](#).

```
{
    "apiVersion": "2016-09-01",
    "type": "Microsoft.ServiceFabric/clusters",
    "name": "[parameters('clusterName')]",
    "location": "[parameters('clusterLocation')]",
    "dependsOn": [
        "[concat('Microsoft.Storage/storageAccounts/', parameters('supportLogStorageAccountName'))]"
    ],
    "properties": {
        ...
        "reverseProxyCertificate": {
            "thumbprint": "[parameters('sfReverseProxyCertificateThumbprint')]",
            "x509StoreName": "[parameters('sfReverseProxyCertificateStoreName')]"
        },
        ...
        "clusterState": "Default",
    }
}
```

## Supporting a reverse proxy certificate that's different from the cluster certificate

If the reverse proxy certificate is different from the certificate that secures the cluster, then the previously specified certificate should be installed on the virtual machine and added to the access control list (ACL) so that Service Fabric can access it. This can be done in the [virtualMachineScaleSets Resource type section](#). For installation, add that certificate to the osProfile. The extension section of the template can update the certificate in the ACL.

```
{  
    "apiVersion": "[variables('vmssApiVersion')]",  
    "type": "Microsoft.Compute/virtualMachineScaleSets",  
    ...  
    "osProfile": {  
        "adminPassword": "[parameters('adminPassword')]",  
        "adminUsername": "[parameters('adminUsername')]",  
        "computerNamePrefix": "[parameters('vmNodeType0Name')]",  
        "secrets": [  
            {  
                "sourceVault": {  
                    "id": "[parameters('sfReverseProxySourceVaultValue')]"  
                },  
                "vaultCertificates": [  
                    {  
                        "certificateStore": "[parameters('sfReverseProxyCertificateStoreValue')]",  
                        "certificateUrl": "[parameters('sfReverseProxyCertificateUrlValue')]"  
                    }  
                ]  
            }  
        ]  
    }  
    ...  
    "extensions": [  
        {  
            "name": "[concat(parameters('vmNodeType0Name'), '_ServiceFabricNode')]",  
            "properties": {  
                "type": "ServiceFabricNode",  
                "autoUpgradeMinorVersion": false,  
                ...  
                "publisher": "Microsoft.Azure.ServiceFabric",  
                "settings": {  
                    "clusterEndpoint": "[reference(parameters('clusterName')).clusterEndpoint]",  
                    "nodeTypeRef": "[parameters('vmNodeType0Name')]",  
                    "dataPath": "D:\\\\\\SvcFab",  
                    "durabilityLevel": "Bronze",  
                    "testExtension": true,  
                    "reverseProxyCertificate": {  
                        "thumbprint": "[parameters('sfReverseProxyCertificateThumbprint')]",  
                        "x509StoreName": "[parameters('sfReverseProxyCertificateStoreValue')]"  
                    },  
                    ...  
                },  
                "typeHandlerVersion": "1.0"  
            }  
        },  
    ]  
}
```

#### NOTE

When you use certificates that are different from the cluster certificate to enable the reverse proxy on an existing cluster, install the reverse proxy certificate and update the ACL on the cluster before you enable the reverse proxy. Complete the [Azure Resource Manager template](#) deployment by using the settings mentioned previously before you start a deployment to enable the reverse proxy in steps 1-4.

## Next steps

- See an example of HTTP communication between services in a [sample project on GitHub](#).
- [Forwarding to secure HTTP service with the reverse proxy](#)
- [Remote procedure calls with Reliable Services remoting](#)
- [Web API that uses OWIN in Reliable Services](#)
- [WCF communication by using Reliable Services](#)
- For additional reverse proxy configuration options, refer ApplicationGateway/Http section in [Customize Service Fabric cluster settings](#).

# Connect to a secure service with the reverse proxy

8/11/2017 • 5 min to read • [Edit Online](#)

This article explains how to establish secure connection between the reverse proxy and services, thus enabling an end to end secure channel.

Connecting to secure services is supported only when reverse proxy is configured to listen on HTTPS. Rest of the document assumes this is the case. Refer to [Reverse proxy in Azure Service Fabric](#) to configure the reverse proxy in Service Fabric.

## Secure connection establishment between the reverse proxy and services

### Reverse proxy authenticating to services:

The reverse proxy identifies itself to services using its certificate, specified with **reverseProxyCertificate** property in the [Cluster Resource type section](#). Services can implement the logic to verify the certificate presented by the reverse proxy. The services can specify the accepted client certificate details as configuration settings in the configuration package. This can be read at runtime and used to validate the certificate presented by the reverse proxy. Refer to [Manage application parameters](#) to add the configuration settings.

### Reverse proxy verifying the service's identity via the certificate presented by the service:

To perform server certificate validation of the certificates presented by the services, reverse proxy supports one of the following options: None, ServiceCommonNameAndIssuer, and ServiceCertificateThumbprints. To select one of these three options, specify the **ApplicationCertificateValidationPolicy** in the parameters section of ApplicationGateway/Http element under [fabricSettings](#).

```
{
  "fabricSettings": [
    ...
    {
      "name": "ApplicationGateway/Http",
      "parameters": [
        {
          "name": "ApplicationCertificateValidationPolicy",
          "value": "None"
        }
      ]
    ],
    ...
  }
}
```

Refer to the next section for details about additional configuration for each of these options.

### Service certificate validation options

- **None**: Reverse proxy skips verification of the proxied service certificate and establishes the secure connection. This is the default behavior. Specify the **ApplicationCertificateValidationPolicy** with value **None** in the parameters section of ApplicationGateway/Http element.
- **ServiceCommonNameAndIssuer**: Reverse proxy verifies the certificate presented by the service based on certificate's common name and immediate issuer's thumbprint: Specify the **ApplicationCertificateValidationPolicy** with value **ServiceCommonNameAndIssuer** in the parameters

section of ApplicationGateway/Http element.

```
{  
  "fabricSettings": [  
    ...  
    {  
      "name": "ApplicationGateway/Http",  
      "parameters": [  
        {  
          "name": "ApplicationCertificateValidationPolicy",  
          "value": "ServiceCommonNameAndIssuer"  
        }  
      ]  
    },  
    ...  
  ]  
}
```

To specify the list of service common name and issuer thumbprints, add a

**ApplicationGateway/Http/ServiceCommonNameAndIssuer** element under fabricSettings, as shown below.

Multiple certificate common name and issuer thumbprint pairs can be added in the parameters array element.

If the endpoint reverse proxy is connecting to presents a certificate who's common name and issuer thumbprint matches any of the values specified here, SSL channel is established. Upon failure to match the certificate details, reverse proxy fails the client's request with a 502 (Bad Gateway) status code. The HTTP status line will also contain the phrase "Invalid SSL Certificate."

```
{  
  "fabricSettings": [  
    ...  
    {  
      "name": "ApplicationGateway/Http/ServiceCommonNameAndIssuer",  
      "parameters": [  
        {  
          "name": "WinFabric-Test-Certificate-CN1",  
          "value": "b3 44 9b 01 8d 0f 68 39 a2 c5 d6 2b 5b 6c 6a c8 22 b4 22 11"  
        },  
        {  
          "name": "WinFabric-Test-Certificate-CN2",  
          "value": "b3 44 9b 01 8d 0f 68 39 a2 c5 d6 2b 5b 6c 6a c8 22 11 33 44"  
        }  
      ]  
    },  
    ...  
  ]  
}
```

- **ServiceCertificateThumbprints:** Reverse proxy will verify the proxied service certificate based on its thumbprint. You can choose to go this route when the services are configured with self signed certificates: Specify the **ApplicationCertificateValidationPolicy** with value **ServiceCertificateThumbprints** in the parameters section of ApplicationGateway/Http element.

```
{
  "fabricSettings": [
    ...
    {
      "name": "ApplicationGateway/Http",
      "parameters": [
        {
          "name": "ApplicationCertificateValidationPolicy",
          "value": "ServiceCertificateThumbprints"
        }
      ]
    },
    ...
  ]
}
```

Also specify the thumbprints with a **ServiceCertificateThumbprints** entry under parameters section of ApplicationGateway/Http element. Multiple thumbprints can be specified as a comma-separated list in the value field, as shown below:

```
{
  "fabricSettings": [
    ...
    {
      "name": "ApplicationGateway/Http",
      "parameters": [
        ...
        {
          "name": "ServiceCertificateThumbprints",
          "value": "78 12 20 5a 39 d2 23 76 da a0 37 f0 5a ed e3 60 1a 7e 64 bf,78 12 20 5a 39 d2 23 76
da a0 37 f0 5a ed e3 60 1a 7e 64 b9"
        }
      ]
    },
    ...
  ]
}
```

If the thumbprint of the server certificate is listed in this config entry, reverse proxy succeeds the SSL connection. Otherwise, it terminates the connection and fails the client's request with a 502 (Bad Gateway). The HTTP status line will also contain the phrase "Invalid SSL Certificate."

## Endpoint selection logic when services expose secure as well as unsecured endpoints

Service fabric supports configuring multiple endpoints for a service. See [Specify resources in a service manifest](#).

Reverse proxy selects one of the endpoints to forward the request based on the **ListenerName** query parameter. If this is not specified, it can pick any endpoint from the endpoints list. Now this can be an HTTP or HTTPS endpoint. There might be scenarios/requirements where you want the reverse proxy to operate in a "secure only mode", i.e you don't want the secure reverse proxy to forward requests to unsecured endpoints. This can be achieved by specifying the **SecureOnlyMode** configuration entry with value **true** in the parameters section of ApplicationGateway/Http element.

```
{
  "fabricSettings": [
    ...
    {
      "name": "ApplicationGateway/Http",
      "parameters": [
        ...
        {
          "name": "SecureOnlyMode",
          "value": true
        }
      ]
    },
    ...
  ]
}
```

When operating in **SecureOnlyMode**, if client has specified a **ListenerName** corresponding to an HTTP(unsecured) endpoint, reverse proxy fails the request with a 404 (Not Found) HTTP status code.

## Setting up client certificate authentication through the reverse proxy

SSL termination happens at the reverse proxy and all the client certificate data is lost. For the services to perform client certificate authentication, set the **ForwardClientCertificate** setting in the parameters section of ApplicationGateway/Http element.

1. When **ForwardClientCertificate** is set to **false**, reverse proxy will not request for the client certificate during its SSL handshake with the client. This is the default behavior.
2. When **ForwardClientCertificate** is set to **true**, reverse proxy requests for the client's certificate during its SSL handshake with the client. It will then forward the client certificate data in a custom HTTP header named **X-Client-Certificate**. The header value is the base64 encoded PEM format string of the client's certificate. The service can succeed/fail the request with appropriate status code after inspecting the certificate data. If the client does not present a certificate, reverse proxy forwards an empty header and let the service handle the case.

Reverse proxy is a mere forwarder. It will not perform any validation of the client's certificate.

## Next steps

- Refer to [Configure reverse proxy to connect to secure services](#) for Azure Resource Manager template samples to configure secure reverse proxy with the different service certificate validation options.
- See an example of HTTP communication between services in a [sample project on GitHub](#).
- [Remote procedure calls with Reliable Services remoting](#)
- [Web API that uses OWIN in Reliable Services](#)
- [Manage cluster certificates](#)

# Monitor and diagnose request processing at the reverse proxy

10/2/2017 • 5 min to read • [Edit Online](#)

Starting with the 5.7 release of Service Fabric, reverse proxy events are available for collection. The events are available in two channels, one with only error events related to request processing failure at the reverse proxy and second channel containing verbose events with entries for both successful and failed requests.

Refer to [Collect reverse proxy events](#) to enable collecting events from these channels in local and Azure Service Fabric clusters.

## Troubleshoot using diagnostics logs

Here are some examples on how to interpret the common failure logs that one can encounter:

1. Reverse proxy returns response status code 504 (Timeout).

One reason could be due to the service failing to reply within the request timeout period. The first event below logs the details of the request received at the reverse proxy. The second event indicates that the request failed while forwarding to service, due to "internal error = ERROR\_WINHTTP\_TIMEOUT"

The payload includes:

- **traceId:** This GUID can be used to correlate all the events corresponding to a single request. In the below two events, the traceId = **2f87b722-e254-4ac2-a802-fd315c1a0271**, implying they belong to the same request.
- **requestUrl:** The URL (Reverse proxy URL) to which the request was sent.
- **verb:** HTTP verb.
- **remoteAddress:** Address of client sending the request.
- **resolvedServiceUrl:** Service endpoint URL to which the incoming request was resolved.
- **errorDetails:** Additional information about the failure.

```
{
  "Timestamp": "2017-07-20T15:57:59.9871163-07:00",
  "ProviderName": "Microsoft-ServiceFabric",
  "Id": 51477,
  "Message": "2f87b722-e254-4ac2-a802-fd315c1a0271 Request url = https://localhost:19081/LocationApp/LocationFEService?zipcode=98052, verb = GET, remote (client) address = ::1, resolved service url = Https://localhost:8491/LocationApp/?zipcode=98052, request processing start time = 15:58:00.074114 (745,608.196 MSec) ",
  "ProcessId": 57696,
  "Level": "Informational",
  "Keywords": "0x1000000000000021",
  "EventName": "ReverseProxy",
  "ActivityID": null,
  "RelatedActivityID": null,
  "Payload": {
    "traceId": "2f87b722-e254-4ac2-a802-fd315c1a0271",
    "requestUrl": "https://localhost:19081/LocationApp/LocationFEService?zipcode=98052",
    "verb": "GET",
    "remoteAddress": "::1",
    "resolvedServiceUrl": "Https://localhost:8491/LocationApp/?zipcode=98052",
    "requestStartTime": "2017-07-20T15:58:00.0741142-07:00"
  }
}

{
  "Timestamp": "2017-07-20T16:00:01.3173605-07:00",
  ...
  "Message": "2f87b722-e254-4ac2-a802-fd315c1a0271 Error while forwarding request to service: response status code = 504, description = Reverse proxy Timeout, phase = FinishSendRequest, internal error = ERROR_WINHTTP_TIMEOUT",
  ...
  "Payload": {
    "traceId": "2f87b722-e254-4ac2-a802-fd315c1a0271",
    "statusCode": 504,
    "description": "Reverse Proxy Timeout",
    "sendRequestPhase": "FinishSendRequest",
    "errorDetails": "internal error = ERROR_WINHTTP_TIMEOUT"
  }
}
```

## 2. Reverse proxy returns response status code 404 (Not Found).

Here is an example event where reverse proxy returns 404 since it failed to find the matching service endpoint. The payload entries of interest here are:

- **processRequestPhase**: Indicates the phase during request processing when the failure occurred, **TryGetEndpoint** i.e while trying to fetch the service endpoint to forward to.
- **errorDetails**: Lists the endpoint search criteria. Here you can see that the **listenerName** specified = **FrontEndListener** whereas the replica endpoint list only contains a listener with the name **OldListener**.

```
{
...
"Message": "c1cca3b7-f85d-4fef-a162-88af23604343 Error while processing request, cannot forward to service: request url = https://localhost:19081/LocationApp/LocationFEService? ListenerName=FrontEndListener&zipcode=98052, verb = GET, remote (client) address = ::1, request processing start time = 16:43:02.686271 (3,448,220.353 MSec), error = FABRIC_E_ENDPOINT_NOT_FOUND, message = , phase = TryGetEndpoint, SecureOnlyMode = false, gateway protocol = https, listenerName = FrontEndListener, replica endpoint = {"Endpoints": {"\\\"": "Https://localhost:8491/LocationApp\\\""}, "ProcessId": 57696, "Level": "Warning", "EventName": "ReverseProxy", "Payload": { "traceId": "c1cca3b7-f85d-4fef-a162-88af23604343", "requestUrl": "https://localhost:19081/LocationApp/LocationFEService? ListenerName>NewListener&zipcode=98052", ... "processRequestPhase": "TryGetEndpoint", "errorDetails": "SecureOnlyMode = false, gateway protocol = https, listenerName = FrontEndListener, replica endpoint = {"Endpoints": {"\\\"OldListener\\\"": "Https://localhost:8491/LocationApp\\\""}, "}
}
}
```

Another example where reverse proxy can return 404 Not Found is: ApplicationGateway\Http configuration parameter **SecureOnlyMode** is set to true with the reverse proxy listening on **HTTPS**, however all of the replica endpoints are unsecure (listening on HTTP). Reverse proxy returns 404 since it cannot find an endpoint listening on HTTPS to forward the request. Analyzing the parameters in the event payload helps to narrow down the issue:

```
"errorDetails": "SecureOnlyMode = true, gateway protocol = https, listenerName = NewListener, replica endpoint = {"Endpoints": {"\\\"OldListener\\\"": "Http://localhost:8491/LocationApp\\\", \"NewListener\\\"": "Http://localhost:8492/LocationApp\\\""}, "
```

3. Request to the reverse proxy fails with a timeout error. The event logs contain an event with the received request details (not shown here). The next event shows that the service responded with a 404 status code and reverse proxy initiates a re-resolve.

```
{
...
"Message": "7ac6212c-c8c4-4c98-9cf7-c187a94f141e Request to service returned: status code = 404, status description = , Reresolving ", "Payload": { "traceId": "7ac6212c-c8c4-4c98-9cf7-c187a94f141e", "statusCode": 404, "statusDescription": "" }, {
...
"Message": "7ac6212c-c8c4-4c98-9cf7-c187a94f141e Re-resolved service url = Https://localhost:8491/LocationApp/?zipcode=98052 ", "Payload": { "traceId": "7ac6212c-c8c4-4c98-9cf7-c187a94f141e", "requestUrl": "Https://localhost:8491/LocationApp/?zipcode=98052" }, }
```

When collecting all the events, you see a train of events showing every resolve and forward attempt. The last event in the series shows the request processing has failed with a timeout, along with the number of

successful resolve attempts.

#### NOTE

It is recommended to keep the verbose channel event collection disabled by default and enable it for troubleshooting on a need basis.

```
{  
  ...  
  "Message": "7ac6212c-c8c4-4c98-9cf7-c187a94f141e Error while processing request: number of successful  
  resolve attempts = 12, error = FABRIC_E_TIMEOUT, message = , phase = ResolveServicePartition, ",  
  "EventName": "ReverseProxy",  
  ...  
  "Payload": {  
    "traceId": "7ac6212c-c8c4-4c98-9cf7-c187a94f141e",  
    "resolveCount": 12,  
    "errorval": -2147017729,  
    "errorMessage": "",  
    "processRequestPhase": "ResolveServicePartition",  
    "errorDetails": ""  
  }  
}
```

If collection is enabled for critical/error events only, you see one event with details about the timeout and the number of resolve attempts.

Services that intend to send a 404 status code back to the user, should add an "X-ServiceFabric" header in the response. After the header is added to the response, reverse proxy forwards the status code back to the client.

#### 4. Cases when the client has disconnected the request.

Following event is recorded when reverse proxy is forwarding the response to client but the client disconnects:

```
{  
  ...  
  "Message": "6e2571a3-14a8-4fc7-93bb-c202c23b50b8 Unable to send response to client: phase =  
  SendResponseHeaders, error = -805306367, internal error = ERROR_SUCCESS ",  
  "ProcessId": 57696,  
  "Level": "Warning",  
  ...  
  "EventName": "ReverseProxy",  
  "Payload": {  
    "traceId": "6e2571a3-14a8-4fc7-93bb-c202c23b50b8",  
    "sendResponsePhase": "SendResponseHeaders",  
    "errorval": -805306367,  
    "winHttpError": "ERROR_SUCCESS"  
  }  
}
```

#### 5. Reverse Proxy returns 404 FABRIC\_E\_SERVICE\_DOES\_NOT\_EXIST

FABRIC\_E\_SERVICE\_DOES\_NOT\_EXIST error is returned if the URI scheme is not specified for the service endpoint in the service manifest.

```
<Endpoint Name="ServiceEndpointHttp" Port="80" Protocol="http" Type="Input"/>
```

To resolve the problem, specify the URI scheme in the manifest.

```
<Endpoint Name="ServiceEndpointHttp" UriScheme="http" Port="80" Protocol="http" Type="Input"/>
```

#### NOTE

Events related to websocket request processing are not currently logged. This will be added in the next release.

## Next steps

- [Event aggregation and collection using Windows Azure Diagnostics](#) for enabling log collection in Azure clusters.
- To view Service Fabric events in Visual Studio, see [Monitor and diagnose locally](#).
- Refer to [Configure reverse proxy to connect to secure services](#) for Azure Resource Manager template samples to configure secure reverse proxy with the different service certificate validation options.
- Read [Service Fabric reverse proxy](#) to learn more.

# Scaling in Service Fabric

8/23/2017 • 14 min to read • [Edit Online](#)

Azure Service Fabric makes it easy to build scalable applications by managing the services, partitions, and replicas on the nodes of a cluster. Running many workloads on the same hardware enables maximum resource utilization, but also provides flexibility in terms of how you choose to scale your workloads.

Scaling in Service Fabric is accomplished several different ways:

1. Scaling by creating or removing stateless service instances
2. Scaling by creating or removing new named services
3. Scaling by creating or removing new named application instances
4. Scaling by using partitioned services
5. Scaling by adding and removing nodes from the cluster
6. Scaling by using Cluster Resource Manager metrics

## Scaling by creating or removing stateless service instances

One of the simplest ways to scale within Service Fabric works with stateless services. When you create a stateless service, you get a chance to define an `InstanceCount`. `InstanceCount` defines how many running copies of that service's code are created when the service starts up. Let's say, for example, that there are 100 nodes in the cluster. Let's also say that a service is created with an `InstanceCount` of 10. During runtime, those 10 running copies of the code could all become too busy (or could be not busy enough). One way to scale that workload is to change the number of instances. For example, some piece of monitoring or management code can change the existing number of instances to 50, or to 5, depending on whether the workload needs to scale in or out based on the load.

C#:

```
StatelessServiceUpdateDescription updateDescription = new StatelessServiceUpdateDescription();
updateDescription.InstanceCount = 50;
await fabricClient.ServiceManager.UpdateServiceAsync(new Uri("fabric:/app/service"), updateDescription);
```

Powershell:

```
Update-ServiceFabricService -Stateless -ServiceName $serviceName -InstanceCount 50
```

### Using Dynamic Instance Count

Specifically for stateless services, Service Fabric offers an automatic way to change the instance count. This allows the service to scale dynamically with the number of nodes that are available. The way to opt into this behavior is to set the instance count = -1. `InstanceCount = -1` is an instruction to Service Fabric that says "Run this stateless service on every node." If the number of nodes changes, Service Fabric automatically changes the instance count to match, ensuring that the service is running on all valid nodes.

C#:

```
StatelessServiceDescription serviceDescription = new StatelessServiceDescription();
//Set other service properties necessary for creation....
serviceDescription.InstanceCount = -1;
await fc.ServiceManager.CreateServiceAsync(serviceDescription);
```

Powershell:

```
New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName
$serviceTypeName -Stateless -PartitionSchemeSingleton -InstanceCount "-1"
```

## Scaling by creating or removing new named services

A named service instance is a specific instance of a service type (see [Service Fabric application life cycle](#)) within some named application instance in the cluster.

New named service instances can be created (or removed) as services become more or less busy. This allows requests to be spread across more service instances, usually allowing load on existing services to decrease. When creating services, the Service Fabric Cluster Resource Manager places the services in the cluster in a distributed fashion. The exact decisions are governed by the [metrics](#) in the cluster and other placement rules. Services can be created several different ways, but the most common are either through administrative actions like someone calling [New-ServiceFabricService](#), or by code calling [CreateServiceAsync](#). [CreateServiceAsync](#) can even be called from within other services running in the cluster.

Creating services dynamically can be used in all sorts of scenarios, and is a common pattern. For example, consider a stateful service that represents a particular workflow. Calls representing work are going to show up to this service, and this service is going to execute the steps to that workflow and record progress.

How would you make this particular service scale? The service could be multi-tenant in some form, and accept calls and kick off steps for many different instances of the same workflow all at once. However, that can make the code more complex, since now it has to worry about many different instances of the same workflow, all at different stages and from different customers. Also, handling multiple workflows at the same time doesn't solve the scale problem. This is because at some point this service will consume too many resources to fit on a particular machine. Many services not built for this pattern in the first place also experience difficulty due to some inherent bottleneck or slowdown in their code. These types of issues cause the service not to work as well when the number of concurrent workflows it is tracking gets larger.

A solution is to create an instance of this service for every different instance of the workflow you want to track. This is a great pattern and works whether the service is stateless or stateful. For this pattern to work, there's usually another service that acts as a "Workload Manager Service". The job of this service is to receive requests and to route those requests to other services. The manager can dynamically create an instance of the workload service when it receives the message, and then pass on requests to those services. The manager service could also receive callbacks when a given workflow service completes its job. When the manager receives these callbacks it could delete that instance of the workflow service, or leave it if more calls are expected.

Advanced versions of this type of manager can even create pools of the services that it manages. The pool helps ensure that when a new request comes in it doesn't have to wait for the service to spin up. Instead, the manager can just pick a workflow service that is not currently busy from the pool, or route randomly. Keeping a pool of services available makes handling new requests faster, since it is less likely that the request has to wait for a new service to be spun up. Creating new services is quick, but not free or instantaneous. The pool helps minimize the amount of time the request has to wait before being serviced. You'll often see this manager and pool pattern when response times matter the most. Queuing the request and creating the service in the background and then passing it on is also a popular manager pattern, as is creating and deleting services based on some tracking of the amount of work that service currently has pending.

# Scaling by creating or removing new named application instances

Creating and deleting whole application instances is similar to the pattern of creating and deleting services. For this pattern, there's some manager service that is making the decision based on the requests that it is seeing and the information it is receiving from the other services inside the cluster.

When should creating a new named application instance be used instead of creating a new named service instances in some already existing application? There's a few cases:

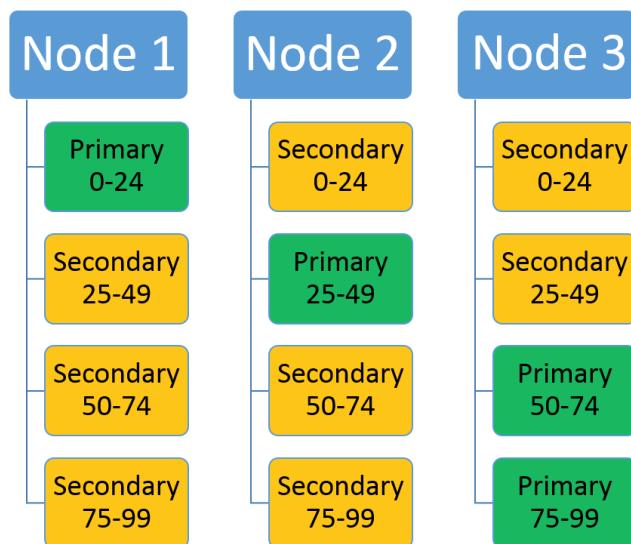
- The new application instance is for a customer whose code needs to run under some particular identity or security settings.
  - Service Fabric allows defining different code packages to run under particular identities. In order to launch the same code package under different identities, the activations need to occur in different application instances. Consider a case where you have an existing customer's workloads deployed. These may be running under a particular identity so you can monitor and control their access to other resources, such as remote databases or other systems. In this case, when a new customer signs up, you probably don't want to activate their code in the same context (process space). While you could, this makes it harder for your service code to act within the context of a particular identity. You typically must have more security, isolation, and identity management code. Instead of using different named service instances within the same application instance and hence the same process space, you can use different named Service Fabric Application instances. This makes it easier to define different identity contexts.
- The new application instance also serves as a means of configuration
  - By default, all of the named service instances of a particular service type within an application instance will run in the same process on a given node. What this means is that while you can configure each service instance differently, doing so is complicated. Services must have some token they use to look up their config within a configuration package. Usually this is just the service's name. This works fine, but it couples the configuration to the names of the individual named service instances within that application instance. This can be confusing and hard to manage since configuration is normally a design time artifact with application instance specific values. Creating more services always means more application upgrades to change the information within the config packages or to deploy new ones so that the new services can look up their specific information. It's often easier to create a whole new named application instance. Then you can use the application parameters to set whatever configuration is necessary for the services. This way all of the services that are created within that named application instance can inherit particular configuration settings. For example, instead of having a single configuration file with the settings and customizations for every customer, such as secrets or per customer resource limits, you'd instead have a different application instance for each customer with these settings overridden.
- The new application serves as an upgrade boundary
  - Within Service Fabric, different named application instances serve as boundaries for upgrade. An upgrade of one named application instance will not impact the code that another named application instance is running. The different applications will end up running different versions of the same code on the same nodes. This can be a factor when you need to make a scaling decision because you can choose whether the new code should follow the same upgrades as another service or not. For example, say that a call arrives at the manager service that is responsible for scaling a particular customer's workloads by creating and deleting services dynamically. In this case however, the call is for a workload associated with a *new* customer. Most customers like being isolated from each other not just for the security and configuration reasons listed previously, but because it provides more flexibility in terms of running specific versions of the software and choosing when they get upgraded. You may also create a new application instance and create the service there simply to further partition the amount of your services that any one upgrade will touch. Separate application instances provide greater granularity when doing application upgrades, and also enable A/B testing and Blue/Green deployments.
- The existing application instance is full
  - In Service Fabric, [application capacity](#) is a concept that you can use to control the amount of resources

available for particular application instances. For example, you may decide that a given service needs to have another instance created in order to scale. However, this application instance is out of capacity for a certain metric. If this particular customer or workload should still be granted more resources, then you could either increase the existing capacity for that application or create a new application.

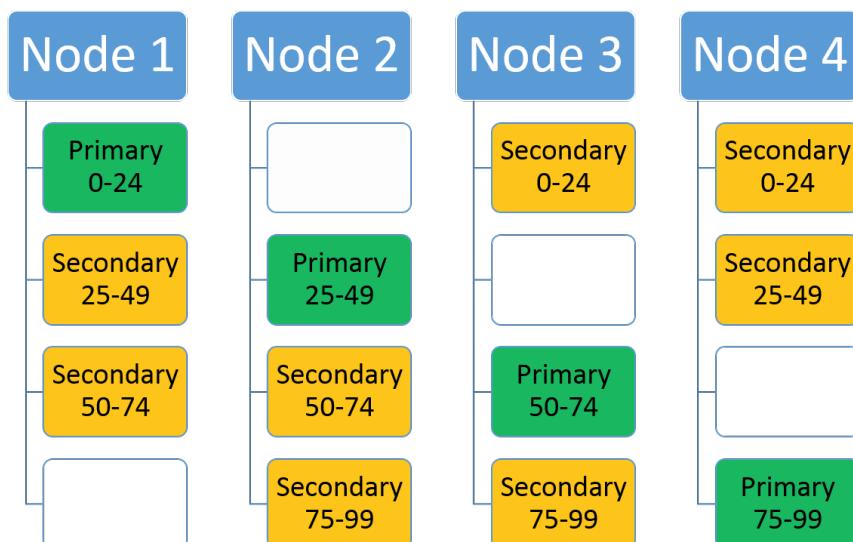
## Scaling at the partition level

Service Fabric supports partitioning. Partitioning splits a service into several logical and physical sections, each of which operates independently. This is useful with stateful services, since no one set of replicas has to handle all the calls and manipulate all of the state at once. The [partitioning overview](#) provides information on the types of partitioning schemes that are supported. The replicas of each partition are spread across the nodes in a cluster, distributing that service's load and ensuring that neither the service as a whole or any partition has a single point of failure.

Consider a service that uses a ranged partitioning scheme with a low key of 0, a high key of 99, and a partition count of 4. In a three-node cluster, the service might be laid out with four replicas that share the resources on each node as shown here:



If you increase the number of nodes, Service Fabric will move some of the existing replicas there. For example, let's say the number of nodes increases to four and the replicas get redistributed. Now the service now has three replicas running on each node, each belonging to different partitions. This allows better resource utilization since the new node isn't cold. Typically, it also improves performance as each service has more resources available to it.



## Scaling by using the Service Fabric Cluster Resource Manager and

## metrics

Metrics are how services express their resource consumption to Service Fabric. Using metrics gives the Cluster Resource Manager an opportunity to reorganize and optimize the layout of the cluster. For example, there may be plenty of resources in the cluster, but they might not be allocated to the services that are currently doing work. Using metrics allows the Cluster Resource Manager to reorganize the cluster to ensure that services have access to the available resources.

## Scaling by adding and removing nodes from the cluster

Another option for scaling with Service Fabric is to change the size of the cluster. Changing the size of the cluster means adding or removing nodes for one or more of the node types in the cluster. For example, consider a case where all of the nodes in the cluster are hot. This means that the cluster's resources are almost all consumed. In this case, adding more nodes to the cluster is the best way to scale. Once the new nodes join the cluster the Service Fabric Cluster Resource Manager moves services to them, resulting in less total load on the existing nodes. For stateless services with instance count = -1, more service instances are automatically created. This allows some calls to move from the existing nodes to the new nodes.

Adding and removing nodes to the cluster can be accomplished via the Service Fabric Azure Resource Manager PowerShell module.

```
Add-AzureRmServiceFabricNode -ResourceGroupName $resourceGroupName -Name $clusterResourceName -NodeType  
$nodeTypeName -NumberOfNodesToAdd 5  
Remove-AzureRmServiceFabricNode -ResourceGroupName $resourceGroupName -Name $clusterResourceName -NodeType  
$nodeTypeName -NumberOfNodesToRemove 5
```

## Putting it all together

Let's take all the ideas that we've discussed here and talk through an example. Consider the following service: you are trying to build a service that acts as an address book, holding on to names and contact information.

Right up front you have a bunch of questions related to scale: How many users are you going to have? How many contacts will each user store? Trying to figure this all out when you are standing up your service for the first time is difficult. Let's say you were going to go with a single static service with a specific partition count. The consequences of picking the wrong partition count could cause you to have scale issues later. Similarly, even if you pick the right count you might not have all the information you need. For example, you also have to decide the size of the cluster up front, both in terms of the number of nodes and their sizes. It's usually hard to predict how many resources a service is going to consume over its lifetime. It can also be hard to know ahead of time the traffic pattern that the service actually sees. For example, maybe people add and remove their contacts only first thing in the morning, or maybe it's distributed evenly over the course of the day. Based on this you might need to scale out and in dynamically. Maybe you can learn to predict when you're going to need to scale out and in, but either way you're probably going to need to react to changing resource consumption by your service. This can involve changing the size of the cluster in order to provide more resources when reorganizing use of existing resources isn't enough.

But why even try to pick a single partition scheme out for all users? Why limit yourself to one service and one static cluster? The real situation is usually more dynamic.

When building for scale, consider the following dynamic pattern. You may need to adapt it to your situation:

1. Instead of trying to pick a partitioning scheme for everyone up front, build a "manager service".
2. The job of the manager service is to look at customer information when they sign up for your service. Then depending on that information the manager service create an instance of your *actual* contact-storage service *just for that customer*. If they require particular configuration, isolation, or upgrades, you can also decide to

spin up an Application instance for this customer.

This dynamic creation pattern many benefits:

- You're not trying to guess the correct partition count for all users up front or build a single service that is infinitely scalable all on its own.
- Different users don't have to have the same partition count, replica count, placement constraints, metrics, default loads, service names, dns settings, or any of the other properties specified at the service or application level.
- You gain additional data segmentation. Each customer has their own copy of the service
  - Each customer service can be configured differently and granted more or fewer resources, with more or fewer partitions or replicas as necessary based on their expected scale.
    - For example, say the customer paid for the "Gold" tier - they could get more replicas or greater partition count, and potentially resources dedicated to their services via metrics and application capacities.
    - Or say they provided information indicating the number of contacts they needed was "Small" - they would get only a few partitions, or could even be put into a shared service pool with other customers.
- You're not running a bunch of service instances or replicas while you're waiting for customers to show up
- If a customer ever leaves, then removing their information from your service is as simple as having the manager delete that service or application that it created.

## Next steps

For more information on Service Fabric concepts, see the following articles:

- [Availability of Service Fabric services](#)
- [Partitioning Service Fabric services](#)

# ASP.NET Core in Service Fabric Reliable Services

6/27/2017 • 15 min to read • [Edit Online](#)

ASP.NET Core is a new open-source and cross-platform framework for building modern cloud-based Internet-connected applications, such as web apps, IoT apps, and mobile backends.

This article is an in-depth guide to hosting ASP.NET Core services in Service Fabric Reliable Services using the **Microsoft.ServiceFabric.AspNetCore.\*** set of NuGet packages.

For an introductory tutorial on ASP.NET Core in Service Fabric and instructions on getting your development environment set up, see [Building a web front-end for your application using ASP.NET Core](#).

The rest of this article assumes you are already familiar with ASP.NET Core. If not, we recommend reading through the [ASP.NET Core fundamentals](#).

## ASP.NET Core in the Service Fabric environment

While ASP.NET Core apps can run on .NET Core or on the full .NET Framework, Service Fabric services currently can only run on the full .NET Framework. This means when you build an ASP.NET Core Service Fabric service, you must still target the full .NET Framework.

ASP.NET Core can be used in two different ways in Service Fabric:

- **Hosted as a guest executable.** This is primarily used to run existing ASP.NET Core applications on Service Fabric with no code changes.
- **Run inside a Reliable Service.** This allows better integration with the Service Fabric runtime and allows stateful ASP.NET Core services.

The rest of this article explains how to use ASP.NET Core inside a Reliable Service using the ASP.NET Core integration components that ship with the Service Fabric SDK.

## Service Fabric service hosting

In Service Fabric, one or more instances and/or replicas of your service run in a *service host process*, an executable file that runs your service code. You, as a service author, own the service host process and Service Fabric activates and monitors it for you.

Traditional ASP.NET (up to MVC 5) is tightly coupled to IIS through System.Web.dll. ASP.NET Core provides a separation between the web server and your web application. This allows web applications to be portable between different web servers and also allows web servers to be *self-hosted*, which means you can start a web server in your own process, as opposed to a process that is owned by dedicated web server software such as IIS.

In order to combine a Service Fabric service and ASP.NET, either as a guest executable or in a Reliable Service, you must be able to start ASP.NET inside your service host process. ASP.NET Core self-hosting allows you to do this.

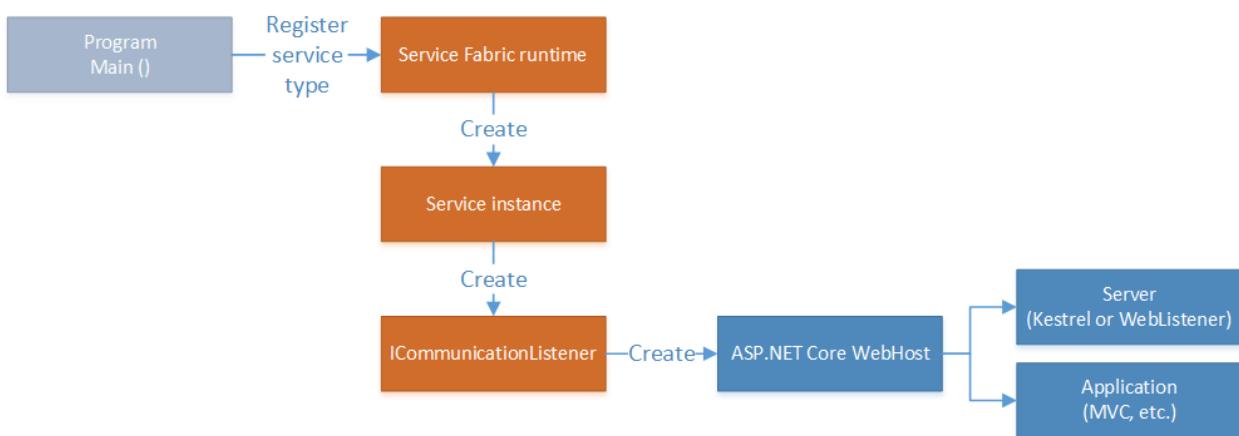
## Hosting ASP.NET Core in a Reliable Service

Typically, self-hosted ASP.NET Core applications create a `WebHost` in an application's entry point, such as the `static void Main()` method in `Program.cs`. In this case, the lifecycle of the `WebHost` is bound to the lifecycle of the process.



However, the application entry point is not the right place to create a WebHost in a Reliable Service, because the application entry point is only used to register a service type with the Service Fabric runtime, so that it may create instances of that service type. The WebHost should be created in a Reliable Service itself. Within the service host process, service instances and/or replicas can go through multiple lifecycles.

A Reliable Service instance is represented by your service class deriving from `StatelessService` or `StatefulService`. The communication stack for a service is contained in an `ICommunicationListener` implementation in your service class. The `Microsoft.ServiceFabric.Services.AspNetCore.*` NuGet packages contain implementations of `ICommunicationListener` that start and manage the ASP.NET Core WebHost for either Kestrel or WebListener in a Reliable Service.



## ASP.NET Core `ICommunicationListeners`

The `ICommunicationListener` implementations for Kestrel and WebListener in the `Microsoft.ServiceFabric.Services.AspNetCore.*` NuGet packages have similar use patterns but perform slightly different actions specific to each web server.

Both communication listeners provide a constructor that takes the following arguments:

- `ServiceContext serviceContext`: The `ServiceContext` object that contains information about the running service.
- `string endpointName`: the name of an `Endpoint` configuration in `ServiceManifest.xml`. This is primarily where the two communication listeners differ: WebListener **requires** an `Endpoint` configuration, while Kestrel does not.
- `Func<string, AspNetCoreCommunicationListener, IWebHost> build`: a lambda that you implement in which you create and return an `IWebHost`. This allows you to configure `IWebHost` the way you normally would in an ASP.NET Core application. The lambda provides a URL which is generated for you depending on the Service Fabric integration options you use and the `Endpoint` configuration you provide. That URL can then be modified or used as-is to start the web server.

## Service Fabric integration middleware

The `Microsoft.ServiceFabric.Services.AspNetCore` NuGet package includes the `UseServiceFabricIntegration` extension method on `IWebHostBuilder` that adds Service Fabric-aware middleware. This middleware configures

the Kestrel or WebListener `ICommunicationListener` to register a unique service URL with the Service Fabric Naming Service and then validates client requests to ensure clients are connecting to the right service. This is necessary in a shared-host environment such as Service Fabric, where multiple web applications can run on the same physical or virtual machine but do not use unique host names, to prevent clients from mistakenly connecting to the wrong service. This scenario is described in more detail in the next section.

### A case of mistaken identity

Service replicas, regardless of protocol, listen on a unique IP:port combination. Once a service replica has started listening on an IP:port endpoint, it reports that endpoint address to the Service Fabric Naming Service where it can be discovered by clients or other services. If services use dynamically-assigned application ports, a service replica may coincidentally use the same IP:port endpoint of another service that was previously on the same physical or virtual machine. This can cause a client to mistakenly connect to the wrong service. This can happen if the following sequence of events occur:

1. Service A listens on 10.0.0.1:30000 over HTTP.
2. Client resolves Service A and gets address 10.0.0.1:30000
3. Service A moves to a different node.
4. Service B is placed on 10.0.0.1 and coincidentally uses the same port 30000.
5. Client attempts to connect to service A with cached address 10.0.0.1:30000.
6. Client is now successfully connected to service B not realizing it is connected to the wrong service.

This can cause bugs at random times that can be difficult to diagnose.

### Using unique service URLs

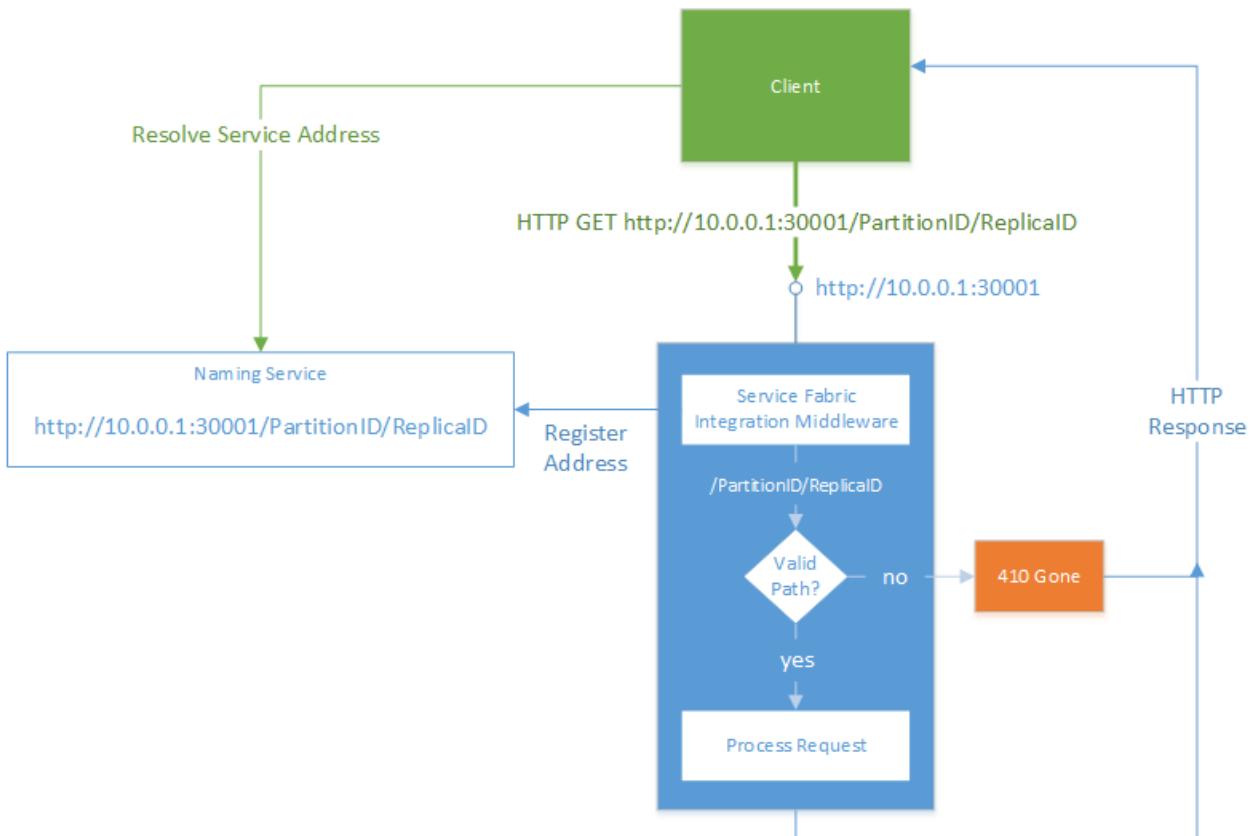
To prevent this, services can post an endpoint to the Naming Service with a unique identifier, and then validate that unique identifier during client requests. This is a cooperative action between services in a non-hostile-tenant trusted environment. This does not provide secure service authentication in a hostile-tenant environment.

In a trusted environment, the middleware that's added by the `UseServiceFabricIntegration` method automatically appends a unique identifier to the address that is posted to the Naming Service and validates that identifier on each request. If the identifier does not match, the middleware immediately returns an HTTP 410 Gone response.

Services that use a dynamically-assigned port should make use of this middleware.

Services that use a fixed unique port do not have this problem in a cooperative environment. A fixed unique port is typically used for externally-facing services that need a well-known port for client applications to connect to. For example, most Internet-facing web applications will use port 80 or 443 for web browser connections. In this case, the unique identifier should not be enabled.

The following diagram shows the request flow with the middleware enabled:



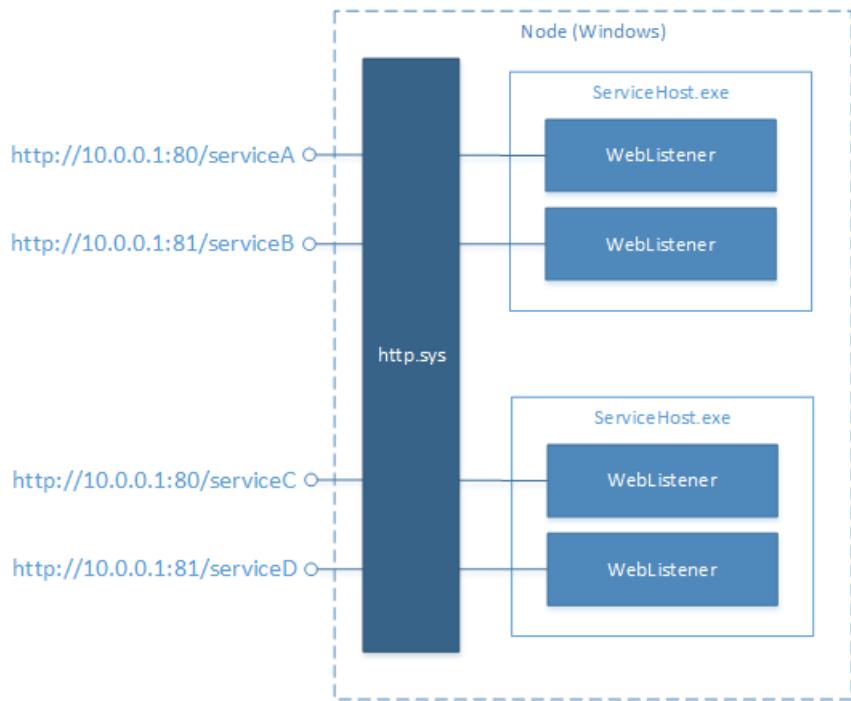
Both Kestrel and WebListener `ICommunicationListener` implementations use this mechanism in exactly the same way. Although WebListener can internally differentiate requests based on unique URL paths using the underlying `http.sys` port sharing feature, that functionality is *not* used by the WebListener `ICommunicationListener` implementation because that will result in HTTP 503 and HTTP 404 error status codes in the scenario described earlier. That in turn makes it very difficult for clients to determine the intent of the error, as HTTP 503 and HTTP 404 are already commonly used to indicate other errors. Thus, both Kestrel and WebListener `ICommunicationListener` implementations standardize on the middleware provided by the `UseServiceFabricIntegration` extension method so that clients only need to perform a service endpoint re-resolve action on HTTP 410 responses.

## WebListener in Reliable Services

WebListener can be used in a Reliable Service by importing the **Microsoft.ServiceFabric.AspNetCore.WebListener** NuGet package. This package contains `WebListenerCommunicationListener`, an implementation of `ICommunicationListener`, that allows you to create an ASP.NET CoreWebHost inside a Reliable Service using WebListener as the web server.

WebListener is built on the [Windows HTTP Server API](#). This uses the `http.sys` kernel driver used by IIS to process HTTP requests and route them to processes running web applications. This allows multiple processes on the same physical or virtual machine to host web applications on the same port, disambiguated by either a unique URL path or hostname. These features are useful in Service Fabric for hosting multiple websites in the same cluster.

The following diagram illustrates how WebListener uses the `http.sys` kernel driver on Windows for port sharing:



## WebListener in a stateless service

To use `WebListener` in a stateless service, override the `CreateServiceInstanceListeners` method and return a `WebListenerCommunicationListener` instance:

```
protected override IEnumerable<ServiceInstanceListener> CreateServiceInstanceListeners()
{
    return new ServiceInstanceListener[]
    {
        new ServiceInstanceListener(serviceContext =>
            new WebListenerCommunicationListener(serviceContext, "ServiceEndpoint", (url, listener) =>
                new WebHostBuilder()
                    .UseWebListener()
                    .ConfigureServices(
                        services => services
                            .AddSingleton<StatelessServiceContext>(serviceContext)
                            .UseContentRoot(Directory.GetCurrentDirectory())
                            .UseServiceFabricIntegration(listener, ServiceFabricIntegrationOptions.None)
                            .UseStartup<Startup>()
                            .UseUrls(url)
                            .Build()))
    };
}
```

## WebListener in a stateful service

`WebListenerCommunicationListener` is currently not designed for use in stateful services due to complications with the underlying `http.sys` port sharing feature. For more information, see the following section on dynamic port allocation with `WebListener`. For stateful services, Kestrel is the recommended web server.

## Endpoint configuration

An `Endpoint` configuration is required for web servers that use the Windows HTTP Server API, including `WebListener`. Web servers that use the Windows HTTP Server API must first reserve their URL with `http.sys` (this is normally accomplished with the `netsh` tool). This action requires elevated privileges that your services by default do not have. The "http" or "https" options for the `Protocol` property of the `Endpoint` configuration in `ServiceManifest.xml` are used specifically to instruct the Service Fabric runtime to register a URL with `http.sys` on your behalf using the `strong wildcard` URL prefix.

For example, to reserve `http://+:80` for a service, the following configuration should be used in

ServiceManifest.xml:

```
<ServiceManifest ... >
...
<Resources>
    <Endpoints>
        <Endpoint Name="ServiceEndpoint" Protocol="http" Port="80" />
    </Endpoints>
</Resources>

</ServiceManifest>
```

And the endpoint name must be passed to the `WebListenerCommunicationListener` constructor:

```
new WebListenerCommunicationListener(serviceContext, "ServiceEndpoint", (url, listener) =>
{
    return new WebHostBuilder()
        .UseWebListener()
        .UseServiceFabricIntegration(listener, ServiceFabricIntegrationOptions.None)
        .UseUrls(url)
        .Build();
})
```

#### Use WebListener with a static port

To use a static port with WebListener, provide the port number in the `Endpoint` configuration:

```
<Resources>
    <Endpoints>
        <Endpoint Protocol="http" Name="ServiceEndpoint" Port="80" />
    </Endpoints>
</Resources>
```

#### Use WebListener with a dynamic port

To use a dynamically assigned port with WebListener, omit the `Port` property in the `Endpoint` configuration:

```
<Resources>
    <Endpoints>
        <Endpoint Protocol="http" Name="ServiceEndpoint" />
    </Endpoints>
</Resources>
```

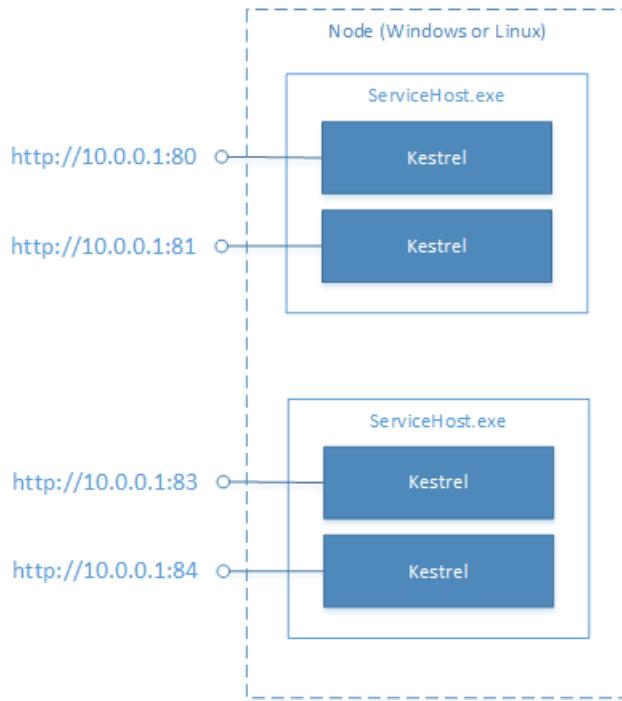
Note that a dynamic port allocated by an `Endpoint` configuration only provides one port *per host process*. The current Service Fabric hosting model allows multiple service instances and/or replicas to be hosted in the same process, meaning that each one will share the same port when allocated through the `Endpoint` configuration. Multiple WebListener instances can share a port using the underlying *http.sys* port sharing feature, but that is not supported by `WebListenerCommunicationListener` due to the complications it introduces for client requests. For dynamic port usage, Kestrel is the recommended web server.

## Kestrel in Reliable Services

Kestrel can be used in a Reliable Service by importing the **Microsoft.ServiceFabric.AspNetCore.Kestrel** NuGet package. This package contains `KestrelCommunicationListener`, an implementation of `ICommunicationListener`, that allows you to create an ASP.NET Core WebHost inside a Reliable Service using Kestrel as the web server.

Kestrel is a cross-platform web server for ASP.NET Core based on libuv, a cross-platform asynchronous I/O library. Unlike WebListener, Kestrel does not use a centralized endpoint manager such as *http.sys*. And unlike WebListener, Kestrel does not support port sharing between multiple processes. Each instance of Kestrel must use

a unique port.



### Kestrel in a stateless service

To use `Kestrel` in a stateless service, override the `CreateServiceInstanceListeners` method and return a `KestrelCommunicationListener` instance:

```
protected override IEnumerable<ServiceInstanceListener> CreateServiceInstanceListeners()
{
    return new ServiceInstanceListener[]
    {
        new ServiceInstanceListener(serviceContext =>
            new KestrelCommunicationListener(serviceContext, "ServiceEndpoint", (url, listener) =>
                new WebHostBuilder()
                    .UseKestrel()
                    .ConfigureServices(
                        services => services
                            .AddSingleton<StatelessServiceContext>(serviceContext)
                            .UseContentRoot(Directory.GetCurrentDirectory())
                            .UseServiceFabricIntegration(listener,
                                ServiceFabricIntegrationOptions.UseUniqueServiceUrl
                                    .UseStartup<Startup>()
                                    .UseUrls(url)
                                    .Build());
            ))
    };
}
```

### Kestrel in a stateful service

To use `Kestrel` in a stateful service, override the `CreateServiceReplicaListeners` method and return a `KestrelCommunicationListener` instance:

```

protected override IEnumerable<ServiceReplicaListener> CreateServiceReplicaListeners()
{
    return new ServiceReplicaListener[]
    {
        new ServiceReplicaListener(serviceContext =>
            new KestrelCommunicationListener(serviceContext, (url, listener) =>
                new WebHostBuilder()
                    .UseKestrel()
                    .ConfigureServices(
                        services => services
                            .AddSingleton<StatefulServiceContext>(serviceContext)
                            .AddSingleton<IReliableStateManager>(this.StateManager))
                    .UseContentRoot(Directory.GetCurrentDirectory())
                    .UseServiceFabricIntegration(listener,
                        ServiceFabricIntegrationOptions.UseUniqueServiceUrl)
                    .UseStartup<Startup>()
                    .UseUrls(url)
                    .Build();
            )));
    };
}

```

In this example, a singleton instance of `IReliableStateManager` is provided to the `WebHost` dependency injection container. This is not strictly necessary, but it allows you to use `IReliableStateManager` and Reliable Collections in your MVC controller action methods.

Note that an `Endpoint` configuration name is **not** provided to `KestrelCommunicationListener` in a stateful service. This is explained in more detail in the following section.

## Endpoint configuration

An `Endpoint` configuration is not required to use Kestrel.

Kestrel is a simple stand-alone web server; unlike `WebListener` (or `HttpListener`), it does not need an `Endpoint` configuration in `ServiceManifest.xml` because it does not require URL registration prior to starting.

### Use Kestrel with a static port

A static port can be configured in the `Endpoint` configuration of `ServiceManifest.xml` for use with Kestrel. Although this is not strictly necessary, it provides two potential benefits:

1. If the port does not fall in the application port range, it is opened through the OS firewall by Service Fabric.
2. The URL provided to you through `KestrelCommunicationListener` will use this port.

```

<Resources>
    <Endpoints>
        <Endpoint Protocol="http" Name="ServiceEndpoint" Port="80" />
    </Endpoints>
</Resources>

```

If an `Endpoint` is configured, its name must be passed into the `KestrelCommunicationListener` constructor:

```
new KestrelCommunicationListener(serviceContext, "ServiceEndpoint", (url, listener) => ...)
```

If an `Endpoint` configuration is not used, omit the name in the `KestrelCommunicationListener` constructor. In this case, a dynamic port will be used. See the next section for more information.

### Use Kestrel with a dynamic port

Kestrel cannot use the automatic port assignment from the `Endpoint` configuration in `ServiceManifest.xml`, because automatic port assignment from an `Endpoint` configuration assigns a unique port per *host process*, and

a single host process can contain multiple Kestrel instances. Since Kestrel does not support port sharing, this does not work as each Kestrel instance must be opened on a unique port.

To use dynamic port assignment with Kestrel, simply omit the `Endpoint` configuration in `ServiceManifest.xml` entirely, and do not pass an endpoint name to the `KestrelCommunicationListener` constructor:

```
new KestrelCommunicationListener(serviceContext, (url, listener) => ...)
```

In this configuration, `KestrelCommunicationListener` will automatically select an unused port from the application port range.

## Scenarios and configurations

This section describes the following scenarios and provides the recommended combination of web server, port configuration, Service Fabric integration options, and miscellaneous settings to achieve a properly functioning service:

- Externally exposed ASP.NET Core stateless service
- Internal-only ASP.NET Core stateless service
- Internal-only ASP.NET Core stateful service

An **externally exposed** service is one that exposes an endpoint reachable from outside the cluster, usually through a load balancer.

An **internal-only** service is one whose endpoint is only reachable from within the cluster.

### NOTE

Stateful service endpoints generally should not be exposed to the Internet. Clusters that are behind load balancers that are unaware of Service Fabric service resolution, such as the Azure Load Balancer, will be unable to expose stateful services because the load balancer will not be able to locate and route traffic to the appropriate stateful service replica.

### Externally exposed ASP.NET Core stateless services

WebListener is the recommended web server for front-end services that expose external, Internet-facing HTTP endpoints on Windows. It provides better protection against attacks and supports features that Kestrel does not, such as Windows Authentication and port sharing.

Kestrel is not supported as an edge (Internet-facing) server at this time. A reverse proxy server such as IIS or Nginx must be used to handle traffic from the public Internet.

When exposed to the Internet, a stateless service should use a well-known and stable endpoint that is reachable through a load balancer. This is the URL you will provide to users of your application. The following configuration is recommended:

		NOTES
Web server	WebListener	If the service is only exposed to a trusted network, such as an intranet, Kestrel may be used. Otherwise, WebListener is the preferred option.
Port configuration	static	A well-known static port should be configured in the <code>Endpoints</code> configuration of <code>ServiceManifest.xml</code> , such as 80 for HTTP or 443 for HTTPS.

		NOTES
ServiceFabricIntegrationOptions	None	The <code>ServiceFabricIntegrationOptions.None</code> option should be used when configuring Service Fabric integration middleware so that the service does not attempt to validate incoming requests for a unique identifier. External users of your application will not know the unique identifying information used by the middleware.
Instance Count	-1	In typical use cases, the instance count setting should be set to "-1" so that an instance is available on all nodes that receive traffic from a load balancer.

If multiple externally exposed services share the same set of nodes, a unique but stable URL path should be used. This can be accomplished by modifying the URL provided when configuring IWebHost. Note this applies to WebListener only.

```
new WebListenerCommunicationListener(serviceContext, "ServiceEndpoint", (url, listener) =>
{
    url += "/MyUniqueServicePath";

    return new WebHostBuilder()
        .UseWebListener()
        ...
        .UseUrls(url)
        .Build();
})
```

### Internal-only stateless ASP.NET Core service

Stateless services that are only called from within the cluster should use unique URLs and dynamically assigned ports to ensure cooperation between multiple services. The following configuration is recommended:

		NOTES
Web server	Kestrel	Although WebListener may be used for internal stateless services, Kestrel is the recommended server to allow multiple service instances to share a host.
Port configuration	dynamically assigned	Multiple replicas of a stateful service may share a host process or host operating system and thus will need unique ports.
ServiceFabricIntegrationOptions	UseUniqueServiceUrl	With dynamic port assignment, this setting prevents the mistaken identity issue described earlier.
InstanceCount	any	The instance count setting can be set to any value necessary to operate the service.

### Internal-only stateful ASP.NET Core service

Stateful services that are only called from within the cluster should use dynamically assigned ports to ensure cooperation between multiple services. The following configuration is recommended:

		<b>NOTES</b>
Web server	Kestrel	The <code>WebListenerCommunicationListener</code> is not designed for use by stateful services in which replicas share a host process.
Port configuration	dynamically assigned	Multiple replicas of a stateful service may share a host process or host operating system and thus will need unique ports.
ServiceFabricIntegrationOptions	UseUniqueServiceUrl	With dynamic port assignment, this setting prevents the mistaken identity issue described earlier.

## Next steps

[Debug your Service Fabric application by using Visual Studio](#)

# Capacity planning for Service Fabric applications

8/15/2017 • 6 min to read • [Edit Online](#)

This document teaches you how to estimate the amount of resources (CPUs, RAM, disk storage) you need to run your Azure Service Fabric applications. It is common for your resource requirements to change over time. You typically require few resources as you develop/test your service, and then require more resources as you go into production and your application grows in popularity. When you design your application, think through the long-term requirements and make choices that allow your service to scale to meet high customer demand.

When you create a Service Fabric cluster, you decide what kinds of virtual machines (VMs) make up the cluster. Each VM comes with a limited amount of resources in the form of CPUs (cores and speed), network bandwidth, RAM, and disk storage. As your service grows over time, you can upgrade to VMs that offer greater resources and/or add more VMs to your cluster. To do the latter, you must architect your service initially so it can take advantage of new VMs that get dynamically added to the cluster.

Some services manage little to no data on the VMs themselves. Therefore, capacity planning for these services should focus primarily on performance, which means selecting the appropriate CPUs (cores and speed) of the VMs. In addition, you should consider network bandwidth, including how frequently network transfers are occurring and how much data is being transferred. If your service needs to perform well as service usage increases, you can add more VMs to the cluster and load balance the network requests across all the VMs.

For services that manage large amounts of data on the VMs, capacity planning should focus primarily on size. Thus, you should carefully consider the capacity of the VM's RAM and disk storage. The virtual memory management system in Windows makes disk space look like RAM to application code. In addition, the Service Fabric runtime provides smart paging keeping only hot data in memory and moving the cold data to disk. Applications can thus use more memory than is physically available on the VM. Having more RAM simply increases performance, since the VM can keep more disk storage in RAM. The VM you select should have a disk large enough to store the data that you want on the VM. Similarly, the VM should have enough RAM to provide you with the performance you desire. If your service's data grows over time, you can add more VMs to the cluster and partition the data across all the VMs.

## Determine how many nodes you need

Partitioning your service allows you to scale out your service's data. For more information on partitioning, see [Partitioning Service Fabric](#). Each partition must fit within a single VM, but multiple (small) partitions can be placed on a single VM. So, having more small partitions gives you greater flexibility than having a few larger partitions. The trade-off is that having lots of partitions increases Service Fabric overhead and you cannot perform transacted operations across partitions. There is also more potential network traffic if your service code frequently needs to access pieces of data that live in different partitions. When designing your service, you should carefully consider these pros and cons to arrive at an effective partitioning strategy.

Let's assume your application has a single stateful service that has a store size that you expect to grow to DB\_Size GB in a year. You are willing to add more applications (and partitions) as you experience growth beyond that year. The replication factor (RF), which determines the number of replicas for your service impacts the total DB\_Size. The total DB\_Size across all replicas is the Replication Factor multiplied by DB\_Size. Node\_Size represents the disk space/RAM per node you want to use for your service. For best performance, the DB\_Size should fit into memory across the cluster, and a Node\_Size that is around the RAM of the VM should be chosen. By allocating a Node\_Size that is larger than the RAM capacity, you are relying on the paging provided by the Service Fabric runtime. Thus, your performance may not be optimal if your entire data is considered to be hot (since then the data is paged in/out). However, for many services where only a fraction of the data is hot, it is more cost-effective.

The number of nodes required for maximum performance can be computed as follows:

$$\text{Number of Nodes} = (\text{DB\_Size} * \text{RF}) / \text{Node\_Size}$$

## Account for growth

You may want to compute the number of nodes based on the DB\_Size that you expect your service to grow to, in addition to the DB\_Size that you began with. Then, grow the number of nodes as your service grows so that you are not over-provisioning the number of nodes. But the number of partitions should be based on the number of nodes that are needed when you're running your service at maximum growth.

It is good to have some extra machines available at any time so that you can handle any unexpected spikes or failure (for example, if a few VMs go down). While the extra capacity should be determined by using your expected spikes, a starting point is to reserve a few extra VMs (5-10 percent extra).

The preceding assumes a single stateful service. If you have more than one stateful service, you have to add the DB\_Size associated with the other services into the equation. Alternatively, you can compute the number of nodes separately for each stateful service. Your service may have replicas or partitions that aren't balanced. Keep in mind that partitions may also have more data than others. For more information on partitioning, see [partitioning article on best practices](#). However, the preceding equation is partition and replica agnostic, because Service Fabric ensures that the replicas are spread out among the nodes in an optimized manner.

## Use a spreadsheet for cost calculation

Now let's put some real numbers in the formula. An [example spreadsheet](#) shows how to plan the capacity for an application that contains three types of data objects. For each object, we approximate its size and how many objects we expect to have. We also select how many replicas we want of each object type. The spreadsheet calculates the total amount of memory to be stored in the cluster.

Then we enter a VM size and monthly cost. Based on the VM size, the spreadsheet tells you the minimum number of partitions you must use to split your data to physically fit on the nodes. You may desire a larger number of partitions to accommodate your application's specific computation and network traffic needs. The spreadsheet shows the number of partitions that are managing the user profile objects has increased from one to six.

Now, based on all this information, the spreadsheet shows that you could physically get all the data with the desired partitions and replicas on a 26-node cluster. However, this cluster would be densely packed, so you may want some additional nodes to accommodate node failures and upgrades. The spreadsheet also shows that having more than 57 nodes provides no additional value because you would have empty nodes. Again, you may want to go above 57 nodes anyway to accommodate node failures and upgrades. You can tweak the spreadsheet to match your application's specific needs.

Data Object Type	Inventory Item	User Profile	Order
Size of Object (KB)	4	2	1
Number of Objects	10,000	3,000,000	45,000,000
Total Size of Object (KB)	40,000	6,000,000	45,000,000
Number of Desired Replicas	3	5	3
Total size of Replicas (KB)	120,000	30,000,000	135,000,000
Total size of data to store in the Cluster (GB)	157		
Node Available RAM/Storage (GB)	6	(Put available RAM or disk size based on your performance needs)	
Node Cost/Month	\$ 229.00	(Ex: Azure A3 pricing)	
Minimum Partitions Required	1	1	8
Number of Desired Partitions	1	6	8
Minimum Number of Nodes	26		
Cost of Nodes/Month	\$ 6,010		
Number of Nodes (1 replica per node)	57		
Cost of Nodes/Month	\$ 13,053		

## Next steps

Check out [Partitioning Service Fabric services](#) to learn more about partitioning your service.

# Service Fabric application lifecycle

9/29/2017 • 6 min to read • [Edit Online](#)

As with other platforms, an application on Azure Service Fabric usually goes through the following phases: design, development, testing, deployment, upgrading, maintenance, and removal. Service Fabric provides first-class support for the full application lifecycle of cloud applications, from development through deployment, daily management, and maintenance to eventual decommissioning. The service model enables several different roles to participate independently in the application lifecycle. This article provides an overview of the APIs and how they are used by the different roles throughout the phases of the Service Fabric application lifecycle.

## IMPORTANT

There are two CLI utilities used to interact with Service Fabric. [Azure CLI](#) is used to manage Azure resources, such as an Azure-hosted Service Fabric cluster. [Service Fabric CLI](#) is used to directly connect to the Service Fabric cluster (regardless of where it's hosted) and manage the cluster, applications, and services.

The following Microsoft Virtual Academy video describes how to manage your application lifecycle:



## Service model roles

The service model roles are:

- **Service developer:** Develops modular and generic services that can be re-purposed and used in multiple applications of the same type or different types. For example, a queue service can be used for creating a ticketing application (helpdesk) or an e-commerce application (shopping cart).
- **Application developer:** Creates applications by integrating a collection of services to satisfy certain specific requirements or scenarios. For example, an e-commerce website might integrate "JSON Stateless Front-End Service," "Auction Stateful Service," and "Queue Stateful Service" to build an auctioning solution.
- **Application administrator:** Makes decisions on the application configuration (filling in the configuration template parameters), deployment (mapping to available resources), and quality of service. For example, an application administrator decides the language locale (English for the United States or Japanese for Japan, for example) of the application. A different deployed application can have different settings.
- **Operator:** Deploys applications based on the application configuration and requirements specified by the application administrator. For example, an operator provisions and deploys the application and ensures that it is running in Azure. Operators monitor application health and performance information and maintain the physical infrastructure as needed.

## Develop

1. A *service developer* develops different types of services using the [Reliable Actors](#) or [Reliable Services](#)

programming model.

2. A *service developer* declaratively describes the developed service types in a service manifest file consisting of one or more code, configuration, and data packages.
3. An *application developer* then builds an application using different service types.
4. An *application developer* declaratively describes the application type in an application manifest by referencing the service manifests of the constituent services and appropriately overriding and parameterizing different configuration and deployment settings of the constituent services.

See [Get started with Reliable Actors](#) and [Get started with Reliable Services](#) for examples.

## Deploy

1. An *application administrator* tailors the application type to a specific application to be deployed to a Service Fabric cluster by specifying the appropriate parameters of the **ApplicationType** element in the application manifest.
2. An *operator* uploads the application package to the cluster image store by using the [\*\*CopyApplicationPackage\*\* method](#) or the [\*\*Copy-ServiceFabricApplicationPackage\*\* cmdlet](#). The application package contains the application manifest and the collection of service packages. Service Fabric deploys applications from the application package stored in the image store, which can be an Azure blob store or the Service Fabric system service.
3. The *operator* then provisions the application type in the target cluster from the uploaded application package using the [\*\*ProvisionApplicationAsync\*\* method](#), the [\*\*Register-ServiceFabricApplicationType\*\* cmdlet](#), or the [\*\*Provision an Application\*\* REST operation](#).
4. After provisioning the application, an *operator* starts the application with the parameters supplied by the *application administrator* using the [\*\*CreateApplicationAsync\*\* method](#), the [\*\*New-ServiceFabricApplication\*\* cmdlet](#), or the [\*\*Create Application\*\* REST operation](#).
5. After the application has been deployed, an *operator* uses the [\*\*CreateServiceAsync\*\* method](#), the [\*\*New-ServiceFabricService\*\* cmdlet](#), or the [\*\*Create Service\*\* REST operation](#) to create new service instances for the application based on available service types.
6. The application is now running in the Service Fabric cluster.

See [Deploy an application](#) for examples.

## Test

1. After deploying to the local development cluster or a test cluster, a *service developer* runs the built-in failover test scenario by using the **FailoverTestScenarioParameters** and **FailoverTestScenario** classes, or the [\*\*Invoke-ServiceFabricFailoverTestScenario\*\* cmdlet](#). The failover test scenario runs a specified service through important transitions and failovers to ensure that it's still available and working.
2. The *service developer* then runs the built-in chaos test scenario using the **ChaosTestScenarioParameters** and **ChaosTestScenario** classes, or the [\*\*Invoke-ServiceFabricChaosTestScenario\*\* cmdlet](#). The chaos test scenario randomly induces multiple node, code package, and replica faults into the cluster.
3. The *service developer* [tests service-to-service communication](#) by authoring test scenarios that move primary replicas around the cluster.

See [Introduction to the Fault Analysis Service](#) for more information.

## Upgrade

1. A *service developer* updates the constituent services of the instantiated application and/or fixes bugs and provides a new version of the service manifest.
2. An *application developer* overrides and parameterizes the configuration and deployment settings of the

consistent services and provides a new version of the application manifest. The application developer then incorporates the new versions of the service manifests into the application and provides a new version of the application type in an updated application package.

3. An *application administrator* incorporates the new version of the application type into the target application by updating the appropriate parameters.
4. An *operator* uploads the updated application package to the cluster image store using the **CopyApplicationPackage** method or the **Copy-ServiceFabricApplicationPackage** cmdlet. The application package contains the application manifest and the collection of service packages.
5. An *operator* provisions the new version of the application in the target cluster by using the **ProvisionApplicationAsync** method, the **Register-ServiceFabricApplicationType** cmdlet, or the **Provision an Application** REST operation.
6. An *operator* upgrades the target application to the new version using the **UpgradeApplicationAsync** method, the **Start-ServiceFabricApplicationUpgrade** cmdlet, or the **Upgrade an Application** REST operation.
7. An *operator* checks the progress of upgrade using the **GetApplicationUpgradeProgressAsync** method, the **Get-ServiceFabricApplicationUpgrade** cmdlet, or the **Get Application Upgrade Progress** REST operation.
8. If necessary, the *operator* modifies and reapplys the parameters of the current application upgrade using the **UpdateApplicationUpgradeAsync** method, the **Update-ServiceFabricApplicationUpgrade** cmdlet, or the **Update Application Upgrade** REST operation.
9. If necessary, the *operator* rolls back the current application upgrade using the **RollbackApplicationUpgradeAsync** method, the **Start-ServiceFabricApplicationRollback** cmdlet, or the **Rollback Application Upgrade** REST operation.
10. Service Fabric upgrades the target application running in the cluster without losing the availability of any of its constituent services.

See the [Application upgrade tutorial](#) for examples.

## Maintain

1. For operating system upgrades and patches, Service Fabric interfaces with the Azure infrastructure to guarantee availability of all the applications running in the cluster.
2. For upgrades and patches to the Service Fabric platform, Service Fabric upgrades itself without losing availability of any of the applications running on the cluster.
3. An *application administrator* approves the addition or removal of nodes from a cluster after analyzing historical capacity utilization data and projected future demand.
4. An *operator* adds and removes nodes specified by the *application administrator*.
5. When new nodes are added to or existing nodes are removed from the cluster, Service Fabric automatically load-balances the running applications across all nodes in the cluster to achieve optimal performance.

## Remove

1. An *operator* can delete a specific instance of a running service in the cluster without removing the entire application using the **DeleteServiceAsync** method, the **Remove-ServiceFabricService** cmdlet, or the **Delete Service** REST operation.
2. An *operator* can also delete an application instance and all of its services using the **DeleteApplicationAsync** method, the **Remove-ServiceFabricApplication** cmdlet, or the **Delete Application** REST operation.
3. Once the application and services have stopped, the *operator* can unprovision the application type using the **UnprovisionApplicationAsync** method, the **Unregister-ServiceFabricApplicationType** cmdlet, or the **Unprovision an Application** REST operation. Unprovisioning the application type does not remove the application package from the ImageStore. You must remove the application package manually.
4. An *operator* removes the application package from the ImageStore using the **RemoveApplicationPackage**

[method](#) or the **Remove-ServiceFabricApplicationPackage** cmdlet.

See [Deploy an application](#) for examples.

## Next steps

For more information on developing, testing, and managing Service Fabric applications and services, see:

- [Reliable Actors](#)
- [Reliable Services](#)
- [Deploy an application](#)
- [Application upgrade](#)
- [Testability overview](#)

# Understand the ImageStoreConnectionString setting

10/4/2017 • 2 min to read • [Edit Online](#)

In some of our documentation, we briefly mention the existence of an "ImageStoreConnectionString" parameter without describing what it really means. And after going through an article like [Deploy and remove applications using PowerShell](#), it looks like all you do is copy/paste the value as it appears in the cluster manifest of the target cluster. So the setting must be configurable per cluster, but when you create a cluster through the [Azure portal](#), there's no option to configure this setting and it's always "fabric:ImageStore". What's the purpose of this setting then?

ESSENTIALS	DETAILS	CLUSTER MAP	METRICS	MANIFEST
				<pre>&lt;Section Name="ImageStoreService"&gt;   &lt;Parameter Name="MinReplicaSetSize" Value="3" /&gt;   &lt;Parameter Name="PlacementConstraints" Value="NodeType==n1" /&gt;   &lt;Parameter Name="TargetReplicaSetSize" Value="5" /&gt; &lt;/Section&gt; &lt;Section Name="Management"&gt;   &lt;Parameter Name="EnableDeploymentAtDataRoot" Value="true" /&gt;   &lt;Parameter Name="ImageStoreConnectionString" Value="fabric:ImageStore" /&gt; &lt;/Section&gt; &lt;Section Name="NamingService"&gt;</pre>

Service Fabric started off as a platform for internal Microsoft consumption by many diverse teams, so some aspects of it are highly customizable - the "Image Store" is one such aspect. Essentially, the Image Store is a pluggable repository for storing application packages. When your application is deployed to a node in the cluster, that node downloads the contents of your application package from the Image Store. The ImageStoreConnectionString is a setting that includes all the necessary information for both clients and nodes to find the correct Image Store for a given cluster.

There are currently three possible kinds of Image Store providers and their corresponding connection strings are as follows:

1. Image Store Service: "fabric:ImageStore"
2. File System: "file:[file system path]"
3. Azure Storage: "xstore:DefaultEndpointsProtocol=https;AccountName=[...];AccountKey=[...];Container=[...]"

The provider type used in production is the Image Store Service, which is a stateful persisted system service that you can see from Service Fabric Explorer.



OK

Warning

Error

Search Cluster



## Cluster

&gt; Applications

&gt; Nodes

## System

&gt; fabric:/System/ClusterManagerService

&gt; fabric:/System/FailoverManagerService

&gt; fabric:/System/FaultAnalysisService

## fabric:/System/ImageStoreService

&gt; 00000000-0000-0000-0000-000000003000

Primary (\_n1\_2)

ActiveSecondary (\_n1\_0)

ActiveSecondary (\_n1\_1)

ActiveSecondary (\_n1\_3)

ActiveSecondary (\_n1\_4)

&gt; fabric:/System/NamingService

Hosting the Image Store in a system service within the cluster itself eliminates external dependencies for the package repository and gives us more control over the locality of storage. Future improvements around the Image Store are likely to target the Image Store provider first, if not exclusively. The connection string for the Image Store Service provider doesn't have any unique information since the client is already connected to the target cluster. The client only needs to know that protocols targeting the system service should be used.

The File System provider is used instead of the Image Store Service for local one-box clusters during development to bootstrap the cluster slightly faster. The difference is typically small, but it's a useful optimization for most folks

during development. It's possible to deploy a local one-box cluster with the other storage provider types as well, but there's usually no reason to do so since the develop/test workflow remains the same regardless of provider. Other than this usage, the File System and Azure Storage providers only exist for legacy support.

So while the `ImageStoreConnectionString` is configurable, you generally just use the default setting. When publishing to Azure through [Visual Studio](#), the parameter is automatically set for you accordingly. For programmatic deployment to clusters hosted in Azure, the connection string is always "fabric:ImageStore". Though when in doubt, its value can always be verified by retrieving the cluster manifest by [PowerShell](#), [.NET](#), or [REST](#). Both on-premises test and production clusters should always be configured to use the Image Store Service provider as well.

## Next steps

[Deploy and remove applications using PowerShell](#)

# Service Fabric application upgrade

10/11/2017 • 5 min to read • [Edit Online](#)

An Azure Service Fabric application is a collection of services. During an upgrade, Service Fabric compares the new [application manifest](#) with the previous version and determines which services in the application require updates. Service Fabric compares the version numbers in the service manifests with the version numbers in the previous version. If a service has not changed, that service is not upgraded.

## Rolling upgrades overview

In a rolling application upgrade, the upgrade is performed in stages. At each stage, the upgrade is applied to a subset of nodes in the cluster, called an update domain. As a result, the application remains available throughout the upgrade. During the upgrade, the cluster may contain a mix of the old and new versions.

For that reason, the two versions must be forward and backward compatible. If they are not compatible, the application administrator is responsible for staging a multiple-phase upgrade to maintain availability. In a multiple-phase upgrade, the first step is upgrading to an intermediate version of the application that is compatible with the previous version. The second step is to upgrade the final version that breaks compatibility with the pre-update version, but is compatible with the intermediate version.

Update domains are specified in the cluster manifest when you configure the cluster. Update domains do not receive updates in a particular order. An update domain is a logical unit of deployment for an application. Update domains allow the services to remain at high availability during an upgrade.

Non-rolling upgrades are possible if the upgrade is applied to all nodes in the cluster, which is the case when the application has only one update domain. This approach is not recommended, since the service goes down and isn't available at the time of upgrade. Additionally, Azure doesn't provide any guarantees when a cluster is set up with only one update domain.

After the upgrade completes, all the services and replicas(instances) would stay in the same version-i.e., if the upgrade succeeds, they will be updated to the new version; if the upgrade fails and is rolled back, they would be rolled back to the old version.

## Health checks during upgrades

For an upgrade, health policies have to be set (or default values may be used). An upgrade is termed successful when all update domains are upgraded within the specified time-outs, and when all update domains are deemed healthy. A healthy update domain means that the update domain passed all the health checks specified in the health policy. For example, a health policy may mandate that all services within an application instance must be *healthy*, as health is defined by Service Fabric.

Health policies and checks during upgrade by Service Fabric are service and application agnostic. That is, no service-specific tests are done. For example, your service might have a throughput requirement, but Service Fabric does not have the information to check throughput. Refer to the [health articles](#) for the checks that are performed. The checks that happen during an upgrade include tests for whether the application package was copied correctly, whether the instance was started, and so on.

The application health is an aggregation of the child entities of the application. In short, Service Fabric evaluates the health of the application through the health that is reported on the application. It also evaluates the health of all the services for the application this way. Service Fabric further evaluates the health of the application services by aggregating the health of their children, such as the service replica. Once the application health

policy is satisfied, the upgrade can proceed. If the health policy is violated, the application upgrade fails.

## Upgrade modes

The mode that we recommend for application upgrade is the monitored mode, which is the commonly used mode. Monitored mode performs the upgrade on one update domain, and if all health checks pass (per the policy specified), moves on to the next update domain automatically. If health checks fail and/or time-outs are reached, the upgrade is either rolled back for the update domain, or the mode is changed to unmonitored manual. You can configure the upgrade to choose one of those two modes for failed upgrades.

Unmonitored manual mode needs manual intervention after every upgrade on an update domain, to kick off the upgrade on the next update domain. No Service Fabric health checks are performed. The administrator performs the health or status checks before starting the upgrade in the next update domain.

## Upgrade default services

Default services within Service Fabric application can be upgraded during the upgrade process of an application. Default services are defined in the [application manifest](#). The standard rules of upgrading default services are:

1. Default services in the new [application manifest](#) that do not exist in the cluster are created.

**TIP**

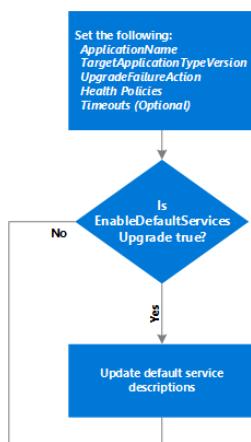
`EnableDefaultServicesUpgrade` needs to be set to true to enable the following rules. This feature is supported from v5.5.

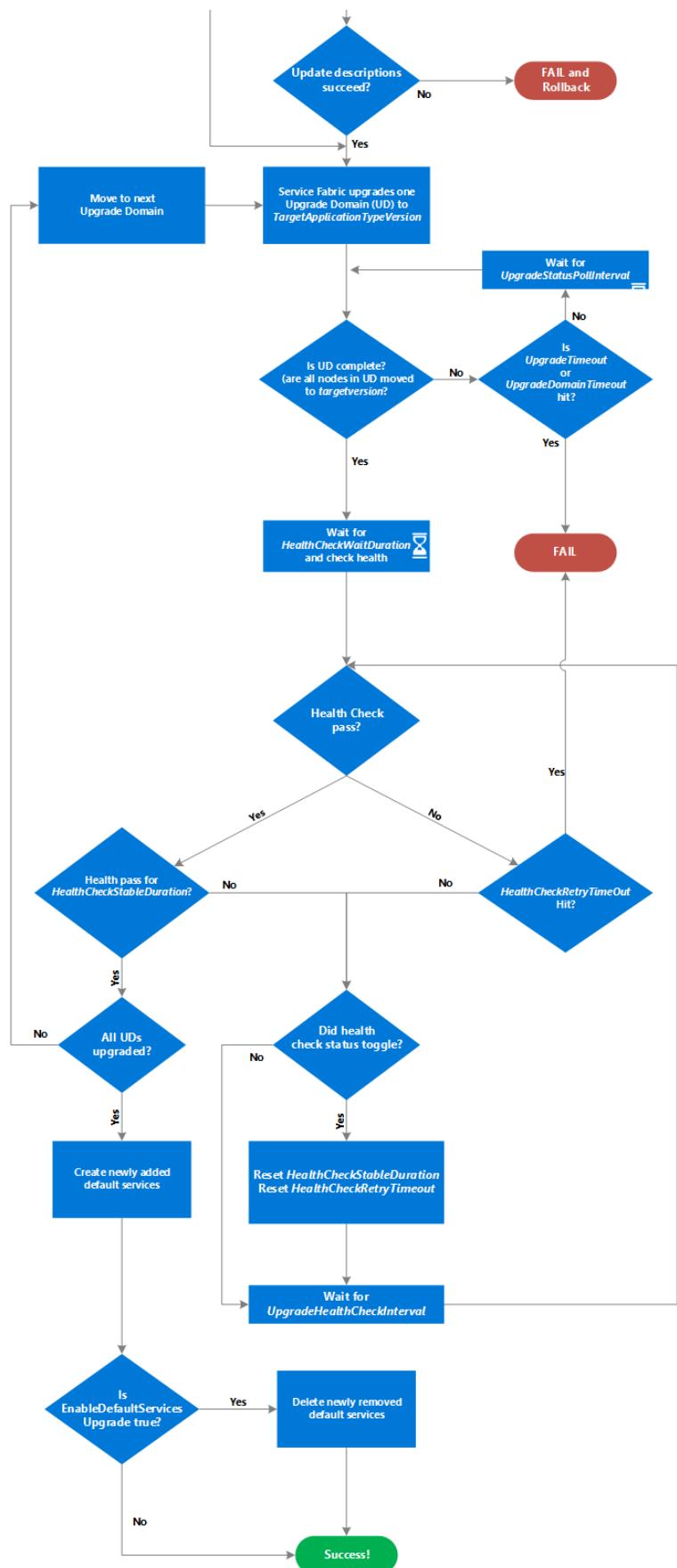
2. Default services existing in both previous [application manifest](#) and new version are updated. Service descriptions in the new version would overwrite those already in the cluster. Application upgrade would rollback automatically upon updating default service failure.
3. Default services in the previous [application manifest](#) but not in the new version are deleted. **Note that this deleting default services can not be reverted.**

In case of an application upgrade is rolled back, default services are reverted to the status before upgrade started. But deleted services can never be created.

## Application upgrade flowchart

The flowchart following this paragraph can help you understand the upgrade process of a Service Fabric application. In particular, the flow describes how the time-outs, including *HealthCheckStableDuration*, *HealthCheckRetryTimeout*, and *UpgradeHealthCheckInterval*, help control when the upgrade in one update domain is considered a success or a failure.





## Next steps

[Upgrading your Application Using Visual Studio](#) walks you through an application upgrade using Visual Studio.

[Upgrading your Application Using PowerShell](#) walks you through an application upgrade using PowerShell.

Control how your application upgrades by using [Upgrade Parameters](#).

Make your application upgrades compatible by learning how to use [Data Serialization](#).

Learn how to use advanced functionality while upgrading your application by referring to [Advanced Topics](#).

Fix common problems in application upgrades by referring to the steps in [Troubleshooting Application Upgrades](#).

# Configure the upgrade of a Service Fabric application in Visual Studio

6/30/2017 • 2 min to read • [Edit Online](#)

Visual Studio tools for Azure Service Fabric provide upgrade support for publishing to local or remote clusters. There are three scenarios in which you want to upgrade your application to a newer version instead of replacing the application during testing and debugging:

- Application data won't be lost during the upgrade.
- Availability remains high so there won't be any service interruption during the upgrade, if there are enough service instances spread across upgrade domains.
- Tests can be run against an application while it's being upgraded.

## Parameters needed to upgrade

You can choose from two types of deployment: regular or upgrade. A regular deployment erases any previous deployment information and data on the cluster, while an upgrade deployment preserves it. When you upgrade a Service Fabric application in Visual Studio, you need to provide application upgrade parameters and health check policies. Application upgrade parameters help control the upgrade, while health check policies determine whether the upgrade was successful. See [Service Fabric application upgrade: upgrade parameters](#) for more details.

There are three upgrade modes: *Monitored*, *UnmonitoredAuto*, and *UnmonitoredManual*.

- A Monitored upgrade automates the upgrade and application health check.
- An UnmonitoredAuto upgrade automates the upgrade, but skips the application health check.
- When you do an UnmonitoredManual upgrade, you need to manually upgrade each upgrade domain.

Each upgrade mode requires different sets of parameters. See [Application upgrade parameters](#) to learn more about the available upgrade options.

## Upgrade a Service Fabric application in Visual Studio

If you're using the Visual Studio Service Fabric tools to upgrade a Service Fabric application, you can specify a publish process to be an upgrade rather than a regular deployment by checking the **Upgrade the application** check box.

### To configure the upgrade parameters

1. Click the **Settings** button next to the check box. The **Edit Upgrade Parameters** dialog box appears. The **Edit Upgrade Parameters** dialog box supports the Monitored, UnmonitoredAuto, and UnmonitoredManual upgrade modes.
2. Select the upgrade mode that you want to use and then fill out the parameter grid.

Each parameter has default values. The optional parameter *DefaultServiceTypeHealthPolicy* takes a hash table input. Here's an example of the hash table input format for *DefaultServiceTypeHealthPolicy*:

```
@{ ConsiderWarningAsError = "false"; MaxPercentUnhealthyDeployedApplications = 0;  
MaxPercentUnhealthyServices = 0; MaxPercentUnhealthyPartitionsPerService = 0;  
MaxPercentUnhealthyReplicasPerPartition = 0 }
```

*ServiceTypeHealthPolicyMap* is another optional parameter that takes a hash table input in the following

format:

```
@ {"ServiceTypeName" :  
  "MaxPercentUnhealthyPartitionsPerService,MaxPercentUnhealthyReplicasPerPartition,MaxPercentUnhealthyServices"}  
}
```

Here's a real-life example:

```
@{ "ServiceTypeName01" = "5,10,5"; "ServiceTypeName02" = "5,5,5" }
```

3. If you select UnmonitoredManual upgrade mode, you must manually start a PowerShell console to continue and finish the upgrade process. Refer to [Service Fabric application upgrade: advanced topics](#) to learn how manual upgrade works.

## Upgrade an application by using PowerShell

You can use PowerShell cmdlets to upgrade a Service Fabric application. See [Service Fabric application upgrade tutorial](#) and [Start-ServiceFabricApplicationUpgrade](#) for detailed information.

## Specify a health check policy in the application manifest file

Every service in a Service Fabric application can have its own health policy parameters that override the default values. You can provide these parameter values in the application manifest file.

The following example shows how to apply a unique health check policy for each service in the application manifest.

```
<Policies>  
  <HealthPolicy ConsiderWarningAsError="false" MaxPercentUnhealthyDeployedApplications="20">  
    <DefaultServiceTypeHealthPolicy MaxPercentUnhealthyServices="20"  
      MaxPercentUnhealthyPartitionsPerService="20"  
      MaxPercentUnhealthyReplicasPerPartition="20" />  
    <ServiceTypeHealthPolicy ServiceTypeName="ServiceTypeName1"  
      MaxPercentUnhealthyServices="20"  
      MaxPercentUnhealthyPartitionsPerService="20"  
      MaxPercentUnhealthyReplicasPerPartition="20" />  
  </HealthPolicy>  
</Policies>
```

## Next steps

For more information about deploying an application, see [Deploy an existing application in Azure Service Fabric](#).

# Application upgrade parameters

8/23/2017 • 5 min to read • [Edit Online](#)

This article describes the various parameters that apply during the upgrade of an Azure Service Fabric application. The parameters include the name and version of the application. They are knobs that control the time-outs and health checks that are applied during the upgrade, and they specify the policies that must be applied when an upgrade fails.

Parameter	Description
ApplicationName	Name of the application that is being upgraded. Examples: fabric:/VisualObjects, fabric:/ClusterMonitor
TargetApplicationTypeVersion	The version of the application type that the upgrade targets.
FailureAction	The action taken by Service Fabric when the upgrade fails. The application may be rolled back to the pre-update version (rollback), or the upgrade may be stopped at the current upgrade domain. In the latter case, the upgrade mode is also changed to Manual. Allowed values are Rollback and Manual.
HealthCheckWaitDurationSec	The time to wait (in seconds) after the upgrade has finished on the upgrade domain before Service Fabric evaluates the health of the application. This duration can also be considered as the time an application should be running before it can be considered healthy. If the health check passes, the upgrade process proceeds to the next upgrade domain. If the health check fails, Service Fabric waits for an interval (the UpgradeHealthCheckInterval) before retrying the health check again until the HealthCheckRetryTimeout is reached. The default and recommended value is 0 seconds.
HealthCheckRetryTimeoutSec	The duration (in seconds) that Service Fabric continues to perform health evaluation before declaring the upgrade as failed. The default is 600 seconds. This duration starts after HealthCheckWaitDuration is reached. Within this HealthCheckRetryTimeout, Service Fabric might perform multiple health checks of the application health. The default value is 10 minutes and should be customized appropriately for your application.
HealthCheckStableDurationSec	The duration (in seconds) to verify that the application is stable before moving to the next upgrade domain or completing the upgrade. This wait duration is used to prevent undetected changes of health right after the health check is performed. The default value is 120 seconds, and should be customized appropriately for your application.
UpgradeDomainTimeoutSec	Maximum time (in seconds) for upgrading a single upgrade domain. If this time-out is reached, the upgrade stops and proceeds based on the setting for UpgradeFailureAction. The default value is never (Infinite) and should be customized appropriately for your application.

PARAMETER	DESCRIPTION
UpgradeTimeout	A time-out (in seconds) that applies for the entire upgrade. If this time-out is reached, the upgrade stops and UpgradeFailureAction is triggered. The default value is never (Infinite) and should be customized appropriately for your application.
UpgradeHealthCheckInterval	The frequency that the health status is checked. This parameter is specified in the ClusterManager section of the <i>cluster manifest</i> , and is not specified as part of the upgrade cmdlet. The default value is 60 seconds.

## Service health evaluation during application upgrade

The health evaluation criteria are optional. If the health evaluation criteria are not specified when an upgrade starts, Service Fabric uses the application health policies specified in the ApplicationManifest.xml of the application instance.

PARAMETER	DESCRIPTION
ConsiderWarningAsError	Default value is False. Treat the warning health events for the application as errors when evaluating the health of the application during upgrade. By default, Service Fabric does not evaluate warning health events to be failures (errors), so the upgrade can proceed even if there are warning events.
MaxPercentUnhealthyDeployedApplications	Default and recommended value is 0. Specify the maximum number of deployed applications (see the <a href="#">Health section</a> ) that can be unhealthy before the application is considered unhealthy and fails the upgrade. This parameter defines the application health on the node and helps detect issues during upgrade. Typically, the replicas of the application get load-balanced to the other node, which allows the application to appear healthy, thus allowing the upgrade to proceed. By specifying a strict MaxPercentUnhealthyDeployedApplications health, Service Fabric can detect a problem with the application package quickly and help produce a fail fast upgrade.
MaxPercentUnhealthyServices	Default and recommended value is 0. Specify the maximum number of services in the application instance that can be unhealthy before the application is considered unhealthy and fails the upgrade.
MaxPercentUnhealthyPartitionsPerService	Default and recommended value is 0. Specify the maximum number of partitions in a service that can be unhealthy before the service is considered unhealthy.
MaxPercentUnhealthyReplicasPerPartition	Default and recommended value is 0. Specify the maximum number of replicas in partition that can be unhealthy before the partition is considered unhealthy.

PARAMETER	DESCRIPTION
UpgradeReplicaSetCheckTimeout	<p><b>Stateless service</b>--Within a single upgrade domain, Service Fabric tries to ensure that additional instances of the service are available. If the target instance count is more than one, Service Fabric waits for more than one instance to be available, up to a maximum time-out value. This time-out is specified by using the UpgradeReplicaSetCheckTimeout property. If the time-out expires, Service Fabric proceeds with the upgrade, regardless of the number of service instances. If the target instance count is one, Service Fabric does not wait, and immediately proceeds with the upgrade. <b>Stateful service</b>--Within a single upgrade domain, Service Fabric tries to ensure that the replica set has a quorum. Service Fabric waits for a quorum to be available, up to a maximum time-out value (specified by the UpgradeReplicaSetCheckTimeout property). If the time-out expires, Service Fabric proceeds with the upgrade, regardless of quorum. This setting is set as never (infinite) when rolling forward, and 900 seconds when rolling back.</p>
ForceRestart	<p>If you update a configuration or data package without updating the service code, the service is restarted only if the ForceRestart property is set to true. When the update is complete, Service Fabric notifies the service that a new configuration package or data package is available. The service is responsible for applying the changes. If necessary, the service can restart itself.</p>

The MaxPercentUnhealthyServices, MaxPercentUnhealthyPartitionsPerService, and MaxPercentUnhealthyReplicasPerPartition criteria can be specified per service type for an application instance. Setting these parameters per-service allows for an application to contain different services types with different evaluation policies. For example, a stateless gateway service type can have a MaxPercentUnhealthyPartitionsPerService that is different from a stateful engine service type for a particular application instance.

## Next steps

[Upgrading your Application Using Visual Studio](#) walks you through an application upgrade using Visual Studio.

[Upgrading your Application Using Powershell](#) walks you through an application upgrade using PowerShell.

[Upgrading your Application using Service Fabric CLI on Linux](#) walks you through an application upgrade using Service Fabric CLI.

[Upgrading your application using Service Fabric Eclipse Plugin](#)

Make your application upgrades compatible by learning how to use [Data Serialization](#).

Learn how to use advanced functionality while upgrading your application by referring to [Advanced Topics](#).

Fix common problems in application upgrades by referring to the steps in [Troubleshooting Application Upgrades](#).

# How data serialization affects an application upgrade

6/30/2017 • 3 min to read • [Edit Online](#)

In a [rolling application upgrade](#), the upgrade is applied to a subset of nodes, one upgrade domain at a time. During this process, some upgrade domains are on the newer version of your application, and some upgrade domains are on the older version of your application. During the rollout, the new version of your application must be able to read the old version of your data, and the old version of your application must be able to read the new version of your data. If the data format is not forward and backward compatible, the upgrade may fail, or worse, data may be lost or corrupted. This article discusses what constitutes your data format and offers best practices for ensuring that your data is forward and backward compatible.

## What makes up your data format?

In Azure Service Fabric, the data that is persisted and replicated comes from your C# classes. For applications that use [Reliable Collections](#), that data is the objects in the reliable dictionaries and queues. For applications that use [Reliable Actors](#), that is the backing state for the actor. These C# classes must be serializable to be persisted and replicated. Therefore, the data format is defined by the fields and properties that are serialized, as well as how they are serialized. For example, in an `IReliableDictionary<int, MyClass>` the data is a serialized `int` and a serialized `MyClass`.

### Code changes that result in a data format change

Since the data format is determined by C# classes, changes to the classes may cause a data format change. Care must be taken to ensure that a rolling upgrade can handle the data format change. Examples that may cause data format changes:

- Adding or removing fields or properties
- Renaming fields or properties
- Changing the types of fields or properties
- Changing the class name or namespace

### Data Contract as the default serializer

The serializer is generally responsible for reading the data and deserializing it into the current version, even if the data is in an older or *newer* version. The default serializer is the [Data Contract serializer](#), which has well-defined versioning rules. Reliable Collections allow the serializer to be overridden, but Reliable Actors currently do not. The data serializer plays an important role in enabling rolling upgrades. The Data Contract serializer is the serializer that we recommend for Service Fabric applications.

## How the data format affects a rolling upgrade

During a rolling upgrade, there are two main scenarios where the serializer may encounter an older or *newer* version of your data:

1. After a node is upgraded and starts back up, the new serializer will load the data that was persisted to disk by the old version.
2. During the rolling upgrade, the cluster will contain a mix of the old and new versions of your code. Since replicas may be placed in different upgrade domains, and replicas send data to each other, the new and/or old version of your data may be encountered by the new and/or old version of your serializer.

#### **NOTE**

The "new version" and "old version" here refer to the version of your code that is running. The "new serializer" refers to the serializer code that is executing in the new version of your application. The "new data" refers to the serialized C# class from the new version of your application.

The two versions of code and data format must be both forward and backward compatible. If they are not compatible, the rolling upgrade may fail or data may be lost. The rolling upgrade may fail because the code or serializer may throw exceptions or a fault when it encounters the other version. Data may be lost if, for example, a new property was added but the old serializer discards it during deserialization.

Data Contract is the recommended solution for ensuring that your data is compatible. It has well-defined versioning rules for adding, removing, and changing fields. It also has support for dealing with unknown fields, hooking into the serialization and deserialization process, and dealing with class inheritance. For more information, see [Using Data Contract](#).

## Next steps

[Upgrading your Application Using Visual Studio](#) walks you through an application upgrade using Visual Studio.

[Upgrading your Application Using Powershell](#) walks you through an application upgrade using PowerShell.

Control how your application upgrades by using [Upgrade Parameters](#).

Learn how to use advanced functionality while upgrading your application by referring to [Advanced Topics](#).

Fix common problems in application upgrades by referring to the steps in [Troubleshooting Application Upgrades](#)

.

# Service Fabric application upgrade: advanced topics

8/15/2017 • 4 min to read • [Edit Online](#)

## Adding or removing services during an application upgrade

If a new service is added to an application that is already deployed, and published as an upgrade, the new service is added to the deployed application. Such an upgrade does not affect any of the services that were already part of the application. However, an instance of the service that was added must be started for the new service to be active (using the `New-ServiceFabricService` cmdlet).

Services can also be removed from an application as part of an upgrade. However, all current instances of the to-be-deleted service must be stopped before proceeding with the upgrade (using the `Remove-ServiceFabricService` cmdlet).

## Manual upgrade mode

### NOTE

The unmonitored manual mode should be considered only for a failed or suspended upgrade. The monitored mode is the recommended upgrade mode for Service Fabric applications.

Azure Service Fabric provides multiple upgrade modes to support development and production clusters.

Deployment options chosen may be different for different environments.

The monitored rolling application upgrade is the most typical upgrade to use in the production environment. When the upgrade policy is specified, Service Fabric ensures that the application is healthy before the upgrade proceeds.

The application administrator can use the manual rolling application upgrade mode to have total control over the upgrade progress through the various upgrade domains. This mode is useful when a customized or complex health evaluation policy is required, or a non-conventional upgrade is happening (for example, the application is already in data loss).

Finally, the automated rolling application upgrade is useful for development or testing environments to provide a fast iteration cycle during service development.

## Change to manual upgrade mode

**Manual**--Stop the application upgrade at the current UD and change the upgrade mode to Unmonitored Manual. The administrator needs to manually call **MoveNextApplicationUpgradeDomainAsync** to proceed with the upgrade or trigger a rollback by initiating a new upgrade. Once the upgrade enters into the Manual mode, it stays in the Manual mode until a new upgrade is initiated. The **GetApplicationUpgradeProgressAsync** command returns FABRIC\_APPLICATION\_UPGRADE\_STATE\_ROLLING\_FORWARD\_PENDING.

## Upgrade with a diff package

A Service Fabric application can be upgraded by provisioning with a full, self-contained application package. An application can also be upgraded by using a diff package that contains only the updated application files, the updated application manifest, and the service manifest files.

A full application package contains all the files necessary to start and run a Service Fabric application. A diff

package contains only the files that changed between the last provision and the current upgrade, plus the full application manifest and the service manifest files. Any reference in the application manifest or service manifest that can't be found in the build layout is searched for in the image store.

Full application packages are required for the first installation of an application to the cluster. Subsequent updates can be either a full application package or a diff package.

Occasions when using a diff package would be a good choice:

- A diff package is preferred when you have a large application package that references several service manifest files and/or several code packages, config packages, or data packages.
- A diff package is preferred when you have a deployment system that generates the build layout directly from your application build process. In this case, even though the code hasn't changed, newly built assemblies get a different checksum. Using a full application package would require you to update the version on all code packages. Using a diff package, you only provide the files that changed and the manifest files where the version has changed.

When an application is upgraded using Visual Studio, the diff package is published automatically. To create a diff package manually, the application manifest, and the service manifests must be updated, but only the changed packages should be included in the final application package.

For example, let's start with the following application (version numbers provided for ease of understanding):

```
app1      1.0.0
  service1  1.0.0
    code     1.0.0
    config   1.0.0
  service2  1.0.0
    code     1.0.0
    config   1.0.0
```

Now, let's assume you wanted to update only the code package of service1 using a diff package using PowerShell. Now, your updated application has the following folder structure:

```
app1      2.0.0      <-- new version
  service1  2.0.0      <-- new version
    code     2.0.0      <-- new version
    config   1.0.0
  service2  1.0.0
    code     1.0.0
    config   1.0.0
```

In this case, you update the application manifest to 2.0.0, and the service manifest for service1 to reflect the code package update. The folder for your application package would have the following structure:

```
app1/
  service1/
    code/
```

## Next steps

[Upgrading your Application Using Visual Studio](#) walks you through an application upgrade using Visual Studio.

[Upgrading your Application Using Powershell](#) walks you through an application upgrade using PowerShell.

Control how your application upgrades by using [Upgrade Parameters](#).

Make your application upgrades compatible by learning how to use [Data Serialization](#).

Fix common problems in application upgrades by referring to the steps in [Troubleshooting Application Upgrades](#).

# Introduction to the Fault Analysis Service

6/27/2017 • 6 min to read • [Edit Online](#)

The Fault Analysis Service is designed for testing services that are built on Microsoft Azure Service Fabric. With the Fault Analysis Service you can induce meaningful faults and run complete test scenarios against your applications. These faults and scenarios exercise and validate the numerous states and transitions that a service will experience throughout its lifetime, all in a controlled, safe, and consistent manner.

Actions are the individual faults targeting a service for testing it. A service developer can use these as building blocks to write complicated scenarios. For example:

- Restart a node to simulate any number of situations where a machine or VM is rebooted.
- Move a replica of your stateful service to simulate load balancing, failover, or application upgrade.
- Invoke quorum loss on a stateful service to create a situation where write operations can't proceed because there aren't enough "back-up" or "secondary" replicas to accept new data.
- Invoke data loss on a stateful service to create a situation where all in-memory state is completely wiped out.

Scenarios are complex operations composed of one or more actions. The Fault Analysis Service provides two built-in complete scenarios:

- Chaos Scenario
- Failover Scenario

## Testing as a service

The Fault Analysis Service is a Service Fabric system service that is automatically started with a Service Fabric cluster. This service acts as the host for fault injection, test scenario execution, and health analysis.

SYSTEM					
Name	Service Type	Version	Service Kind	Health State	Status
fabric/System/ClusterManagerService	ClusterManagerServiceType	5.0.135.9590	Stateful	OK	Active
fabric/System/FailoverManagerService	FMServiceType	5.0.135.9590	Stateful	OK	Active
fabric/System/FaultAnalysisService	FaultAnalysisServiceType	5.0.135.9590	Stateful	OK	Active
fabric/System/NamingService	NamingStoreService	5.0.135.9590	Stateful	OK	Active

When a fault action or test scenario is initiated, a command is sent to the Fault Analysis Service to run the fault action or test scenario. The Fault Analysis Service is stateful so that it can reliably run faults and scenarios and validate results. For example, a long-running test scenario can be reliably executed by the Fault Analysis Service. And because tests are being executed inside the cluster, the service can examine the state of the cluster and your services to provide more in-depth information about failures.

## Testing distributed systems

Service Fabric makes the job of writing and managing distributed scalable applications significantly easier. The Fault Analysis Service makes testing a distributed application similarly easier. There are three main issues that need to be solved while testing:

1. Simulating/generating failures that might occur in real-world scenarios: One of the important aspects of Service Fabric is that it enables distributed applications to recover from various failures. However, to test that the application is able to recover from these failures, we need a mechanism to simulate/generate these real-world failures in a controlled test environment.

2. The ability to generate correlated failures: Basic failures in the system, such as network failures and machine failures, are easy to produce individually. Generating a significant number of scenarios that can happen in the real world as a result of the interactions of these individual failures is non-trivial.
3. Unified experience across various levels of development and deployment: There are many fault injection systems that can do various types of failures. However, the experience in all of these is poor when moving from one-box developer scenarios, to running the same tests in large test environments, to using them for tests in production.

While there are many mechanisms to solve these problems, a system that does the same with required guarantees--all the way from a one-box developer environment, to test in production clusters--is missing. The Fault Analysis Service helps the application developers concentrate on testing their business logic. The Fault Analysis Service provides all the capabilities needed to test the interaction of the service with the underlying distributed system.

### **Simulating/generating real-world failure scenarios**

To test the robustness of a distributed system against failures, we need a mechanism to generate failures. While in theory, generating a failure like a node down seems easy, it starts hitting the same set of consistency problems that Service Fabric is trying to solve. As an example, if we want to shut down a node, the required workflow is the following:

1. From the client, issue a shutdown node request.
2. Send the request to the right node.
  - a. If the node is not found, it should fail.
  - b. If the node is found, it should return only if the node is shut down.

To verify the failure from a test perspective, the test needs to know that when this failure is induced, the failure actually happens. The guarantee that Service Fabric provides is that either the node will go down or was already down when the command reached the node. In either case the test should be able to correctly reason about the state and succeed or fail correctly in its validation. A system implemented outside of Service Fabric to do the same set of failures could hit many network, hardware, and software issues, which would prevent it from providing the preceding guarantees. In the presence of the issues stated before, Service Fabric will reconfigure the cluster state to work around the issues, and hence the Fault Analysis Service will still be able to give the right set of guarantees.

### **Generating required events and scenarios**

While simulating a real-world failure consistently is tough to start with, the ability to generate correlated failures is even tougher. For example, a data loss happens in a stateful persisted service when the following things happen:

1. Only a write quorum of the replicas are caught up on replication. All the secondary replicas lag behind the primary.
2. The write quorum goes down because of the replicas going down (due to a code package or node going down).
3. The write quorum cannot come back up because the data for the replicas is lost (due to disk corruption or machine reimaging).

These correlated failures do happen in the real world but not as frequently as individual failures. The ability to test for these scenarios before they happen in production is critical. Even more important is the ability to simulate these scenarios with production workloads in controlled circumstances (in the middle of the day with all engineers on deck). That is much better than having it happen for the first time in production at 2:00 A.M.

### **Unified experience across different environments**

The practice traditionally has been to create three different sets of experiences, one for the development environment, one for tests, and one for production. The model was:

1. In the development environment, produce state transitions that allow unit tests of individual methods.
2. In the test environment, produce failures to allow end-to-end tests that exercise various failure scenarios.
3. Keep the production environment pristine to prevent any non-natural failures and to ensure that there is extremely quick human response to failure.

In Service Fabric, through the Fault Analysis Service, we are proposing to turn this around and use the same methodology from developer environment to production. There are two ways to achieve this:

1. To induce controlled failures, use the Fault Analysis Service APIs from a one-box environment all the way to production clusters.
2. To give the cluster a fever that causes automatic induction of failures, use the Fault Analysis Service to generate automatic failures. Controlling the rate of failures through configuration enables the same service to be tested differently in different environments.

With Service Fabric, though the scale of failures would be different in the different environments, the actual mechanisms would be identical. This allows for a much quicker code-to-deployment pipeline and the ability to test the services under real-world loads.

## Using the Fault Analysis Service

### C#

Fault Analysis Service features are in the System.Fabric namespace in the Microsoft.ServiceFabric NuGet package. To use the Fault Analysis Service features, include the nuget package as a reference in your project.

### PowerShell

To use PowerShell, you must install the Service Fabric SDK. After the SDK is installed, the ServiceFabric PowerShell module is auto loaded for you to use.

## Next steps

To create truly cloud-scale services, it is critical to ensure, both before and after deployment, that services can withstand real world failures. In the services world today, the ability to innovate quickly and move code to production quickly is very important. The Fault Analysis Service helps service developers to do precisely that.

Begin testing your applications and services using the built-in [test scenarios](#), or author your own test scenarios using the [fault actions](#) provided by the Fault Analysis Service.

# Create Service Fabric clusters on Windows Server or Linux

9/25/2017 • 4 min to read • [Edit Online](#)

An Azure Service Fabric cluster is a network-connected set of virtual or physical machines into which your microservices are deployed and managed. A machine or VM that is part of a cluster is called a cluster node. Clusters can scale to thousands of nodes. If you add new nodes to the cluster, Service Fabric rebalances the service partition replicas and instances across the increased number of nodes. Overall application performance improves and contention for access to memory decreases. If the nodes in the cluster are not being used efficiently, you can decrease the number of nodes in the cluster. Service Fabric again rebalances the partition replicas and instances across the decreased number of nodes to make better use of the hardware on each node.

Service Fabric allows for the creation of Service Fabric clusters on any VMs or computers running Windows Server or Linux. This means you are able to deploy and run Service Fabric applications in any environment where you have a set of Windows Server or Linux computers that are interconnected, be it on-premises, Microsoft Azure or with any cloud provider.

## Create Service Fabric clusters on Azure

Creating a cluster on Azure is done either via a Resource Model template or the [Azure portal](#). Read [Create a Service Fabric cluster by using a Resource Manager template](#) or [Create a Service Fabric cluster from the Azure portal](#) for more information.

## Supported operating systems for clusters on Azure

You are able to create clusters on virtual machines running these operating systems:

- Windows Server 2012 R2
- Windows Server 2016
- Linux Ubuntu 16.04

## Create Service Fabric standalone clusters on-premises or with any cloud provider

Service Fabric provides an install package for you to create standalone Service Fabric clusters on-premises or on any cloud provider.

For more information on setting up standalone Service Fabric clusters on Windows Server, read [Service Fabric cluster creation for Windows Server](#)

### NOTE

Standalone clusters currently aren't supported for Linux. Linux is supported on one-box for development and Azure Linux multi-machine clusters.

## Any cloud deployments vs. on-premises deployments

The process for creating a Service Fabric cluster on-premises is similar to the process of creating a cluster on any cloud of your choice with a set of VMs. The initial steps to provision the VMs are governed by the cloud provider or on-premises environment that you are using. Once you have a set of VMs with network connectivity enabled

between them, then the steps to set up the Service Fabric package, edit the cluster settings, and run the cluster creation and management scripts are identical. This ensures that your knowledge and experience of operating and managing Service Fabric clusters is transferable when you choose to target new hosting environments.

### Benefits of creating standalone Service Fabric clusters

- You are free to choose any cloud provider to host your cluster.
- Service Fabric applications, once written, can be run in multiple hosting environments with minimal to no changes.
- Knowledge of building Service Fabric applications carries over from one hosting environment to another.
- Operational experience of running and managing Service Fabric clusters carries over from one environment to another.
- Broad customer reach is unbounded by hosting environment constraints.
- An extra layer of reliability and protection against widespread outages exists because you can move the services over to another deployment environment if a data center or cloud provider has a blackout.

## Supported operating systems for standalone clusters

You are able to create clusters on VMs or computers running these operating systems (Linux is not yet supported):

- Windows Server 2012 R2
- Windows Server 2016

## Advantages of Service Fabric clusters on Azure over standalone Service Fabric clusters created on-premises

Running Service Fabric clusters on Azure provides advantages over the on-premises option, so if you don't have specific needs for where you run your clusters, then we suggest that you run them on Azure. On Azure, we provide integration with other Azure features and services, which makes operations and management of the cluster easier and more reliable.

- **Azure portal:** Azure portal makes it easy to create and manage clusters.
- **Azure Resource Manager:** Use of Azure Resource Manager allows easy management of all resources used by the cluster as a unit and simplifies cost tracking and billing.
- **Service Fabric Cluster as an Azure Resource** A Service Fabric cluster is an Azure resource, so you can model it like you do other resources in Azure.
- **Integration with Azure Infrastructure** Service Fabric coordinates with the underlying Azure infrastructure for OS, network, and other upgrades to improve availability and reliability of your applications.
- **Diagnostics:** On Azure, we provide integration with Azure diagnostics and Log Analytics.
- **Auto-scaling:** For clusters on Azure, we provide built-in auto-scaling functionality due to Virtual Machine scale-sets. In on-premises and other cloud environments, you have to build your own auto-scaling feature or scale manually using the APIs that Service Fabric exposes for scaling clusters.

## Next steps

- Create a cluster on VMs or computers running Windows Server: [Service Fabric cluster creation for Windows Server](#)
- Create a cluster on VMs or computers running Linux: [Create a Linux cluster](#)
- Learn about [Service Fabric support options](#)

# Service Fabric cluster capacity planning considerations

10/30/2017 • 16 min to read • [Edit Online](#)

For any production deployment, capacity planning is an important step. Here are some of the items that you have to consider as a part of that process.

- The number of node types your cluster needs to start out with
- The properties of each of node type (size, primary, internet facing, number of VMs, etc.)
- The reliability and durability characteristics of the cluster

Let us briefly review each of these items.

## The number of node types your cluster needs to start out with

First, you need to figure out what the cluster you are creating is going to be used for and what kinds of applications you are planning to deploy into this cluster. If you are not clear on the purpose of the cluster, you are most likely not yet ready to enter the capacity planning process.

Establish the number of node types your cluster needs to start out with. Each node type is mapped to a Virtual Machine Scale Set. Each node type can then be scaled up or down independently, have different sets of ports open, and can have different capacity metrics. So the decision of the number of node types essentially comes down to the following considerations:

- Does your application have multiple services, and do any of them need to be public or internet facing? Typical applications contain a front-end gateway service that receives input from a client, and one or more back-end services that communicate with the front-end services. So in this case, you end up having at least two node types.
- Do your services (that make up your application) have different infrastructure needs such as greater RAM or higher CPU cycles? For example, let us assume that the application that you want to deploy contains a front-end service and a back-end service. The front-end service can run on smaller VMs (VM sizes like D2) that have ports open to the internet. The back-end service, however, is computation intensive and needs to run on larger VMs (with VM sizes like D4, D6, D15) that are not internet facing.

In this example, although you can decide to put all the services on one node type, we recommended that you place them in a cluster with two node types. This allows for each node type to have distinct properties such as internet connectivity or VM size. The number of VMs can be scaled independently, as well.

- Since you cannot predict the future, go with facts you know of and decide on the number of node types that your applications need to start with. You can always add or remove node types later. A Service Fabric cluster must have at least one node type.

## The properties of each node type

The **node type** can be seen as equivalent to roles in Cloud Services. Node types define the VM sizes, the number of VMs, and their properties. Every node type that is defined in a Service Fabric cluster is set up as a separate virtual machine scale set. Virtual machine scale set is an Azure compute resource you can use to deploy and manage a collection of virtual machines as a set. Being defined as distinct virtual machine scale set, each node type can then be scaled up or down independently, have different sets of ports open, and can have different capacity metrics.

Read [this document](#) for more details on the relationship of Nodetypes to virtual machine scale set, how to RDP into one of the instances, open new ports etc.

Your cluster can have more than one node type, but the primary node type (the first one that you define on the portal) must have at least five VMs for clusters used for production workloads (or at least three VMs for test clusters). If you are creating the cluster using a Resource Manager template, then look for **is Primary** attribute under the node type definition. The primary node type is the node type where Service Fabric system services are placed.

### Primary node type

For a cluster with multiple node types, you need to choose one of them to be primary. Here are the characteristics of a primary node type:

- The **minimum size of VMs** for the primary node type is determined by the **durability tier** you choose. The default for the durability tier is Bronze. Scroll down for details on what the durability tier is and the values it can take.
- The **minimum number of VMs** for the primary node type is determined by the **reliability tier** you choose. The default for the reliability tier is Silver. Scroll down for details on what the reliability tier is and the values it can take.
- The Service Fabric system services (for example, the Cluster Manager service or Image Store service) are placed on the primary node type and so the reliability and durability of the cluster is determined by the reliability tier value and durability tier value you select for the primary node type.

The screenshot shows the Service Fabric Explorer interface for a cluster named 'fabric:/System/ClusterManagerService'. The left sidebar shows a tree view of the cluster structure, with a red box highlighting the 'System' node under 'Applications'. The main pane displays the 'ESSENTIALS' tab for the 'ClusterManagerServiceType' service. Key details shown include:

- Name: fabric:/System/ClusterManagerService
- Service Type: ClusterManagerServiceType
- Health State: OK
- Status: Active
- Service Kind: Stateful
- Service Type Version: 4.5.192.9590
- Minimum Replica Size: 3
- Target Replica Size: 5

The 'PARTITIONS' section shows a single partition with a green 'OK' status. The bottom of the pane has checkboxes for 'OK', 'Warning', and 'Error' status filters.

### Non-primary node type

For a cluster with multiple node types, there is one primary node type and the rest of them are non-primary. Here are the characteristics of a non-primary node type:

- The minimum size of VMs for this node type is determined by the durability tier you choose. The default for the durability tier is Bronze. Scroll down for details on what the durability tier is and the values it can take.
- The minimum number of VMs for this node type can be one. However you should choose this number based on the number of replicas of the application/services that you would like to run in this node type. The number of VMs in a node type can be increased after you have deployed the cluster.

# The durability characteristics of the cluster

The durability tier is used to indicate to the system the privileges that your VMs have with the underlying Azure infrastructure. In the primary node type, this privilege allows Service Fabric to pause any VM level infrastructure request (such as a VM reboot, VM reimage, or VM migration) that impact the quorum requirements for the system services and your stateful services. In the non-primary node types, this privilege allows Service Fabric to pause any VM level infrastructure requests like VM reboot, VM reimage, VM migration etc., that impact the quorum requirements for your stateful services running in it.

This privilege is expressed in the following values:

- Gold - The infrastructure Jobs can be paused for a duration of two hours per UD. Gold durability can be enabled only on full node VM skus like D15\_V2, G5 etc.
- Silver - The infrastructure Jobs can be paused for a duration of 10 minutes per UD and is available on all standard VMs of single core and above.
- Bronze - No privileges. This is the default. Only use this durability level for Node Types that run *only* stateless workloads.

## WARNING

NodeTypes running with Bronze durability obtain *no privileges*. This means that infrastructure jobs that impact your stateless workloads will not be stopped or delayed. It is possible that such jobs can still impact your workloads, causing downtime or other issues. For any sort of production workload, running with at least Silver is recommended. You must maintain a minimum count of 5 nodes for any node-type that has a durability of Gold or Silver.

You get to choose durability level for each of your node-types. You can choose one node-type to have a durability level of Gold or silver and the other have Bronze in the same cluster. **You must maintain a minimum count of 5 nodes for any node-type that has a durability of Gold or silver.**

## Advantages of using Silver or Gold durability levels

1. Reduces the number of required steps in a scale-in operation (that is, node deactivation and RemoveServiceFabricNodeState is called automatically)
2. Reduces the risk of data loss due to a customer-initiated in-place VM SKU change operation or Azure infrastructure operations.

## Disadvantages of using Silver or Gold durability levels

1. Deployments to your Virtual Machine Scale Set and other related Azure resources) can be delayed, can time out, or can be blocked entirely by problems in your cluster or at the infrastructure level.
2. Increases the number of [replica lifecycle events](#) (for example, primary swaps) due to automated node deactivations during Azure infrastructure operations.

## Recommendations on when to use Silver or Gold durability levels

Use Silver or Gold durability for all node types that host stateful services you expect to scale-in (reduce VM instance count) frequently, and you would prefer that deployment operations be delayed in favor of simplifying these scale-in operations. The scale-out scenarios (adding VMs instances) do not play into your choice of the durability tier, only scale-in does.

## Changing durability levels

- Node types with durability levels of Silver or Gold cannot be downgraded to Bronze.
- Upgrading from Bronze to Silver or Gold can take a few hours.
- When changing durability level, be sure to update it in both the Service Fabric extension configuration in your VMSS resource, and in the node type definition in your Service Fabric cluster resource. These values must

match.

### **Operational Recommendations for the node type that you have set to silver or gold durability level.**

1. Keep your cluster and applications healthy at all times, and make sure that applications respond to all [Service replica lifecycle events](#) (like replica in build is stuck) in a timely fashion.
2. Adopt safer ways to make a VM SKU change (Scale up/down): Changing the VM SKU of a Virtual Machine Scale Set is inherently an unsafe operation and so should be avoided if possible. Here is the process you can follow to avoid common issues.
  - **For non-primary nodetypes:** It is recommended that you create new Virtual Machine Scale Set, modify the service placement constraint to include the new Virtual Machine Scale Set/node type and then reduce the old Virtual Machine Scale Set instance count to 0, one node at a time (this is to make sure that removal of the nodes do not impact the reliability of the cluster).
  - **For the primary nodetype:** Our recommendation is that you do not change VM SKU of the primary node type. Changing of the primary node type SKU is not supported. If the reason for the new SKU is capacity, we recommend adding more instances. If that not possible, create a new cluster and [restore application state](#) (if applicable) from your old cluster. You do not need to restore any system service state, they are recreated when you deploy your applications to your new cluster. If you were just running stateless applications on your cluster, then all you do is deploy your applications to the new cluster, you have nothing to restore. If you decide to go the unsupported route and want to change the VM SKU, then make modifications to the Virtual Machine Scale Set Model definition to reflect the new SKU. If your cluster has only one nodetype, then make sure that all your stateful applications respond to all [Service replica lifecycle events](#) (like replica in build is stuck) in a timely fashion and that your service replica rebuild duration is less than five minutes (for Silver durability level).

#### **WARNING**

Changing the VM SKU Size for VM Scale Sets not running at least Silver durability is not recommended. Changing VM SKU Size is a data-destructive in-place infrastructure operation. Without at least some ability to delay or monitor this change, it is possible that the operation can cause data loss for stateful services or cause other unforeseen operational issues, even for stateless workloads.

1. Maintain a minimum count of five nodes for any Virtual Machine Scale Set that has durability level of Gold or Silver enabled
2. Do not delete random VM instances, always use Virtual Machine Scale Set scale down feature. The deletion of random VM instances has a potential of creating imbalances in the VM instance spread across UD and FD. This imbalance could adversely affect the systems ability to properly load balance amongst the service instances/Service replicas.
3. If using Autoscale, then set the rules such that scale in (removing of VM instances) are done only one node at a time. Scaling down more than one instance at a time is not safe.
4. If Scaling down a primary node type, you should never scale it down more than what the reliability tier allows.

## The reliability characteristics of the cluster

The reliability tier is used to set the number of replicas of the system services that you want to run in this cluster on the primary node type. The more the number of replicas, the more reliable the system services are in your cluster.

The reliability tier can take the following values:

- Platinum - Run the System services with a target replica set count of 9
- Gold - Run the System services with a target replica set count of 7
- Silver - Run the System services with a target replica set count of 5

- Bronze - Run the System services with a target replica set count of 3

**NOTE**

The reliability tier you choose determines the minimum number of nodes your primary node type must have.

### Recommendations for the reliability tier.

When you increase or decrease the size of your cluster (the sum of VM instances in all node types), you must update the reliability of your cluster from one tier to another. Doing this triggers the cluster upgrades needed to change the system services replica set count. Wait for the upgrade in progress to complete before making any other changes to the cluster, like adding nodes. You can monitor the progress of the upgrade on Service Fabric Explorer or by running [Get-ServiceFabricClusterUpgrade](#)

Here is the recommendation on choosing the reliability tier.

CLUSTER SIZE	RELIABILITY TIER
1	Do not specify the Reliability Tier parameter, the system calculates it
3	Bronze
5 or 6	Silver
7 or 8	Gold
9 and up	Platinum

## Primary node type - Capacity Guidance

Here is the guidance for planning the primary node type capacity

1. **Number of VM instances to run any production workload in Azure:** You must specify a minimum Primary Node type size of 5.
2. **Number of VM instances to run test workloads in Azure** You can specify a minimum primary node type size of 1 or 3. The one node cluster, runs with a special configuration and so, scale out of that cluster is not supported. The one node cluster, has no reliability and so in your Resource Manager template, you have to remove/not specify that configuration (not setting the configuration value is not enough). If you set up the one node cluster set up via portal, then the configuration is automatically taken care of. 1 and 3 node clusters are not supported for running production workloads.
3. **VM SKU:** Primary node type is where the system services run, so the VM SKU you choose for it, must take into account the overall peak load you plan to place into the cluster. Here is an analogy to illustrate what I mean here - Think of the primary node type as your "Lungs", it is what provides oxygen to your brain, and so if the brain does not get enough oxygen, your body suffers.

Since the capacity needs of a cluster is determined by workload you plan to run in the cluster, we cannot provide you with qualitative guidance for your specific workload, however here is the broad guidance to help you get started

For production workloads

- The recommended VM SKU is Standard D3 or Standard D3\_V2 or equivalent with a minimum of 14 GB of local SSD.
- The minimum supported use VM SKU is Standard D1 or Standard D1\_V2 or equivalent with a minimum of 14

GB of local SSD.

- Partial core VM SKUs like Standard A0 are not supported for production workloads.
- Standard A1 SKU is not supported for production workloads for performance reasons.

#### WARNING

Currently, changing the Primary node VM SKU size on a running cluster is not supported. So choose the primary node type VM SKU carefully, taking into account your capacity future needs. At this time, the only supported way to move your primary node type to a new VM SKU (smaller or larger) is to create a new cluster with the right capacity, deploy your applications to it and then restoring the application state (if applicable) from the [latest service backups](#) you have taken from the old cluster. You do not need to restore any system service state, they are recreated when you deploy applications to your new cluster. If you were just running stateless applications on your cluster, then all you do is deploy your applications to the new cluster, you have nothing to restore.

## Non-Primary node type - Capacity Guidance for stateful workloads

This guidance is for stateful Workloads using Service fabric [reliable collections or reliable Actors](#) that you are running in the non-primary node type.

**Number of VM instances:** For production workloads that are stateful, it is recommended that you run them with a minimum and target replica count of 5. This means that in steady state you end up with a replica (from a replica set) in each fault domain and upgrade domain. The whole reliability tier concept for the primary node type is a way to specify this setting for system services. So the same consideration applies to your stateful services as well.

So for production workloads, the minimum recommended non-Primary Node type size is 5, if you are running stateful workloads in it.

**VM SKU:** This is the node type where your application services are running, so the VM SKU you choose for it, must take into account the peak load you plan to place into each Node. The capacity needs of the nodetype, is determined by workload you plan to run in the cluster, so we cannot provide you with qualitative guidance for your specific workload, however here is the broad guidance to help you get started

For production workloads

- The recommended VM SKU is Standard D3 or Standard D3\_V2 or equivalent with a minimum of 14 GB of local SSD.
- The minimum supported use VM SKU is Standard D1 or Standard D1\_V2 or equivalent with a minimum of 14 GB of local SSD.
- Partial core VM SKUs like Standard A0 are not supported for production workloads.
- Standard A1 SKU is specifically not supported for production workloads for performance reasons.

## Non-Primary node type - Capacity Guidance for stateless workloads

This guidance of stateless Workloads that you are running on the non-primary nodetype.

**Number of VM instances:** For production workloads that are stateless, the minimum supported non-Primary Node type size is 2. This allows you to run two stateless instances of your application and allowing your service to survive the loss of a VM instance.

#### NOTE

If your cluster is running on a service fabric version less than 5.6, due to a defect in the runtime (this issue is fixed in 5.6), scaling down a non-primary node type to less than 5, results in cluster health turning unhealthy, till you call [Remove-ServiceFabricNodeState cmd](#) with the appropriate node name. Read [perform Service Fabric cluster in or out](#) for more details

**VM SKU:** This is the node type where your application services are running, so the VM SKU you choose for it, must take into account the peak load you plan to place into each Node. The capacity needs of the nodetype, is determined by workload you plan to run in the cluster, So we cannot provide you with qualitative guidance for your specific workload, however here is the broad guidance to help you get started

For production workloads

- The recommended VM SKU is Standard D3 or Standard D3\_V2 or equivalent.
- The minimum supported use VM SKU is Standard D1 or Standard D1\_V2 or equivalent.
- Partial core VM SKUs like Standard A0 are not supported for production workloads.
- Standard A1 SKU is not supported for production workloads for performance reasons.

## Next steps

Once you finish your capacity planning and set up a cluster, read the following:

- [Service Fabric cluster security](#)
- [Disaster recovery planning](#)
- [Relationship of Nodetypes to Virtual machine scale set](#)

# Disaster recovery in Azure Service Fabric

9/5/2017 • 17 min to read • [Edit Online](#)

A critical part of delivering high-availability is ensuring that services can survive all different types of failures. This is especially important for failures that are unplanned and outside of your control. This article describes some common failure modes that could be disasters if not modeled and managed correctly. It also discusses mitigations and actions to take if a disaster happened anyway. The goal is to limit or eliminate the risk of downtime or data loss when they occur, planned or otherwise, occur.

## Avoiding disaster

Service Fabric's primary goal is to help you model both your environment and your services in such a way that common failure types are not disasters.

In general there are two types of disaster/failure scenarios:

1. Hardware or software faults
2. Operational faults

### **Hardware and software faults**

Hardware and software faults are unpredictable. The easiest way to survive faults is running more copies of the service spanned across hardware or software fault boundaries. For example, if your service is running only on one particular machine, then the failure of that one machine is a disaster for that service. The simple way to avoid this disaster is to ensure that the service is actually running on multiple machines. Testing is also necessary to ensure the failure of one machine doesn't disrupt the running service. Capacity planning ensures a replacement instance can be created elsewhere and that reduction in capacity doesn't overload the remaining services. The same pattern works regardless of what you're trying to avoid the failure of. For example, if you're concerned about the failure of a SAN, you run across multiple SANs. If you're concerned about the loss of a rack of servers, you run across multiple racks. If you're worried about the loss of datacenters, your service should run across multiple Azure regions or datacenters.

When running in this type of spanned mode, you're still subject to some types of simultaneous failures, but single and even multiple failures of a particular type (ex: a single VM or network link failing) are automatically handled (and so no longer a "disaster"). Service Fabric provides many mechanisms for expanding the cluster and handles bringing failed nodes and services back. Service Fabric also allows running many instances of your services in order to avoid these types of unplanned failures from turning into real disasters.

There may be reasons why running a deployment large enough to span over failures is not feasible. For example, it may take more hardware resources than you're willing to pay for relative to the chance of failure. When dealing with distributed applications, it could be that additional communication hops or state replication costs across geographic distances causes unacceptable latency. Where this line is drawn differs for each application. For software faults specifically, the fault could be in the service that you are trying to scale. In this case more copies don't prevent the disaster, since the failure condition is correlated across all the instances.

### **Operational faults**

Even if your service is spanned across the globe with many redundancies, it can still experience disastrous events. For example, if someone accidentally reconfigures the dns name for the service, or deletes it outright. As an example, let's say you had a stateful Service Fabric service, and someone deleted that service accidentally. Unless there's some other mitigation, that service and all of the state it had is now gone. These types of operational disasters ("oops") require different mitigations and steps for recovery than regular unplanned failures.

The best ways to avoid these types of operational faults are to

1. restrict operational access to the environment
2. strictly audit dangerous operations
3. impose automation, prevent manual or out of band changes, and validate specific changes against the actual environment before enacting them
4. ensure that destructive operations are "soft". Soft operations don't take effect immediately or can be undone within some time window

Service Fabric provides some mechanisms to prevent operational faults, such as providing [role-based](#) access control for cluster operations. However, most of these operational faults require organizational efforts and other systems. Service Fabric does provide some mechanism for surviving operational faults, most notably backup and restore for stateful services.

## Managing failures

The goal of Service Fabric is almost always automatic management of failures. However, in order to handle some types of failures, services must have additional code. Other types of failures should *not* be automatically addressed because of safety and business continuity reasons.

### Handling single failures

Single machines can fail for all sorts of reasons. Some of these are hardware causes, like power supplies and networking hardware failures. Other failures are in software. These include failures of the actual operating system and the service itself. Service Fabric automatically detects these types of failures, including cases where the machine becomes isolated from other machines due to network issues.

Regardless of the type of service, running a single instance results in downtime for that service if that single copy of the code fails for any reason.

In order to handle any single failure, the simplest thing you can do is to ensure that your services run on more than one node by default. For stateless services, this can be accomplished by having an `InstanceCount` greater than 1. For stateful services, the minimum recommendation is always a `TargetReplicaSetSize` and `MinReplicaSetSize` of at least 3. Running more copies of your service code ensures that your service can handle any single failure automatically.

### Handling coordinated failures

Coordinated failures can happen in a cluster due to either planned or unplanned infrastructure failures and changes, or planned software changes. Service Fabric models infrastructure zones that experience coordinated failures as Fault Domains. Areas that will experience coordinated software changes are modeled as Upgrade Domains. More information about fault and upgrade domains is in [this document](#) that describes cluster topology and definition.

By default Service Fabric considers fault and upgrade domains when planning where your services should run. By default, Service Fabric tries to ensure that your services run across several fault and upgrade domains so if planned or unplanned changes happen your services remain available.

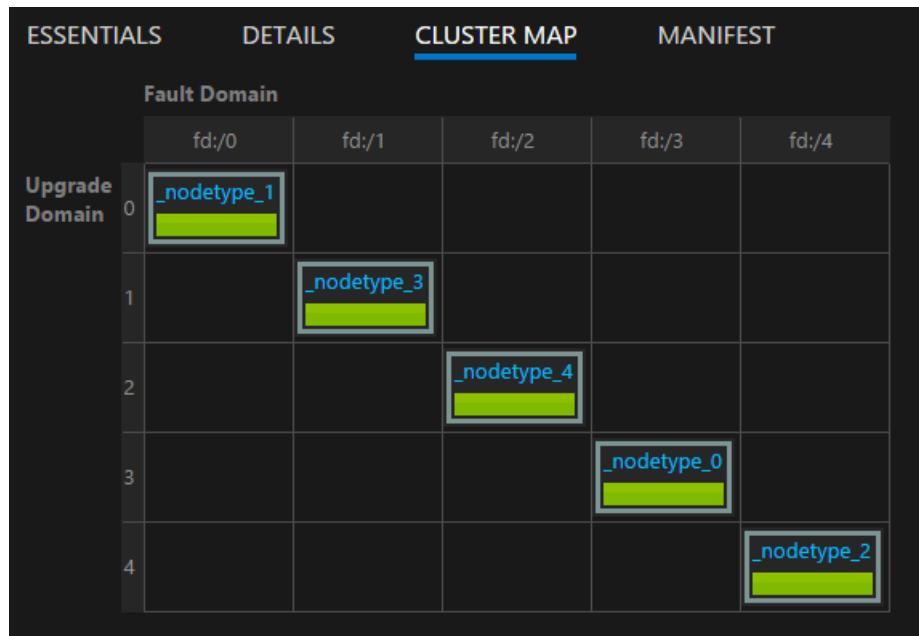
For example, let's say that failure of a power source causes a rack of machines to fail simultaneously. With multiple copies of the service running the loss of many machines in fault domain failure turns into just another example of single failure for a given service. This is why managing fault domains is critical to ensuring high availability of your services. When running Service Fabric in Azure, fault domains are managed automatically. In other environments they may not be. If you're building your own clusters on premises, be sure to map and plan your fault domain layout correctly.

Upgrade Domains are useful for modeling areas where software is going to be upgraded at the same time. Because of this, Upgrade Domains also often define the boundaries where software is taken down during planned upgrades.

Upgrades of both Service Fabric and your services follow the same model. For more on rolling upgrades, upgrade domains, and the Service Fabric health model that helps prevent unintended changes from impacting the cluster and your service, see these documents:

- [Application Upgrade](#)
- [Application Upgrade Tutorial](#)
- [Service Fabric Health Model](#)

You can visualize the layout of your cluster using the cluster map provided in [Service Fabric Explorer](#):



#### NOTE

Modeling areas of failure, rolling upgrades, running many instances of your service code and state, placement rules to ensure your services run across fault and upgrade domains, and built-in health monitoring are just **some** of the features that Service Fabric provides in order to keep normal operational issues and failures from turning into disasters.

### Handling simultaneous hardware or software failures

Above we talked about single failures. As you can see, are easy to handle for both stateless and stateful services just by keeping more copies of the code (and state) running across fault and upgrade domains. Multiple simultaneous random failures can also happen. These are more likely to lead to an actual disaster.

### Random failures leading to service failures

Let's say that the service had an `InstanceCount` of 5, and several nodes running those instances all failed at the same time. Service Fabric responds by automatically creating replacement instances on other nodes. It will continue creating replacements until the service is back to its desired instance count. As another example, let's say there was a stateless service with an `InstanceCount` of -1, meaning it runs on all valid nodes in the cluster. Let's say that some of those instances were to fail. In this case, Service Fabric notices that the service is not in its desired state, and tries to create the instances on the nodes where they are missing.

For stateful services the situation depends on whether the service has persisted state or not. It also depends on how many replicas the service had and how many failed. Determining whether a disaster occurred for a stateful service and managing it follows three stages:

1. Determining if there has been quorum loss or not
  - A quorum loss is any time a majority of the replicas of a stateful service are down at the same time, including the Primary.
2. Determining if the quorum loss is permanent or not

- Most of the time, failures are transient. Processes are restarted, nodes are restarted, VMs are relaunched, network partitions heal. Sometimes though, failures are permanent.
  - For services without persisted state, a failure of a quorum or more of replicas results *immediately* in permanent quorum loss. When Service Fabric detects quorum loss in a stateful non-persistent service, it immediately proceeds to step 3 by declaring (potential) dataloss. Proceeding to dataloss makes sense because Service Fabric knows that there's no point in waiting for the replicas to come back, because even if they were recovered they would be empty.
  - For stateful persistent services, a failure of a quorum or more of replicas causes Service Fabric to start waiting for the replicas to come back and restore quorum. This results in a service outage for any *writes* to the affected partition (or "replica set") of the service. However, reads may still be possible with reduced consistency guarantees. The default amount of time that Service Fabric waits for quorum to be restored is infinite, since proceeding is a (potential) dataloss event and carries other risks. Overriding the default `QuorumLossWaitDuration` value is possible but is not recommended. Instead at this time, all efforts should be made to restore the down replicas. This requires bringing the nodes that are down back up, and ensuring that they can remount the drives where they stored the local persistent state. If the quorum loss is caused by process failure, Service Fabric automatically tries to recreate the processes and restart the replicas inside them. If this fails, Service Fabric reports health errors. If these can be resolved then the replicas usually come back. Sometimes, though, the replicas can't be brought back. For example, the drives may all have failed, or the machines physically destroyed somehow. In these cases, we now have a permanent quorum loss event. To tell Service Fabric to stop waiting for the down replicas to come back, a cluster administrator must determine which partitions of which services are affected and call the `Repair-ServiceFabricPartition -PartitionId` or `System.Fabric.FabricClient.ClusterManagementClient.RecoverPartitionAsync(Guid partitionId)` API.

This API allows specifying the ID of the partition to move out of QuorumLoss and into potential dataloss.

#### NOTE

It is *never* safe to use this API other than in a targeted way against specific partitions.

## 1. Determining if there has been actual data loss, and restoring from backups

- When Service Fabric calls the `OnDataLossAsync` method it is always because of *suspected* dataloss. Service Fabric ensures that this call is delivered to the *best* remaining replica. This is whichever replica has made the most progress. The reason we always say *suspected* dataloss is that it is possible that the remaining replica actually has all same state as the Primary did when it went down. However, without that state to compare it to, there's no good way for Service Fabric or operators to know for sure. At this point, Service Fabric also knows the other replicas are not coming back. That was the decision made when we stopped waiting for the quorum loss to resolve itself. The best course of action for the service is usually to freeze and wait for specific administrative intervention. So what does a typical implementation of the `OnDataLossAsync` method do?
  - First, log that `OnDataLossAsync` has been triggered, and fire off any necessary administrative alerts.
  - Usually at this point, to pause and wait for further decisions and manual actions to be taken. This is because even if backups are available they may need to be prepared. For example, if two different services coordinate information, those backups may need to be modified in order to ensure that once the restore happens that the information those two services care about is consistent.
  - Often there is also some other telemetry or exhaust from the service. This metadata may be contained in other services or in logs. This information can be used needed to determine if there were any calls received and processed at the primary that were not present in the backup or replicated to this particular replica. These may need to be replayed or added to the backup before restoration is feasible.
  - Comparisons of the remaining replica's state to that contained in any backups that are available. If using

the Service Fabric reliable collections then there are tools and processes available for doing so, described in [this article](#). The goal is to see if the state within the replica is sufficient, or also what the backup may be missing.

- Once the comparison is done, and if necessary the restore completed, the service code should return true if any state changes were made. If the replica determined that it was the best available copy of the state and made no changes, then return false. True indicates that any *other* remaining replicas may now be inconsistent with this one. They will be dropped and rebuilt from this replica. False indicates that no state changes were made, so the other replicas can keep what they have.

It is critically important that service authors practice potential dataloss and failure scenarios before services are ever deployed in production. To protect against the possibility of dataloss, it is important to periodically [back up the state](#) of any of your stateful services to a geo-redundant store. You must also ensure that you have the ability to restore it. Since backups of many different services are taken at different times, you need to ensure that after a restore your services have a consistent view of each other. For example, consider a situation where one service generates a number and stores it, then sends it to another service that also stores it. After a restore, you might discover that the second service has the number but the first does not, because its backup didn't include that operation.

If you find out that the remaining replicas are insufficient to continue from in a dataloss scenario, and you can't reconstruct service state from telemetry or exhaust the frequency of your backups determines your best possible recovery point objective (RPO). Service Fabric provides many tools for testing various failure scenarios, including permanent quorum and dataloss requiring restoration from a backup. These scenarios are included as a part of Service Fabric's testability tools, managed by the Fault Analysis Service. More info on those tools and patterns is available [here](#).

#### NOTE

System services can also suffer quorum loss, with the impact being specific to the service in question. For instance, quorum loss in the naming service impacts name resolution, whereas quorum loss in the failover manager service blocks new service creation and failovers. While the Service Fabric system services follow the same pattern as your services for state management, it is not recommended that you should attempt to move them out of Quorum Loss and into potential dataloss. The recommendation is instead to [seek support](#) to determine a solution that is targeted to your specific situation. Usually it is preferable to simply wait until the down replicas return.

## Availability of the Service Fabric cluster

Generally speaking, the Service Fabric cluster itself is a highly distributed environment with no single points of failure. A failure of any one node will not cause availability or reliability issues for the cluster, primarily because the Service Fabric system services follow the same guidelines provided earlier: they always run with three or more replicas by default, and those system services that are stateless run on all nodes. The underlying Service Fabric networking and failure detection layers are fully distributed. Most system services can be rebuilt from metadata in the cluster, or know how to resynchronize their state from other places. The availability of the cluster can become compromised if system services get into quorum loss situations like those described above. In these cases you may not be able to perform certain operations on the cluster like starting an upgrade or deploying new services, but the cluster itself is still up. Services on already running will remain running in these conditions unless they require writes to the system services to continue functioning. For example, if the Failover Manager is in quorum loss all services will continue to run, but any services that fail will not be able to automatically restart, since this requires the involvement of the Failover Manager.

### Failures of a datacenter or Azure region

In rare cases, a physical data center can become temporarily unavailable due to loss of power or network connectivity. In these cases, your Service Fabric clusters and services in that datacenter or Azure region will be unavailable. However, *your data is preserved*. For clusters running in Azure, you can view updates on outages on

the [Azure status page](#). In the highly unlikely event that a physical data center is partially or fully destroyed, any Service Fabric clusters hosted there or the services inside them could be lost. This includes any state not backed up outside of that datacenter or region.

There's two different strategies for surviving the permanent or sustained failure of a single datacenter or region.

1. Run separate Service Fabric clusters in multiple such regions, and utilize some mechanism for failover and fail-back between these environments. This sort of multi-cluster active-active or active-passive model requires additional management and operations code. This also requires coordination of backups from the services in one datacenter or region so that they are available in other datacenters or regions when one fails.
2. Run a single Service Fabric cluster that spans multiple datacenters or regions. The minimum supported configuration for this is three datacenters or regions. The recommended number of regions or datacenters is five. This requires a more complex cluster topology. However, the benefit of this model is that failure of one datacenter or region is converted from a disaster into a normal failure. These failures can be handled by the mechanisms that work for clusters within a single region. Fault domains, upgrade domains, and Service Fabric's placement rules ensure workloads are distributed so that they tolerate normal failures. For more information on policies that can help operate services in this type of cluster, read up on [placement policies](#)

### **Random failures leading to cluster failures**

Service Fabric has the concept of Seed Nodes. These are nodes that maintain the availability of the underlying cluster. These nodes help to ensure the cluster remains up by establishing leases with other nodes and serving as tiebreakers during certain kinds of network failures. If random failures remove a majority of the seed nodes in the cluster and they are not brought back, the cluster automatically shuts down. In Azure, Seed Nodes are automatically managed: they are distributed over the available fault and upgrade domains, and if a single seed node is removed from the cluster another one will be created in its place.

In both standalone Service Fabric clusters and Azure, the "Primary Node Type" is the one that runs the seeds. When defining a primary node type, Service Fabric will automatically take advantage of the number of nodes provided by creating up to 9 seed nodes and 9 replicas of each of the system services. If a set of random failures takes out a majority of those system service replicas simultaneously, the system services will enter quorum loss, as we described above. If a majority of the seed nodes are lost, the cluster will shut down soon after.

## **Next steps**

- Learn how to simulate various failures using the [testability framework](#)
- Read other disaster-recovery and high-availability resources. Microsoft has published a large amount of guidance on these topics. While some of these documents refer to specific techniques for use in other products, they contain many general best practices you can apply in the Service Fabric context as well:
  - [Availability checklist](#)
  - [Performing a disaster recovery drill](#)
  - [Disaster recovery and high availability for Azure applications](#)
- Learn about [Service Fabric support options](#)

# Describing a service fabric cluster

10/26/2017 • 23 min to read • [Edit Online](#)

The Service Fabric Cluster Resource Manager provides several mechanisms for describing a cluster. During runtime, the Cluster Resource Manager uses this information to ensure high availability of the services running in the cluster. While enforcing these important rules, it also attempts to optimize the resource consumption within the cluster.

## Key concepts

The Cluster Resource Manager supports several features that describe a cluster:

- Fault Domains
- Upgrade Domains
- Node Properties
- Node Capacities

## Fault domains

A Fault Domain is any area of coordinated failure. A single machine is a Fault Domain (since it can fail on its own for various reasons, from power supply failures to drive failures to bad NIC firmware). Machines connected to the same Ethernet switch are in the same Fault Domain, as are machines sharing a single source of power or in a single location. Since it's natural for hardware faults to overlap, Fault Domains are inherently hierachal and are represented as URLs in Service Fabric.

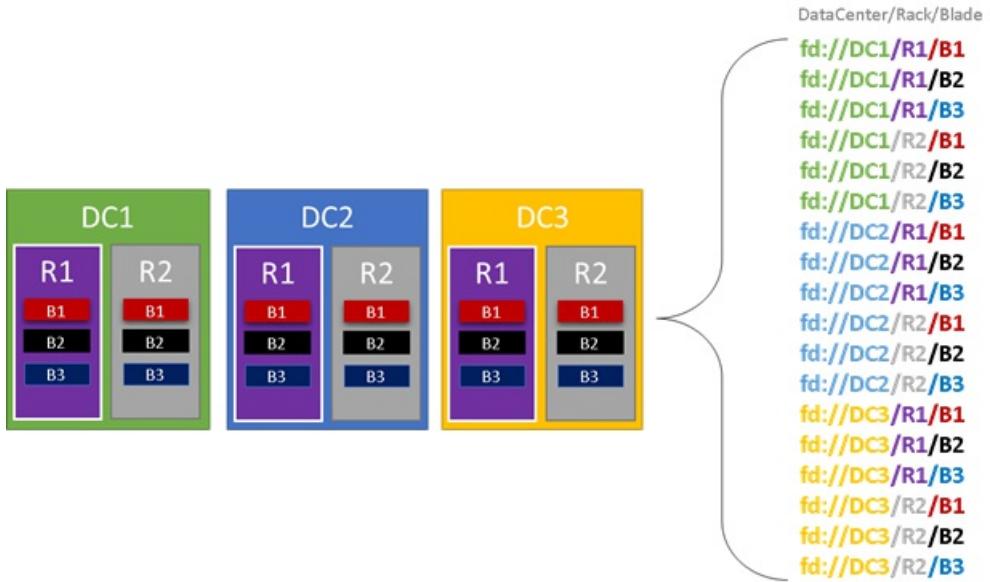
It is important that Fault Domains are set up correctly since Service Fabric uses this information to safely place services. Service Fabric doesn't want to place services such that the loss of a Fault Domain (caused by the failure of some component) causes a service to go down. In the Azure environment Service Fabric uses the Fault Domain information provided by the environment to correctly configure the nodes in the cluster on your behalf. For Service Fabric Standalone, Fault Domains are defined at the time that the cluster is set up

### WARNING

It is important that the Fault Domain information provided to Service Fabric is accurate. For example, let's say that your Service Fabric cluster's nodes are running inside 10 virtual machines, running on five physical hosts. In this case, even though there are 10 virtual machines, there are only 5 different (top level) fault domains. Sharing the same physical host causes VMs to share the same root fault domain, since the VMs experience coordinated failure if their physical host fails.

Service Fabric expects the Fault Domain of a node not to change. Other mechanisms of ensuring high availability of the VMs such as [HA-VMs](#) may cause conflicts with Service Fabric, as they use transparent migration of VMs from one host to another. These mechanisms do not reconfigure or notify the running code inside the VM. As such, they are **not supported** as environments for running Service Fabric clusters. Service Fabric should be the only high-availability technology employed. Mechanisms like live VM migration, SANs, or others are not necessary. If used in conjunction with Service Fabric, these mechanisms *reduce* application availability and reliability because they introduce additional complexity, add centralized sources of failure, and utilize reliability and availability strategies that conflict with those in Service Fabric.

In the graphic below we color all the entities that contribute to Fault Domains and list all the different Fault Domains that result. In this example, we have datacenters ("DC"), racks ("R"), and blades ("B"). Conceivably, if each blade holds more than one virtual machine, there could be another layer in the Fault Domain hierarchy.

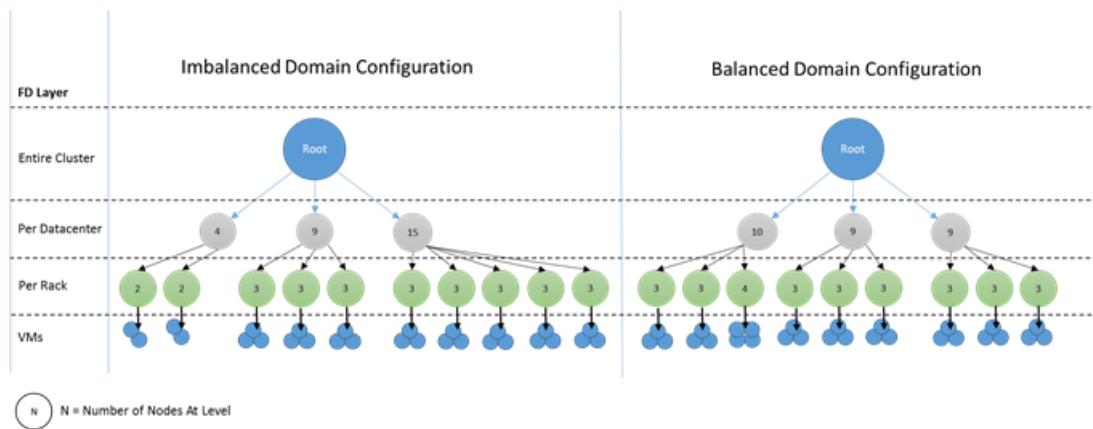


During runtime, the Service Fabric Cluster Resource Manager considers the Fault Domains in the cluster and plans layouts. The stateful replicas or stateless instances for a given service are distributed so they are in separate Fault Domains. Distributing the service across fault domains ensures the availability of the service is not compromised when a Fault Domain fails at any level of the hierarchy.

Service Fabric's Cluster Resource Manager doesn't care how many layers there are in the Fault Domain hierarchy. However, it tries to ensure that the loss of any one portion of the hierarchy doesn't impact services running in it.

It is best if there are the same number of nodes at each level of depth in the Fault Domain hierarchy. If the "tree" of Fault Domains is unbalanced in your cluster, it makes it harder for the Cluster Resource Manager to figure out the best allocation of services. Imbalanced Fault Domains layouts mean that the loss of some domains impact the availability of services more than other domains. As a result, the Cluster Resource Manager is torn between two goals: It wants to use the machines in that "heavy" domain by placing services on them, and it wants to place services in other domains so that the loss of a domain doesn't cause problems.

What do imbalanced domains look like? In the diagram below, we show two different cluster layouts. In the first example, the nodes are distributed evenly across the Fault Domains. In the second example, one Fault Domain has many more nodes than the other Fault Domains.



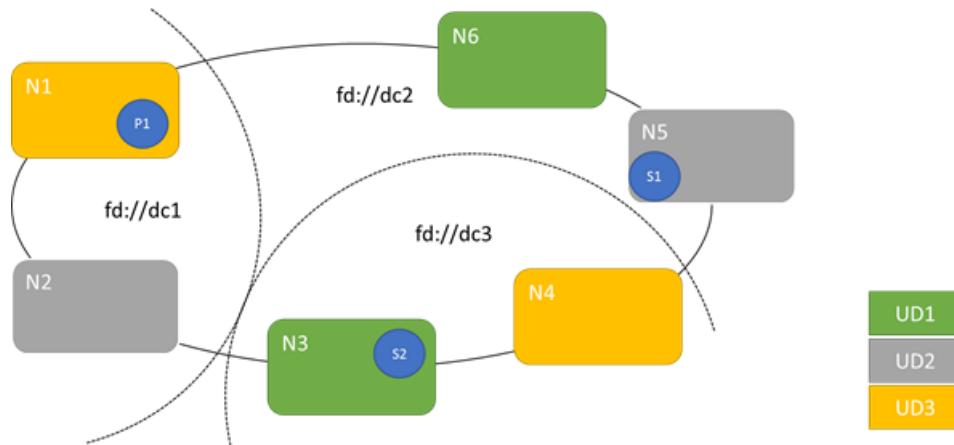
In Azure, the choice of which Fault Domain contains a node is managed for you. However, depending on the number of nodes that you provision you can still end up with Fault Domains with more nodes in them than others. For example, say you have five Fault Domains in the cluster but provision seven nodes for a given NodeType. In this case, the first two Fault Domains end up with more nodes. If you continue to deploy more NodeTypes with only a couple instances, the problem gets worse. For this reason it's recommended that the number of nodes in each node type is a multiple of the number of Fault Domains.

## Upgrade domains

Upgrade Domains are another feature that helps the Service Fabric Cluster Resource Manager understand the layout of the cluster. Upgrade Domains define sets of nodes that are upgraded at the same time. Upgrade Domains help the Cluster Resource Manager understand and orchestrate management operations like upgrades.

Upgrade Domains are a lot like Fault Domains, but with a couple key differences. First, areas of coordinated hardware failures define Fault Domains. Upgrade Domains, on the other hand, are defined by policy. You get to decide how many you want, rather than it being dictated by the environment. You could have as many Upgrade Domains as you do nodes. Another difference between Fault Domains and Upgrade Domains is that Upgrade Domains are not hierarchical. Instead, they are more like a simple tag.

The following diagram shows three Upgrade Domains striped across three Fault Domains. It also shows one possible placement for three different replicas of a stateful service, where each ends up in different Fault and Upgrade Domains. This placement allows the loss of a Fault Domain while in the middle of a service upgrade and still have one copy of the code and data.



There are pros and cons to having large numbers of Upgrade Domains. More Upgrade Domains means each step of the upgrade is more granular and therefore affects a smaller number of nodes or services. As a result, fewer services have to move at a time, introducing less churn into the system. This tends to improve reliability, since less of the service is impacted by any issue introduced during the upgrade. More Upgrade Domains also means that you need less available buffer on other nodes to handle the impact of the upgrade. For example, if you have five Upgrade Domains, the nodes in each are handling roughly 20% of your traffic. If you need to take down that Upgrade Domain for an upgrade, that load usually needs to go somewhere. Since you have four remaining Upgrade Domains, each must have room for about 5% of the total traffic. More Upgrade Domains means you need less buffer on the nodes in the cluster. For example, consider if you had 10 Upgrade Domains instead. In that case, each UD would only be handling about 10% of the total traffic. When an upgrade steps through the cluster, each domain would only need to have room for about 1.1% of the total traffic. More Upgrade Domains generally allow you to run your nodes at higher utilization, since you need less reserved capacity. The same is true for Fault Domains.

The downside of having many Upgrade Domains is that upgrades tend to take longer. Service Fabric waits a short period of time after an Upgrade Domain is completed and performs checks before starting to upgrade the next one. These delays enable detecting issues introduced by the upgrade before the upgrade proceeds. The tradeoff is acceptable because it prevents bad changes from affecting too much of the service at a time.

Too few Upgrade Domains has many negative side effects – while each individual Upgrade Domain is down and being upgraded a large portion of your overall capacity is unavailable. For example, if you only have three Upgrade Domains you are taking down about 1/3 of your overall service or cluster capacity at a time. Having so much of your service down at once isn't desirable since you have to have enough capacity in the rest of your cluster to handle the workload. Maintaining that buffer means that during normal operation those nodes are less loaded than they would be otherwise. This increases the cost of running your service.

There's no real limit to the total number of fault or Upgrade Domains in an environment, or constraints on how they overlap. That said, there are several common patterns:

- Fault Domains and Upgrade Domains mapped 1:1
- One Upgrade Domain per Node (physical or virtual OS instance)
- A “striped” or “matrix” model where the Fault Domains and Upgrade Domains form a matrix with machines usually running down the diagonals

UD Per Node		FD1	FD2	FD3
UD1	Node1			
UD2		Node2		
UD3			Node3	
UD4	Node4			
UD5		Node5		
UD6			Node6	
UD7	Node7			
UD8		Node8		
UD9			Node9	

FD/UD Matrix (Sparse/Diagonal)		FD1	FD2	FD3	FD4	FD5
UD1	Node1					
UD2		Node2				
UD3			Node3			
UD4				Node4		
UD5					Node5	

1:1 FD & UD

FD1/UD1	FD2/UD2	FD3/UD3
Node1	Node2	Node3
Node4	Node5	Node6
Node7	Node8	Node9

FD/UD Matrix

	FD1	FD2	FD3
UD1	Node1	Node2	Node3
UD2	Node4	Node5	Node6
UD3	Node7	Node8	Node9

There's no best answer which layout to choose, each has some pros and cons. For example, the 1FD:1UD model is simple to set up. The 1 Upgrade Domain per Node model is most like what people are used to. During upgrades each node is updated independently. This is similar to how small sets of machines were upgraded manually in the past.

The most common model is the FD/UD matrix, where the FDs and UD form a table and nodes are placed starting along the diagonal. This is the model used by default in Service Fabric clusters in Azure. For clusters with many nodes everything ends up looking like the dense matrix pattern above.

## Fault and Upgrade Domain constraints and resulting behavior

The Cluster Resource Manager treats the desire to keep a service balanced across fault and Upgrade Domains as a constraint. You can find out more about constraints in [this article](#). The Fault and Upgrade Domain constraints state: "For a given service partition there should never be a difference *greater than one* in the number of service objects (stateless service instances or stateful service replicas) between two domains." This prevents certain moves or arrangements that violate this constraint.

Let's look at one example. Let's say that we have a cluster with six nodes, configured with five Fault Domains and five Upgrade Domains.

	FD0	FD1	FD2	FD3	FD4
UD0	N1				
UD1	N6	N2			
UD2			N3		
UD3				N4	
UD4					N5

Now let's say that we create a service with a TargetReplicaSetSize (or, for a stateless service an InstanceCount) of five. The replicas land on N1-N5. In fact, N6 is never used no matter how many services like this you create. But why? Let's look at the difference between the current layout and what would happen if N6 is chosen.

Here's the layout we got and the total number of replicas per Fault and Upgrade Domain:

	<b>FD0</b>	<b>FD1</b>	<b>FD2</b>	<b>FD3</b>	<b>FD4</b>	<b>UDTOTAL</b>
<b>UD0</b>	R1					1
<b>UD1</b>		R2				1
<b>UD2</b>			R3			1
<b>UD3</b>				R4		1
<b>UD4</b>					R5	1
<b>FTotal</b>	1	1	1	1	1	-

This layout is balanced in terms of nodes per Fault Domain and Upgrade Domain. It is also balanced in terms of the number of replicas per Fault and Upgrade Domain. Each domain has the same number of nodes and the same number of replicas.

Now, let's look at what would happen if we'd used N6 instead of N2. How would the replicas be distributed then?

	<b>FD0</b>	<b>FD1</b>	<b>FD2</b>	<b>FD3</b>	<b>FD4</b>	<b>UDTOTAL</b>
<b>UD0</b>	R1					1
<b>UD1</b>	R5					1
<b>UD2</b>			R2			1
<b>UD3</b>				R3		1
<b>UD4</b>					R4	1
<b>FTotal</b>	2	0	1	1	1	-

This layout violates our definition for the Fault Domain constraint. FD0 has two replicas, while FD1 has zero, making the difference between FD0 and FD1 a total of two. The Cluster Resource Manager does not allow this arrangement. Similarly if we picked N2 and N6 (instead of N1 and N2) we'd get:

	<b>FD0</b>	<b>FD1</b>	<b>FD2</b>	<b>FD3</b>	<b>FD4</b>	<b>UDTOTAL</b>
<b>UD0</b>						0
<b>UD1</b>	R5	R1				2
<b>UD2</b>			R2			1
<b>UD3</b>				R3		1

	<b>FD0</b>	<b>FD1</b>	<b>FD2</b>	<b>FD3</b>	<b>FD4</b>	<b>UDTOTAL</b>
<b>UD4</b>					R4	1
<b>FDTOTAL</b>	1	1	1	1	1	-

This layout is balanced in terms of Fault Domains. However, now it's violating the Upgrade Domain constraint. This is because UD0 has zero replicas while UD1 has two. Therefore, this layout is also invalid and won't be picked by the Cluster Resource Manager.

## Configuring fault and Upgrade Domains

Defining Fault Domains and Upgrade Domains is done automatically in Azure hosted Service Fabric deployments. Service Fabric picks up and uses the environment information from Azure.

If you're creating your own cluster (or want to run a particular topology in development), you can provide the Fault Domain and Upgrade Domain information yourself. In this example, we define a nine node local development cluster that spans three "datacenters" (each with three racks). This cluster also has three Upgrade Domains striped across those three datacenters. An example of the configuration is below:

ClusterManifest.xml

```

<Infrastructure>
    <!-- IsScaleMin indicates that this cluster runs on one-box /one single server -->
    <WindowsServer IsScaleMin="true">
        <NodeList>
            <Node NodeName="Node01" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType01"
FaultDomain="fd:/DC01/Rack01" UpgradeDomain="UpgradeDomain1" IsSeedNode="true" />
            <Node NodeName="Node02" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType02"
FaultDomain="fd:/DC01/Rack02" UpgradeDomain="UpgradeDomain2" IsSeedNode="true" />
            <Node NodeName="Node03" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType03"
FaultDomain="fd:/DC01/Rack03" UpgradeDomain="UpgradeDomain3" IsSeedNode="true" />
            <Node NodeName="Node04" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType04"
FaultDomain="fd:/DC02/Rack01" UpgradeDomain="UpgradeDomain1" IsSeedNode="true" />
            <Node NodeName="Node05" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType05"
FaultDomain="fd:/DC02/Rack02" UpgradeDomain="UpgradeDomain2" IsSeedNode="true" />
            <Node NodeName="Node06" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType06"
FaultDomain="fd:/DC02/Rack03" UpgradeDomain="UpgradeDomain3" IsSeedNode="true" />
            <Node NodeName="Node07" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType07"
FaultDomain="fd:/DC03/Rack01" UpgradeDomain="UpgradeDomain1" IsSeedNode="true" />
            <Node NodeName="Node08" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType08"
FaultDomain="fd:/DC03/Rack02" UpgradeDomain="UpgradeDomain2" IsSeedNode="true" />
            <Node NodeName="Node09" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType09"
FaultDomain="fd:/DC03/Rack03" UpgradeDomain="UpgradeDomain3" IsSeedNode="true" />
        </NodeList>
    </WindowsServer>
</Infrastructure>
```

via ClusterConfig.json for Standalone deployments

```
"nodes": [
  {
    "nodeName": "vm1",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc1/r0",
    "upgradeDomain": "UD1"
  },
  {
    "nodeName": "vm2",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc1/r0",
    "upgradeDomain": "UD2"
  },
  {
    "nodeName": "vm3",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc1/r0",
    "upgradeDomain": "UD3"
  },
  {
    "nodeName": "vm4",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc2/r0",
    "upgradeDomain": "UD1"
  },
  {
    "nodeName": "vm5",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc2/r0",
    "upgradeDomain": "UD2"
  },
  {
    "nodeName": "vm6",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc2/r0",
    "upgradeDomain": "UD3"
  },
  {
    "nodeName": "vm7",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc3/r0",
    "upgradeDomain": "UD1"
  },
  {
    "nodeName": "vm8",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc3/r0",
    "upgradeDomain": "UD2"
  },
  {
    "nodeName": "vm9",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc3/r0",
    "upgradeDomain": "UD3"
  }
],
```

## NOTE

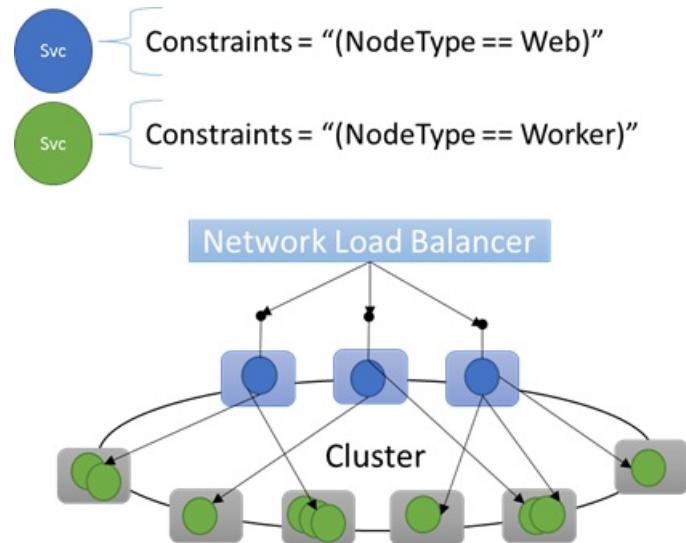
When defining clusters via Azure Resource Manager, Fault Domains and Upgrade Domains are assigned by Azure. Therefore, the definition of your Node Types and Virtual Machine Scale Sets in your Azure Resource Manager template does not include Fault Domain or Upgrade Domain information.

## Node properties and placement constraints

Sometimes (in fact, most of the time) you're going to want to ensure that certain workloads run only on certain types of nodes in the cluster. For example, some workload may require GPUs or SSDs while others may not. A great example of targeting hardware to particular workloads is almost every n-tier architecture out there. Certain machines serve as the front end or API serving side of the application and are exposed to the clients or the internet. Different machines, often with different hardware resources, handle the work of the compute or storage layers. These are usually *not* directly exposed to clients or the internet. Service Fabric expects that there are cases where particular workloads need to run on particular hardware configurations. For example:

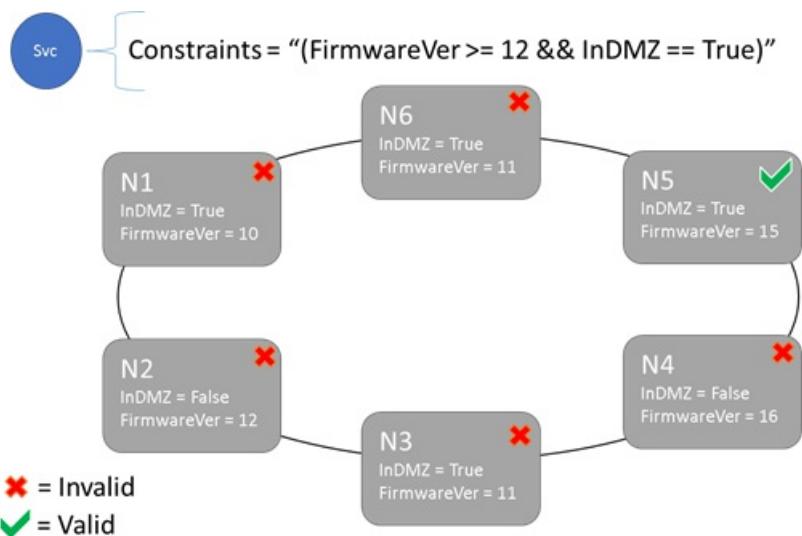
- an existing n-tier application has been "lifted and shifted" into a Service Fabric environment
- a workload wants to run on specific hardware for performance, scale, or security isolation reasons
- A workload should be isolated from other workloads for policy or resource consumption reasons

To support these sorts of configurations, Service Fabric has a first class notion of tags that can be applied to nodes. These tags are called **node properties**. **Placement constraints** are the statements attached to individual services that select for one or more node properties. Placement constraints define where services should run. The set of constraints is extensible - any key/value pair can work.



### Built in node properties

Service Fabric defines some default node properties that can be used automatically without the user having to define them. The default properties defined at each node are the **NodeType** and the **NodeName**. So for example you could write a placement constraint as `"(NodeType == NodeType03)"`. Generally we have found NodeType to be one of the most commonly used properties. It is useful since it corresponds 1:1 with a type of a machine. Each type of machine corresponds to a type of workload in a traditional n-tier application.



## Placement Constraint and Node Property Syntax

The value specified in the node property can be a string, bool, or signed long. The statement at the service is called a placement *constraint* since it constrains where the service can run in the cluster. The constraint can be any Boolean statement that operates on the different node properties in the cluster. The valid selectors in these boolean statements are:

- 1) conditional checks for creating particular statements

STATEMENT	SYNTAX
"equal to"	"=="
"not equal to"	"!="
"greater than"	
"greater than or equal to"	>=
"less than"	<
"less than or equal to"	<=

- 2) boolean statements for grouping and logical operations

STATEMENT	SYNTAX
"and"	"&&"
"or"	"  "
"not"	"!"
"group as single statement"	"()"

Here are some examples of basic constraint statements.

- "Value >= 5"

- "NodeColor != green"
- "((OneProperty < 100) || ((AnotherProperty == false) && (OneProperty >= 100)))"

Only nodes where the overall placement constraint statement evaluates to "True" can have the service placed on it. Nodes that do not have a property defined do not match any placement constraint containing that property.

Let's say that the following node properties were defined for a given node type:

ClusterManifest.xml

```
<NodeType Name="NodeType01">
  <PlacementProperties>
    <Property Name="HasSSD" Value="true"/>
    <Property Name="NodeColor" Value="green"/>
    <Property Name="SomeProperty" Value="5"/>
  </PlacementProperties>
</NodeType>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters.

#### **NOTE**

In your Azure Resource Manager template the node type is usually parameterized. It would look like "[parameters('vmNodeType1Name')]" rather than "NodeType01".

```
"nodeTypes": [
  {
    "name": "NodeType01",
    "placementProperties": {
      "HasSSD": "true",
      "NodeColor": "green",
      "SomeProperty": "5"
    }
  }
],
```

You can create service placement *constraints* for a service like as follows:

C#

```
FabricClient fabricClient = new FabricClient();
StatefulServiceDescription serviceDescription = new StatefulServiceDescription();
serviceDescription.PlacementConstraints = "(HasSSD == true && SomeProperty >= 4)";
// add other required servicedescription fields
//...
await fabricClient.ServiceManager.CreateServiceAsync(serviceDescription);
```

Powershell:

```
New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName
$serviceType -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -
PlacementConstraint "HasSSD == true && SomeProperty >= 4"
```

If all nodes of NodeType01 are valid, you can also select that node type with the constraint "(NodeType == NodeType01)".

One of the cool things about a service's placement constraints is that they can be updated dynamically during

runtime. So if you need to, you can move a service around in the cluster, add and remove requirements, etc. Service Fabric takes care of ensuring that the service stays up and available even when these types of changes are made.

C#:

```
StatefulServiceUpdateDescription updateDescription = new StatefulServiceUpdateDescription();
updateDescription.PlacementConstraints = "NodeType == NodeType01";
await fabricClient.ServiceManager.UpdateServiceAsync(new Uri("fabric:/app/service"), updateDescription);
```

Powershell:

```
Update-ServiceFabricService -Stateful -ServiceName $serviceName -PlacementConstraints "NodeType == NodeType01"
```

Placement constraints are specified for every different named service instance. Updates always take the place of (overwrite) what was previously specified.

The cluster definition defines the properties on a node. Changing a node's properties requires a cluster configuration upgrade. Upgrading a node's properties requires each affected node to restart to report its new properties. These rolling upgrades are managed by Service Fabric.

## Describing and Managing Cluster Resources

One of the most important jobs of any orchestrator is to help manage resource consumption in the cluster. Managing cluster resources can mean a couple of different things. First, there's ensuring that machines are not overloaded. This means making sure that machines aren't running more services than they can handle. Second, there's balancing and optimization which is critical to running services efficiently. Cost effective or performance sensitive service offerings can't allow some nodes to be hot while others are cold. Hot nodes lead to resource contention and poor performance, and cold nodes represent wasted resources and increased costs.

Service Fabric represents resources as [Metrics](#). Metrics are any logical or physical resource that you want to describe to Service Fabric. Examples of metrics are things like "WorkQueueDepth" or "MemoryInMb". For information about the physical resources that Service Fabric can govern on nodes, see [resource governance](#). For information on configuring custom metrics and their uses, see [this article](#)

Metrics are different from placements constraints and node properties. Node properties are static descriptors of the nodes themselves. Metrics describe resources that nodes have and that services consume when they are run on a node. A node property could be "HasSSD" and could be set to true or false. The amount of space available on that SSD and how much is consumed by services would be a metric like "DriveSpaceInMb".

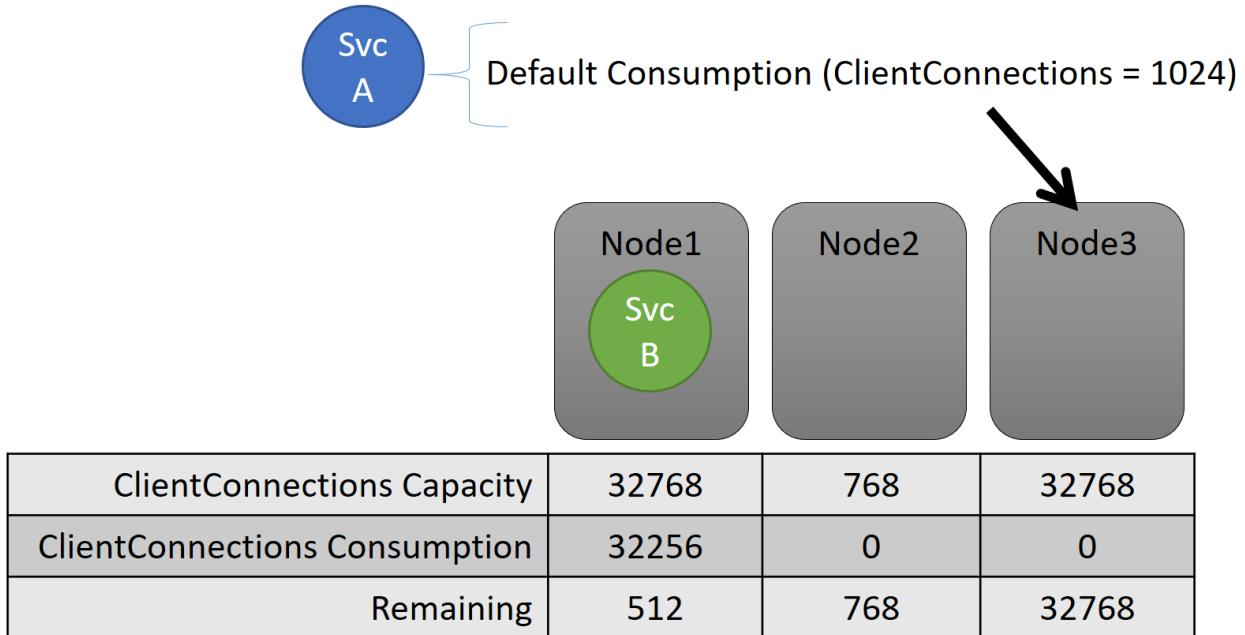
It is important to note that just like for placement constraints and node properties, the Service Fabric Cluster Resource Manager doesn't understand what the names of the metrics mean. Metric names are just strings. It is a good practice to declare units as a part of the metric names that you create when it could be ambiguous.

## Capacity

If you turned off all resource *balancing*, Service Fabric's Cluster Resource Manager would still ensure that no node ended up over its capacity. Managing capacity overruns is possible unless the cluster is too full or the workload is larger than any node. Capacity is another *constraint* that the Cluster Resource Manager uses to understand how much of a resource a node has. Remaining capacity is also tracked for the cluster as a whole. Both the capacity and the consumption at the service level are expressed in terms of metrics. So for example, the metric might be "ClientConnections" and a given Node may have a capacity for "ClientConnections" of 32768. Other nodes can have other limits. Some service running on that node can say it is currently consuming 32256 of the metric "ClientConnections".

During runtime, the Cluster Resource Manager tracks remaining capacity in the cluster and on nodes. In order to

track capacity the Cluster Resource Manager subtracts each service's usage from node's capacity where the service runs. With this information, the Service Fabric Cluster Resource Manager can figure out where to place or move replicas so that nodes don't go over capacity.



C#:

```
StatefulServiceDescription serviceDescription = new StatefulServiceDescription();
ServiceLoadMetricDescription metric = new ServiceLoadMetricDescription();
metric.Name = "ClientConnections";
metric.PrimaryDefaultLoad = 1024;
metric.SecondaryDefaultLoad = 0;
metric.Weight = ServiceLoadMetricWeight.High;
serviceDescription.Metrics.Add(metric);
await fabricClient.ServiceManager.CreateServiceAsync(serviceDescription);
```

Powershell:

```
New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName
$serviceTypeName -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -Metric
@("ClientConnections,High,1024,0)
```

You can see capacities defined in the cluster manifest:

ClusterManifest.xml

```
<NodeType Name="NodeType03">
<Capacities>
<Capacity Name="ClientConnections" Value="65536"/>
</Capacities>
</NodeType>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters.

```

"nodeTypes": [
    {
        "name": "NodeType03",
        "capacities": {
            "ClientConnections": "65536",
        }
    }
],

```

Commonly a service's load changes dynamically. Say that a replica's load of "ClientConnections" changed from 1024 to 2048, but the node it was running on then only had 512 capacity remaining for that metric. Now that replica or instance's placement is invalid, since there's not enough room on that node. The Cluster Resource Manager has to kick in and get the node back below capacity. It reduces load on the node that is over capacity by moving one or more of the replicas or instances from that node to other nodes. When moving replicas, the Cluster Resource Manager tries to minimize the cost of those movements. Movement cost is discussed in [this article](#) and more about the Cluster Resource Manager's rebalancing strategies and rules is described [here](#).

## Cluster capacity

So how does the Service Fabric Cluster Resource Manager keep the overall cluster from being too full? Well, with dynamic load there's not a lot it can do. Services can have their load spike independently of actions taken by the Cluster Resource Manager. As a result, your cluster with plenty of headroom today may be underpowered when you become famous tomorrow. That said, there are some controls that are baked in to prevent problems. The first thing we can do is prevent the creation of new workloads that would cause the cluster to become full.

Say that you create a stateless service and it has some load associated with it. Let's say that the service cares about the "DiskSpaceInMb" metric. Let's also say that it is going to consume five units of "DiskSpaceInMb" for every instance of the service. You want to create three instances of the service. Great! So that means that we need 15 units of "DiskSpaceInMb" to be present in the cluster in order for us to even be able to create these service instances. The Cluster Resource Manager continually calculates the capacity and consumption of each metric so it can determine the remaining capacity in the cluster. If there isn't enough space, the Cluster Resource Manager rejects the create service call.

Since the requirement is only that there be 15 units available, this space could be allocated many different ways. For example, there could be one remaining unit of capacity on 15 different nodes, or three remaining units of capacity on five different nodes. If the Cluster Resource Manager can rearrange things so there's five units available on three nodes, it places the service. Rearranging the cluster is usually possible unless the cluster is almost full or the existing services can't be consolidated for some reason.

## Buffered Capacity

Buffered capacity is another feature of the Cluster Resource Manager. It allows reservation of some portion of the overall node capacity. This capacity buffer is only used to place services during upgrades and node failures. Buffered Capacity is specified globally per metric for all nodes. The value you pick for the reserved capacity is a function of the number of Fault and Upgrade Domains you have in the cluster. More Fault and Upgrade Domains means that you can pick a lower number for your buffered capacity. If you have more domains, you can expect smaller amounts of your cluster to be unavailable during upgrades and failures. Specifying Buffered Capacity only makes sense if you have also specified the node capacity for a metric.

Here's an example of how to specify buffered capacity:

ClusterManifest.xml

```

<Section Name="NodeBufferPercentage">
    <Parameter Name="SomeMetric" Value="0.15" />
    <Parameter Name="SomeOtherMetric" Value="0.20" />
</Section>

```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters:

```

"fabricSettings": [
{
    "name": "NodeBufferPercentage",
    "parameters": [
        {
            "name": "SomeMetric",
            "value": "0.15"
        },
        {
            "name": "SomeOtherMetric",
            "value": "0.20"
        }
    ]
}
]

```

The creation of new services fails when the cluster is out of buffered capacity for a metric. Preventing the creation of new services to preserve the buffer ensures that upgrades and failures don't cause nodes to go over capacity. Buffered capacity is optional but is recommended in any cluster that defines a capacity for a metric.

The Cluster Resource Manager exposes this load information. For each metric, this information includes:

- the buffered capacity settings
- the total capacity
- the current consumption
- whether each metric is considered balanced or not
- statistics about the standard deviation
- the nodes which have the most and least load

Below we see an example of that output:

```
PS C:\Users\user> Get-ServiceFabricClusterLoadInformation
LastBalancingStartTimeUtc : 9/1/2016 12:54:59 AM
LastBalancingEndTimeUtc   : 9/1/2016 12:54:59 AM
LoadMetricInformation     :
    LoadMetricName      : Metric1
    IsBalancedBefore    : False
    IsBalancedAfter     : False
    DeviationBefore     : 0.192450089729875
    DeviationAfter      : 0.192450089729875
    BalancingThreshold  : 1
    Action              : NoActionNeeded
    ActivityThreshold   : 0
    ClusterCapacity     : 189
    ClusterLoad         : 45
    ClusterRemainingCapacity : 144
    NodeBufferPercentage : 10
    ClusterBufferedCapacity : 170
    ClusterRemainingBufferedCapacity : 125
    ClusterCapacityViolation : False
    MinNodeLoadValue    : 0
    MinNodeLoadNodeId   : 3ea71e8e01f4b0999b121abcbf27d74d
    MaxNodeLoadValue    : 15
    MaxNodeLoadNodeId   : 2cc648b6770be1bc9824fa995d5b68b1
```

## Next steps

- For information on the architecture and information flow within the Cluster Resource Manager, check out [this article](#)
- Defining Defragmentation Metrics is one way to consolidate load on nodes instead of spreading it out. To learn how to configure defragmentation, refer to [this article](#)
- Start from the beginning and [get an Introduction to the Service Fabric Cluster Resource Manager](#)
- To find out about how the Cluster Resource Manager manages and balances load in the cluster, check out the article on [balancing load](#)

# Service Fabric cluster security scenarios

9/5/2017 • 6 min to read • [Edit Online](#)

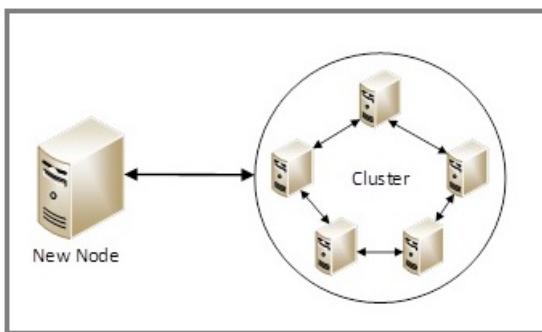
An Azure Service Fabric cluster is a resource that you own. You must secure your clusters to help prevent unauthorized users from connecting to them. A secure cluster is especially important when you are running production workloads on the cluster. Although it's possible to create an unsecured cluster, if the cluster exposes management endpoints to the public internet, anonymous users can connect to it.

This article is an overview of security scenarios for Azure clusters and standalone clusters, and the various technologies you can use to implement them:

- Node-to-node security
- Client-to-node security
- Role-Based Access Control (RBAC)

## Node-to-node security

Node-to-node security helps secure communication between the VMs or computers in a cluster. This security scenario ensures that only computers that are authorized to join the cluster can participate in hosting applications and services in the cluster.



Clusters running on Azure and standalone clusters running on Windows both can use either [certificate security](#) or [Windows security](#) for Windows Server computers.

### Node-to-node certificate security

Service Fabric uses X.509 server certificates that you specify as part of the node-type configuration when you create a cluster. At the end of this article, you can see a brief overview of what these certificates are and how you can acquire or create them.

Set up certificate security when you create the cluster, either in the Azure portal, by using an Azure Resource Manager template, or by using a standalone JSON template. You can set a primary certificate and an optional secondary certificate, which is used for certificate rollovers. The primary and secondary certificates you set should be different from the admin client and read-only client certificates that you set for [client-to-node security](#).

To learn how to set up certificate security in a cluster for Azure, see [Set up a cluster by using an Azure Resource Manager template](#).

To learn how to set up certificate security in a cluster for a standalone Windows Server cluster, see [Secure a standalone cluster on Windows by using X.509 certificates](#).

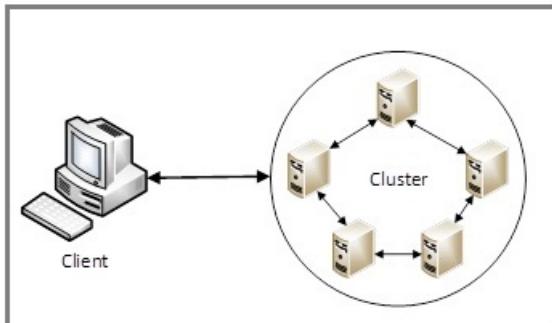
### Node-to-node Windows security

To learn how to set up Windows security for a standalone Windows Server cluster, see [Secure a standalone](#)

cluster on Windows by using Windows security.

## Client-to-node security

Client-to-node security authenticates clients and helps secure communication between a client and individual nodes in the cluster. This type of security helps ensure that only authorized users can access the cluster and the applications that are deployed on the cluster. Clients are uniquely identified through either their Windows security credentials or their certificate security credentials.



Clusters running on Azure and standalone clusters running on Windows both can use either [certificate security](#) or [Windows security](#).

### Client-to-node certificate security

Set up client-to-node certificate security when you create the cluster, either in the Azure portal, by using a Resource Manager template, or by using a standalone JSON template. To create the certificate, specify an admin client certificate or a user client certificate. As a best practice, the admin client and user client certificates you specify should be different from the primary and secondary certificates you specify for [node-to-node security](#). By default, the cluster certificates for node-to-node security are added to the allowed client admin certificates list.

Clients that connect to the cluster by using the admin certificate have full access to management capabilities. Clients that connect to the cluster by using the read-only user client certificate have only read access to management capabilities. These certificates are used for the RBAC that we described later in this article.

To learn how to set up certificate security in a cluster for Azure, see [Set up a cluster by using an Azure Resource Manager template](#).

To learn how to set up certificate security in a cluster for a standalone Windows Server cluster, see [Secure a standalone cluster on Windows by using X.509 certificates](#).

### Client-to-node Azure Active Directory security on Azure

For clusters running on Azure, you also can secure access to management endpoints by using Azure Active Directory (Azure AD). To learn how to create the required Azure AD artifacts, how to populate them when you create the cluster, and how to connect to the clusters afterward, see [Set up a cluster by using an Azure Resource Manager template](#).

## Security recommendations

For Azure clusters, for node-to-node security, we recommend that you use Azure AD security to authenticate clients and certificates.

For standalone Windows Server clusters, if you have Windows Server 2012 R2 and Windows Active Directory, we recommend that you use Windows security with group Managed Service Accounts. Otherwise, use Windows security with Windows accounts.

## Role-Based Access Control (RBAC)

You can use access control to limit access to certain cluster operations for different groups of users. This helps make the cluster more secure. Two access control types are supported for clients that connect to a cluster: Administrator role and User role.

Users who are assigned the Administrator role have full access to management capabilities, including read and write capabilities. Users who are assigned the User role, by default, have only read access to management capabilities (for example, query capabilities). They also can resolve applications and services.

Set the Administrator and User client roles when you create the cluster. Assign roles by providing separate identities (for example, by using certificates or Azure AD) for each role type. For more information about default access control settings and how to change default settings, see [Role-Based Access Control for Service Fabric clients](#).

## X.509 certificates and Service Fabric

X.509 digital certificates commonly are used to authenticate clients and servers. They also are used to encrypt and digitally sign messages. For more information about X.509 digital certificates, see [Working with certificates](#).

Some important things to consider:

- To create certificates for clusters that are running production workloads, use a correctly configured Windows Server certificate service, or one from an approved [certificate authority \(CA\)](#).
- Never use any temporary or test certificates that you create by using tools like MakeCert.exe in a production environment.
- You can use a self-signed certificate, but only in a test cluster. Do not use a self-signed certificate in production.

### Server X.509 certificates

Server certificates have the primary task of authenticating a server (node) to clients, or authenticating a server (node) to a server (node). When a client or node authenticates a node, one of the initial checks is the value of the common name in the **Subject** field. Either this common name or one of the certificates' Subject Alternative Names (SANs) must be present in the list of allowed common names.

To learn how to generate certificates that have SANs, see [How to add a Subject Alternative Name to a secure LDAP certificate](#).

The **Subject** field can have multiple values. Each value is prefixed with an initialization to indicate the value type. Usually, the initialization is **CN** (for *common name*); for example, **CN = www.contoso.com**. The **Subject** field can be blank. If the optional **Subject Alternative Name** field is populated, it must have both the common name of the certificate and one entry per SAN. These are entered as **DNS Name** values.

The value of the **Intended Purposes** field of the certificate should include an appropriate value, such as **Server Authentication** or **Client Authentication**.

### Client X.509 certificates

Client certificates typically are not issued by a third-party CA. Instead, the Personal store of the current user location typically contains client certificates placed there by a root authority, with an **Intended Purposes** value of **Client Authentication**. The client can use this certificate when mutual authentication is required.

#### NOTE

All management operations on a Service Fabric cluster require server certificates. Client certificates cannot be used for management.

## Next steps

- Create a cluster in Azure by using a Resource Manager template
- Create a cluster by using the Azure portal

# Differences between Service Fabric on Linux and Windows

9/25/2017 • 1 min to read • [Edit Online](#)

There are some features that are supported on Windows, but not yet on Linux. Eventually, the feature sets will be at parity and with each release this feature gap will shrink. The following differences exist between the latest available releases (that is, between version 6.0 on Windows and version 6.0 on Linux):

- All programming models are in preview (Java/C# Reliable Actors, Reliable Stateless Services and Reliable Stateful Services)
- Envoy (ReverseProxy) is in preview on Linux
- Standalone installer for Linux is not yet available on Linux
- Console redirection (not supported in Linux or Windows production clusters)
- The Fault Analysis Service (FAS) on Linux
- DNS service for Service Fabric services (DNS service is supported for containers on Linux)
- CLI command equivalents of certain Powershell commands (list below, most of which apply only to standalone clusters)

Development tooling is also different between Windows and Linux. Visual Studio, Powershell, VSTS, and ETW are used on Windows while Yeoman, Eclipse, Jenkins, and LTTng are used on Linux.

## Powershell cmdlets that do not work against a Linux Service Fabric cluster

- Invoke-ServiceFabricChaosTestScenario
- Invoke-ServiceFabricFailoverTestScenario
- Invoke-ServiceFabricPartitionDataLoss
- Invoke-ServiceFabricPartitionQuorumLoss
- Restart-ServiceFabricPartition
- Start-ServiceFabricNode
- Stop-ServiceFabricNode
- Get-ServiceFabricImageStoreContent
- Get-ServiceFabricChaosReport
- Get-ServiceFabricNodeTransitionProgress
- Get-ServiceFabricPartitionDataLossProgress
- Get-ServiceFabricPartitionQuorumLossProgress
- Get-ServiceFabricPartitionRestartProgress
- Get-ServiceFabricTestCommandStatusList
- Remove-ServiceFabricTestState
- Start-ServiceFabricChaos
- Start-ServiceFabricNodeTransition
- Start-ServiceFabricPartitionDataLoss
- Start-ServiceFabricPartitionQuorumLoss
- Start-ServiceFabricPartitionRestart
- Stop-ServiceFabricChaos

- Stop-ServiceFabricTestCommand
- Get-ServiceFabricNodeConfiguration
- Get-ServiceFabricClusterConfiguration
- Get-ServiceFabricClusterConfigurationUpgradeStatus
- Get-ServiceFabricPackageDebugParameters
- New-ServiceFabricPackageDebugParameter
- New-ServiceFabricPackageSharingPolicy
- Add-ServiceFabricNode
- Copy-ServiceFabricClusterPackage
- Get-ServiceFabricRuntimeSupportedVersion
- Get-ServiceFabricRuntimeUpgradeVersion
- New-ServiceFabricCluster
- New-ServiceFabricNodeConfiguration
- Remove-ServiceFabricCluster
- Remove-ServiceFabricClusterPackage
- Remove-ServiceFabricNodeConfiguration
- Test-ServiceFabricClusterManifest
- Test-ServiceFabricConfiguration
- Update-ServiceFabricNodeConfiguration
- Approve-ServiceFabricRepairTask
- Complete-ServiceFabricRepairTask
- Get-ServiceFabricRepairTask
- Invoke-ServiceFabricDecryptText
- Invoke-ServiceFabricEncryptSecret
- Invoke-ServiceFabricEncryptText
- Invoke-ServiceFabricInfrastructureCommand
- Invoke-ServiceFabricInfrastructureQuery
- Remove-ServiceFabricRepairTask
- Start-ServiceFabricRepairTask
- Stop-ServiceFabricRepairTask
- Update-ServiceFabricRepairTaskHealthPolicy

## Next steps

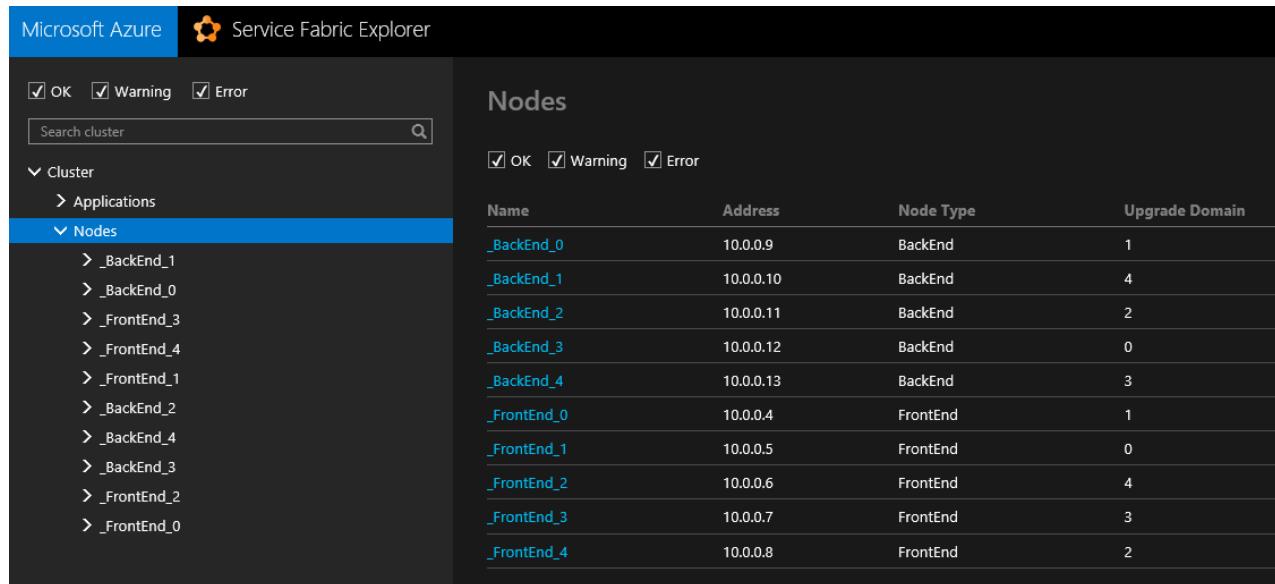
- [Prepare your development environment on Linux](#)
- [Prepare your development environment on OSX](#)
- [Create and deploy your first Service Fabric Java application on Linux using Yeoman](#)
- [Create and deploy your first Service Fabric Java application on Linux using Service Fabric Plugin for Eclipse](#)
- [Create your first CSharp application on Linux](#)
- [Use the Service Fabric CLI to manage your applications](#)

# Azure Service Fabric node types and virtual machine scale sets

9/14/2017 • 4 min to read • [Edit Online](#)

Virtual machine scale sets are an Azure compute resource. You can use scale sets to deploy and manage a collection of virtual machines as a set. Set up a separate scale set for each node type that you define in an Azure Service Fabric cluster. You can independently scale each node type up or down, have different sets of ports open, and use different capacity metrics.

The following figure shows a cluster that has two node types, named FrontEnd and BackEnd. Each node type has five nodes.



The screenshot shows the Microsoft Azure Service Fabric Explorer interface. On the left, there's a navigation pane with 'Microsoft Azure' at the top, followed by 'Service Fabric Explorer'. Below that are checkboxes for 'OK', 'Warning', and 'Error', and a search bar labeled 'Search cluster'. The main area is titled 'Nodes' and contains a table with four columns: 'Name', 'Address', 'Node Type', and 'Upgrade Domain'. The table lists ten nodes, grouped into two categories: BackEnd and FrontEnd. The BackEnd category has five nodes: '\_BackEnd\_0' (Address 10.0.0.9, Node Type BackEnd, Upgrade Domain 1), '\_BackEnd\_1' (Address 10.0.0.10, Node Type BackEnd, Upgrade Domain 4), '\_BackEnd\_2' (Address 10.0.0.11, Node Type BackEnd, Upgrade Domain 2), '\_BackEnd\_3' (Address 10.0.0.12, Node Type BackEnd, Upgrade Domain 0), and '\_BackEnd\_4' (Address 10.0.0.13, Node Type BackEnd, Upgrade Domain 3). The FrontEnd category has five nodes: '\_FrontEnd\_0' (Address 10.0.0.4, Node Type FrontEnd, Upgrade Domain 1), '\_FrontEnd\_1' (Address 10.0.0.5, Node Type FrontEnd, Upgrade Domain 0), '\_FrontEnd\_2' (Address 10.0.0.6, Node Type FrontEnd, Upgrade Domain 4), '\_FrontEnd\_3' (Address 10.0.0.7, Node Type FrontEnd, Upgrade Domain 3), and '\_FrontEnd\_4' (Address 10.0.0.8, Node Type FrontEnd, Upgrade Domain 2).

Name	Address	Node Type	Upgrade Domain
_BackEnd_0	10.0.0.9	BackEnd	1
_BackEnd_1	10.0.0.10	BackEnd	4
_BackEnd_2	10.0.0.11	BackEnd	2
_BackEnd_3	10.0.0.12	BackEnd	0
_BackEnd_4	10.0.0.13	BackEnd	3
_FrontEnd_0	10.0.0.4	FrontEnd	1
_FrontEnd_1	10.0.0.5	FrontEnd	0
_FrontEnd_2	10.0.0.6	FrontEnd	4
_FrontEnd_3	10.0.0.7	FrontEnd	3
_FrontEnd_4	10.0.0.8	FrontEnd	2

## Map virtual machine scale set instances to nodes

As shown in the preceding figure, scale set instances start at instance 0, and then increase by 1. The numbering is reflected in the node names. For example, node BackEnd\_0 is instance 0 of the BackEnd scale set. This particular scale set has five instances, named BackEnd\_0, BackEnd\_1, BackEnd\_2, BackEnd\_3, and BackEnd\_4.

When you scale up a scale set, a new instance is created. The new scale set instance name typically is the scale set name plus the next instance number. In our example, it is BackEnd\_5.

## Map scale set load balancers to node types and scale sets

If you deployed your cluster in the Azure portal or used the sample Azure Resource Manager template, all resources under a resource group are listed. You can see the load balancers for each scale set or node type. The load balancer name uses the following format: **LB-<node type name>**. An example is LB-sfcluster4doc-0, as shown in the following figure:

The screenshot shows the Azure Resource Group 'sfcluster4doc' interface. At the top, there are 'Settings', 'Add', and 'Delete' buttons. Below that is the 'Essentials' section with 'Subscription name: Build Demo', 'Last deployment: 2/11/2016 (Succeeded)', 'Subscription ID', and 'Location: West US'. A 'All settings →' button is also present. The main area is titled 'Summary' and contains a 'Resources' list. The list includes: BackEnd, FrontEnd, LB-sfcluster4doc-0 (highlighted with a red box), LB-sfcluster4doc-1 (highlighted with a red box), LBIP-sfcluster4doc-0, LBIP-sfcluster4doc-1, VNet-sfcluster4doc, sfcluster4doc, and sfcluster4doc2678. There is also an 'Add a group +' button at the bottom.

## Remote connect to a virtual machine scale set instance or a cluster node

Set up a separate scale set for each node type that you defined in a cluster. You can independently scale the node types up or down. You also can use different VM SKUs. Unlike single-instance VMs, scale set instances don't have their own virtual IP addresses. This can be challenging when you are looking for an IP address and port that you can use to remotely connect to a specific instance.

To find an IP address and port that you can use to remotely connect to a specific instance, complete the following steps.

**Step 1:** Find the virtual IP address for the node type by getting the inbound NAT rules for Remote Desktop Protocol (RDP).

First, get the inbound NAT rules values that were defined as part of the resource definition for `Microsoft.Network/loadBalancers`.

In the Azure portal, on the load balancer page, select **Settings > Inbound NAT rules**. This gives you the IP address and port that you can use to remotely connect to the first scale set instance.

The screenshot shows the Azure portal interface for managing a Load Balancer. The left pane displays the 'Essentials' section for the load balancer 'LB-sfcluster4doc-0'. It includes details like IP address (104.42.106.156), protocol/port (TCP/19000, TCP/19080, 4 more), backend pool (LoadBalancerBEAddressPool), probe (AppPortProbe1), and NAT rules (5 inbound). A blue button labeled 'All settings →' is visible. The right pane is titled 'Settings' and lists various configuration options under sections like SUPPORT & TROUBLESHOOTING, GENERAL, and MONITORING. The 'Inbound NAT rules' option is highlighted with a red box.

In the following figure, the IP address and port are **104.42.106.156** and **3389**.

The screenshot shows the 'Inbound NAT rules' page for the load balancer. The table lists two rules:

NAME	DESTINATION	TARGET	SERVICE
LoadBalancerBEAdd...	104.42.106.156	Loading...	RDP (TCP/3389)
LoadBalancerBEAdd...	104.42.106.156	Loading...	Custom (TCP/3390)

**Step 2:** Find the port that you can use to remotely connect to the specific scale set instance or node.

Scale set instances map to nodes. Use the scale set information to determine the exact port to use.

Ports are allocated in an ascending order that matches the scale set instance. For the earlier example of the FrontEnd node type, the following table lists the ports for each of the five node instances. Apply the same mapping to your scale set instance.

VIRTUAL MACHINE SCALE SET INSTANCE	PORT
FrontEnd_0	3389
FrontEnd_1	3390
FrontEnd_2	3391
FrontEnd_3	3392

VIRTUAL MACHINE SCALE SET INSTANCE	PORT
FrontEnd_4	3393
FrontEnd_5	3394

**Step 3:** Remotely connect to the specific scale set instance.

The following figure demonstrates using Remote Desktop Connection to connect to the FrontEnd\_1 scale set instance:



## Change the RDP port range values

### Before cluster deployment

When you set up the cluster by using a Resource Manager template, specify the range in `inboundNatPools`.

Go to the resource definition for `Microsoft.Network/loadBalancers`. Locate the description for `inboundNatPools`. Replace the `frontendPortRangeStart` and `frontendPortRangeEnd` values.

```

        },
      ],
      "inboundNatPools": [
        {
          "name": "LoadBalancerBEAddressNatPool",
          "properties": {
            "backendPort": "3389",
            "frontendIPConfiguration": {
              "id": "[variables('lbIPConfig0')]"
            },
            "frontendPortRangeEnd": "4500",
            "frontendPortRangeStart": "3389",
            "protocol": "tcp"
          }
        }
      ]
    },
  ],
  "loadBalancingRules": [
    {
      "name": "LoadBalancerLBAddressRule"
    }
  ]
}

```

### After cluster deployment

Changing the RDP port range values after the cluster has been deployed is more complex. To ensure that you don't recycle the VMs, use Azure PowerShell to set new values.

#### NOTE

Ensure that you have Azure PowerShell 1.0 or a later version installed on your computer. If you don't have Azure PowerShell 1.0 or a later version, we recommend that you follow the steps described in [How to install and configure Azure PowerShell](#).

1. Sign in to your Azure account. If the following PowerShell command fails, verify that you installed PowerShell correctly.

```
Login-AzureRmAccount
```

2. To get details about your load balancer, and to see the values for the description for `inboundNatPools`, run the following code:

```
Get-AzureRmResource -ResourceGroupName <resource group name> -ResourceType Microsoft.Network/loadBalancers -ResourceName <load balancer name>
```

3. Set `frontendPortRangeEnd` and `frontendPortRangeStart` to the values that you want.

```
$PropertiesObject = @{
    #Property = value;
}
Set-AzureRmResource -PropertyObject $PropertiesObject -ResourceGroupName <resource group name> -ResourceType Microsoft.Network/loadBalancers -ResourceName <load balancer name> -ApiVersion <use the API version that is returned> -Force
```

## Change the RDP user name and password for nodes

To change the password for all nodes of a specific node type, complete the following steps. These changes will apply to all current and future nodes in the scale set.

1. Open PowerShell as an administrator.
2. To log in and select your subscription for the session, run the following commands. Change the `SUBSCRIPTIONID` parameter to your subscription ID.

```
Login-AzureRmAccount
Get-AzureRmSubscription -SubscriptionId 'SUBSCRIPTIONID' | Select-AzureRmSubscription
```

3. Run the following script. Use the relevant `NODETYPENAME`, `RESOURCEGROUP`, `USERNAME`, and `PASSWORD` values. The `USERNAME` and `PASSWORD` values are the new credentials that you use in future RDP sessions.

```
$nodeTypeNames = 'NODETYPENAME'
$resourceGroup = 'RESOURCEGROUP'
$publicConfig = @{'UserName' = 'USERNAME'}
$privateConfig = @{'Password' = 'PASSWORD'}
$extName = 'VMAccessAgent'
$publisher = 'Microsoft.Compute'
$node = Get-AzureRmVmss -ResourceGroupName $resourceGroup -VMScaleSetName $nodeTypeNames
$node = Add-AzureRmVmssExtension -VirtualMachineScaleSet $node -Name $extName -Publisher $publisher -Setting $publicConfig -ProtectedSetting $privateConfig -Type $extName -TypeHandlerVersion '2.0' -AutoUpgradeMinorVersion $true

Update-AzureRmVmss -ResourceGroupName $resourceGroup -Name $nodeTypeNames -VirtualMachineScaleSet $node
```

## Next steps

- See the [overview of the "Deploy anywhere" feature and a comparison with Azure-managed clusters](#).
- Learn about [cluster security](#).
- Learn about the [Service Fabric SDK and getting started](#).

# Service Fabric networking patterns

8/31/2017 • 11 min to read • [Edit Online](#)

You can integrate your Azure Service Fabric cluster with other Azure networking features. In this article, we show you how to create clusters that use the following features:

- [Existing virtual network or subnet](#)
- [Static public IP address](#)
- [Internal-only load balancer](#)
- [Internal and external load balancer](#)

Service Fabric runs in a standard virtual machine scale set. Any functionality that you can use in a virtual machine scale set, you can use with a Service Fabric cluster. The networking sections of the Azure Resource Manager templates for virtual machine scale sets and Service Fabric are identical. After you deploy to an existing virtual network, it's easy to incorporate other networking features, like Azure ExpressRoute, Azure VPN Gateway, a network security group, and virtual network peering.

Service Fabric is unique from other networking features in one aspect. The [Azure portal](#) internally uses the Service Fabric resource provider to call to a cluster to get information about nodes and applications. The Service Fabric resource provider requires publicly accessible inbound access to the HTTP gateway port (port 19080, by default) on the management endpoint. [Service Fabric Explorer](#) uses the management endpoint to manage your cluster. The Service Fabric resource provider also uses this port to query information about your cluster, to display in the Azure portal.

If port 19080 is not accessible from the Service Fabric resource provider, a message like *Nodes Not Found* appears in the portal, and your node and application list appears empty. If you want to see your cluster in the Azure portal, your load balancer must expose a public IP address, and your network security group must allow incoming port 19080 traffic. If your setup does not meet these requirements, the Azure portal does not display the status of your cluster.

## Templates

All Service Fabric templates are in [one download file](#). You should be able to deploy the templates as-is by using the following PowerShell commands. If you are deploying the existing Azure Virtual Network template or the static public IP template, first read the [Initial setup](#) section of this article.

## Initial setup

### Existing virtual network

In the following example, we start with an existing virtual network named ExistingRG-vnet, in the **ExistingRG** resource group. The subnet is named default. These default resources are created when you use the Azure portal to create a standard virtual machine (VM). You could create the virtual network and subnet without creating the VM, but the main goal of adding a cluster to an existing virtual network is to provide network connectivity to other VMs. Creating the VM gives a good example of how an existing virtual network typically is used. If your Service Fabric cluster uses only an internal load balancer, without a public IP address, you can use the VM and its public IP as a secure *jump box*.

### Static public IP address

A static public IP address generally is a dedicated resource that's managed separately from the VM or VMs it's assigned to. It's provisioned in a dedicated networking resource group (as opposed to in the Service Fabric cluster

resource group itself). Create a static public IP address named staticIP1 in the same ExistingRG resource group, either in the Azure portal or by using PowerShell:

```
PS C:\Users\user> New-AzureRmPublicIpAddress -Name staticIP1 -ResourceGroupName ExistingRG -Location westus -AllocationMethod Static -DomainNameLabel sfnetworking

Name          : staticIP1
ResourceGroupName : ExistingRG
Location       : westus
Id            : /subscriptions/1237f4d2-3dce-1236-ad95-123f764e7123/resourceGroups/ExistingRG/providers/Microsoft.Network/publicIPAddresses/staticIP1
Etag          : W/"fc8b0c77-1f84-455d-9930-0404ebba1b64"
ResourceGuid   : 77c26c06-c0ae-496c-9231-b1a114e08824
ProvisioningState : Succeeded
Tags          :
PublicIpAllocationMethod : Static
IpAddress       : 40.83.182.110
PublicIpAddressVersion : IPv4
IdleTimeoutInMinutes : 4
IpConfiguration  : null
DnsSettings    : {
    "DomainNameLabel": "sfnetworking",
    "Fqdn": "sfnetworking.westus.cloudapp.azure.com"
}
```

## Service Fabric template

In the examples in this article, we use the Service Fabric template.json. You can use the standard portal wizard to download the template from the portal before you create a cluster. You also can use one of the templates in the [template gallery](#), like the [five-node Service Fabric cluster](#).

## Existing virtual network or subnet

1. Change the subnet parameter to the name of the existing subnet, and then add two new parameters to reference the existing virtual network:

```
"subnet0Name": {
    "type": "string",
    "defaultValue": "default"
},
"existingVNetRGName": {
    "type": "string",
    "defaultValue": "ExistingRG"
},

"existingVNetName": {
    "type": "string",
    "defaultValue": "ExistingRG-vnet"
},
/*
"subnet0Name": {
    "type": "string",
    "defaultValue": "Subnet-0"
},
"subnet0Prefix": {
    "type": "string",
    "defaultValue": "10.0.0.0/24"
},*/
}
```

2. Change the `vnetID` variable to point to the existing virtual network:

```

/*old "vnetID": "
[resourceId('Microsoft.Network/virtualNetworks',parameters('virtualNetworkName'))]",*/
    "vnetID": "[concat('/subscriptions/', subscription().subscriptionId, '/resourceGroups/',
parameters('existingVNetRGName'), '/providers/Microsoft.Network/virtualNetworks/',
parameters('existingVNetName'))]",

```

3. Remove `Microsoft.Network/virtualNetworks` from your resources, so Azure does not create a new virtual network:

```

/*
"apiVersion": "[variables('vNetApiVersion')]",
"type": "Microsoft.Network/virtualNetworks",
"name": "[parameters('virtualNetworkName')]",
"location": "[parameters('computeLocation')]",
"properties": {
    "addressSpace": {
        "addressPrefixes": [
            "[parameters('addressPrefix')]"
        ]
    },
    "subnets": [
        {
            "name": "[parameters('subnet0Name')]",
            "properties": {
                "addressPrefix": "[parameters('subnet0Prefix')]"
            }
        }
    ]
},
"tags": {
    "resourceType": "Service Fabric",
    "clusterName": "[parameters('clusterName')]"
}
},*/

```

4. Comment out the virtual network from the `dependsOn` attribute of `Microsoft.Compute/virtualMachineScaleSets`, so you don't depend on creating a new virtual network:

```

"apiVersion": "[variables('vmssApiVersion')]",
"type": "Microsoft.Compute/virtualMachineScaleSets",
"name": "[parameters('vmNodeType0Name')]",
"location": "[parameters('computeLocation')]",
"dependsOn": [
    /*"[concat('Microsoft.Network/virtualNetworks/', parameters('virtualNetworkName'))]",*/
    /*
    "[Concat('Microsoft.Storage/storageAccounts/', variables('uniqueStringArray0')[0])]"*/
]

```

5. Deploy the template:

```

New-AzureRmResourceGroup -Name sfnetworkingexistingvnet -Location westus
New-AzureRmResourceGroupDeployment -Name deployment -ResourceGroupName sfnetworkingexistingvnet -
TemplateFile C:\SFSamples\Final\template\_existingvnet.json

```

After deployment, your virtual network should include the new scale set VMs. The virtual machine scale set node type should show the existing virtual network and subnet. You also can use Remote Desktop Protocol (RDP) to access the VM that was already in the virtual network, and to ping the new scale set VMs:

```
C:>\Users\users>ping 10.0.0.5 -n 1  
C:>\Users\users>ping N0de1000000 -n 1
```

For another example, see [one that is not specific to Service Fabric](#).

## Static public IP address

1. Add parameters for the name of the existing static IP resource group, name, and fully qualified domain name (FQDN):

```
"existingStaticIPResourceGroup": {  
    "type": "string"  
},  
"existingStaticIPName": {  
    "type": "string"  
},  
"existingStaticIPDnsFQDN": {  
    "type": "string"  
}
```

2. Remove the `dnsName` parameter. (The static IP address already has one.)

```
/*  
"dnsName": {  
    "type": "string"  
},  
*/
```

3. Add a variable to reference the existing static IP address:

```
"existingStaticIP": "[concat('/subscriptions/', subscription().subscriptionId, '/resourceGroups/',  
parameters('existingStaticIPResourceGroup'), '/providers/Microsoft.Network/publicIPAddresses/',  
parameters('existingStaticIPName'))]",
```

4. Remove `Microsoft.Network/publicIPAddresses` from your resources, so Azure does not create a new IP address:

```
/*  
{  
    "apiVersion": "[variables('publicIPApiVersion')]",  
    "type": "Microsoft.Network/publicIPAddresses",  
    "name": "[concat(parameters('lbIPName'),)-', '0')]",  
    "location": "[parameters('computeLocation')]",  
    "properties": {  
        "dnsSettings": {  
            "domainNameLabel": "[parameters('dnsName')]"  
        },  
        "publicIPAllocationMethod": "Dynamic"  
    },  
    "tags": {  
        "resourceType": "Service Fabric",  
        "clusterName": "[parameters('clusterName')]"  
    }  
}, */
```

5. Comment out the IP address from the `dependsOn` attribute of `Microsoft.Network/loadBalancers`, so you don't depend on creating a new IP address:

```

"apiVersion": "[variables('lbIPApiVersion')]",
"type": "Microsoft.Network/loadBalancers",
"name": "[concat('LB', '-', parameters('clusterName'), '-', parameters('vmNodeType0Name'))]",
"location": "[parameters('computeLocation')]",
/*
"dependsOn": [
    "[concat('Microsoft.Network/publicIPAddresses/', concat(parameters('lbIPName'), '-', '0'))]"
], */
"properties": {

```

6. In the `Microsoft.Network/loadBalancers` resource, change the `publicIPAddress` element of `frontendIPConfigurations` to reference the existing static IP address instead of a newly created one:

```

"frontendIPConfigurations": [
{
    "name": "LoadBalancerIPConfig",
    "properties": {
        "publicIPAddress": {
            /*"id": "*/[resourceId('Microsoft.Network/publicIPAddresses', concat(parameters('lbIPName'), '-', '0'))]/*"
            "id": "[variables('existingStaticIP')]"
        }
    }
},
],

```

7. In the `Microsoft.ServiceFabric/clusters` resource, change `managementEndpoint` to the DNS FQDN of the static IP address. If you are using a secure cluster, make sure you change `http://` to `https://`. (Note that this step applies only to Service Fabric clusters. If you are using a virtual machine scale set, skip this step.)

```

"fabricSettings": [],
/*"managementEndpoint": "[concat('http://',reference(concat(parameters('lbIPName'), '-',
', '0')).dnsSettings.fqdn,':',parameters('nt0fabricHttpGatewayPort'))]",*/
"managementEndpoint": "[concat('http://',parameters('existingStaticIPDnsFQDN'),':',parameters('nt0fabricHttpGatewayPort'))]",
```

8. Deploy the template:

```

New-AzureRmResourceGroup -Name sfnetworkingstaticip -Location westus

$staticip = Get-AzureRmPublicIpAddress -Name staticIP1 -ResourceGroupName ExistingRG

$staticip

New-AzureRmResourceGroupDeployment -Name deployment -ResourceGroupName sfnetworkingstaticip -
TemplateFile C:\SFSamples\Final\template\_staticip.json -existingStaticIPResourceGroup
$staticip.ResourceGroupName -existingStaticIPName $staticip.Name -existingStaticIPDnsFQDN
$staticip.DnsSettings.Fqdn

```

After deployment, you can see that your load balancer is bound to the public static IP address from the other resource group. The Service Fabric client connection endpoint and [Service Fabric Explorer](#) endpoint point to the DNS FQDN of the static IP address.

## Internal-only load balancer

This scenario replaces the external load balancer in the default Service Fabric template with an internal-only load balancer. For implications for the Azure portal and for the Service Fabric resource provider, see the preceding

section.

1. Remove the `dnsName` parameter. (It's not needed.)

```
/*
"dnsName": {
    "type": "string"
},
*/
```

2. Optionally, if you use a static allocation method, you can add a static IP address parameter. If you use a dynamic allocation method, you do not need to do this step.

```
"internalLBAddress": {
    "type": "string",
    "defaultValue": "10.0.0.250"
}
```

3. Remove `Microsoft.Network/publicIPAddresses` from your resources, so Azure does not create a new IP address:

```
/*
{
    "apiVersion": "[variables('publicIPApiVersion')]",
    "type": "Microsoft.Network/publicIPAddresses",
    "name": "[concat(parameters('lbIPName'),)-', '0')]",
    "location": "[parameters('computeLocation')]",
    "properties": {
        "dnsSettings": {
            "domainNameLabel": "[parameters('dnsName')]"
        },
        "publicIPAllocationMethod": "Dynamic"
    },
    "tags": {
        "resourceType": "Service Fabric",
        "clusterName": "[parameters('clusterName')]"
    }
}, */
```

4. Remove the IP address `dependsOn` attribute of `Microsoft.Network/loadBalancers`, so you don't depend on creating a new IP address. Add the virtual network `dependsOn` attribute because the load balancer now depends on the subnet from the virtual network:

```
"apiVersion": "[variables('lbApiVersion')]",
"type": "Microsoft.Network/loadBalancers",
"name": "[concat('LB', '-', parameters('clusterName'), '-', parameters('vmNodeType0Name'))]",
"location": "[parameters('computeLocation')]",
"dependsOn": [
    /*"[concat('Microsoft.Network/publicIPAddresses/',concat(parameters('lbIPName'),'-',
', '0'))]"*/
    "[concat('Microsoft.Network/virtualNetworks/',parameters('virtualNetworkName'))]"
],
```

5. Change the load balancer's `frontendIPConfigurations` setting from using a `publicIPAddress`, to using a subnet and `privateIPAddress`. `privateIPAddress` uses a predefined static internal IP address. To use a dynamic IP address, remove the `privateIPAddress` element, and then change `privateIPAllocationMethod` to **Dynamic**.

```

"frontendIPConfigurations": [
    {
        "name": "LoadBalancerIPConfig",
        "properties": {
            /*
            "publicIPAddress": {
                "id": "
[resourceId('Microsoft.Network/publicIPAddresses',concat(parameters('lbIPName'),'-','0'))]"
            } */
            "subnet" :{
                "id": "[variables('subnet0Ref')]"
            },
            "privateIPAddress": "[parameters('internalLBAddress')]",
            "privateIPAllocationMethod": "Static"
        }
    }
],

```

- In the `Microsoft.ServiceFabric/clusters` resource, change `managementEndpoint` to point to the internal load balancer address. If you use a secure cluster, make sure you change `http://` to `https://`. (Note that this step applies only to Service Fabric clusters. If you are using a virtual machine scale set, skip this step.)

```

"fabricSettings": [],
/*"managementEndpoint": "[concat('http://',reference(concat(parameters('lbIPName'),'-
','0')).dnsSettings.fqdn,':',parameters('nt0fabricHttpGatewayPort'))]",*/
"managementEndpoint": "
[concat('http://',reference(variables('lbID0')).frontEndIPConfigurations[0].properties.privateIPAddress,
':',parameters('nt0fabricHttpGatewayPort'))]",

```

- Deploy the template:

```

New-AzureRmResourceGroup -Name sfnetworkinginternallb -Location westus

New-AzureRmResourceGroupDeployment -Name deployment -ResourceGroupName sfnetworkinginternallb - 
TemplateFile C:\SFSamples\Final\template\_internalonlyLB.json

```

After deployment, your load balancer uses the private static 10.0.0.250 IP address. If you have another machine in that same virtual network, you can go to the internal [Service Fabric Explorer](#) endpoint. Note that it connects to one of the nodes behind the load balancer.

## Internal and external load balancer

In this scenario, you start with the existing single-node type external load balancer, and add an internal load balancer for the same node type. A back-end port attached to a back-end address pool can be assigned only to a single load balancer. Choose which load balancer will have your application ports, and which load balancer will have your management endpoints (ports 19000 and 19080). If you put the management endpoints on the internal load balancer, keep in mind the Service Fabric resource provider restrictions discussed earlier in the article. In the example we use, the management endpoints remain on the external load balancer. You also add a port 80 application port, and place it on the internal load balancer.

In a two-node-type cluster, one node type is on the external load balancer. The other node type is on the internal load balancer. To use a two-node-type cluster, in the portal-created two-node-type template (which comes with two load balancers), switch the second load balancer to an internal load balancer. For more information, see the [Internal-only load balancer](#) section.

- Add the static internal load balancer IP address parameter. (For notes related to using a dynamic IP address, see earlier sections of this article.)

```

    "internalLBAddress": {
        "type": "string",
        "defaultValue": "10.0.0.250"
    }
}

```

2. Add an application port 80 parameter.

3. To add internal versions of the existing networking variables, copy and paste them, and add "-Int" to the name:

```

/* Add internal load balancer networking variables */
    "lbID0-Int": "[resourceId('Microsoft.Network/loadBalancers', concat('LB','-', parameters('clusterName'), '-', parameters('vmNodeType0Name'), '-Internal'))]",
    "lbIPConfig0-Int": "[concat(variables('lbID0-Int'), '/frontendIPConfigurations/LoadBalancerIPConfig')]",
    "lbPoolID0-Int": "[concat(variables('lbID0-Int'), '/backendAddressPools/LoadBalancerBEAddressPool')]",
    "lbProbeID0-Int": "[concat(variables('lbID0-Int'), '/probes/FabricGatewayProbe')]",
    "lbHttpProbeID0-Int": "[concat(variables('lbID0-Int'), '/probes/FabricHttpGatewayProbe')]",
    "lbNatPoolID0-Int": "[concat(variables('lbID0-Int'), '/inboundNatPools/LoadBalancerBEAddressNatPool')]",
    /* Internal load balancer networking variables end */
}

```

4. If you start with the portal-generated template that uses application port 80, the default portal template adds AppPort1 (port 80) on the external load balancer. In this case, remove AppPort1 from the external load balancer `loadBalancingRules` and probes, so you can add it to the internal load balancer:

```

"loadBalancingRules": [
{
    "name": "LBHttpRule",
    "properties": {
        "backendAddressPool": {
            "id": "[variables('lbPoolID0')]"
        },
        "backendPort": "[parameters('nt0fabricHttpGatewayPort')]",
        "enableFloatingIP": "false",
        "frontendIPConfiguration": {
            "id": "[variables('lbIPConfig0')]"
        },
        "frontendPort": "[parameters('nt0fabricHttpGatewayPort')]",
        "idleTimeoutInMinutes": "5",
        "probe": {
            "id": "[variables('lbHttpProbeID0')]"
        },
        "protocol": "tcp"
    }
} /* Remove AppPort1 from the external load balancer.
{
    "name": "AppPortLBRule1",
    "properties": {
        "backendAddressPool": {
            "id": "[variables('lbPoolID0')]"
        },
        "backendPort": "[parameters('loadBalancedAppPort1')]",
        "enableFloatingIP": "false",
        "frontendIPConfiguration": {
            "id": "[variables('lbIPConfig0')]"
        },
        "frontendPort": "[parameters('loadBalancedAppPort1')]",
        "idleTimeoutInMinutes": "5",
        "probe": {
            "id": "[concat(variables('lbID0'), '/probes/AppPortProbe1')]"
        },
        "protocol": "tcp"
    }
}

```

```

        "protocol": "tcp"
    }
} */

],
"probes": [
{
    "name": "FabricGatewayProbe",
    "properties": {
        "intervalInSeconds": 5,
        "numberOfProbes": 2,
        "port": "[parameters('nt0fabricTcpGatewayPort')]",
        "protocol": "tcp"
    }
},
{
    "name": "FabricHttpGatewayProbe",
    "properties": {
        "intervalInSeconds": 5,
        "numberOfProbes": 2,
        "port": "[parameters('nt0fabricHttpGatewayPort')]",
        "protocol": "tcp"
    }
}
] /* Remove AppPort1 from the external load balancer.
{
    "name": "AppPortProbe1",
    "properties": {
        "intervalInSeconds": 5,
        "numberOfProbes": 2,
        "port": "[parameters('loadBalancedAppPort1')]",
        "protocol": "tcp"
    }
}
} */

],
"inboundNatPools": [

```

5. Add a second `Microsoft.Network/loadBalancers` resource. It looks similar to the internal load balancer created in the [Internal-only load balancer](#) section, but it uses the "-Int" load balancer variables, and implements only the application port 80. This also removes `inboundNatPools`, to keep RDP endpoints on the public load balancer. If you want RDP on the internal load balancer, move `inboundNatPools` from the external load balancer to this internal load balancer:

```

/* Add a second load balancer, configured with a static privateIPAddress and the "-Int" load
balancer variables. */
{
    "apiVersion": "[variables('lbApiVersion')]",
    "type": "Microsoft.Network/loadBalancers",
    /* Add "-Internal" to the name. */
    "name": "[concat('LB','-', parameters('clusterName'),'-',parameters('vmNodeType0Name'), '-
Internal')]",
    "location": "[parameters('computeLocation')]",
    "dependsOn": [
        /* Remove public IP dependsOn, add vnet dependsOn
        "[concat('Microsoft.Network/publicIPAddresses/',concat(parameters('lbIPName'), '-
','0'))]"
        */
        "[concat('Microsoft.Network/virtualNetworks/',parameters('virtualNetworkName'))]"
    ],
    "properties": {
        "frontendIPConfigurations": [
            {
                "name": "LoadBalancerIPConfig",
                "properties": {
                    /* Switch from Public to Private IP address
                    */

```

```

        "publicIPAddress": {
            "id": "
        }
        /*
        "subnet" :{
            "id": "[variables('subnet0Ref')]"
        },
        "privateIPAddress": "[parameters('internalLBAddress')]",
        "privateIPAllocationMethod": "Static"
    }
}
],
"backendAddressPools": [
{
    "name": "LoadBalancerBEAddressPool",
    "properties": {}
},
],
"loadBalancingRules": [
/* Add the AppPort rule. Be sure to reference the "-Int" versions of
backendAddressPool, frontendIPConfiguration, and the probe variables. */
{
    "name": "AppPortLBRule1",
    "properties": {
        "backendAddressPool": {
            "id": "[variables('lbPoolID0-Int')]"
        },
        "backendPort": "[parameters('loadBalancedAppPort1')]",
        "enableFloatingIP": "false",
        "frontendIPConfiguration": {
            "id": "[variables('lbIPConfig0-Int')]"
        },
        "frontendPort": "[parameters('loadBalancedAppPort1')]",
        "idleTimeoutInMinutes": "5",
        "probe": {
            "id": "[concat(variables('lbID0-Int'), '/probes/AppPortProbe1')]"
        },
        "protocol": "tcp"
    }
},
],
"probes": [
/* Add the probe for the app port. */
{
    "name": "AppPortProbe1",
    "properties": {
        "intervalInSeconds": 5,
        "numberOfProbes": 2,
        "port": "[parameters('loadBalancedAppPort1')]",
        "protocol": "tcp"
    }
},
],
"inboundNatPools": [
]
},
"tags": {
    "resourceType": "Service Fabric",
    "clusterName": "[parameters('clusterName')]"
}
},
},

```

6. In `networkProfile` for the `Microsoft.Compute/virtualMachineScaleSets` resource, add the internal back-end address pool:

```
"loadBalancerBackendAddressPools": [
    {
        "id": "[variables('lbPoolID0')]"
    },
    {
        /* Add internal BE pool */
        "id": "[variables('lbPoolID0-Int')]"
    }
],
]
```

## 7. Deploy the template:

```
New-AzureRmResourceGroup -Name sfnetworkinginternalexternallb -Location westus

New-AzureRmResourceGroupDeployment -Name deployment -ResourceGroupName sfnetworkinginternalexternallb -TemplateFile C:\SFSamples\Final\template\_internalexternalLB.json
```

After deployment, you can see two load balancers in the resource group. If you browse the load balancers, you can see the public IP address and management endpoints (ports 19000 and 19080) assigned to the public IP address. You also can see the static internal IP address and application endpoint (port 80) assigned to the internal load balancer. Both load balancers use the same virtual machine scale set back-end pool.

## Next steps

[Create a cluster](#)

# Introducing the Service Fabric cluster resource manager

8/18/2017 • 6 min to read • [Edit Online](#)

Traditionally managing IT systems or online services meant dedicating specific physical or virtual machines to those specific services or systems. Services were architected as tiers. There would be a "web" tier and a "data" or "storage" tier. Applications would have a messaging tier where requests flowed in and out, as well as a set of machines dedicated to caching. Each tier or type of workload had specific machines dedicated to it: the database got a couple machines dedicated to it, the web servers a few. If a particular type of workload caused the machines it was on to run too hot, then you added more machines with that same configuration to that tier. However, not all workloads could be scaled out so easily - particularly with the data tier you would typically replace machines with larger machines. Easy. If a machine failed, that part of the overall application ran at lower capacity until the machine could be restored. Still fairly easy (if not necessarily fun).

Now however the world of service and software architecture has changed. It's more common that applications have adopted a scale-out design. Building applications with containers or microservices (or both) is common. Now, while you may still have only a few machines, they're not running just a single instance of a workload. They may even be running multiple different workloads at the same time. You now have dozens of different types of services (none consuming a full machine's worth of resources), perhaps hundreds of different instances of those services. Each named instance has one or more instances or replicas for High Availability (HA). Depending on the sizes of those workloads, and how busy they are, you may find yourself with hundreds or thousands of machines.

Suddenly managing your environment is not so simple as managing a few machines dedicated to single types of workloads. Your servers are virtual and no longer have names (you have switched mindsets from [pets to cattle](#) after all). Configuration is less about the machines and more about the services themselves. Hardware that is dedicated to a single instance of a workload is largely a thing of the past. Services themselves have become small distributed systems that span multiple smaller pieces of commodity hardware.

Because your app is no longer a series of monoliths spread across several tiers, you now have many more combinations to deal with. Who decides what types of workloads can run on which hardware, or how many? Which workloads work well on the same hardware, and which conflict? When a machine goes down how do you know what was running there on that machine? Who is in charge of making sure that workload starts running again? Do you wait for the (virtual?) machine to come back or do your workloads automatically fail over to other machines and keep running? Is human intervention required? What about upgrades in this environment?

As developers and operators dealing in this environment, we're going to want help managing this complexity. A hiring binge and trying to hide the complexity with people is probably not the right answer, so what do we do?

## Introducing orchestrators

An "Orchestrator" is the general term for a piece of software that helps administrators manage these types of environments. Orchestrators are the components that take in requests like "I would like five copies of this service running in my environment." They try to make the environment match the desired state, no matter what happens.

Orchestrators (not humans) are what take action when a machine fails or a workload terminates for some unexpected reason. Most orchestrators do more than just deal with failure. Other features they have are managing new deployments, handling upgrades, and dealing with resource consumption and governance. All orchestrators are fundamentally about maintaining some desired state of configuration in the environment. You want to be able to tell an orchestrator what you want and have it do the heavy lifting. Aurora on top of Mesos, Docker Datacenter/Docker Swarm, Kubernetes, and Service Fabric are all examples of orchestrators. These orchestrators are

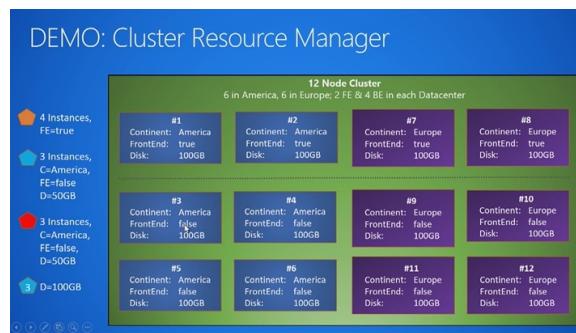
being actively developed to meet the needs of real workloads in production environments.

## Orchestration as a service

The Cluster Resource Manager is the system component that handles orchestration in Service Fabric. The Cluster Resource Manager's job is broken down into three parts:

1. Enforcing Rules
2. Optimizing Your Environment
3. Helping with Other Processes

To see how the Cluster Resource Manager works, watch the following Microsoft Virtual Academy video:



### What it isn't

In traditional N tier applications, there's always a [Load Balancer](#). Usually this was a Network Load Balancer (NLB) or an Application Load Balancer (ALB) depending on where it sat in the networking stack. Some load balancers are Hardware-based like F5's BigIP offering, others are software-based such as Microsoft's NLB. In other environments, you might see something like HAProxy, nginx, Istio, or Envoy in this role. In these architectures, the job of load balancing is to ensure stateless workloads receive (roughly) the same amount of work. Strategies for balancing load varied. Some balancers would send each different call to a different server. Others provided session pinning/stickiness. More advanced balancers use actual load estimation or reporting to route a call based on its expected cost and current machine load.

Network balancers or message routers tried to ensure that the web/worker tier remained roughly balanced. Strategies for balancing the data tier were different and depended on the data storage mechanism. Balancing the data tier relied on data sharding, caching, managed views, stored procedures, and other store-specific mechanisms.

While some of these strategies are interesting, the Service Fabric Cluster Resource Manager is not anything like a network load balancer or a cache. A Network Load Balancer balances frontends by spreading traffic across frontends. The Service Fabric Cluster Resource Manager has a different strategy. Fundamentally, Service Fabric moves *services* to where they make the most sense, expecting traffic or load to follow. For example, it might move services to nodes that are currently cold because the services that are there are not doing much work. The nodes may be cold since the services that were present were deleted or moved elsewhere. As another example, the Cluster Resource Manager could also move a service away from a machine. Perhaps the machine is about to be upgraded, or is overloaded due to a spike in consumption by the services running on it. Alternatively, the service's resource requirements may have increased. As a result there aren't sufficient resources on this machine to continue running it.

Because the Cluster Resource Manager is responsible for moving services around, it contains a different feature set compared to what you would find in a network load balancer. This is because network load balancers deliver network traffic to where services already are, even if that location is not ideal for running the service itself. The Service Fabric Cluster Resource Manager employs fundamentally different strategies for ensuring that the resources in the cluster are efficiently utilized.

## Next steps

- For information on the architecture and information flow within the Cluster Resource Manager, check out [this article](#)
- The Cluster Resource Manager has many options for describing the cluster. To find out more about metrics, check out this article on [describing a Service Fabric cluster](#)
- For more information on configuring services, [Learn about configuring Services](#)(service-fabric-cluster-resource-manager-configure-services.md)
- Metrics are how the Service Fabric Cluster Resource Manager manages consumption and capacity in the cluster. To learn more about metrics and how to configure them check out [this article](#)
- The Cluster Resource Manager works with Service Fabric's management capabilities. To find out more about that integration, read [this article](#)
- To find out about how the Cluster Resource Manager manages and balances load in the cluster, check out the article on [balancing load](#)

# Cluster resource manager architecture overview

8/18/2017 • 3 min to read • [Edit Online](#)

The Service Fabric Cluster Resource Manager is a central service that runs in the cluster. It manages the desired state of the services in the cluster, particularly with respect to resource consumption and any placement rules.

To manage the resources in your cluster, the Service Fabric Cluster Resource Manager must have several pieces of information:

- Which services currently exist
- Each service's current (or default) resource consumption
- The remaining cluster capacity
- The capacity of the nodes in the cluster
- The amount of resources consumed on each node

The resource consumption of a given service can change over time, and services usually care about more than one type of resource. Across different services, there may be both real physical and physical resources being measured. Services may track physical metrics like memory and disk consumption. More commonly, services may care about logical metrics - things like "WorkQueueDepth" or "TotalRequests". Both logical and physical metrics can be used in the same cluster. Metrics can be shared across many services or be specific to a particular service.

## Other considerations

The owners and operators of the cluster can be different from the service and application authors, or at a minimum are the same people wearing different hats. When you develop your application you know a few things about what it requires. You have an estimate of the resources it will consume and how different services should be deployed. For example, the web tier needs to run on nodes exposed to the Internet, while the database services should not. As another example, the web services are probably constrained by CPU and network, while the data tier services care more about memory and disk consumption. However, the person handling a live-site incident for that service in production, or who is managing an upgrade to the service has a different job to do, and requires different tools.

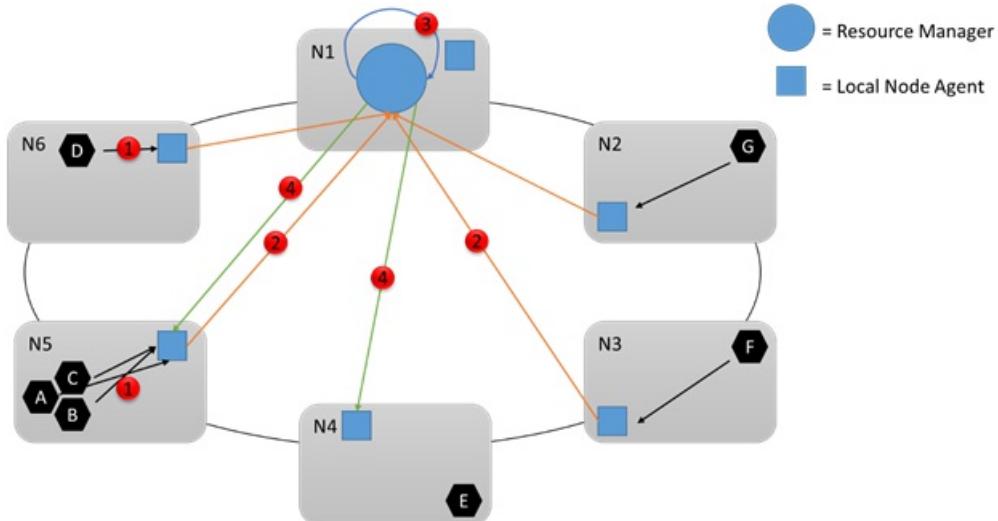
Both the cluster and services are dynamic:

- The number of nodes in the cluster can grow and shrink
- Nodes of different sizes and types can come and go
- Services can be created, removed, and change their desired resource allocations and placement rules
- Upgrades or other management operations can roll through the cluster at the application or infrastructure levels
- Failures can happen at any time.

## Cluster resource manager components and data flow

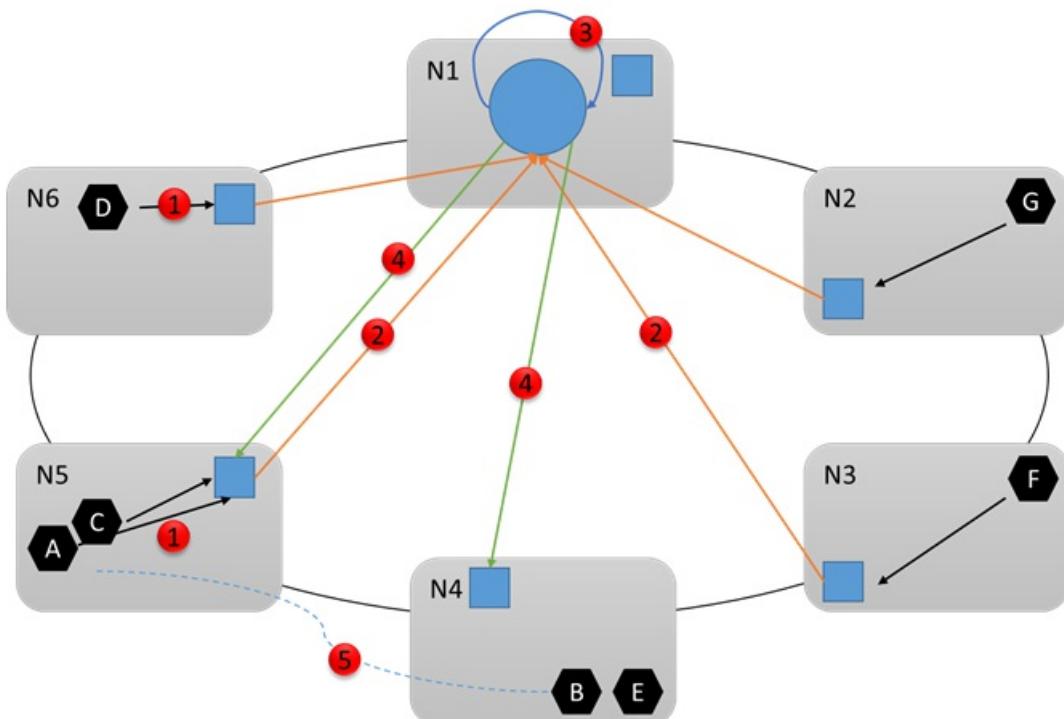
The Cluster Resource Manager has to track the requirements of each service and the consumption of resources by each service object within those services. The Cluster Resource Manager has two conceptual parts: agents that run on each node and a fault-tolerant service. The agents on each node track load reports from services, aggregate them, and periodically report them. The Cluster Resource Manager service aggregates all the information from the local agents and reacts based on its current configuration.

Let's look at the following diagram:



During runtime, there are many changes that could happen. For example, let's say the amount of resources some services consume changes, some services fail, and some nodes join and leave the cluster. All the changes on a node are aggregated and periodically sent to the Cluster Resource Manager service (1,2) where they are aggregated again, analyzed, and stored. Every few seconds that service looks at the changes and determines if any actions are necessary (3). For example, it could notice that some empty nodes have been added to the cluster. As a result, it decides to move some services to those nodes. The Cluster Resource Manager could also notice that a particular node is overloaded, or that certain services have failed or been deleted, freeing up resources elsewhere.

Let's look at the following diagram and see what happens next. Let's say that the Cluster Resource Manager determines that changes are necessary. It coordinates with other system services (in particular the Failover Manager) to make the necessary changes. Then the necessary commands are sent to the appropriate nodes (4). For example, let's say the Resource Manager noticed that Node5 was overloaded, and so decided to move service B from Node5 to Node4. At the end of the reconfiguration (5), the cluster looks like this:



## Next steps

- The Cluster Resource Manager has many options for describing the cluster. To find out more about them, check out this article on [describing a Service Fabric cluster](#)
- The Cluster Resource Manager's primary duties are rebalancing the cluster and enforcing placement rules. For

more information on configuring these behaviors, see [balancing your Service Fabric cluster](#)

# Describing a service fabric cluster

10/26/2017 • 23 min to read • [Edit Online](#)

The Service Fabric Cluster Resource Manager provides several mechanisms for describing a cluster. During runtime, the Cluster Resource Manager uses this information to ensure high availability of the services running in the cluster. While enforcing these important rules, it also attempts to optimize the resource consumption within the cluster.

## Key concepts

The Cluster Resource Manager supports several features that describe a cluster:

- Fault Domains
- Upgrade Domains
- Node Properties
- Node Capacities

## Fault domains

A Fault Domain is any area of coordinated failure. A single machine is a Fault Domain (since it can fail on its own for various reasons, from power supply failures to drive failures to bad NIC firmware). Machines connected to the same Ethernet switch are in the same Fault Domain, as are machines sharing a single source of power or in a single location. Since it's natural for hardware faults to overlap, Fault Domains are inherently hierachal and are represented as URLs in Service Fabric.

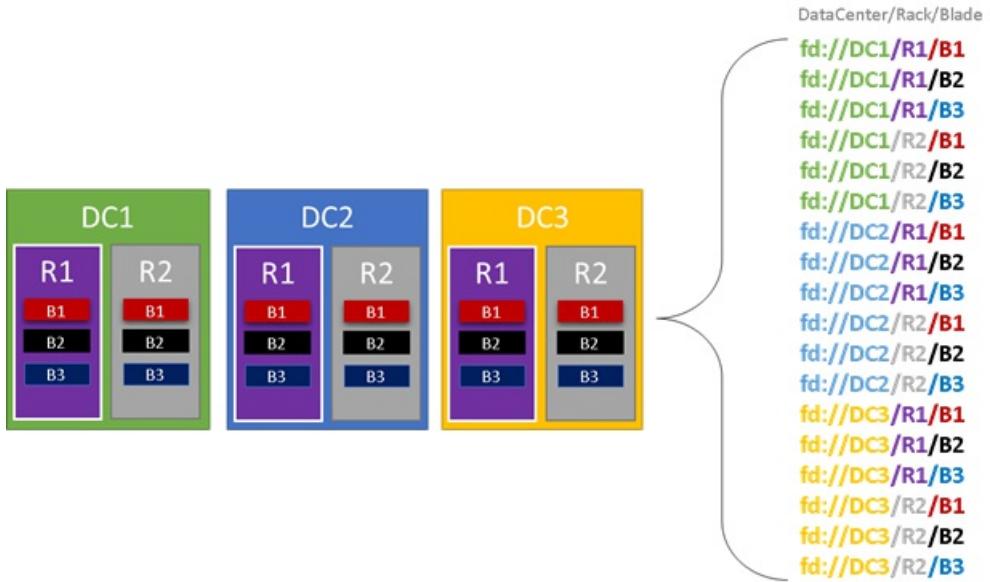
It is important that Fault Domains are set up correctly since Service Fabric uses this information to safely place services. Service Fabric doesn't want to place services such that the loss of a Fault Domain (caused by the failure of some component) causes a service to go down. In the Azure environment Service Fabric uses the Fault Domain information provided by the environment to correctly configure the nodes in the cluster on your behalf. For Service Fabric Standalone, Fault Domains are defined at the time that the cluster is set up

### WARNING

It is important that the Fault Domain information provided to Service Fabric is accurate. For example, let's say that your Service Fabric cluster's nodes are running inside 10 virtual machines, running on five physical hosts. In this case, even though there are 10 virtual machines, there are only 5 different (top level) fault domains. Sharing the same physical host causes VMs to share the same root fault domain, since the VMs experience coordinated failure if their physical host fails.

Service Fabric expects the Fault Domain of a node not to change. Other mechanisms of ensuring high availability of the VMs such as [HA-VMs](#) may cause conflicts with Service Fabric, as they use transparent migration of VMs from one host to another. These mechanisms do not reconfigure or notify the running code inside the VM. As such, they are **not supported** as environments for running Service Fabric clusters. Service Fabric should be the only high-availability technology employed. Mechanisms like live VM migration, SANs, or others are not necessary. If used in conjunction with Service Fabric, these mechanisms *reduce* application availability and reliability because they introduce additional complexity, add centralized sources of failure, and utilize reliability and availability strategies that conflict with those in Service Fabric.

In the graphic below we color all the entities that contribute to Fault Domains and list all the different Fault Domains that result. In this example, we have datacenters ("DC"), racks ("R"), and blades ("B"). Conceivably, if each blade holds more than one virtual machine, there could be another layer in the Fault Domain hierarchy.

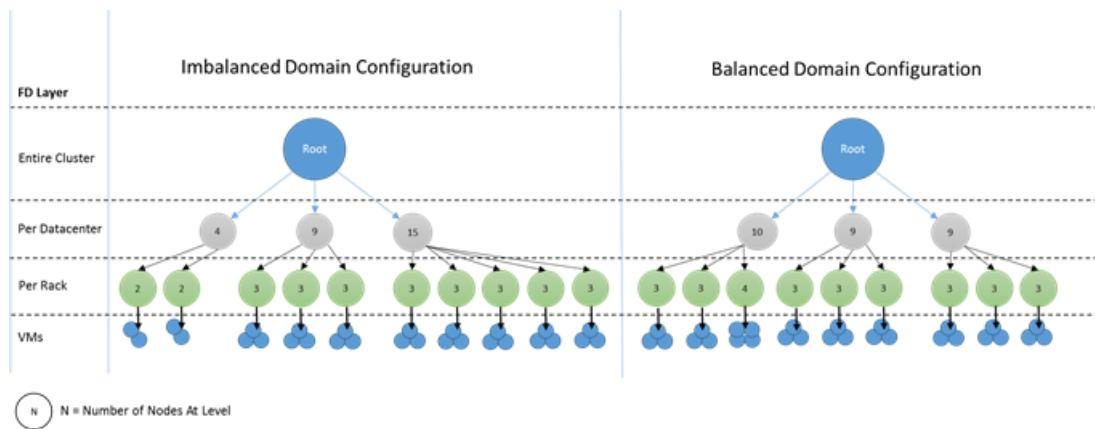


During runtime, the Service Fabric Cluster Resource Manager considers the Fault Domains in the cluster and plans layouts. The stateful replicas or stateless instances for a given service are distributed so they are in separate Fault Domains. Distributing the service across fault domains ensures the availability of the service is not compromised when a Fault Domain fails at any level of the hierarchy.

Service Fabric's Cluster Resource Manager doesn't care how many layers there are in the Fault Domain hierarchy. However, it tries to ensure that the loss of any one portion of the hierarchy doesn't impact services running in it.

It is best if there are the same number of nodes at each level of depth in the Fault Domain hierarchy. If the "tree" of Fault Domains is unbalanced in your cluster, it makes it harder for the Cluster Resource Manager to figure out the best allocation of services. Imbalanced Fault Domains layouts mean that the loss of some domains impact the availability of services more than other domains. As a result, the Cluster Resource Manager is torn between two goals: It wants to use the machines in that "heavy" domain by placing services on them, and it wants to place services in other domains so that the loss of a domain doesn't cause problems.

What do imbalanced domains look like? In the diagram below, we show two different cluster layouts. In the first example, the nodes are distributed evenly across the Fault Domains. In the second example, one Fault Domain has many more nodes than the other Fault Domains.



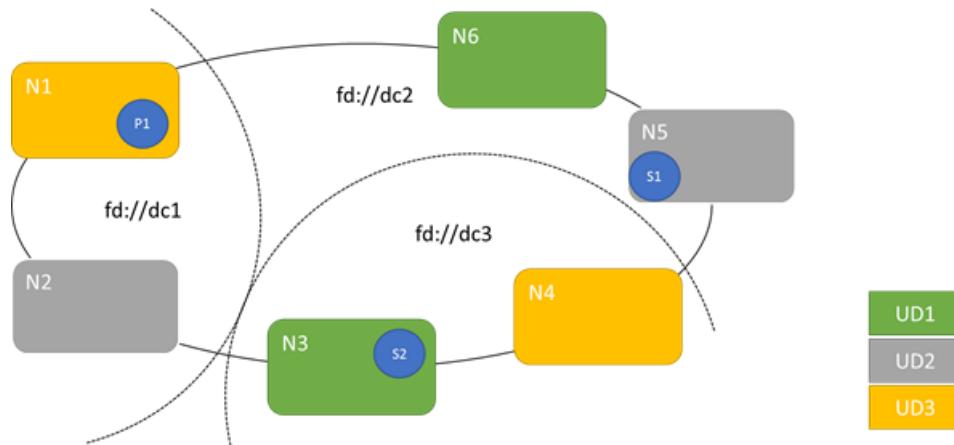
In Azure, the choice of which Fault Domain contains a node is managed for you. However, depending on the number of nodes that you provision you can still end up with Fault Domains with more nodes in them than others. For example, say you have five Fault Domains in the cluster but provision seven nodes for a given NodeType. In this case, the first two Fault Domains end up with more nodes. If you continue to deploy more NodeTypes with only a couple instances, the problem gets worse. For this reason it's recommended that the number of nodes in each node type is a multiple of the number of Fault Domains.

## Upgrade domains

Upgrade Domains are another feature that helps the Service Fabric Cluster Resource Manager understand the layout of the cluster. Upgrade Domains define sets of nodes that are upgraded at the same time. Upgrade Domains help the Cluster Resource Manager understand and orchestrate management operations like upgrades.

Upgrade Domains are a lot like Fault Domains, but with a couple key differences. First, areas of coordinated hardware failures define Fault Domains. Upgrade Domains, on the other hand, are defined by policy. You get to decide how many you want, rather than it being dictated by the environment. You could have as many Upgrade Domains as you do nodes. Another difference between Fault Domains and Upgrade Domains is that Upgrade Domains are not hierarchical. Instead, they are more like a simple tag.

The following diagram shows three Upgrade Domains striped across three Fault Domains. It also shows one possible placement for three different replicas of a stateful service, where each ends up in different Fault and Upgrade Domains. This placement allows the loss of a Fault Domain while in the middle of a service upgrade and still have one copy of the code and data.



There are pros and cons to having large numbers of Upgrade Domains. More Upgrade Domains means each step of the upgrade is more granular and therefore affects a smaller number of nodes or services. As a result, fewer services have to move at a time, introducing less churn into the system. This tends to improve reliability, since less of the service is impacted by any issue introduced during the upgrade. More Upgrade Domains also means that you need less available buffer on other nodes to handle the impact of the upgrade. For example, if you have five Upgrade Domains, the nodes in each are handling roughly 20% of your traffic. If you need to take down that Upgrade Domain for an upgrade, that load usually needs to go somewhere. Since you have four remaining Upgrade Domains, each must have room for about 5% of the total traffic. More Upgrade Domains means you need less buffer on the nodes in the cluster. For example, consider if you had 10 Upgrade Domains instead. In that case, each UD would only be handling about 10% of the total traffic. When an upgrade steps through the cluster, each domain would only need to have room for about 1.1% of the total traffic. More Upgrade Domains generally allow you to run your nodes at higher utilization, since you need less reserved capacity. The same is true for Fault Domains.

The downside of having many Upgrade Domains is that upgrades tend to take longer. Service Fabric waits a short period of time after an Upgrade Domain is completed and performs checks before starting to upgrade the next one. These delays enable detecting issues introduced by the upgrade before the upgrade proceeds. The tradeoff is acceptable because it prevents bad changes from affecting too much of the service at a time.

Too few Upgrade Domains has many negative side effects – while each individual Upgrade Domain is down and being upgraded a large portion of your overall capacity is unavailable. For example, if you only have three Upgrade Domains you are taking down about 1/3 of your overall service or cluster capacity at a time. Having so much of your service down at once isn't desirable since you have to have enough capacity in the rest of your cluster to handle the workload. Maintaining that buffer means that during normal operation those nodes are less loaded than they would be otherwise. This increases the cost of running your service.

There's no real limit to the total number of fault or Upgrade Domains in an environment, or constraints on how they overlap. That said, there are several common patterns:

- Fault Domains and Upgrade Domains mapped 1:1
- One Upgrade Domain per Node (physical or virtual OS instance)
- A “striped” or “matrix” model where the Fault Domains and Upgrade Domains form a matrix with machines usually running down the diagonals

UD Per Node		FD1	FD2	FD3
UD1	Node1			
UD2		Node2		
UD3			Node3	
UD4	Node4			
UD5		Node5		
UD6			Node6	
UD7	Node7			
UD8		Node8		
UD9			Node9	

FD/UD Matrix (Sparse/Diagonal)		FD1	FD2	FD3	FD4	FD5
UD1	Node1					
UD2		Node2				
UD3			Node3			
UD4				Node4		
UD5					Node5	

1:1 FD & UD

FD1/UD1	FD2/UD2	FD3/UD3
Node1	Node2	Node3
Node4	Node5	Node6
Node7	Node8	Node9

FD/UD Matrix

	FD1	FD2	FD3
UD1	Node1	Node2	Node3
UD2	Node4	Node5	Node6
UD3	Node7	Node8	Node9

There's no best answer which layout to choose, each has some pros and cons. For example, the 1FD:1UD model is simple to set up. The 1 Upgrade Domain per Node model is most like what people are used to. During upgrades each node is updated independently. This is similar to how small sets of machines were upgraded manually in the past.

The most common model is the FD/UD matrix, where the FDs and UD form a table and nodes are placed starting along the diagonal. This is the model used by default in Service Fabric clusters in Azure. For clusters with many nodes everything ends up looking like the dense matrix pattern above.

## Fault and Upgrade Domain constraints and resulting behavior

The Cluster Resource Manager treats the desire to keep a service balanced across fault and Upgrade Domains as a constraint. You can find out more about constraints in [this article](#). The Fault and Upgrade Domain constraints state: "For a given service partition there should never be a difference *greater than one* in the number of service objects (stateless service instances or stateful service replicas) between two domains." This prevents certain moves or arrangements that violate this constraint.

Let's look at one example. Let's say that we have a cluster with six nodes, configured with five Fault Domains and five Upgrade Domains.

	FD0	FD1	FD2	FD3	FD4
UD0	N1				
UD1	N6	N2			
UD2			N3		
UD3				N4	
UD4					N5

Now let's say that we create a service with a TargetReplicaSetSize (or, for a stateless service an InstanceCount) of five. The replicas land on N1-N5. In fact, N6 is never used no matter how many services like this you create. But why? Let's look at the difference between the current layout and what would happen if N6 is chosen.

Here's the layout we got and the total number of replicas per Fault and Upgrade Domain:

	<b>FD0</b>	<b>FD1</b>	<b>FD2</b>	<b>FD3</b>	<b>FD4</b>	<b>UDTOTAL</b>
<b>UD0</b>	R1					1
<b>UD1</b>		R2				1
<b>UD2</b>			R3			1
<b>UD3</b>				R4		1
<b>UD4</b>					R5	1
<b>FTotal</b>	1	1	1	1	1	-

This layout is balanced in terms of nodes per Fault Domain and Upgrade Domain. It is also balanced in terms of the number of replicas per Fault and Upgrade Domain. Each domain has the same number of nodes and the same number of replicas.

Now, let's look at what would happen if we'd used N6 instead of N2. How would the replicas be distributed then?

	<b>FD0</b>	<b>FD1</b>	<b>FD2</b>	<b>FD3</b>	<b>FD4</b>	<b>UDTOTAL</b>
<b>UD0</b>	R1					1
<b>UD1</b>	R5					1
<b>UD2</b>			R2			1
<b>UD3</b>				R3		1
<b>UD4</b>					R4	1
<b>FTotal</b>	2	0	1	1	1	-

This layout violates our definition for the Fault Domain constraint. FD0 has two replicas, while FD1 has zero, making the difference between FD0 and FD1 a total of two. The Cluster Resource Manager does not allow this arrangement. Similarly if we picked N2 and N6 (instead of N1 and N2) we'd get:

	<b>FD0</b>	<b>FD1</b>	<b>FD2</b>	<b>FD3</b>	<b>FD4</b>	<b>UDTOTAL</b>
<b>UD0</b>						0
<b>UD1</b>	R5	R1				2
<b>UD2</b>			R2			1
<b>UD3</b>				R3		1

	<b>FD0</b>	<b>FD1</b>	<b>FD2</b>	<b>FD3</b>	<b>FD4</b>	<b>UDTOTAL</b>
<b>UD4</b>					R4	1
<b>FDTOTAL</b>	1	1	1	1	1	-

This layout is balanced in terms of Fault Domains. However, now it's violating the Upgrade Domain constraint. This is because UD0 has zero replicas while UD1 has two. Therefore, this layout is also invalid and won't be picked by the Cluster Resource Manager.

## Configuring fault and Upgrade Domains

Defining Fault Domains and Upgrade Domains is done automatically in Azure hosted Service Fabric deployments. Service Fabric picks up and uses the environment information from Azure.

If you're creating your own cluster (or want to run a particular topology in development), you can provide the Fault Domain and Upgrade Domain information yourself. In this example, we define a nine node local development cluster that spans three "datacenters" (each with three racks). This cluster also has three Upgrade Domains striped across those three datacenters. An example of the configuration is below:

ClusterManifest.xml

```

<Infrastructure>
    <!-- IsScaleMin indicates that this cluster runs on one-box /one single server -->
    <WindowsServer IsScaleMin="true">
        <NodeList>
            <Node NodeName="Node01" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType01"
FaultDomain="fd:/DC01/Rack01" UpgradeDomain="UpgradeDomain1" IsSeedNode="true" />
            <Node NodeName="Node02" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType02"
FaultDomain="fd:/DC01/Rack02" UpgradeDomain="UpgradeDomain2" IsSeedNode="true" />
            <Node NodeName="Node03" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType03"
FaultDomain="fd:/DC01/Rack03" UpgradeDomain="UpgradeDomain3" IsSeedNode="true" />
            <Node NodeName="Node04" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType04"
FaultDomain="fd:/DC02/Rack01" UpgradeDomain="UpgradeDomain1" IsSeedNode="true" />
            <Node NodeName="Node05" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType05"
FaultDomain="fd:/DC02/Rack02" UpgradeDomain="UpgradeDomain2" IsSeedNode="true" />
            <Node NodeName="Node06" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType06"
FaultDomain="fd:/DC02/Rack03" UpgradeDomain="UpgradeDomain3" IsSeedNode="true" />
            <Node NodeName="Node07" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType07"
FaultDomain="fd:/DC03/Rack01" UpgradeDomain="UpgradeDomain1" IsSeedNode="true" />
            <Node NodeName="Node08" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType08"
FaultDomain="fd:/DC03/Rack02" UpgradeDomain="UpgradeDomain2" IsSeedNode="true" />
            <Node NodeName="Node09" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType09"
FaultDomain="fd:/DC03/Rack03" UpgradeDomain="UpgradeDomain3" IsSeedNode="true" />
        </NodeList>
    </WindowsServer>
</Infrastructure>
```

via ClusterConfig.json for Standalone deployments

```
"nodes": [
  {
    "nodeName": "vm1",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc1/r0",
    "upgradeDomain": "UD1"
  },
  {
    "nodeName": "vm2",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc1/r0",
    "upgradeDomain": "UD2"
  },
  {
    "nodeName": "vm3",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc1/r0",
    "upgradeDomain": "UD3"
  },
  {
    "nodeName": "vm4",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc2/r0",
    "upgradeDomain": "UD1"
  },
  {
    "nodeName": "vm5",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc2/r0",
    "upgradeDomain": "UD2"
  },
  {
    "nodeName": "vm6",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc2/r0",
    "upgradeDomain": "UD3"
  },
  {
    "nodeName": "vm7",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc3/r0",
    "upgradeDomain": "UD1"
  },
  {
    "nodeName": "vm8",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc3/r0",
    "upgradeDomain": "UD2"
  },
  {
    "nodeName": "vm9",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc3/r0",
    "upgradeDomain": "UD3"
  }
],
```

## NOTE

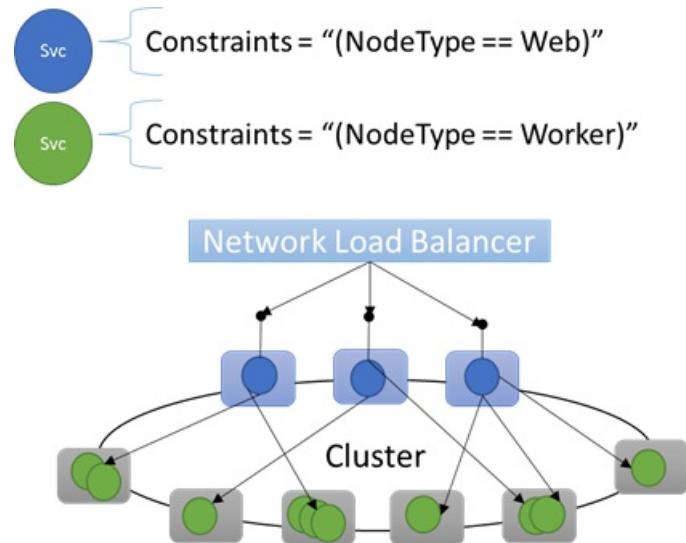
When defining clusters via Azure Resource Manager, Fault Domains and Upgrade Domains are assigned by Azure. Therefore, the definition of your Node Types and Virtual Machine Scale Sets in your Azure Resource Manager template does not include Fault Domain or Upgrade Domain information.

## Node properties and placement constraints

Sometimes (in fact, most of the time) you're going to want to ensure that certain workloads run only on certain types of nodes in the cluster. For example, some workload may require GPUs or SSDs while others may not. A great example of targeting hardware to particular workloads is almost every n-tier architecture out there. Certain machines serve as the front end or API serving side of the application and are exposed to the clients or the internet. Different machines, often with different hardware resources, handle the work of the compute or storage layers. These are usually *not* directly exposed to clients or the internet. Service Fabric expects that there are cases where particular workloads need to run on particular hardware configurations. For example:

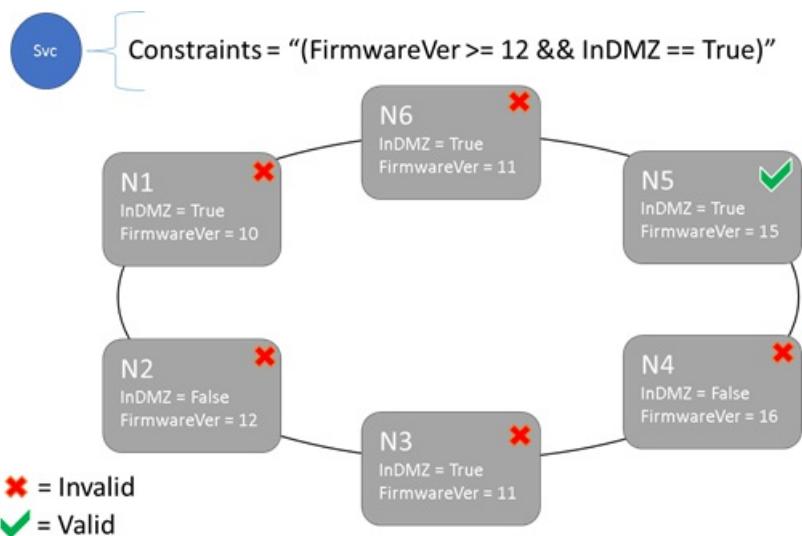
- an existing n-tier application has been "lifted and shifted" into a Service Fabric environment
- a workload wants to run on specific hardware for performance, scale, or security isolation reasons
- A workload should be isolated from other workloads for policy or resource consumption reasons

To support these sorts of configurations, Service Fabric has a first class notion of tags that can be applied to nodes. These tags are called **node properties**. **Placement constraints** are the statements attached to individual services that select for one or more node properties. Placement constraints define where services should run. The set of constraints is extensible - any key/value pair can work.



### Built in node properties

Service Fabric defines some default node properties that can be used automatically without the user having to define them. The default properties defined at each node are the **NodeType** and the **NodeName**. So for example you could write a placement constraint as `"(NodeType == NodeType03)"`. Generally we have found NodeType to be one of the most commonly used properties. It is useful since it corresponds 1:1 with a type of a machine. Each type of machine corresponds to a type of workload in a traditional n-tier application.



## Placement Constraint and Node Property Syntax

The value specified in the node property can be a string, bool, or signed long. The statement at the service is called a placement *constraint* since it constrains where the service can run in the cluster. The constraint can be any Boolean statement that operates on the different node properties in the cluster. The valid selectors in these boolean statements are:

- 1) conditional checks for creating particular statements

STATEMENT	SYNTAX
"equal to"	"=="
"not equal to"	"!="
"greater than"	
"greater than or equal to"	>="
"less than"	<"
"less than or equal to"	<="

- 2) boolean statements for grouping and logical operations

STATEMENT	SYNTAX
"and"	"&&"
"or"	"  "
"not"	"!"
"group as single statement"	"()"

Here are some examples of basic constraint statements.

- "Value >= 5"

- "NodeColor != green"
- "((OneProperty < 100) || ((AnotherProperty == false) && (OneProperty >= 100)))"

Only nodes where the overall placement constraint statement evaluates to "True" can have the service placed on it. Nodes that do not have a property defined do not match any placement constraint containing that property.

Let's say that the following node properties were defined for a given node type:

ClusterManifest.xml

```
<NodeType Name="NodeType01">
  <PlacementProperties>
    <Property Name="HasSSD" Value="true"/>
    <Property Name="NodeColor" Value="green"/>
    <Property Name="SomeProperty" Value="5"/>
  </PlacementProperties>
</NodeType>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters.

#### **NOTE**

In your Azure Resource Manager template the node type is usually parameterized. It would look like "[parameters('vmNodeType1Name')]" rather than "NodeType01".

```
"nodeTypes": [
  {
    "name": "NodeType01",
    "placementProperties": {
      "HasSSD": "true",
      "NodeColor": "green",
      "SomeProperty": "5"
    }
  }
],
```

You can create service placement *constraints* for a service like as follows:

C#

```
FabricClient fabricClient = new FabricClient();
StatefulServiceDescription serviceDescription = new StatefulServiceDescription();
serviceDescription.PlacementConstraints = "(HasSSD == true && SomeProperty >= 4)";
// add other required servicedescription fields
//...
await fabricClient.ServiceManager.CreateServiceAsync(serviceDescription);
```

Powershell:

```
New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName
$serviceType -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -
PlacementConstraint "HasSSD == true && SomeProperty >= 4"
```

If all nodes of NodeType01 are valid, you can also select that node type with the constraint "(NodeType == NodeType01)".

One of the cool things about a service's placement constraints is that they can be updated dynamically during

runtime. So if you need to, you can move a service around in the cluster, add and remove requirements, etc. Service Fabric takes care of ensuring that the service stays up and available even when these types of changes are made.

C#:

```
StatefulServiceUpdateDescription updateDescription = new StatefulServiceUpdateDescription();
updateDescription.PlacementConstraints = "NodeType == NodeType01";
await fabricClient.ServiceManager.UpdateServiceAsync(new Uri("fabric:/app/service"), updateDescription);
```

Powershell:

```
Update-ServiceFabricService -Stateful -ServiceName $serviceName -PlacementConstraints "NodeType == NodeType01"
```

Placement constraints are specified for every different named service instance. Updates always take the place of (overwrite) what was previously specified.

The cluster definition defines the properties on a node. Changing a node's properties requires a cluster configuration upgrade. Upgrading a node's properties requires each affected node to restart to report its new properties. These rolling upgrades are managed by Service Fabric.

## Describing and Managing Cluster Resources

One of the most important jobs of any orchestrator is to help manage resource consumption in the cluster. Managing cluster resources can mean a couple of different things. First, there's ensuring that machines are not overloaded. This means making sure that machines aren't running more services than they can handle. Second, there's balancing and optimization which is critical to running services efficiently. Cost effective or performance sensitive service offerings can't allow some nodes to be hot while others are cold. Hot nodes lead to resource contention and poor performance, and cold nodes represent wasted resources and increased costs.

Service Fabric represents resources as [Metrics](#). Metrics are any logical or physical resource that you want to describe to Service Fabric. Examples of metrics are things like "WorkQueueDepth" or "MemoryInMb". For information about the physical resources that Service Fabric can govern on nodes, see [resource governance](#). For information on configuring custom metrics and their uses, see [this article](#)

Metrics are different from placements constraints and node properties. Node properties are static descriptors of the nodes themselves. Metrics describe resources that nodes have and that services consume when they are run on a node. A node property could be "HasSSD" and could be set to true or false. The amount of space available on that SSD and how much is consumed by services would be a metric like "DriveSpaceInMb".

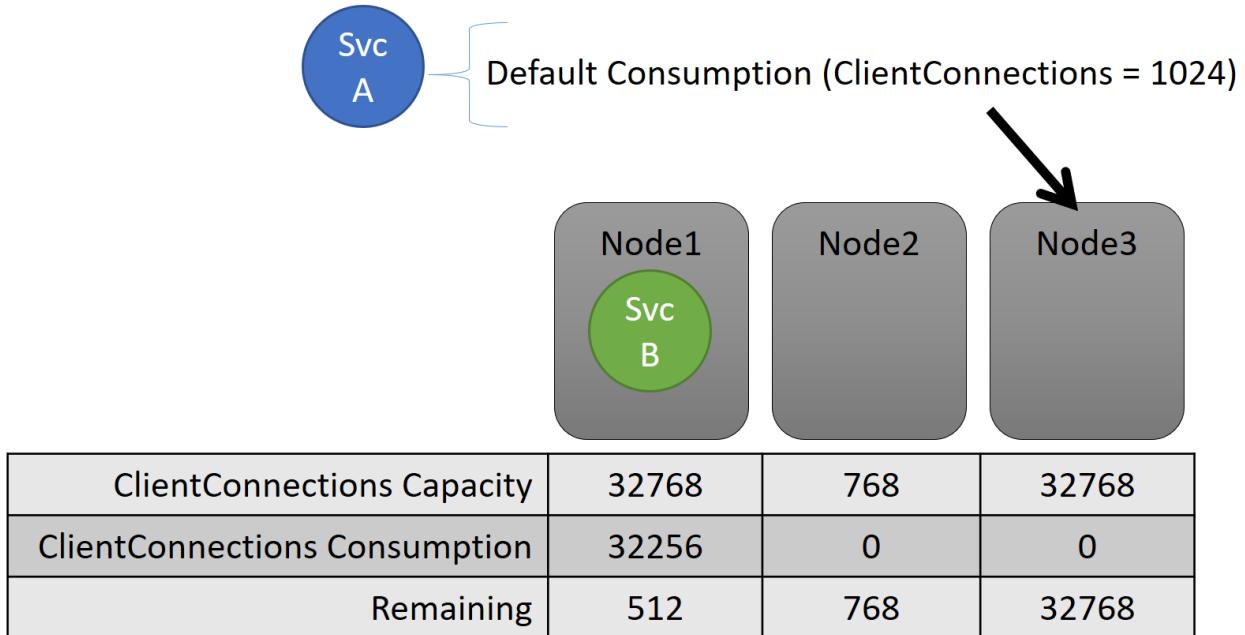
It is important to note that just like for placement constraints and node properties, the Service Fabric Cluster Resource Manager doesn't understand what the names of the metrics mean. Metric names are just strings. It is a good practice to declare units as a part of the metric names that you create when it could be ambiguous.

## Capacity

If you turned off all resource *balancing*, Service Fabric's Cluster Resource Manager would still ensure that no node ended up over its capacity. Managing capacity overruns is possible unless the cluster is too full or the workload is larger than any node. Capacity is another *constraint* that the Cluster Resource Manager uses to understand how much of a resource a node has. Remaining capacity is also tracked for the cluster as a whole. Both the capacity and the consumption at the service level are expressed in terms of metrics. So for example, the metric might be "ClientConnections" and a given Node may have a capacity for "ClientConnections" of 32768. Other nodes can have other limits. Some service running on that node can say it is currently consuming 32256 of the metric "ClientConnections".

During runtime, the Cluster Resource Manager tracks remaining capacity in the cluster and on nodes. In order to

track capacity the Cluster Resource Manager subtracts each service's usage from node's capacity where the service runs. With this information, the Service Fabric Cluster Resource Manager can figure out where to place or move replicas so that nodes don't go over capacity.



C#:

```
StatefulServiceDescription serviceDescription = new StatefulServiceDescription();
ServiceLoadMetricDescription metric = new ServiceLoadMetricDescription();
metric.Name = "ClientConnections";
metric.PrimaryDefaultLoad = 1024;
metric.SecondaryDefaultLoad = 0;
metric.Weight = ServiceLoadMetricWeight.High;
serviceDescription.Metrics.Add(metric);
await fabricClient.ServiceManager.CreateServiceAsync(serviceDescription);
```

Powershell:

```
New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName
$serviceTypeName -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -Metric
@("ClientConnections,High,1024,0)
```

You can see capacities defined in the cluster manifest:

ClusterManifest.xml

```
<NodeType Name="NodeType03">
<Capacities>
<Capacity Name="ClientConnections" Value="65536"/>
</Capacities>
</NodeType>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters.

```

"nodeTypes": [
    {
        "name": "NodeType03",
        "capacities": {
            "ClientConnections": "65536",
        }
    }
],

```

Commonly a service's load changes dynamically. Say that a replica's load of "ClientConnections" changed from 1024 to 2048, but the node it was running on then only had 512 capacity remaining for that metric. Now that replica or instance's placement is invalid, since there's not enough room on that node. The Cluster Resource Manager has to kick in and get the node back below capacity. It reduces load on the node that is over capacity by moving one or more of the replicas or instances from that node to other nodes. When moving replicas, the Cluster Resource Manager tries to minimize the cost of those movements. Movement cost is discussed in [this article](#) and more about the Cluster Resource Manager's rebalancing strategies and rules is described [here](#).

## Cluster capacity

So how does the Service Fabric Cluster Resource Manager keep the overall cluster from being too full? Well, with dynamic load there's not a lot it can do. Services can have their load spike independently of actions taken by the Cluster Resource Manager. As a result, your cluster with plenty of headroom today may be underpowered when you become famous tomorrow. That said, there are some controls that are baked in to prevent problems. The first thing we can do is prevent the creation of new workloads that would cause the cluster to become full.

Say that you create a stateless service and it has some load associated with it. Let's say that the service cares about the "DiskSpaceInMb" metric. Let's also say that it is going to consume five units of "DiskSpaceInMb" for every instance of the service. You want to create three instances of the service. Great! So that means that we need 15 units of "DiskSpaceInMb" to be present in the cluster in order for us to even be able to create these service instances. The Cluster Resource Manager continually calculates the capacity and consumption of each metric so it can determine the remaining capacity in the cluster. If there isn't enough space, the Cluster Resource Manager rejects the create service call.

Since the requirement is only that there be 15 units available, this space could be allocated many different ways. For example, there could be one remaining unit of capacity on 15 different nodes, or three remaining units of capacity on five different nodes. If the Cluster Resource Manager can rearrange things so there's five units available on three nodes, it places the service. Rearranging the cluster is usually possible unless the cluster is almost full or the existing services can't be consolidated for some reason.

## Buffered Capacity

Buffered capacity is another feature of the Cluster Resource Manager. It allows reservation of some portion of the overall node capacity. This capacity buffer is only used to place services during upgrades and node failures. Buffered Capacity is specified globally per metric for all nodes. The value you pick for the reserved capacity is a function of the number of Fault and Upgrade Domains you have in the cluster. More Fault and Upgrade Domains means that you can pick a lower number for your buffered capacity. If you have more domains, you can expect smaller amounts of your cluster to be unavailable during upgrades and failures. Specifying Buffered Capacity only makes sense if you have also specified the node capacity for a metric.

Here's an example of how to specify buffered capacity:

ClusterManifest.xml

```

<Section Name="NodeBufferPercentage">
    <Parameter Name="SomeMetric" Value="0.15" />
    <Parameter Name="SomeOtherMetric" Value="0.20" />
</Section>

```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters:

```

"fabricSettings": [
{
    "name": "NodeBufferPercentage",
    "parameters": [
        {
            "name": "SomeMetric",
            "value": "0.15"
        },
        {
            "name": "SomeOtherMetric",
            "value": "0.20"
        }
    ]
}
]

```

The creation of new services fails when the cluster is out of buffered capacity for a metric. Preventing the creation of new services to preserve the buffer ensures that upgrades and failures don't cause nodes to go over capacity. Buffered capacity is optional but is recommended in any cluster that defines a capacity for a metric.

The Cluster Resource Manager exposes this load information. For each metric, this information includes:

- the buffered capacity settings
- the total capacity
- the current consumption
- whether each metric is considered balanced or not
- statistics about the standard deviation
- the nodes which have the most and least load

Below we see an example of that output:

```
PS C:\Users\user> Get-ServiceFabricClusterLoadInformation
LastBalancingStartTimeUtc : 9/1/2016 12:54:59 AM
LastBalancingEndTimeUtc   : 9/1/2016 12:54:59 AM
LoadMetricInformation     :
    LoadMetricName      : Metric1
    IsBalancedBefore    : False
    IsBalancedAfter     : False
    DeviationBefore     : 0.192450089729875
    DeviationAfter      : 0.192450089729875
    BalancingThreshold  : 1
    Action              : NoActionNeeded
    ActivityThreshold   : 0
    ClusterCapacity     : 189
    ClusterLoad         : 45
    ClusterRemainingCapacity : 144
    NodeBufferPercentage : 10
    ClusterBufferedCapacity : 170
    ClusterRemainingBufferedCapacity : 125
    ClusterCapacityViolation : False
    MinNodeLoadValue    : 0
    MinNodeLoadNodeId   : 3ea71e8e01f4b0999b121abcbf27d74d
    MaxNodeLoadValue    : 15
    MaxNodeLoadNodeId   : 2cc648b6770be1bc9824fa995d5b68b1
```

## Next steps

- For information on the architecture and information flow within the Cluster Resource Manager, check out [this article](#)
- Defining Defragmentation Metrics is one way to consolidate load on nodes instead of spreading it out. To learn how to configure defragmentation, refer to [this article](#)
- Start from the beginning and [get an Introduction to the Service Fabric Cluster Resource Manager](#)
- To find out about how the Cluster Resource Manager manages and balances load in the cluster, check out the article on [balancing load](#)

# Introduction to Application Groups

8/18/2017 • 7 min to read • [Edit Online](#)

Service Fabric's Cluster Resource Manager typically manages cluster resources by spreading the load (represented via [Metrics](#)) evenly throughout the cluster. Service Fabric manages the capacity of the nodes in the cluster and the cluster as a whole via [capacity](#). Metrics and capacity work great for many workloads, but patterns that make heavy use of different Service Fabric Application Instances sometimes bring in additional requirements. For example you may want to:

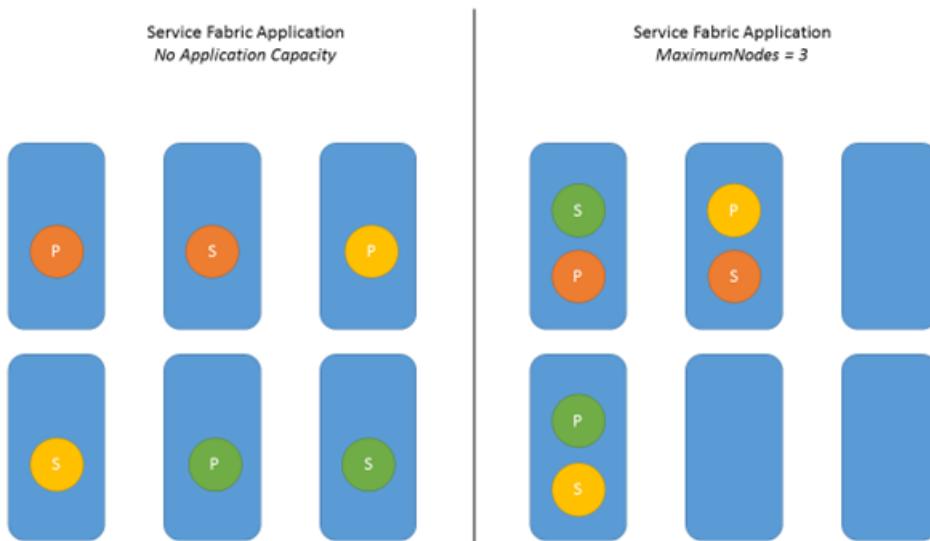
- Reserve some capacity on the nodes in the cluster for the services within some named application instance
- Limit the total number of nodes that the services within a named application instance run on (instead of spreading them out over the entire cluster)
- Define capacities on the named application instance itself to limit the number of services or total resource consumption of the services inside it

To meet these requirements, the Service Fabric Cluster Resource Manager supports a feature called Application Groups.

## Limiting the maximum number of nodes

The simplest use case for Application capacity is when an application instance needs to be limited to a certain maximum number of nodes. This consolidates all services within that application instance onto a set number of machines. Consolidation is useful when you're trying to predict or cap physical resource use by the services within that named application instance.

The following image shows an application instance with and without a maximum number of nodes defined:



In the left example, the application doesn't have a maximum number of nodes defined, and it has three services. The Cluster Resource Manager has spread out all replicas across six available nodes to achieve the best balance in the cluster (the default behavior). In the right example, we see the same application limited to three nodes.

The parameter that controls this behavior is called `MaximumNodes`. This parameter can be set during application creation, or updated for an application instance that was already running.

Powershell

```
New-ServiceFabricApplication -ApplicationName fabric:/AppName -ApplicationTypeName AppType1 -  
ApplicationTypeVersion 1.0.0.0 -MaximumNodes 3  
Update-ServiceFabricApplication -Name fabric:/AppName -MaximumNodes 5
```

C#

```
ApplicationDescription ad = new ApplicationDescription();  
ad.ApplicationName = new Uri("fabric:/AppName");  
ad.ApplicationTypeName = "AppType1";  
ad.ApplicationTypeVersion = "1.0.0.0";  
ad.MaximumNodes = 3;  
await fc.ApplicationManager.CreateApplicationAsync(ad);  
  
ApplicationUpdateDescription adUpdate = new ApplicationUpdateDescription(new Uri("fabric:/AppName"));  
adUpdate.MaximumNodes = 5;  
await fc.ApplicationManager.UpdateApplicationAsync(adUpdate);
```

Within the set of nodes, the Cluster Resource Manager doesn't guarantee which service objects get placed together or which nodes get used.

## Application Metrics, Load, and Capacity

Application Groups also allow you to define metrics associated with a given named application instance, and that application instance's capacity for those metrics. Application metrics allow you to track, reserve, and limit the resource consumption of the services inside that application instance.

For each application metric, there are two values that can be set:

- **Total Application Capacity** – This setting represents the total capacity of the application for a particular metric. The Cluster Resource Manager disallows the creation of any new services within this application instance that would cause total load to exceed this value. For example, let's say the application instance had a capacity of 10 and already had load of five. The creation of a service with a total default load of 10 would be disallowed.
- **Maximum Node Capacity** – This setting specifies the maximum total load for the application on a single node. If load goes over this capacity, the Cluster Resource Manager moves replicas to other nodes so that the load decreases.

Powershell:

```
New-ServiceFabricApplication -ApplicationName fabric:/AppName -ApplicationTypeName AppType1 -  
ApplicationTypeVersion 1.0.0.0 -Metrics  
@("MetricName:Metric1,MaximumNodeCapacity:100,MaximumApplicationCapacity:1000")
```

C#:

```
ApplicationDescription ad = new ApplicationDescription();  
ad.ApplicationName = new Uri("fabric:/AppName");  
ad.ApplicationTypeName = "AppType1";  
ad.ApplicationTypeVersion = "1.0.0.0";  
  
var appMetric = new ApplicationMetricDescription();  
appMetric.Name = "Metric1";  
appMetric.TotalApplicationCapacity = 1000;  
appMetric.MaximumNodeCapacity = 100;  
ad.Metrics.Add(appMetric);  
await fc.ApplicationManager.CreateApplicationAsync(ad);
```

# Reserving Capacity

Another common use for application groups is to ensure that resources within the cluster are reserved for a given application instance. The space is always reserved when the application instance is created.

Reserving space in the cluster for the application happens immediately even when:

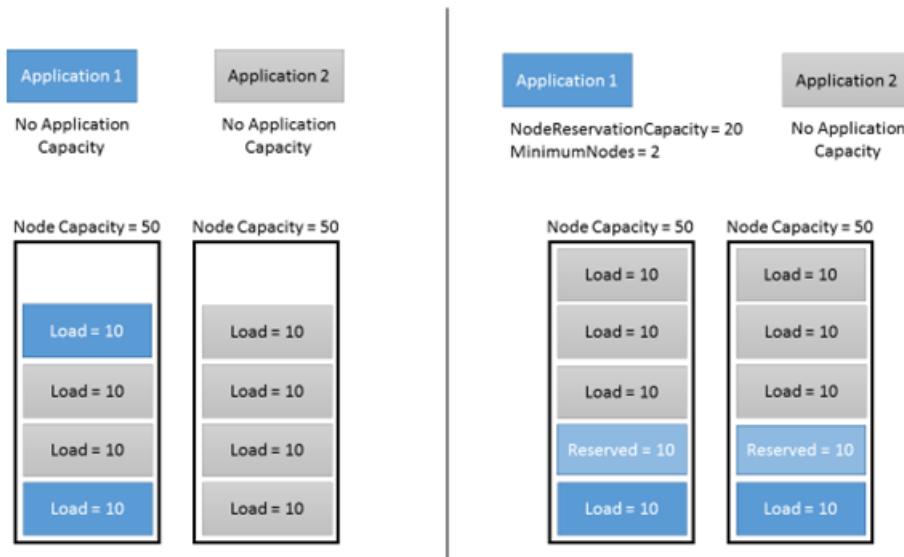
- the application instance is created but doesn't have any services within it yet
- the number of services within the application instance changes every time
- the services exist but aren't consuming the resources

Reserving resources for an application instance requires specifying two additional parameters: *MinimumNodes* and *NodeReservationCapacity*

- **MinimumNodes** - Defines the minimum number of nodes that the application instance should run on.
- **NodeReservationCapacity** - This setting is per metric for the application. The value is the amount of that metric reserved for the application on any node where that the services in that application run.

Combining **MinimumNodes** and **NodeReservationCapacity** guarantees a minimum load reservation for the application within the cluster. If there's less remaining capacity in the cluster than the total reservation required, creation of the application fails.

Let's look at an example of capacity reservation:



In the left example, applications do not have any Application Capacity defined. The Cluster Resource Manager balances everything according to normal rules.

In the example on the right, let's say that Application1 was created with the following settings:

- MinimumNodes set to two
- An application Metric defined with
  - NodeReservationCapacity of 20

Powershell

```
New-ServiceFabricApplication -ApplicationName fabric:/AppName -ApplicationTypeName AppType1 -  
ApplicationTypeVersion 1.0.0.0 -MinimumNodes 2 -Metrics @("MetricName:Metric1,NodeReservationCapacity:20")
```

C#

```

ApplicationDescription ad = new ApplicationDescription();
ad.ApplicationName = new Uri("fabric:/AppName");
ad.ApplicationTypeName = "AppType1";
ad.ApplicationTypeVersion = "1.0.0.0";
ad.MinimumNodes = 2;

var appMetric = new ApplicationMetricDescription();
appMetric.Name = "Metric1";
appMetric.NodeReservationCapacity = 20;

ad.Metrics.Add(appMetric);

await fc.ApplicationManager.CreateApplicationAsync(ad);

```

Service Fabric reserves capacity on two nodes for Application1, and doesn't allow services from Application2 to consume that capacity even if there are no load is being consumed by the services inside Application1 at the time. This reserved application capacity is considered consumed and counts against the remaining capacity on that node and within the cluster. The reservation is deducted from the remaining cluster capacity immediately, however the reserved consumption is deducted from the capacity of a specific node only when at least one service object is placed on it. This later reservation allows for flexibility and better resource utilization since resources are only reserved on nodes when needed.

## Obtaining the application load information

For each application that has an Application Capacity defined for one or more metrics you can obtain the information about the aggregate load reported by replicas of its services.

Powershell:

```
Get-ServiceFabricApplicationLoad -ApplicationName fabric:/MyApplication1
```

C#

```

var v = await fc.QueryManager.GetApplicationLoadInformationAsync("fabric:/MyApplication1");
var metrics = v.ApplicationLoadMetricInformation;
foreach (ApplicationLoadMetricInformation metric in metrics)
{
    Console.WriteLine(metric.ApplicationCapacity); //total capacity for this metric in this application
    instance
    Console.WriteLine(metric.ReservationCapacity); //reserved capacity for this metric in this application
    instance
    Console.WriteLine(metric.ApplicationLoad); //current load for this metric in this application instance
}

```

The ApplicationLoad query returns the basic information about Application Capacity that was specified for the application. This information includes the Minimum Nodes and Maximum Nodes info, and the number that the application is currently occupying. It also includes information about each application load metric, including:

- Metric Name: Name of the metric.
- Reservation Capacity: Cluster Capacity that is reserved in the cluster for this Application.
- Application Load: Total Load of this Application's child replicas.
- Application Capacity: Maximum permitted value of Application Load.

## Removing Application Capacity

Once the Application Capacity parameters are set for an application, they can be removed using Update Application

APIs or PowerShell cmdlets. For example:

```
Update-ServiceFabricApplication -Name fabric:/MyApplication1 -RemoveApplicationCapacity
```

This command removes all Application capacity management parameters from the application instance. This includes MinimumNodes, MaximumNodes, and the Application's metrics, if any. The effect of the command is immediate. After this command completes, the Cluster Resource Manager uses the default behavior for managing applications. Application Capacity parameters can be specified again via `Update-ServiceFabricApplication / System.Fabric.FabricClient.ApplicationManagementClient.UpdateApplicationAsync()`.

### Restrictions on Application Capacity

There are several restrictions on Application Capacity parameters that must be respected. If there are validation errors no changes take place.

- All integer parameters must be non-negative numbers.
- MinimumNodes must never be greater than MaximumNodes.
- If capacities for a load metric are defined, then they must follow these rules:
  - Node Reservation Capacity must not be greater than Maximum Node Capacity. For example, you cannot limit the capacity for the metric "CPU" on the node to two units and try to reserve three units on each node.
  - If MaximumNodes is specified, then the product of MaximumNodes and Maximum Node Capacity must not be greater than Total Application Capacity. For example, let's say the Maximum Node Capacity for load metric "CPU" is set to eight. Let's also say you set the Maximum Nodes to 10. In this case, Total Application Capacity must be greater than 80 for this load metric.

The restrictions are enforced both during application creation and updates.

### How not to use Application Capacity

- Do not try to use the Application Group features to constrain the application to a *specific* subset of nodes. In other words, you can specify that the application runs on at most five nodes, but not which specific five nodes in the cluster. Constraining an application to specific nodes can be achieved using placement constraints for services.
- Do not try to use the Application Capacity to ensure that two services from the same application are placed on the same nodes. Instead use [affinity](#) or [placement constraints](#).

### Next steps

- For more information on configuring services, [Learn about configuring Services](#)
- To find out about how the Cluster Resource Manager manages and balances load in the cluster, check out the article on [balancing load](#)
- Start from the beginning and [get an Introduction to the Service Fabric Cluster Resource Manager](#)
- For more information on how metrics work generally, read up on [Service Fabric Load Metrics](#)
- The Cluster Resource Manager has many options for describing the cluster. To find out more about them, check out this article on [describing a Service Fabric cluster](#)

# Configuring cluster resource manager settings for Service Fabric services

8/18/2017 • 2 min to read • [Edit Online](#)

The Service Fabric Cluster Resource Manager allows fine-grained control over the rules that govern every individual named service. Each named service can specify rules for how it should be allocated in the cluster. Each named service can also define the set of metrics that it wants to report, including how important they are to that service. Configuring services breaks down into three different tasks:

1. Configuring placement constraints
2. Configuring metrics
3. Configuring advanced placement policies and other rules (less common)

## Placement constraints

Placement constraints are used to control which nodes in the cluster a service can actually run on. Typically a particular named service instance or all services of a given type constrained to run on a particular type of node. Placement constraints are extensible. You can define any set of properties per node type, and then select for them with constraints when creating services. You can also change a service's placement constraints while it is running. This allows you to respond to changes in the cluster or the requirements of the service. The properties of a given node can also be updated dynamically in the cluster. More information on placement constraints and how to configure them can be found in [this article](#)

## Metrics

Metrics are the set of resources that a given named service needs. A service's metric configuration includes how much of that resource each stateful replica or stateless instance of that service consumes by default. Metrics also include a weight that indicates how important balancing that metric is to that service, in case tradeoffs are necessary.

## Advanced placement rules

There are other types of placement rules that are useful in less common scenarios. Some examples are:

- Constraints that help with geographically distributed clusters
- Certain application architectures

Other placement rules are configured via either Correlations or Policies.

## Next steps

- Metrics are how the Service Fabric Cluster Resource Manager manages consumption and capacity in the cluster. To learn more about metrics and how to configure them, check out [this article](#)
- Affinity is one mode you can configure for your services. It is not common, but if you need it you can learn about it [here](#)
- There are many different placement rules that can be configured on your service to handle additional scenarios. You can find out about those different placement policies [here](#)
- Start from the beginning and [get an Introduction to the Service Fabric Cluster Resource Manager](#)
- To find out about how the Cluster Resource Manager manages and balances load in the cluster, check out the

article on [balancing load](#)

- The Cluster Resource Manager has many options for describing the cluster. To find out more about them, check out this article on [describing a Service Fabric cluster](#)

# Managing resource consumption and load in Service Fabric with metrics

8/18/2017 • 17 min to read • [Edit Online](#)

Metrics are the resources that your services care about and which are provided by the nodes in the cluster. A metric is anything that you want to manage in order to improve or monitor the performance of your services. For example, you might watch memory consumption to know if your service is overloaded. Another use is to figure out whether the service could move elsewhere where memory is less constrained in order to get better performance.

Things like Memory, Disk, and CPU usage are examples of metrics. These metrics are physical metrics, resources that correspond to physical resources on the node that need to be managed. Metrics can also be (and commonly are) logical metrics. Logical metrics are things like "MyWorkQueueDepth" or "MessagesToProcess" or "TotalRecords". Logical metrics are application-defined and indirectly correspond to some physical resource consumption. Logical metrics are common because it can be hard to measure and report consumption of physical resources on a per-service basis. The complexity of measuring and reporting your own physical metrics is also why Service Fabric provides some default metrics.

## Default metrics

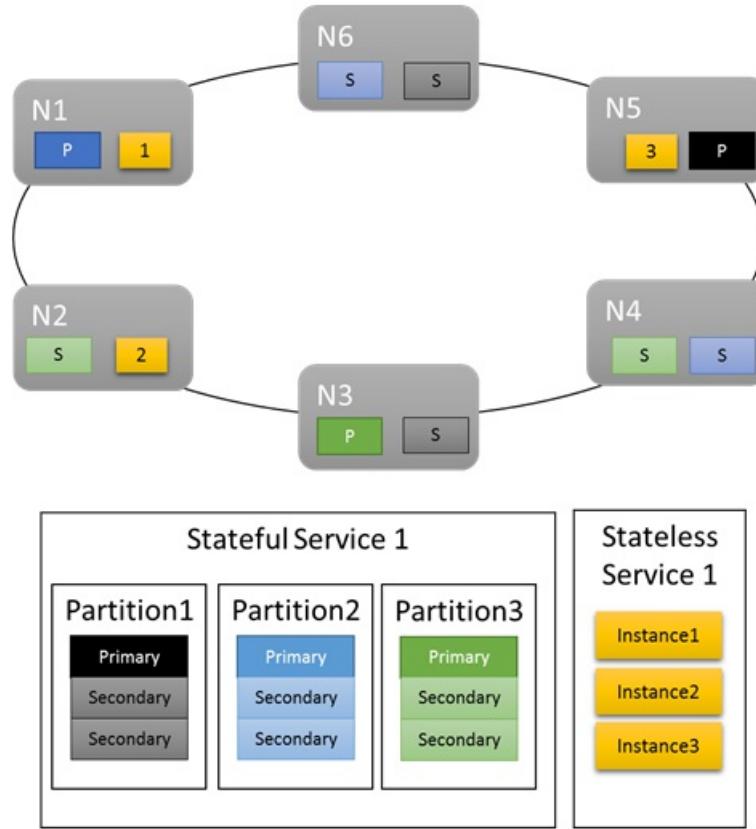
Let's say that you want to get started writing and deploying your service. At this point you don't know what physical or logical resources it consumes. That's fine! The Service Fabric Cluster Resource Manager uses some default metrics when no other metrics are specified. They are:

- PrimaryCount - count of Primary replicas on the node
- ReplicaCount - count of total stateful replicas on the node
- Count - count of all service objects (stateless and stateful) on the node

METRIC	STATELESS INSTANCE LOAD	STATEFUL SECONDARY LOAD	STATEFUL PRIMARY LOAD
PrimaryCount	0	0	1
ReplicaCount	0	1	1
Count	1	1	1

For basic workloads, the default metrics provide a decent distribution of work in the cluster. In the following example, let's see what happens when we create two services and rely on the default metrics for balancing. The first service is a stateful service with three partitions and a target replica set size of three. The second service is a stateless service with one partition and an instance count of three.

Here's what you get:



Some things to note:

- Primary replicas for the stateful service are distributed across several nodes
- Replicas for the same partition are on different nodes
- The total number of primaries and secondaries is distributed in the cluster
- The total number of service objects are evenly allocated on each node

Good!

The default metrics work great as a start. However, the default metrics will only carry you so far. For example: What's the likelihood that the partitioning scheme you picked results in perfectly even utilization by all partitions? What's the chance that the load for a given service is constant over time, or even just the same across multiple partitions right now?

You could run with just the default metrics. However, doing so usually means that your cluster utilization is lower and more uneven than you'd like. This is because the default metrics aren't adaptive and presume everything is equivalent. For example, a Primary that is busy and one that is not both contribute "1" to the PrimaryCount metric. In the worst case, using only the default metrics can also result in overscheduled nodes resulting in performance issues. If you're interested in getting the most out of your cluster and avoiding performance issues, you need to use custom metrics and dynamic load reporting.

## Custom metrics

Metrics are configured on a per-named-service-instance basis when you're creating the service.

Any metric has some properties that describe it: a name, a weight, and a default load.

- **Metric Name:** The name of the metric. The metric name is a unique identifier for the metric within the cluster from the Resource Manager's perspective.
- **Weight:** Metric weight defines how important this metric is relative to the other metrics for this service.
- **Default Load:** The default load is represented differently depending on whether the service is stateless or stateful.
  - For stateless services, each metric has a single property named DefaultLoad

- For stateful services you define:
  - PrimaryDefaultLoad: The default amount of this metric this service consumes when it is a Primary
  - SecondaryDefaultLoad: The default amount of this metric this service consumes when it is a Secondary

#### NOTE

If you define custom metrics and you want to *also* use the default metrics, you need to *explicitly* add the default metrics back and define weights and values for them. This is because you must define the relationship between the default metrics and your custom metrics. For example, maybe you care about ConnectionCount or WorkQueueDepth more than Primary distribution. By default the weight of the PrimaryCount metric is High, so you want to reduce it to Medium when you add your other metrics to ensure they take precedence.

### Defining metrics for your service - an example

Let's say you want the following configuration:

- Your service reports a metric named "ConnectionCount"
- You also want to use the default metrics
- You've done some measurements and know that normally a Primary replica of that service takes up 20 units of "ConnectionCount"
- Secondaries use 5 units of "ConnectionCount"
- You know that "ConnectionCount" is the most important metric in terms of managing the performance of this particular service
- You still want Primary replicas balanced. Balancing primary replicas is generally a good idea no matter what. This helps prevent the loss of some node or fault domain from impacting a majority of primary replicas along with it.
- Otherwise, the default metrics are fine

Here's the code that you would write to create a service with that metric configuration:

Code:

```

StatefulServiceDescription serviceDescription = new StatefulServiceDescription();
StatefulServiceLoadMetricDescription connectionMetric = new StatefulServiceLoadMetricDescription();
connectionMetric.Name = "ConnectionCount";
connectionMetric.PrimaryDefaultLoad = 20;
connectionMetric.SecondaryDefaultLoad = 5;
connectionMetric.Weight = ServiceLoadMetricWeight.High;

StatefulServiceLoadMetricDescription primaryCountMetric = new StatefulServiceLoadMetricDescription();
primaryCountMetric.Name = "PrimaryCount";
primaryCountMetric.PrimaryDefaultLoad = 1;
primaryCountMetric.SecondaryDefaultLoad = 0;
primaryCountMetric.Weight = ServiceLoadMetricWeight.Medium;

StatefulServiceLoadMetricDescription replicaCountMetric = new StatefulServiceLoadMetricDescription();
replicaCountMetric.Name = "ReplicaCount";
replicaCountMetric.PrimaryDefaultLoad = 1;
replicaCountMetric.SecondaryDefaultLoad = 1;
replicaCountMetric.Weight = ServiceLoadMetricWeight.Low;

StatefulServiceLoadMetricDescription totalCountMetric = new StatefulServiceLoadMetricDescription();
totalCountMetric.Name = "Count";
totalCountMetric.PrimaryDefaultLoad = 1;
totalCountMetric.SecondaryDefaultLoad = 1;
totalCountMetric.Weight = ServiceLoadMetricWeight.Low;

serviceDescription.Metrics.Add(connectionMetric);
serviceDescription.Metrics.Add(primaryCountMetric);
serviceDescription.Metrics.Add(replicaCountMetric);
serviceDescription.Metrics.Add(totalCountMetric);

await fabricClient.ServiceManager.CreateServiceAsync(serviceDescription);

```

Powershell:

```

New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName
$serviceName -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -Metric
@("ConnectionCount,High,20,5","PrimaryCount,Medium,1,0","ReplicaCount,Low,1,1","Count,Low,1,1")

```

#### **NOTE**

The above examples and the rest of this document describe managing metrics on a per-named-service basis. It is also possible to define metrics for your services at the service *type* level. This is accomplished by specifying them in your service manifests. Defining type level metrics is not recommended for several reasons. The first reason is that metric names are frequently environment-specific. Unless there is a firm contract in place, you cannot be sure that the metric "Cores" in one environment isn't "MiliCores" or "CoReS" in others. If your metrics are defined in your manifest you need to create new manifests per environment. This usually leads to a proliferation of different manifests with only minor differences, which can lead to management difficulties.

Metric loads are commonly assigned on a per-named-service-instance basis. For example, let's say you create one instance of the service for CustomerA who plans to use it only lightly. Let's also say you create another for CustomerB who has a larger workload. In this case, you'd probably want to tweak the default loads for those services. If you have metrics and loads defined via manifests and you want to support this scenario, it requires different application and service types for each customer. The values defined at service creation time override those defined in the manifest, so you could use that to set the specific defaults. However, doing that causes the values declared in the manifests to not match those the service actually runs with. This can lead to confusion.

As a reminder: if you just want to use the default metrics, you don't need to touch the metrics collection at all or do anything special when creating your service. The default metrics get used automatically when no others are defined.

Now, let's go through each of these settings in more detail and talk about the behavior that it influences.

# Load

The whole point of defining metrics is to represent some load. *Load* is how much of a given metric is consumed by some service instance or replica on a given node. Load can be configured at almost any point. For example:

- Load can be defined when a service is created. This is called *default load*.
- The metric information, including default loads, for a service can be updated after the service is created. This is called *updating a service*.
- The loads for a given partition can be reset to the default values for that service. This is called *resetting partition load*.
- Load can be reported on a per service object basis dynamically during runtime. This is called *reporting load*.

All of these strategies can be used within the same service over its lifetime.

## Default load

*Default load* is how much of the metric each service object (stateless instance or stateful replica) of this service consumes. The Cluster Resource Manager uses this number for the load of the service object until it receives other information, such as a dynamic load report. For simpler services, the default load is a static definition. The default load is never updated and is used for the lifetime of the service. Default loads work great for simple capacity planning scenarios where certain amounts of resources are dedicated to different workloads and do not change.

### NOTE

For more information on capacity management and defining capacities for the nodes in your cluster, please see [this article](#).

The Cluster Resource Manager allows stateful services to specify a different default load for their Primaries and Secondaries. Stateless services can only specify one value that applies to all instances. For stateful services, the default load for Primary and Secondary replicas are typically different since replicas do different kinds of work in each role. For example, Primaries usually serve both reads and writes, and handle most of the computational burden, while secondaries do not. Usually the default load for a primary replica is higher than the default load for secondary replicas. The real numbers should depend on your own measurements.

## Dynamic load

Let's say that you've been running your service for a while. With some monitoring, you've noticed that:

1. Some partitions or instances of a given service consume more resources than others
2. Some services have load that varies over time.

There's lots of things that could cause these types of load fluctuations. For example, different services or partitions are associated with different customers with different requirements. Load could also change because the amount of work the service does varies over the course of the day. Regardless of the reason, there's usually no single number that you can use for default. This is especially true if you want to get the most utilization out of the cluster. Any value you pick for default load is wrong some of the time. Incorrect default loads result in the Cluster Resource Manager either over or under allocating resources. As a result, you have nodes that are over or under utilized even though the Cluster Resource Manager thinks the cluster is balanced. Default loads are still good since they provide some information for initial placement, but they're not a complete story for real workloads. To accurately capture changing resource requirements, the Cluster Resource Manager allows each service object to update its own load during runtime. This is called dynamic load reporting.

Dynamic load reports allow replicas or instances to adjust their allocation/reported load of metrics over their lifetime. A service replica or instance that was cold and not doing any work would usually report that it was using low amounts of a given metric. A busy replica or instance would report that they are using more.

Reporting load per replica or instance allows the Cluster Resource Manager to reorganize the individual service objects in the cluster. Reorganizing the services helps ensure that they get the resources they require. Busy services effectively get to "reclaim" resources from other replicas or instances that are currently cold or doing less work.

Within Reliable Services, the code to report load dynamically looks like this:

Code:

```
this.Partition.ReportLoad(new List<LoadMetric> { new LoadMetric("CurrentConnectionCount", 1234), new LoadMetric("metric1", 42) });
```

A service can report on any of the metrics defined for it at creation time. If a service reports load for a metric that it is not configured to use, Service Fabric ignores that report. If there are other metrics reported at the same time that are valid, those reports are accepted. Service code can measure and report all the metrics it knows how to, and operators can specify the metric configuration to use without having to change the service code.

### Updating a service's metric configuration

The list of metrics associated with the service, and the properties of those metrics can be updated dynamically while the service is live. This allows for experimentation and flexibility. Some examples of when this is useful are:

- disabling a metric with a buggy report for a particular service
- reconfiguring the weights of metrics based on desired behavior
- enabling a new metric only after the code has already been deployed and validated via other mechanisms
- changing the default load for a service based on observed behavior and consumption

The main APIs for changing metric configuration are `FabricClient.ServiceManagementClient.UpdateServiceAsync` in C# and `Update-ServiceFabricService` in PowerShell. Whatever information you specify with these APIs replaces the existing metric information for the service immediately.

## Mixing default load values and dynamic load reports

Default load and dynamic loads can be used for the same service. When a service utilizes both default load and dynamic load reports, default load serves as an estimate until dynamic reports show up. Default load is good because it gives the Cluster Resource Manager something to work with. The default load allows the Cluster Resource Manager to place the service objects in good locations when they are created. If no default load information is provided, placement of services is effectively random. When load reports arrive later the initial random placement is often wrong and the Cluster Resource Manager has to move services.

Let's take our previous example and see what happens when we add some custom metrics and dynamic load reporting. In this example, we use "MemoryInMb" as an example metric.

#### NOTE

Memory is one of the system metrics that Service Fabric can [resource govern](#), and reporting it yourself is typically difficult. We don't actually expect you to report on Memory consumption; Memory is used here as an aid to learning about the capabilities of the Cluster Resource Manager.

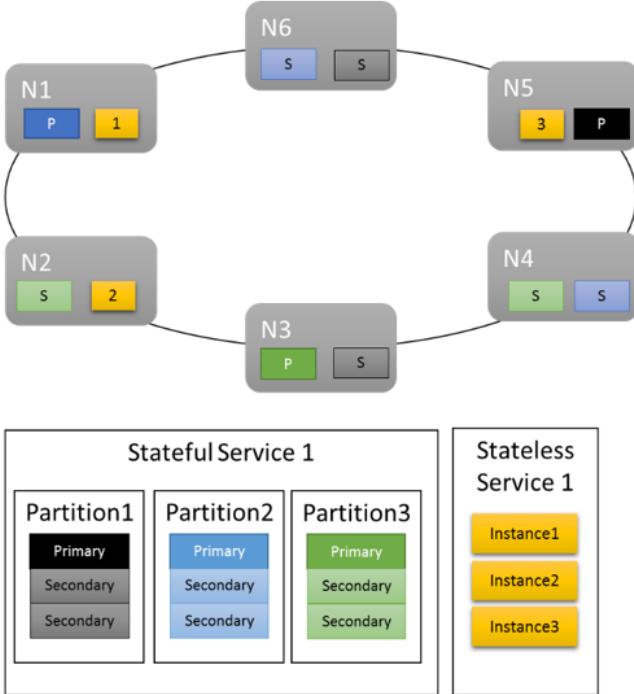
Let's presume that we initially created the stateful service with the following command:

Powershell:

```
New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName $serviceTypeName -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -Metric @("MemoryInMb,High,21,11","PrimaryCount,Medium,1,0","ReplicaCount,Low,1,1","Count,Low,1,1")
```

As a reminder, this syntax is ("MetricName, MetricWeight, PrimaryDefaultLoad, SecondaryDefaultLoad").

Let's see what one possible cluster layout could look like:



Some things that are worth noting:

- Secondary replicas within a partition can each have their own load
- Overall the metrics look balanced. For Memory, the ratio between the maximum and minimum load is 1.75 (the node with the most load is N3, the least is N2, and  $28/16 = 1.75$ ).

There are some things that we still need to explain:

- What determined whether a ratio of 1.75 was reasonable or not? How does the Cluster Resource Manager know if that's good enough or if there is more work to do?
- When does balancing happen?
- What does it mean that Memory was weighted "High"?

## Metric weights

Tracking the same metrics across different services is important. That global view is what allows the Cluster Resource Manager to track consumption in the cluster, balance consumption across nodes, and ensure that nodes don't go over capacity. However, services may have different views as to the importance of the same metric. Also, in a cluster with many metrics and lots of services, perfectly balanced solutions may not exist for all metrics. How should the Cluster Resource Manager handle these situations?

Metric weights allow the Cluster Resource Manager to decide how to balance the cluster when there's no perfect answer. Metric weights also allow the Cluster Resource Manager to balance specific services differently. Metrics can have four different weight levels: Zero, Low, Medium, and High. A metric with a weight of Zero contributes nothing when considering whether things are balanced or not. However, its load does still contribute to capacity management. Metrics with Zero weight are still useful and are frequently used as a part of service behavior and performance monitoring. [This article](#) provides more information on the use of metrics for monitoring and diagnostics of your services.

The real impact of different metric weights in the cluster is that the Cluster Resource Manager generates different solutions. Metric weights tell the Cluster Resource Manager that certain metrics are more important than others. When there's no perfect solution the Cluster Resource Manager can prefer solutions which balance the higher weighted metrics better. If a service thinks a particular metric is unimportant, it may find their use of that metric imbalanced. This allows another service to get an even distribution of some metric that is important to it.

Let's look at an example of some load reports and how different metric weights results in different allocations in the cluster. In this example, we see that switching the relative weight of the metrics causes the Cluster Resource Manager to create different arrangements of services.

Services & Load	Metric Weight Selection	Placement & Total Load	Results
 	MetricA Weight = "HIGH" MetricB Weight = "LOW"	 <b>MetricA = 16 MetricB = 40</b>	<u>MetricA more balanced</u> RatioA = 16/14 = 1.067 RatioB = 40/20 = 2
 	MetricA Weight = "LOW" MetricB Weight = "HIGH"	 <b>MetricA = 20 MetricB = 30</b>	<u>MetricB more balanced</u> RatioA = 20/10 = 2 RatioB = 30/30 = 1

In this example, there are four different services, all reporting different values for two different metrics, MetricA and MetricB. In one case, all the services define MetricA as the most important one (Weight = High) and MetricB as unimportant (Weight = Low). As a result, we see that the Cluster Resource Manager places the services so that MetricA is better balanced than MetricB. "Better balanced" means that MetricA has a lower standard deviation than MetricB. In the second case, we reverse the metric weights. As a result, the Cluster Resource Manager swaps services A and B to come up with an allocation where MetricB is better balanced than MetricA.

#### NOTE

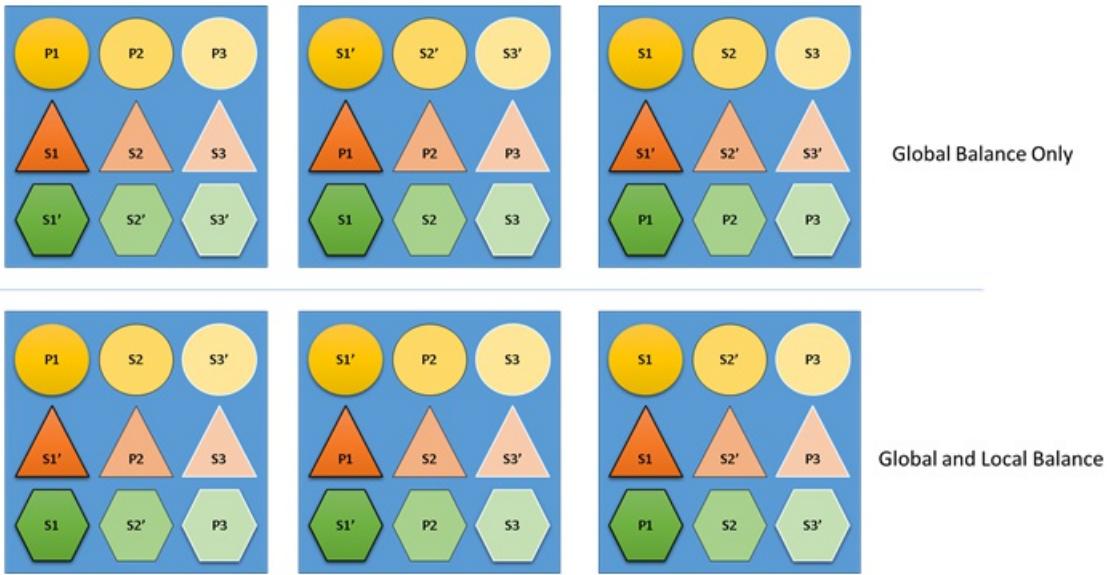
Metric weights determine how the Cluster Resource Manager should balance, but not when balancing should happen. For more information on balancing, check out [this article](#)

### Global metric weights

Let's say ServiceA defines MetricA as weight High, and ServiceB sets the weight for MetricA to Low or Zero. What's the actual weight that ends up getting used?

There are multiple weights that are tracked for every metric. The first weight is the one defined for the metric when the service is created. The other weight is a global weight, which is computed automatically. The Cluster Resource Manager uses both these weights when scoring solutions. Taking both weights into account is important. This allows the Cluster Resource Manager to balance each service according to its own priorities, and also ensure that the cluster as a whole is allocated correctly.

What would happen if the Cluster Resource Manager didn't care about both global and local balance? Well, it's easy to construct solutions that are globally balanced, but which result in poor resource balance for individual services. In the following example, let's look at a service configured with just the default metrics, and see what happens when only global balance is considered:



In the top example based only on global balance, the cluster as a whole is indeed balanced. All nodes have the same count of primaries and the same number total replicas. However, if you look at the actual impact of this allocation it's not so good: the loss of any node impacts a particular workload disproportionately, because it takes out all of its primaries. For example, if the first node fails the three primaries for the three different partitions of the Circle service would all be lost. Conversely, the Triangle and Hexagon services have their partitions lose a replica. This causes no disruption, other than having to recover the down replica.

In the bottom example, the Cluster Resource Manager has distributed the replicas based on both the global and per-service balance. When calculating the score of the solution it gives most of the weight to the global solution, and a (configurable) portion to individual services. Global balance for a metric is calculated based on the average of the metric weights from each service. Each service is balanced according to its own defined metric weights. This ensures that the services are balanced within themselves according to their own needs. As a result, if the same first node fails the failure is distributed across all partitions of all services. The impact to each is the same.

## Next steps

- For more information on configuring services, [Learn about configuring Services](#)(service-fabric-cluster-resource-manager-configure-services.md)
- Defining Defragmentation Metrics is one way to consolidate load on nodes instead of spreading it out. To learn how to configure defragmentation, refer to [this article](#)
- To find out about how the Cluster Resource Manager manages and balances load in the cluster, check out the article on [balancing load](#)
- Start from the beginning and [get an Introduction to the Service Fabric Cluster Resource Manager](#)
- Movement Cost is one way of signaling to the Cluster Resource Manager that certain services are more expensive to move than others. To learn more about movement cost, refer to [this article](#)

# Configuring and using service affinity in Service Fabric

8/18/2017 • 5 min to read • [Edit Online](#)

Affinity is a control that is provided mainly to help ease the transition of larger monolithic applications into the cloud and microservices world. It is also used as an optimization for improving the performance of services, although doing so can have side effects.

Let's say you're bringing a larger app, or one that just wasn't designed with microservices in mind, to Service Fabric (or any distributed environment). This type of transition is common. You start by lifting the entire app into the environment, packaging it, and making sure it is running smoothly. Then you start breaking it down into different smaller services that all talk to each other.

Eventually you may find that the application is experiencing some issues. The issues usually fall into one of these categories:

1. Some component X in the monolithic app had an undocumented dependency on component Y, and you just turned those components into separate services. Since these services are now running on different nodes in the cluster, they're broken.
2. These components communicate via (local named pipes | shared memory | files on disk) and they really need to be able to write to a shared local resource for performance reasons right now. That hard dependency gets removed later, maybe.
3. Everything is fine, but it turns out that these two components are actually chatty/performance sensitive. When they moved them into separate services overall application performance tanked or latency increased. As a result, the overall application is not meeting expectations.

In these cases, we don't want to lose our refactoring work, and don't want to go back to the monolith. The last condition may even be desirable as a plain optimization. However, until we can redesign the components to work naturally as services (or until we can solve the performance expectations some other way) we're going to need some sense of locality.

What to do? Well, you could try turning on affinity.

## How to configure affinity

To set up affinity, you define an affinity relationship between two different services. You can think of affinity as "pointing" one service at another and saying "This service can only run where that service is running." Sometimes we refer to affinity as a parent/child relationship (where you point the child at the parent). Affinity ensures that the replicas or instances of one service are placed on the same nodes as those of another service.

```
ServiceCorrelationDescription affinityDescription = new ServiceCorrelationDescription();
affinityDescription.Scheme = ServiceCorrelationScheme.Affinity;
affinityDescription.ServiceName = new Uri("fabric:/otherApplication/parentService");
serviceDescription.Correlations.Add(affinityDescription);
await fabricClient.ServiceManager.CreateServiceAsync(serviceDescription);
```

#### NOTE

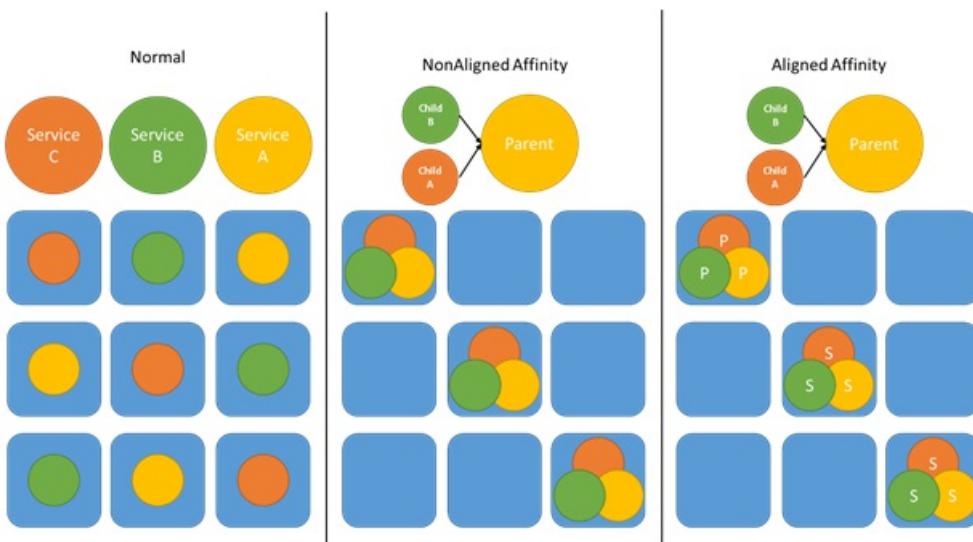
A child service can only participate in a single affinity relationship. If you wanted the child to be affinitized to two parent services at once you have a couple options:

- Reverse the relationships (have parentService1 and parentService2 point at the current child service), or
- Designate one of the parents as a hub by convention and have all services point at that service.

The resulting placement behavior in the cluster should be the same.

## Different affinity options

Affinity is represented via one of several correlation schemes, and has two different modes. The most common mode of affinity is what we call NonAlignedAffinity. In NonAlignedAffinity, the replicas or instances of the different services are placed on the same nodes. The other mode is AlignedAffinity. Aligned Affinity is useful only with stateful services. Configuring two stateful services to have aligned affinity ensures that the primaries of those services are placed on the same nodes as each other. It also causes each pair of secondaries for those services to be placed on the same nodes. It is also possible (though less common) to configure NonAlignedAffinity for stateful services. For NonAlignedAffinity, the different replicas of the two stateful services would run on the same nodes, but their primaries could end up on different nodes.

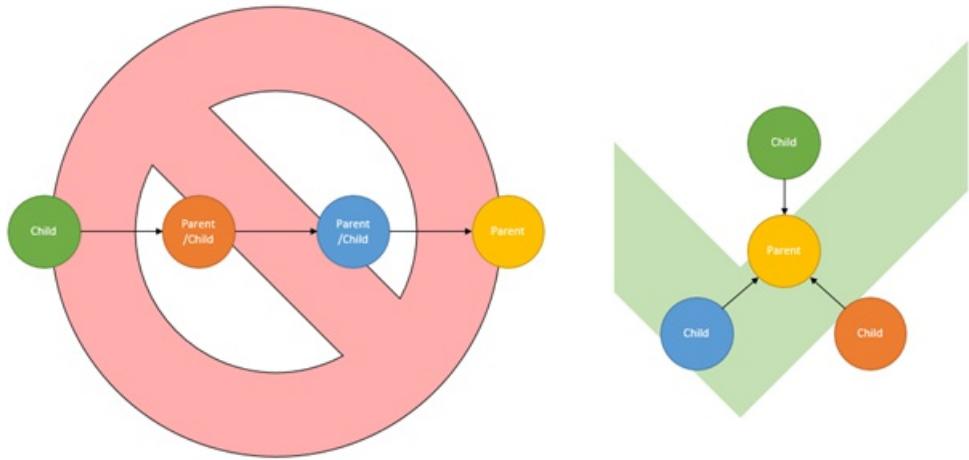


### Best effort desired state

An affinity relationship is best effort. It does not provide the same guarantees of collocation or reliability that running in the same executable process does. The services in an affinity relationship are fundamentally different entities that can fail and be moved independently. An affinity relationship could also break, though these breaks are temporary. For example, capacity limitations may mean that only some of the service objects in the affinity relationship can fit on a given node. In these cases even though there's an affinity relationship in place, it can't be enforced due to the other constraints. If it is possible to do so, the violation is automatically corrected later.

### Chains vs. stars

Today the Cluster Resource Manager isn't able to model chains of affinity relationships. What this means is that a service that is a child in one affinity relationship can't be a parent in another affinity relationship. If you want to model this type of relationship, you effectively have to model it as a star, rather than a chain. To move from a chain to a star, the bottommost child would be parented to the first child's parent instead. Depending on the arrangement of your services, you may have to do this multiple times. If there's no natural parent service, you may have to create one that serves as a placeholder. Depending on your requirements, you may also want to look into [Application Groups](#).



Another thing to note about affinity relationships today is that they are directional. This means that the affinity rule only enforces that the child placed with the parent. It does not ensure that the parent is located with the child. It is also important to note that the affinity relationship can't be perfect or instantly enforced since different services have with different lifecycles and can fail and move independently. For example, let's say the parent suddenly fails over to another node because it crashed. The Cluster Resource Manager and Failover Manager handle the failover first, since keeping the services up, consistent, and available is the priority. Once the failover completes, the affinity relationship is broken, but the Cluster Resource Manager thinks everything is fine until it notices that the child is not located with the parent. These sorts of checks are performed periodically. More information on how the Cluster Resource Manager evaluates constraints is available in [this article](#), and [this one](#) talks more about how to configure the cadence on which these constraints are evaluated.

### **Partitioning support**

The final thing to notice about affinity is that affinity relationships aren't supported where the parent is partitioned. Partitioned parent services may be supported eventually, but today it is not allowed.

## Next steps

- For more information on configuring services, [Learn about configuring Services](#)
- To limit services to a small set of machines or aggregating the load of services, use [Application Groups](#)

# Placement policies for service fabric services

8/18/2017 • 5 min to read • [Edit Online](#)

Placement policies are additional rules that can be used to govern service placement in some specific, less-common scenarios. Some examples of those scenarios are:

- Your Service Fabric cluster spans geographic distances, such as multiple on-premises datacenters or across Azure regions
- Your environment spans multiple areas of geopolitical or legal control, or some other case where you have policy boundaries you need to enforce
- There are communication performance or latency considerations due to large distances or use of slower or less reliable network links
- You need to keep certain workloads collocated as a best effort, either with other workloads or in proximity to customers

Most of these requirements align with the physical layout of the cluster, represented as the fault domains of the cluster.

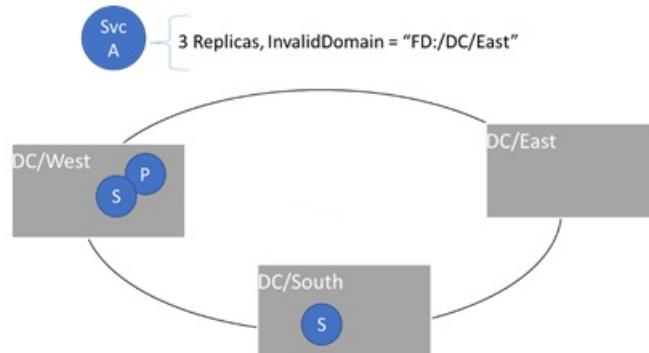
The advanced placement policies that help address these scenarios are:

1. Invalid domains
2. Required domains
3. Preferred domains
4. Disallowing replica packing

Most of the following controls could be configured via node properties and placement constraints, but some are more complicated. To make things simpler, the Service Fabric Cluster Resource Manager provides these additional placement policies. Placement policies are configured on a per-named service instance basis. They can also be updated dynamically.

## Specifying invalid domains

The **InvalidDomain** placement policy allows you to specify that a particular Fault Domain is invalid for a specific service. This policy ensures that a particular service never runs in a particular area, for example for geopolitical or corporate policy reasons. Multiple invalid domains may be specified via separate policies.



Code:

```

ServicePlacementInvalidDomainPolicyDescription invalidDomain = new
ServicePlacementInvalidDomainPolicyDescription();
invalidDomain.DomainName = "fd:/DCEast"; //regulations prohibit this workload here
serviceDescription.PlacementPolicies.Add(invalidDomain);

```

Powershell:

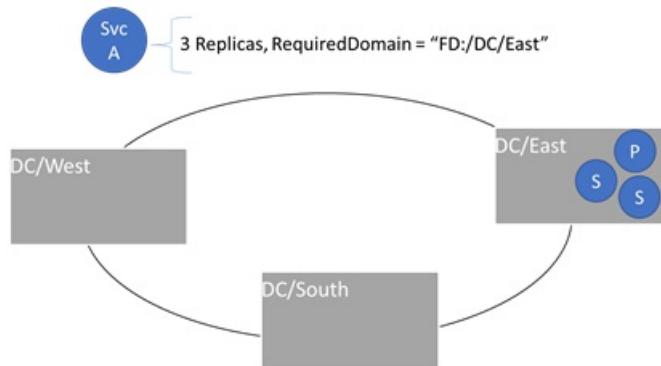
```

New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName
$serviceName -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -
PlacementPolicy @("InvalidDomain,fd:/DCEast")

```

## Specifying required domains

The required domain placement policy requires that the service is present only in the specified domain. Multiple required domains can be specified via separate policies.



Code:

```

ServicePlacementRequiredDomainPolicyDescription requiredDomain = new
ServicePlacementRequiredDomainPolicyDescription();
requiredDomain.DomainName = "fd:/DC01/RK03/BL2";
serviceDescription.PlacementPolicies.Add(requiredDomain);

```

Powershell:

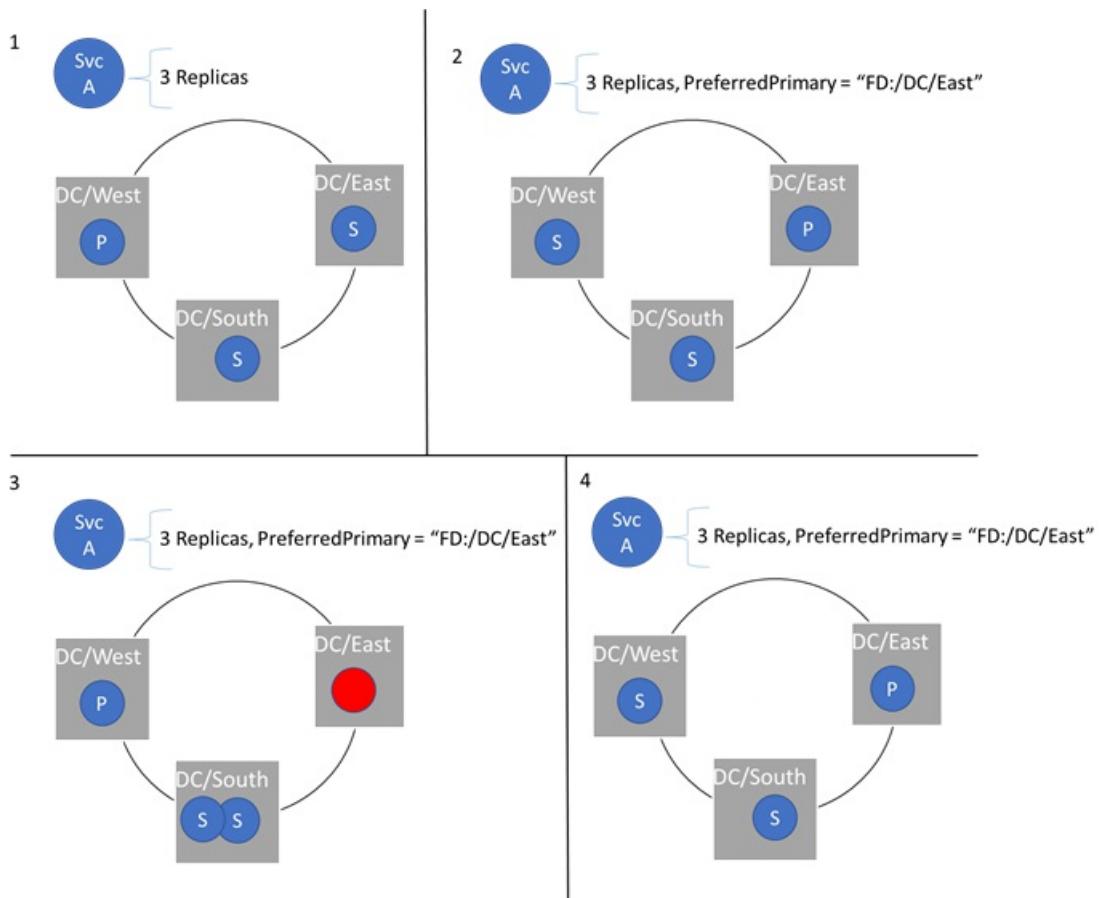
```

New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName
$serviceName -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -
PlacementPolicy @("RequiredDomain,fd:/DC01/RK03/BL2")

```

## Specifying a preferred domain for the primary replicas of a stateful service

The Preferred Primary Domain specifies the fault domain to place the Primary in. The Primary ends up in this domain when everything is healthy. If the domain or the Primary replica fails or shuts down, the Primary moves to some other location, ideally in the same domain. If this new location isn't in the preferred domain, the Cluster Resource Manager moves it back to the preferred domain as soon as possible. Naturally this setting only makes sense for stateful services. This policy is most useful in clusters that are spanned across Azure regions or multiple datacenters but have services that prefer placement in a certain location. Keeping Primaries close to their users or other services helps provide lower latency, especially for reads, which are handled by Primaries by default.



```
ServicePlacementPreferPrimaryDomainPolicyDescription primaryDomain = new
ServicePlacementPreferPrimaryDomainPolicyDescription();
primaryDomain.DomainName = "fd:/EastUS/";
serviceDescription.PlacementPolicies.Add(invalidDomain);
```

Powershell:

```
New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName
$serviceTypeName -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -
PlacementPolicy @("PreferredPrimaryDomain,fd:/EastUS")
```

## Requiring replica distribution and disallowing packing

Replicas are *normally* distributed across fault and upgrade domains when the cluster is healthy. However, there are cases where more than one replica for a given partition may end up temporarily packed into a single domain. For example, let's say that the cluster has nine nodes in three fault domains, fd:/0, fd:/1, and fd:/2. Let's also say that your service has three replicas. Let's say that the nodes that were being used for those replicas in fd:/1 and fd:/2 went down. Normally the Cluster Resource Manager would prefer other nodes in those same fault domains. In this case, let's say due to capacity issues none of the other nodes in those domains were valid. If the Cluster Resource Manager builds replacements for those replicas, it would have to choose nodes in fd:/0. However, doing *that* creates a situation where the Fault Domain constraint is violated. Packing replicas increases the chance that the whole replica set could go down or be lost.

### NOTE

For more information on constraints and constraint priorities generally, check out [this topic](#).

If you've ever seen a health message such as "

```
The Load Balancer has detected a Constraint Violation for this Replica:fabric:/<some service name> Secondary Partition <some partition ID> is violating the Constraint: FaultDomain
```

", then you've hit this condition or something like it. Usually only one or two replicas are packed together temporarily. So long as there are fewer than a quorum of replicas in a given domain, you're safe. Packing is rare, but it can happen, and usually these situations are transient since the nodes come back. If the nodes do stay down and the Cluster Resource Manager needs to build replacements, usually there are other nodes available in the ideal fault domains.

Some workloads would prefer always having the target number of replicas, even if they are packed into fewer domains. These workloads are betting against total simultaneous permanent domain failures and can usually recover local state. Other workloads would rather take the downtime earlier than risk correctness or loss of data. Most production workloads run with more than three replicas, more than three fault domains, and many valid nodes per fault domain. Because of this, the default behavior allows domain packing by default. The default behavior allows normal balancing and failover to handle these extreme cases, even if that means temporary domain packing.

If you want to disable such packing for a given workload, you can specify the `RequireDomainDistribution` policy on the service. When this policy is set, the Cluster Resource Manager ensures no two replicas from the same partition run in the same fault or upgrade domain.

Code:

```
ServicePlacementRequireDomainDistributionPolicyDescription distributeDomain = new ServicePlacementRequireDomainDistributionPolicyDescription(); serviceDescription.PlacementPolicies.Add(distributeDomain);
```

Powershell:

```
New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName $serviceTypeName -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -PlacementPolicy @("RequiredDomainDistribution")
```

Now, would it be possible to use these configurations for services in a cluster that was not geographically spanned? You could, but there's not a great reason too. The required, invalid, and preferred domain configurations should be avoided unless the scenarios require them. It doesn't make any sense to try to force a given workload to run in a single rack, or to prefer some segment of your local cluster over another. Different hardware configurations should be spread across fault domains and handled via normal placement constraints and node properties.

## Next steps

- For more information on configuring services, [Learn about configuring Services](#)

# Cluster resource manager integration with Service Fabric cluster management

8/18/2017 • 11 min to read • [Edit Online](#)

The Service Fabric Cluster Resource Manager doesn't drive upgrades in Service Fabric, but it is involved. The first way that the Cluster Resource Manager helps with management is by tracking the desired state of the cluster and the services inside it. The Cluster Resource Manager sends out health reports when it cannot put the cluster into the desired configuration. For example, if there is insufficient capacity the Cluster Resource Manager sends out health warnings and errors indicating the problem. Another piece of integration has to do with how upgrades work. The Cluster Resource Manager alters its behavior slightly during upgrades.

## Health integration

The Cluster Resource Manager constantly tracks the rules you have defined for placing your services. It also tracks the remaining capacity for each metric on the nodes and in the cluster and in the cluster as a whole. If it can't satisfy those rules or if there is insufficient capacity, health warnings and errors are emitted. For example, if a node is over capacity and the Cluster Resource Manager will try to fix the situation by moving services. If it can't correct the situation it emits a health warning indicating which node is over capacity, and for which metrics.

Another example of the Resource Manager's health warnings is violations of placement constraints. For example, if you have defined a placement constraint (such as `"NodeColor == Blue"`) and the Resource Manager detects a violation of that constraint, it emits a health warning. This is true for custom constraints and the default constraints (like the Fault Domain and Upgrade Domain constraints).

Here's an example of one such health report. In this case, the health report is for one of the system service's partitions. The health message indicates the replicas of that partition are temporarily packed into too few Upgrade Domains.

```

PS C:\Users\User > Get-WindowsFabricPartitionHealth -PartitionId '00000000-0000-0000-0000-000000000001'

PartitionId          : 00000000-0000-0000-0000-000000000001
AggregatedHealthState : Warning
UnhealthyEvaluations :
    Unhealthy event: SourceId='System.PLB',
Property='ReplicaConstraintViolation_UpgradeDomain', HealthState='Warning', ConsiderWarningAsError=false.

ReplicaHealthStates   :
    ReplicaId          : 130766528804733380
    AggregatedHealthState : Ok

    ReplicaId          : 130766528804577821
    AggregatedHealthState : Ok

    ReplicaId          : 130766528854889931
    AggregatedHealthState : Ok

    ReplicaId          : 130766528804577822
    AggregatedHealthState : Ok

    ReplicaId          : 130837073190680024
    AggregatedHealthState : Ok

HealthEvents          :
    SourceId           : System.PLB
    Property            : ReplicaConstraintViolation_UpgradeDomain
    HealthState         : Warning
    SequenceNumber      : 130837100116930204
    SentAt              : 8/10/2015 7:53:31 PM
    ReceivedAt          : 8/10/2015 7:53:33 PM
    TTL                 : 00:01:05
    Description          : The Load Balancer has detected a Constraint Violation for this
Replica: fabric:/System/FailoverManagerService Secondary Partition 00000000-0000-0000-000000000001 is
          violating the Constraint: UpgradeDomain Details: UpgradeDomain ID -- 4, Replica on
nodeName -- Node.8 Currently Upgrading -- false Distribution Policy -- Packing
    RemoveWhenExpired   : True
    IsExpired           : False
    Transitions          : Ok->Warning = 8/10/2015 7:13:02 PM, LastError = 1/1/0001
12:00:00 AM

```

Here's what this health message is telling us is:

1. All the replicas themselves are healthy: Each has AggregatedHealthState : Ok
2. The Upgrade Domain distribution constraint is currently being violated. This means a particular Upgrade Domain has more replicas from this partition than it should.
3. Which node contains the replica causing the violation. In this case it's the node with the name "Node.8"
4. Whether an upgrade is currently happening for this partition ("Currently Upgrading -- false")
5. The distribution policy for this service: "Distribution Policy -- Packing". This is governed by the [RequireDomainDistribution](#) [placement policy](#). "Packing" indicates that in this case DomainDistribution was *not* required, so we know that placement policy was not specified for this service.
6. When the report happened - 8/10/2015 7:13:02 PM

Information like this powers alerts that fire in production to let you know something has gone wrong and is also used to detect and halt bad upgrades. In this case, we'd want to see if we can figure out why the Resource Manager had to pack the replicas into the Upgrade Domain. Usually packing is transient because the nodes in the other Upgrade Domains were down, for example.

Let's say the Cluster Resource Manager is trying to place some services, but there aren't any solutions that work. When services can't be placed, it is usually for one of the following reasons:

1. Some transient condition has made it impossible to place this service instance or replica correctly
2. The service's placement requirements are unsatisfiable.

In these cases, health reports from the Cluster Resource Manager help you determine why the service can't be placed. We call this process the constraint elimination sequence. During it, the system walks through the configured constraints affecting the service and records what they eliminate. This way when services aren't able to be placed, you can see which nodes were eliminated and why.

## Constraint types

Let's talk about each of the different constraints in these health reports. You will see health messages related to these constraints when replicas can't be placed.

- **ReplicaExclusionStatic** and **ReplicaExclusionDynamic**: These constraints indicates that a solution was rejected because two service objects from the same partition would have to be placed on the same node. This isn't allowed because then failure of that node would overly impact that partition. ReplicaExclusionStatic and ReplicaExclusionDynamic are almost the same rule and the differences don't really matter. If you are seeing a constraint elimination sequence containing either the ReplicaExclusionStatic or ReplicaExclusionDynamic constraint, the Cluster Resource Manager thinks that there aren't enough nodes. This requires remaining solutions to use these invalid placements which are disallowed. The other constraints in the sequence will usually tell us why nodes are being eliminated in the first place.
- **PlacementConstraint**: If you see this message, it means that we eliminated some nodes because they didn't match the service's placement constraints. We trace out the currently configured placement constraints as a part of this message. This is normal if you have a placement constraint defined. However, if placement constraint is incorrectly causing too many nodes to be eliminated this is how you would notice.
- **NodeCapacity**: This constraint means that the Cluster Resource Manager couldn't place the replicas on the indicated nodes because that would put them over capacity.
- **Affinity**: This constraint indicates that we couldn't place the replica on the affected nodes since it would cause a violation of the affinity constraint. More information on affinity is in [this article](#)
- **FaultDomain** and **UpgradeDomain**: This constraint eliminates nodes if placing the replica on the indicated nodes would cause packing in a particular fault or upgrade domain. Several examples discussing this constraint are presented in the topic on [fault and upgrade domain constraints and resulting behavior](#)
- **PreferredLocation**: You shouldn't normally see this constraint removing nodes from the solution since it runs as an optimization by default. The preferred location constraint is also present during upgrades. During upgrade it is used to move services back to where they were when the upgrade started.

## Blocklisting Nodes

Another health message the Cluster Resource Manager reports is when nodes are blocklisted. You can think of blocklisting as a temporary constraint that is automatically applied for you. Nodes get blocklisted when they experience repeated failures when launching instances of that service type. Nodes are blocklisted on a per-service-type basis. A node may be blocklisted for one service type but not another.

You'll see blocklisting kick in often during development: some bug causes your service host to crash on startup. Service Fabric tries to create the service host a few times, and the failure keeps occurring. After a few attempts, the node gets blocklisted, and the Cluster Resource Manager will try to create the service elsewhere. If that failure keeps happening on multiple nodes, it's possible that all of the valid nodes in the cluster end up blocked. Blocklisting can also remove so many nodes that not enough can successfully launch the service to meet the desired scale. You'll typically see additional errors or warnings from the Cluster Resource Manager indicating that the service is below the desired replica or instance count, as well as health messages indicating what the failure is that's leading to the blocklisting in the first place.

Blocklisting is not a permanent condition. After a few minutes, the node is removed from the blocklist and Service

Fabric may activate the services on that node again. If services continue to fail, the node is blacklisted for that service type again.

## Constraint priorities

### WARNING

Changing constraint priorities is not recommended and may have significant adverse effects on your cluster. The below information is provided for reference of the default constraint priorities and their behavior.

With all of these constraints, you may have been thinking "Hey – I think that fault domain constraints are the most important thing in my system. In order to ensure the fault domain constraint isn't violated, I'm willing to violate other constraints."

Constraints can be configured with different priority levels. These are:

- "hard" (0)
- "soft" (1)
- "optimization" (2)
- "off" (-1).

Most of the constraints are configured as hard constraints by default.

Changing the priority of constraints is uncommon. There have been times where constraint priorities needed to change, usually to work around some other bug or behavior that was impacting the environment. Generally the flexibility of the constraint priority infrastructure has worked very well, but it isn't needed often. Most of the time everything sits at their default priorities.

The priority levels don't mean that a given constraint *will* be violated, nor that it will always be met. Constraint priorities define an order in which constraints are enforced. Priorities define the tradeoffs when it is impossible to satisfy all constraints. Usually all the constraints can be satisfied unless there's something else going on in the environment. Some examples of scenarios that will lead to constraint violations are conflicting constraints, or large numbers of concurrent failures.

In advanced situations, you can change the constraint priorities. For example, say you wanted to ensure that affinity would always be violated when necessary to solve node capacity issues. To achieve this, you could set the priority of the affinity constraint to "soft" (1) and leave the capacity constraint set to "hard" (0).

The default priority values for the different constraints are specified in the following config:

ClusterManifest.xml

```
<Section Name="PlacementAndLoadBalancing">
    <Parameter Name="PlacementConstraintPriority" Value="0" />
    <Parameter Name="CapacityConstraintPriority" Value="0" />
    <Parameter Name="AffinityConstraintPriority" Value="0" />
    <Parameter Name="FaultDomainConstraintPriority" Value="0" />
    <Parameter Name="UpgradeDomainConstraintPriority" Value="1" />
    <Parameter Name="PreferredLocationConstraintPriority" Value="2" />
</Section>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters:

```

"fabricSettings": [
  {
    "name": "PlacementAndLoadBalancing",
    "parameters": [
      {
        "name": "PlacementConstraintPriority",
        "value": "0"
      },
      {
        "name": "CapacityConstraintPriority",
        "value": "0"
      },
      {
        "name": "AffinityConstraintPriority",
        "value": "0"
      },
      {
        "name": "FaultDomainConstraintPriority",
        "value": "0"
      },
      {
        "name": "UpgradeDomainConstraintPriority",
        "value": "1"
      },
      {
        "name": "PreferredLocationConstraintPriority",
        "value": "2"
      }
    ]
  }
]

```

## Fault domain and upgrade domain constraints

The Cluster Resource Manager wants to keep services spread out among fault and upgrade domains. It models this as a constraint inside the Cluster Resource Manager's engine. For more information on how they are used and their specific behavior, check out the article on [cluster configuration](#).

The Cluster Resource Manager may need to pack a couple replicas into an upgrade domain in order to deal with upgrades, failures, or other constraint violations. Packing into fault or upgrade domains normally happens only when there are several failures or other churn in the system preventing correct placement. If you wish to prevent packing even during these situations, you can utilize the [RequireDomainDistribution](#) [placement policy](#). Note that this may affect service availability and reliability as a side effect, so consider it carefully.

If the environment is configured correctly, all constraints are fully respected, even during upgrades. The key thing is that the Cluster Resource Manager is watching out for your constraints. When it detects a violation it immediately reports it and tries to correct the issue.

## The preferred location constraint

The PreferredLocation constraint is a little different, as it has two different uses. One use of this constraint is during application upgrades. The Cluster Resource Manager automatically manages this constraint during upgrades. It is used to ensure that when upgrades are complete that replicas return to their initial locations. The other use of the PreferredLocation constraint is for the [PreferredPrimaryDomain](#) [placement policy](#). Both of these are optimizations, and hence the PreferredLocation constraint is the only constraint set to "Optimization" by default.

## Upgrades

The Cluster Resource Manager also helps during application and cluster upgrades, during which it has two jobs:

- ensure that the rules of the cluster are not compromised
- try to help the upgrade go smoothly

### **Keep enforcing the rules**

The main thing to be aware of is that the rules – the strict constraints like placement constraints and capacities - are still enforced during upgrades. Placement constraints ensure that your workloads only run where they are allowed to, even during upgrades. When services are highly constrained, upgrades can take longer. When the service or the node it is running on is brought down for an update there may be few options for where it can go.

### **Smart replacements**

When an upgrade starts, the Resource Manager takes a snapshot of the current arrangement of the cluster. As each Upgrade Domain completes, it attempts to return the services that were in that Upgrade Domain to their original arrangement. This way there are at most two transitions for a service during the upgrade. There is one move out of the affected node and one move back in. Returning the cluster or service to how it was before the upgrade also ensures the upgrade doesn't impact the layout of the cluster.

### **Reduced churn**

Another thing that happens during upgrades is that the Cluster Resource Manager turns off balancing. Preventing balancing prevents unnecessary reactions to the upgrade itself, like moving services into nodes that were emptied for the upgrade. If the upgrade in question is a Cluster upgrade, then the entire cluster is not balanced during the upgrade. Constraint checks stay active, only movement based on the proactive balancing of metrics is disabled.

### **Buffered Capacity & Upgrade**

Generally you want the upgrade to complete even if the cluster is constrained or close to full. Managing the capacity of the cluster is even more important during upgrades than usual. Depending on the number of upgrade domains, between 5 and 20 percent of capacity must be migrated as the upgrade rolls through the cluster. That work has to go somewhere. This is where the notion of [buffered capacities](#) is useful. Buffered capacity is respected during normal operation. The Cluster Resource Manager may fill nodes up to their total capacity (consuming the buffer) during upgrades if necessary.

## **Next steps**

- Start from the beginning and [get an Introduction to the Service Fabric Cluster Resource Manager](#)

# Defragmentation of metrics and load in Service Fabric

8/18/2017 • 4 min to read • [Edit Online](#)

The Service Fabric Cluster Resource Manager's default strategy for managing load metrics in the cluster is to distribute the load. Ensuring that nodes are evenly utilized avoids hot and cold spots that lead to both contention and wasted resources. Distributing workloads in the cluster is also the safest in terms of surviving failures since it ensures that a failure doesn't take out a large percentage of a given workload.

The Service Fabric Cluster Resource Manager does support a different strategy for managing load, which is defragmentation. Defragmentation means that instead of trying to distribute the utilization of a metric across the cluster, it is consolidated. Consolidation is just an inversion of the default balancing strategy – instead of minimizing the average standard deviation of metric load, the Cluster Resource Manager tries to increase it.

## When to use defragmentation

Distributing load in the cluster consumes some of the resources on each node. Some workloads create services that are exceptionally large and consume most or all of a node. In these cases, it's possible that when there are large workloads getting created that there isn't enough space on any node to run them. Large workloads aren't a problem in Service Fabric; in these cases the Cluster Resource Manager determines that it needs to reorganize the cluster to make room for this large workload. However, in the meantime that workload has to wait to be scheduled in the cluster.

If there are many services and state to move around, then it could take a long time for the large workload to be placed in the cluster. This is more likely if other workloads in the cluster are also large and so take longer to reorganize. The Service Fabric team measured creation times in simulations of this scenario. We found that creating large services took much longer as soon as cluster utilization got above between 30% and 50%. To handle this scenario, we introduced defragmentation as a balancing strategy. We found that for large workloads, especially ones where creation time was important, defragmentation really helped those new workloads get scheduled in the cluster.

You can configure defragmentation metrics to have the Cluster Resource Manager to proactively try to condense the load of the services into fewer nodes. This helps ensure that there is almost always room for large services without reorganizing the cluster. Not having to reorganize the cluster allows creating large workloads quickly.

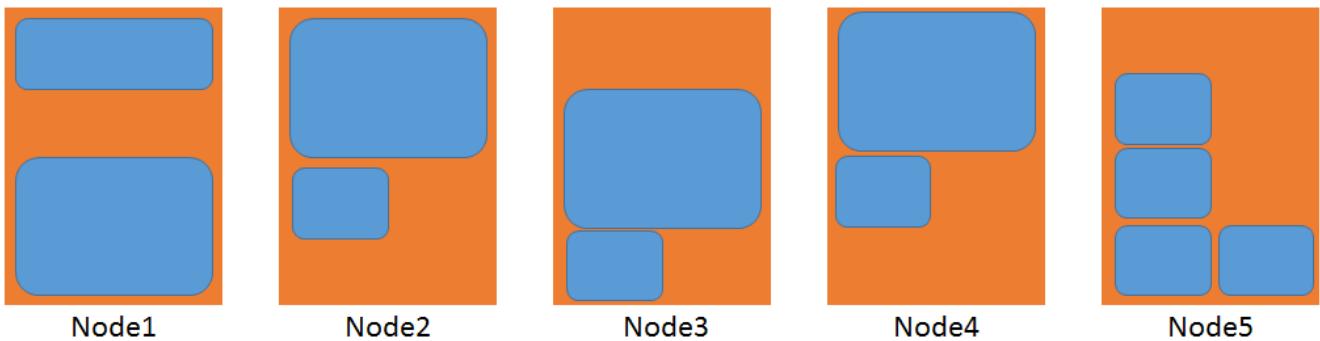
Most people don't need defragmentation. Services are usually small, so it's not hard to find room for them in the cluster. When reorganization is possible, it goes quickly, again because most services are small and can be moved quickly and in parallel. However, if you have large services and need them created quickly then the defragmentation strategy is for you. We'll discuss the tradeoffs of using defragmentation next.

## Defragmentation tradeoffs

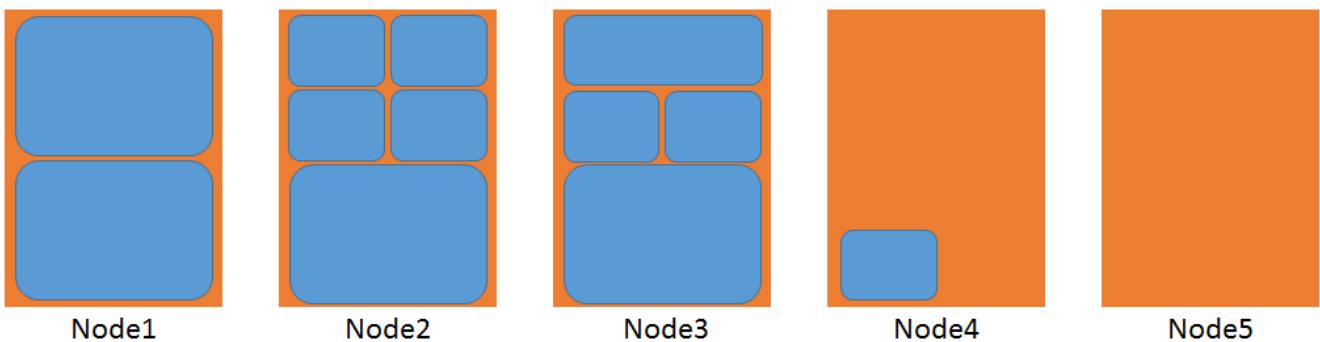
Defragmentation can increase impactfulness of failures, since more services are running on nodes that fail. Defragmentation can also increase costs, since resources in the cluster must be held in reserve, waiting for the creation of large workloads.

The following diagram gives a visual representation of two clusters, one that is defragmented and one that is not.

# Balanced Cluster



# Defragmented Cluster



In the balanced case, consider the number of movements that would be necessary to place one of the largest service objects. In the defragmented cluster, the large workload could be placed on nodes four or five without having to wait for any other services to move.

## Defragmentation pros and cons

So what are those other conceptual tradeoffs? Here's a quick table of things to think about:

DEFRAAGMENTATION PROS	DEFRAAGMENTATION CONS
Allows faster creation of large services	Concentrates load onto fewer nodes, increasing contention
Enables lower data movement during creation	Failures can impact more services and cause more churn
Allows rich description of requirements and reclamation of space	More complex overall Resource Management configuration

You can mix defragmented and normal metrics in the same cluster. The Cluster Resource Manager tries to consolidate the defragmentation metrics as much as possible while spreading out the others. The results of mixing defragmentation and balancing strategies depends on several factors, including:

- the number of balancing metrics vs. the number of defragmentation metrics
- Whether any service uses both types of metrics
- the metric weights
- current metric loads

Experimentation is required to determine the exact configuration necessary. We recommend thorough measurement of your workloads before you enable defragmentation metrics in production. This is especially true when mixing defragmentation and balanced metrics within the same service.

## Configuring defragmentation metrics

Configuring defragmentation metrics is a global decision in the cluster, and individual metrics can be selected for defragmentation. The following config snippets show how to configure metrics for defragmentation. In this case, "Metric1" is configured as a defragmentation metric, while "Metric2" will continue to be balanced normally.

ClusterManifest.xml:

```
<Section Name="DefragmentationMetrics">
    <Parameter Name="Metric1" Value="true" />
    <Parameter Name="Metric2" Value="false" />
</Section>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters:

```
"fabricSettings": [
{
    "name": "DefragmentationMetrics",
    "parameters": [
        {
            "name": "Metric1",
            "value": "true"
        },
        {
            "name": "Metric2",
            "value": "false"
        }
    ]
}]
```

## Next steps

- The Cluster Resource Manager has man options for describing the cluster. To find out more about them, check out this article on [describing a Service Fabric cluster](#)
- Metrics are how the Service Fabric Cluster Resource Manger manages consumption and capacity in the cluster. To learn more about metrics and how to configure them, check out [this article](#)

# Balancing your service fabric cluster

9/5/2017 • 8 min to read • [Edit Online](#)

The Service Fabric Cluster Resource Manager supports dynamic load changes, reacting to additions or removals of nodes or services. It also automatically corrects constraint violations, and proactively rebalances the cluster. But how often are these actions taken, and what triggers them?

There are three different categories of work that the Cluster Resource Manager performs. They are:

1. Placement – this stage deals with placing any stateful replicas or stateless instances that are missing. Placement includes both new services and handling stateful replicas or stateless instances that have failed. Deleting and dropping replicas or instances are handled here.
2. Constraint Checks – this stage checks for and corrects violations of the different placement constraints (rules) within the system. Examples of rules are things like ensuring that nodes are not over capacity and that a service's placement constraints are met.
3. Balancing – this stage checks to see if rebalancing is necessary based on the configured desired level of balance for different metrics. If so it attempts to find an arrangement in the cluster that is more balanced.

## Configuring Cluster Resource Manager Timers

The first set of controls around balancing are a set of timers. These timers govern how often the Cluster Resource Manager examines the cluster and takes corrective actions.

Each of these different types of corrections the Cluster Resource Manager can make is controlled by a different timer that governs its frequency. When each timer fires, the task is scheduled. By default the Resource Manager:

- scans its state and applies updates (like recording that a node is down) every 1/10th of a second
- sets the placement check flag
- sets the constraint check flag every second
- sets the balancing flag every five seconds.

Examples of the configuration governing these timers are below:

ClusterManifest.xml:

```
<Section Name="PlacementAndLoadBalancing">
  <Parameter Name="PLBRefreshGap" Value="0.1" />
  <Parameter Name="MinPlacementInterval" Value="1.0" />
  <Parameter Name="MinConstraintCheckInterval" Value="1.0" />
  <Parameter Name="MinLoadBalancingInterval" Value="5.0" />
</Section>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters:

```

"fabricSettings": [
  {
    "name": "PlacementAndLoadBalancing",
    "parameters": [
      {
        "name": "PLBRefreshGap",
        "value": "0.10"
      },
      {
        "name": "MinPlacementInterval",
        "value": "1.0"
      },
      {
        "name": "MinConstraintCheckInterval",
        "value": "1.0"
      },
      {
        "name": "MinLoadBalancingInterval",
        "value": "5.0"
      }
    ]
  }
]

```

Today the Cluster Resource Manager only performs one of these actions at a time, sequentially. This is why we refer to these timers as "minimum intervals" and the actions that get taken when the timers go off as "setting flags". For example, the Cluster Resource Manager takes care of pending requests to create services before balancing the cluster. As you can see by the default time intervals specified, the Cluster Resource Manager scans for anything it needs to do frequently. Normally this means that the set of changes made during each step is small. Making small changes frequently allows the Cluster Resource Manager to be responsive when things happen in the cluster. The default timers provide some batching since many of the same types of events tend to occur simultaneously.

For example, when nodes fail they can do so entire fault domains at a time. All these failures are captured during the next state update after the *PLBRefreshGap*. The corrections are determined during the following placement, constraint check, and balancing runs. By default the Cluster Resource Manager is not scanning through hours of changes in the cluster and trying to address all changes at once. Doing so would lead to bursts of churn.

The Cluster Resource Manager also needs some additional information to determine if the cluster imbalanced. For that we have two other pieces of configuration: *BalancingThresholds* and *ActivityThresholds*.

## Balancing thresholds

A Balancing Threshold is the main control for triggering rebalancing. The Balancing Threshold for a metric is a *ratio*. If the load for a metric on the most loaded node divided by the amount of load on the least loaded node exceeds that metric's *BalancingThreshold*, then the cluster is imbalanced. As a result balancing is triggered the next time the Cluster Resource Manager checks. The *MinLoadBalancingInterval* timer defines how often the Cluster Resource Manager should check if rebalancing is necessary. Checking doesn't mean that anything happens.

Balancing Thresholds are defined on a per-metric basis as a part of the cluster definition. For more information on metrics, check out [this article](#).

ClusterManifest.xml

```

<Section Name="MetricBalancingThresholds">
  <Parameter Name="MetricName1" Value="2"/>
  <Parameter Name="MetricName2" Value="3.5"/>
</Section>

```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters:

```

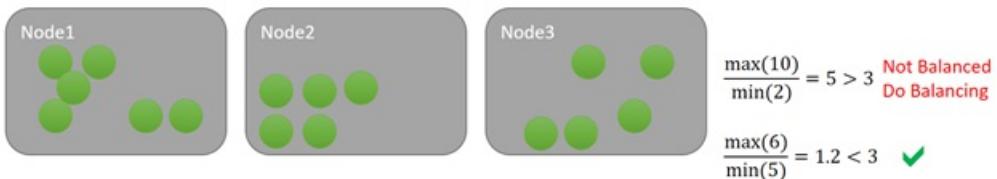
"fabricSettings": [
  {
    "name": "MetricBalancingThresholds",
    "parameters": [
      {
        "name": "MetricName1",
        "value": "2"
      },
      {
        "name": "MetricName2",
        "value": "3.5"
      }
    ]
  }
]

```



In this example, each service is consuming one unit of some metric. In the top example, the maximum load on a node is five and the minimum is two. Let's say that the balancing threshold for this metric is three. Since the ratio in the cluster is  $5/2 = 2.5$  and that is less than the specified balancing threshold of three, the cluster is balanced. No rebalancing is triggered when the Cluster Resource Manager checks.

In the bottom example, the maximum load on a node is ten, while the minimum is two, resulting in a ratio of five. Five is greater than the designated balancing threshold of three for that metric. As a result, a rebalancing run will be scheduled next time the balancing timer fires. In a situation like this some load is usually distributed to Node3. Because the Service Fabric Cluster Resource Manager doesn't use a greedy approach, some load could also be distributed to Node2.



#### NOTE

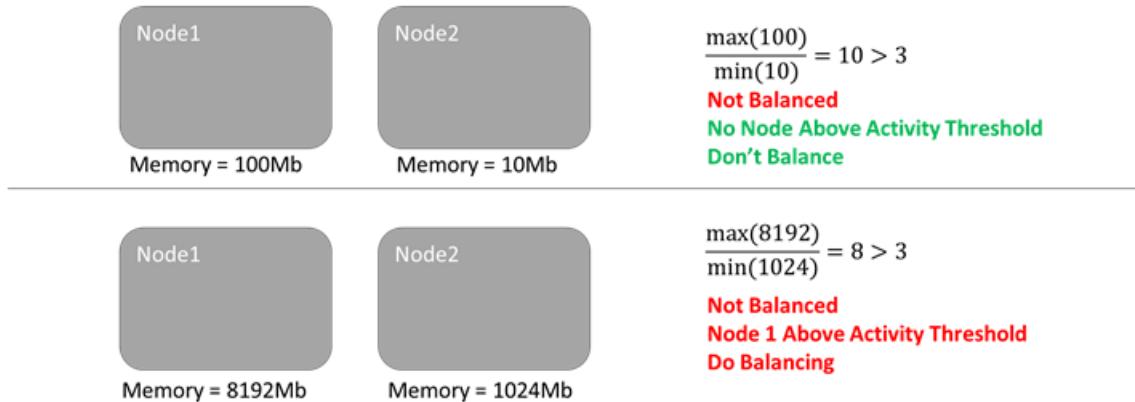
"Balancing" handles two different strategies for managing load in your cluster. The default strategy that the Cluster Resource Manager uses is to distribute load across the nodes in the cluster. The other strategy is [defragmentation](#). Defragmentation is performed during the same balancing run. The balancing and defragmentation strategies can be used for different metrics within the same cluster. A service can have both balancing and defragmentation metrics. For defragmentation metrics, the ratio of the loads in the cluster triggers rebalancing when it is *below* the balancing threshold.

Getting below the balancing threshold is not an explicit goal. Balancing Thresholds are just a *trigger*. When balancing runs, the Cluster Resource Manager determines what improvements it can make, if any. Just because a balancing search is kicked off doesn't mean anything moves. Sometimes the cluster is imbalanced but too constrained to correct. Alternatively, the improvements require movements that are too [costly](#)).

## Activity thresholds

Sometimes, although nodes are relatively imbalanced, the *total* amount of load in the cluster is low. The lack of load could be a transient dip, or because the cluster is new and just getting bootstrapped. In either case, you may not want to spend time balancing the cluster because there's little to be gained. If the cluster underwent balancing, you'd spend network and compute resources to move things around without making any large *absolute* difference. To avoid unnecessary moves, there's another control known as Activity Thresholds. Activity Thresholds allows you to specify some absolute lower bound for activity. If no node is over this threshold, balancing isn't triggered even if the Balancing Threshold is met.

Let's say that we retain our Balancing Threshold of three for this metric. Let's also say we have an Activity Threshold of 1536. In the first case, while the cluster is imbalanced per the Balancing Threshold there's no node meets that Activity Threshold, so nothing happens. In the bottom example, Node1 is over the Activity Threshold. Since both the Balancing Threshold and the Activity Threshold for the metric are exceeded, balancing is scheduled. As an example, let's look at the following diagram:



Just like Balancing Thresholds, Activity Thresholds are defined per-metric via the cluster definition:

ClusterManifest.xml

```
<Section Name="MetricActivityThresholds">
  <Parameter Name="Memory" Value="1536"/>
</Section>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters:

```
"fabricSettings": [
  {
    "name": "MetricActivityThresholds",
    "parameters": [
      {
        "name": "Memory",
        "value": "1536"
      }
    ]
  }
]
```

Balancing and activity thresholds are both tied to a specific metric - balancing is triggered only if both the Balancing Threshold and Activity Threshold is exceeded for the same metric.

#### NOTE

When not specified, the Balancing Threshold for a metric is 1, and the Activity Threshold is 0. This means that the Cluster Resource Manager will try to keep that metric perfectly balanced for any given load. If you are using custom metrics it is recommended that you explicitly define your own balancing and activity thresholds for your metrics.

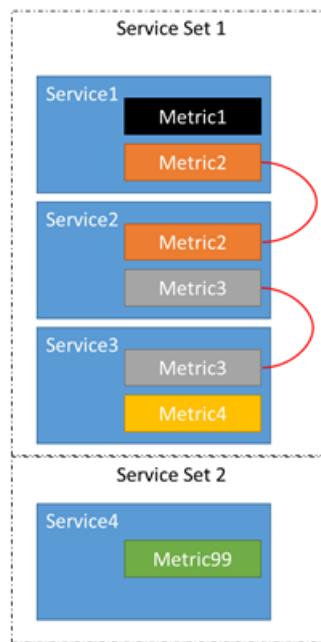
## Balancing services together

Whether the cluster is imbalanced or not is a cluster-wide decision. However, the way we go about fixing it is moving individual service replicas and instances around. This makes sense, right? If memory is stacked up on one node, multiple replicas or instances could be contributing to it. Fixing the imbalance could require moving any of the stateful replicas or stateless instances that use the imbalanced metric.

Occasionally though, a service that wasn't itself imbalanced gets moved (remember the discussion of local and global weights earlier). Why would a service get moved when all that service's metrics were balanced? Let's see an example:

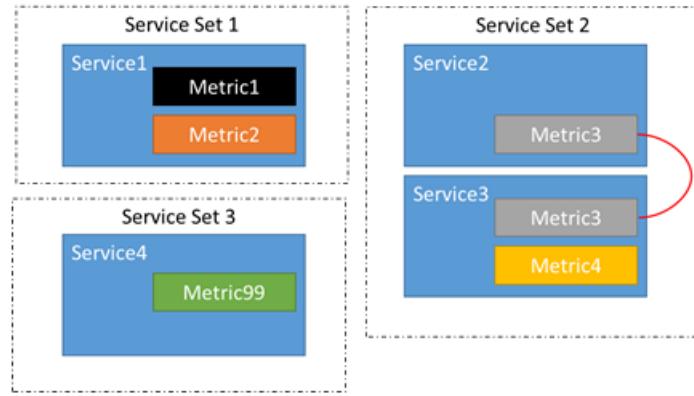
- Let's say there are four services, Service1, Service2, Service3, and Service4.
- Service1 reports metrics Metric1 and Metric2.
- Service2 reports metrics Metric2 and Metric3.
- Service3 reports metrics Metric3 and Metric4.
- Service4 reports metric Metric99.

Surely you can see where we're going here: There's a chain! We don't really have four independent services, we have three services that are related and one that is off on its own.



Because of this chain, it's possible that an imbalance in metrics 1-4 can cause replicas or instances belonging to services 1-3 to move around. We also know that an imbalance in Metrics 1, 2, or 3 can't cause movements in Service4. There would be no point since moving the replicas or instances belonging to Service4 around can do absolutely nothing to impact the balance of Metrics 1-3.

The Cluster Resource Manager automatically figures out what services are related. Adding, removing, or changing the metrics for services can impact their relationships. For example, between two runs of balancing Service2 may have been updated to remove Metric2. This breaks the chain between Service1 and Service2. Now instead of two groups of related services, there are three:



## Next steps

- Metrics are how the Service Fabric Cluster Resource Manager manages consumption and capacity in the cluster. To learn more about metrics and how to configure them, check out [this article](#)
- Movement Cost is one way of signaling to the Cluster Resource Manager that certain services are more expensive to move than others. For more about movement cost, refer to [this article](#)
- The Cluster Resource Manager has several throttles that you can configure to slow down churn in the cluster. They're not normally necessary, but if you need them you can learn about them [here](#)

# Throttling the Service Fabric Cluster Resource Manager

8/18/2017 • 4 min to read • [Edit Online](#)

Even if you've configured the Cluster Resource Manager correctly, the cluster can get disrupted. For example, there could be simultaneous node and fault domain failures - what would happen if that occurred during an upgrade? The Cluster Resource Manager always tries to fix everything, consuming the cluster's resources trying to reorganize and fix the cluster. Throttles help provide a backstop so that the cluster can use resources to stabilize - the nodes come back, the network partitions heal, corrected bits get deployed.

To help with these sorts of situations, the Service Fabric Cluster Resource Manager includes several throttles. These throttles are all fairly large hammers. Generally they shouldn't be changed without careful planning and testing.

If you change the Cluster Resource Manager's throttles, you should tune them to your expected actual load. You may determine you need to have some throttles in place, even if it means the cluster takes longer to stabilize in some situations. Testing is required to determine the correct values for throttles. Throttles need to be high enough to allow the cluster to respond to changes in a reasonable amount of time, and low enough to actually prevent too much resource consumption.

Most of the time we've seen customers use throttles it has been because they were already in a resource constrained environment. Some examples would be limited network bandwidth for individual nodes, or disks that aren't able to build many stateful replicas in parallel due to throughput limitations. Without throttles, operations could overwhelm these resources, causing operations to fail or be slow. In these situations customers used throttles and knew they were extending the amount of time it would take the cluster to reach a stable state. Customers also understood they could end up running at lower overall reliability while they were throttled.

## Configuring the throttles

Service Fabric has two mechanisms for throttling the number of replica movements. The default mechanism that existed before Service Fabric 5.7 represents throttling as an absolute number of moves allowed. This does not work for clusters of all sizes. In particular, for large clusters the default value can be too small, significantly slowing down balancing even when it is necessary, while having no effect in smaller clusters. This prior mechanism has been superseded by percentage-based throttling, which scales better with dynamic clusters in which the number of services and nodes change regularly.

The throttles are based on a percentage of the number of replicas in the clusters. Percentage based throttles enable expressing the rule: "do not move more than 10% of replicas in a 10 minute interval", for example.

The configuration settings for percentage-based throttling are:

- `GlobalMovementThrottleThresholdPercentage` - Maximum number of movements allowed in cluster at any time, expressed as percentage of total number of replicas in the cluster. 0 indicates no limit. The default value is 0. If both this setting and `GlobalMovementThrottleThreshold` are specified, then the more conservative limit is used.
- `GlobalMovementThrottleThresholdPercentageForPlacement` - Maximum number of movements allowed during the placement phase, expressed as percentage of total number of replicas in the cluster. 0 indicates no limit. The default value is 0. If both this setting and `GlobalMovementThrottleThresholdForPlacement` are specified, then the more conservative limit is used.
- `GlobalMovementThrottleThresholdPercentageForBalancing` - Maximum number of movements allowed during the balancing phase, expressed as percentage of total number of replicas in the cluster. 0 indicates no limit. The default value is 0. If both this setting and `GlobalMovementThrottleThresholdForBalancing` are specified, then the

more conservative limit is used.

When specifying the throttle percentage, you'd specify 5% as 0.05. The interval on which these throttles are governed is the GlobalMovementThrottleCountingInterval, which is specified in seconds.

```
<Section Name="PlacementAndLoadBalancing">
  <Parameter Name="GlobalMovementThrottleThresholdPercentage" Value="0" />
  <Parameter Name="GlobalMovementThrottleThresholdPercentageForPlacement" Value="0" />
  <Parameter Name="GlobalMovementThrottleThresholdPercentageForBalancing" Value="0" />
  <Parameter Name="GlobalMovementThrottleCountingInterval" Value="600" />
</Section>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters:

```
"fabricSettings": [
  {
    "name": "PlacementAndLoadBalancing",
    "parameters": [
      {
        "name": "GlobalMovementThrottleThresholdPercentage",
        "value": "0.0"
      },
      {
        "name": "GlobalMovementThrottleThresholdPercentageForPlacement",
        "value": "0.0"
      },
      {
        "name": "GlobalMovementThrottleThresholdPercentageForBalancing",
        "value": "0.0"
      },
      {
        "name": "GlobalMovementThrottleCountingInterval",
        "value": "600"
      }
    ]
  }
]
```

## Default count based throttles

This information is provided in case you have older clusters or still retain these configurations in clusters that have since been upgraded. In general, it is recommended that these are replaced with the percentage-based throttles above. Since percentage-based throttling is disabled by default, these throttles remain the default throttles for a cluster until they are disabled and replaced with the percentage-based throttles.

- GlobalMovementThrottleThreshold – this setting controls the total number of movements in the cluster over some time. The amount of time is specified in seconds as the GlobalMovementThrottleCountingInterval. The default value for the GlobalMovementThrottleThreshold is 1000 and the default value for the GlobalMovementThrottleCountingInterval is 600.
- MovementPerPartitionThrottleThreshold – this setting controls the total number of movements for any service partition over some time. The amount of time is specified in seconds as the MovementPerPartitionThrottleCountingInterval. The default value for the MovementPerPartitionThrottleThreshold is 50 and the default value for the MovementPerPartitionThrottleCountingInterval is 600.

The configuration for these throttles follows the same pattern as the percentage-based throttling.

## Next steps

- To find out about how the Cluster Resource Manager manages and balances load in the cluster, check out the

article on [balancing load](#)

- The Cluster Resource Manager has many options for describing the cluster. To find out more about them, check out this article on [describing a Service Fabric cluster](#)

# Service movement cost

8/18/2017 • 3 min to read • [Edit Online](#)

A factor that the Service Fabric Cluster Resource Manager considers when trying to determine what changes to make to a cluster is the cost of those changes. The notion of "cost" is traded off against how much the cluster can be improved. Cost is factored in when moving services for balancing, defragmentation, and other requirements. The goal is to meet the requirements in the least disruptive or expensive way.

Moving services costs CPU time and network bandwidth at a minimum. For stateful services, it requires copying the state of those services, consuming additional memory and disk. Minimizing the cost of solutions that the Azure Service Fabric Cluster Resource Manager comes up with helps ensure that the cluster's resources aren't spent unnecessarily. However, you also don't want to ignore solutions that would significantly improve the allocation of resources in the cluster.

The Cluster Resource Manager has two ways of computing costs and limiting them while it tries to manage the cluster. The first mechanism is simply counting every move that it would make. If two solutions are generated with about the same balance (score), then the Cluster Resource Manager prefers the one with the lowest cost (total number of moves).

This strategy works well. But as with default or static loads, it's unlikely in any complex system that all moves are equal. Some are likely to be much more expensive.

## Setting Move Costs

You can specify the default move cost for a service when it is created:

PowerShell:

```
New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName  
$serviceTypeName -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -  
DefaultMoveCost Medium
```

C#:

```
FabricClient fabricClient = new FabricClient();  
StatefulServiceDescription serviceDescription = new StatefulServiceDescription();  
//set up the rest of the ServiceDescription  
serviceDescription.DefaultMoveCost = MoveCost.Medium;  
await fabricClient.ServiceManager.CreateServiceAsync(serviceDescription);
```

You can also specify or update MoveCost dynamically for a service after the service has been created:

PowerShell:

```
Update-ServiceFabricService -Stateful -ServiceName "fabric:/AppName/ServiceName" -DefaultMoveCost High
```

C#:

```

StatefulServiceUpdateDescription updateDescription = new StatefulServiceUpdateDescription();
updateDescription.DefaultMoveCost = MoveCost.High;
await fabricClient.ServiceManager.UpdateServiceAsync(new Uri("fabric:/AppName/ServiceName"),
updateDescription);

```

## Dynamically specifying move cost on a per-replica basis

The preceding snippets are all for specifying MoveCost for a whole service at once from outside the service itself. However, move cost is most useful is when the move cost of a specific service object changes over its lifespan. Since the services themselves probably have the best idea of how costly they are to move a given time, there's an API for services to report their own individual move cost during runtime.

C#:

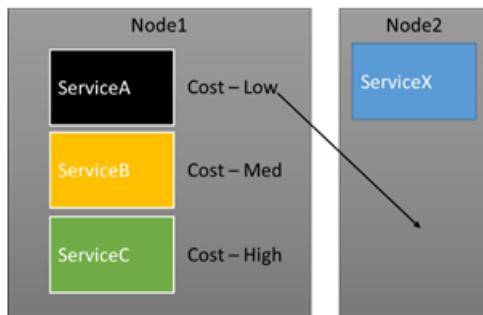
```

this.Partition.ReportMoveCost(MoveCost.Medium);

```

## Impact of move cost

MoveCost has four levels: Zero, Low, Medium, and High. MoveCosts are relative to each other, except for Zero. Zero move cost means that movement is free and should not count against the score of the solution. Setting your move cost to High does *not* guarantee that the replica stays in one place.



MoveCost helps you find the solutions that cause the least disruption overall and are easiest to achieve while still arriving at equivalent balance. A service's notion of cost can be relative to many things. The most common factors in calculating your move cost are:

- The amount of state or data that the service has to move.
- The cost of disconnection of clients. Moving a primary replica is usually more costly than the cost of moving a secondary replica.
- The cost of interrupting an in-flight operation. Some operations at the data store level or operations performed in response to a client call are costly. After a certain point, you don't want to stop them if you don't have to. So while the operation is going on, you increase the move cost of this service object to reduce the likelihood that it moves. When the operation is done, you set the cost back to normal.

## Enabling move cost in your cluster

In order for the more granular MoveCosts to be taken into account, MoveCost must be enabled in your cluster. Without this setting, the default mode of counting moves is used for calculating MoveCost, and MoveCost reports are ignored.

ClusterManifest.xml:

```
<Section Name="PlacementAndLoadBalancing">
    <Parameter Name="UseMoveCostReports" Value="true" />
</Section>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters:

```
"fabricSettings": [
{
    "name": "PlacementAndLoadBalancing",
    "parameters": [
        {
            "name": "UseMoveCostReports",
            "value": "true"
        }
    ]
}]
```

## Next steps

- Service Fabric Cluster Resource Manager uses metrics to manage consumption and capacity in the cluster. To learn more about metrics and how to configure them, check out [Managing resource consumption and load in Service Fabric with metrics](#).
- To learn about how the Cluster Resource Manager manages and balances load in the cluster, check out [Balancing your Service Fabric cluster](#).

# Service Fabric with Azure API Management overview

6/27/2017 • 6 min to read • [Edit Online](#)

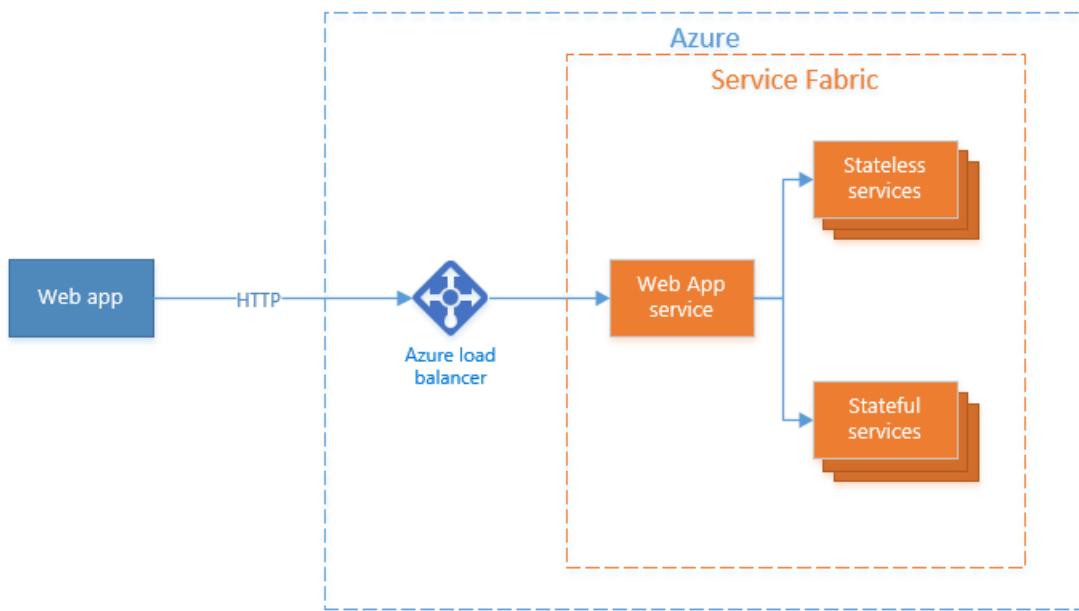
Cloud applications typically need a front-end gateway to provide a single point of ingress for users, devices, or other applications. In Service Fabric, a gateway can be any stateless service such as an [ASP.NET Core application](#), or another service designed for traffic ingress, such as [Event Hubs](#), [IoT Hub](#), or [Azure API Management](#).

This article is an introduction to using Azure API Management as a gateway to your Service Fabric applications. API Management integrates directly with Service Fabric, allowing you to publish APIs with a rich set of routing rules to your back-end Service Fabric services.

## Architecture

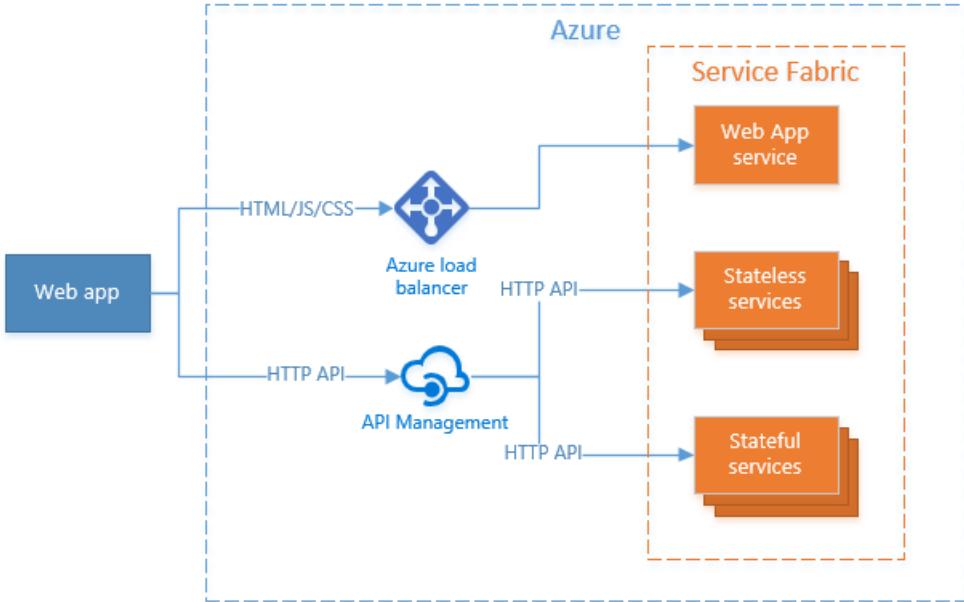
A common Service Fabric architecture uses a single-page web application that makes HTTP calls to back-end services that expose HTTP APIs. The [Service Fabric getting-started sample application](#) shows an example of this architecture.

In this scenario, a stateless web service serves as the gateway into the Service Fabric application. This approach requires you to write a web service that can proxy HTTP requests to back-end services, as shown in the following diagram:



As applications grow in complexity, so do the gateways that must present an API in front of myriad back-end services. Azure API Management is designed to handle complex APIs with routing rules, access control, rate limiting, monitoring, event logging, and response caching with minimal work on your part. Azure API Management supports Service Fabric service discovery, partition resolution, and replica selection to intelligently route requests directly to back-end services in Service Fabric so you don't have to write your own stateless API gateway.

In this scenario, the web UI is still served through a web service, while HTTP API calls are managed and routed through Azure API Management, as shown in the following diagram:



## Application scenarios

Services in Service Fabric may be either stateless or stateful, and they may be partitioned using one of three schemes: singleton, int-64 range, and named. Service endpoint resolution requires identifying a specific partition of a specific service instance. When resolving an endpoint of a service, both the service instance name (for example, `fabric:/myapp/myservice`) as well as the specific partition of the service must be specified, except in the case of singleton partition.

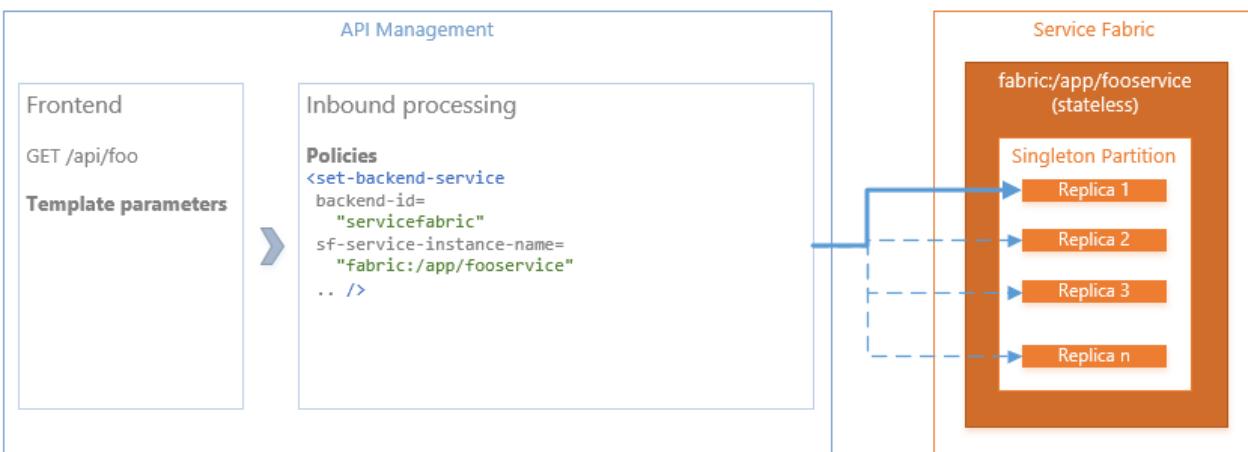
Azure API Management can be used with any combination of stateless services, stateful services, and any partitioning scheme.

## Send traffic to a stateless service

In the simplest case, traffic is forwarded to a stateless service instance. To achieve this, an API Management operation contains an inbound processing policy with a Service Fabric back-end that maps to a specific stateless service instance in the Service Fabric back-end. Requests sent to that service are sent to a random replica of the stateless service instance.

### Example

In the following scenario, a Service Fabric application contains a stateless service named `fabric:/app/fooservice`, that exposes an internal HTTP API. The service instance name is well known and can be hard-coded directly in the API Management inbound processing policy.



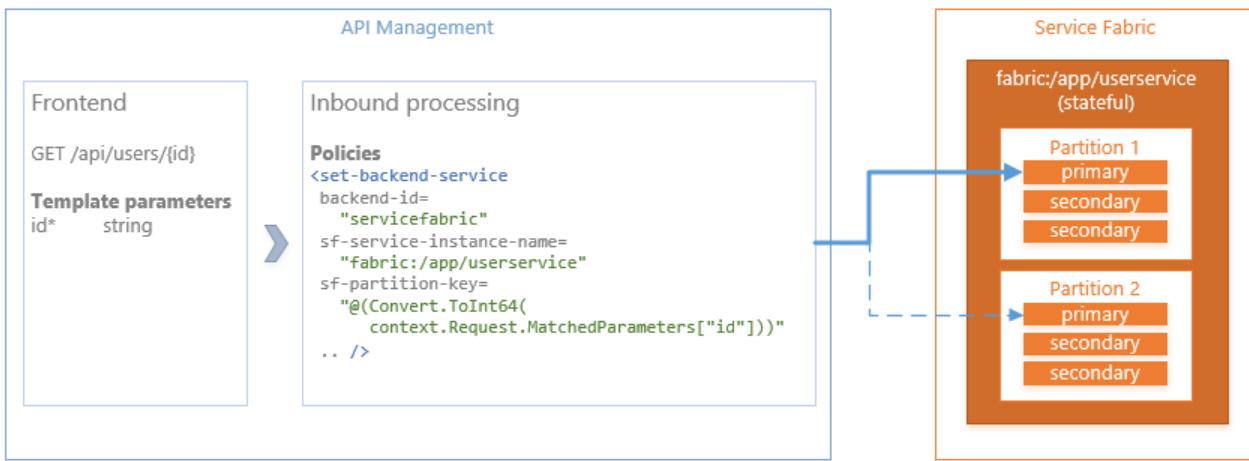
## Send traffic to a stateful service

Similar to the stateless service scenario, traffic can be forwarded to a stateful service instance. In this case, an API Management operation contains an inbound processing policy with a Service Fabric back-end that maps a request to a specific partition of a specific *stateful* service instance. The partition to map each request to is computed via a lambda method using some input from the incoming HTTP request, such as a value in the URL path. The policy may be configured to send requests to the primary replica only, or to a random replica for read operations.

#### Example

In the following scenario, a Service Fabric application contains a partitioned stateful service named `fabric:/app/userservice` that exposes an internal HTTP API. The service instance name is well known and can be hard-coded directly in the API Management inbound processing policy.

The service is partitioned using the Int64 partition scheme with two partitions and a key range that spans `Int64.MinValue` to `Int64.MaxValue`. The back-end policy computes a partition key within that range by converting the `id` value provided in the URL request path to a 64-bit integer, although any algorithm can be used here to compute the partition key.



## Send traffic to multiple stateless services

In more advanced scenarios, you can define an API Management operation that maps requests to more than one service instance. In this case, each operation contains a policy that maps requests to a specific service instance based on values from the incoming HTTP request, such as the URL path or query string, and in the case of stateful services, a partition within the service instance.

To achieve this, an API Management operation contains an inbound processing policy with a Service Fabric back-end that maps to a stateless service instance in the Service Fabric back-end based on values retrieved from the incoming HTTP request. Requests to a service instance are sent to a random replica of the service instance.

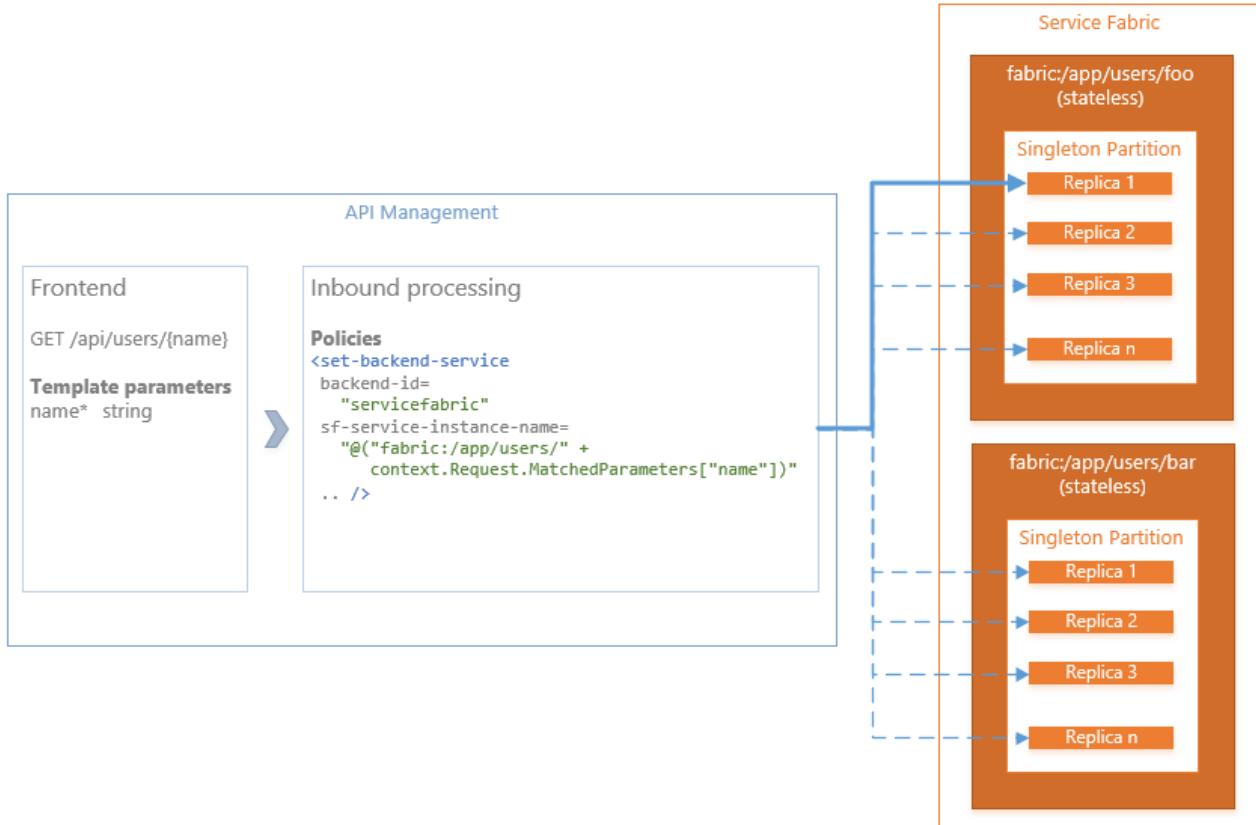
#### Example

In this example, a new stateless service instance is created for each user of an application with a dynamically generated name using the following formula:

- `fabric:/app/users/<username>`

Each service has a unique name, but the names are not known up-front because the services are created in response to user or admin input and thus cannot be hard-coded into APIM policies or routing rules. Instead, the name of the service to which to send a request is generated in the back-end policy definition from the `name` value provided in the URL request path. For example:

- A request to `/api/users/foo` is routed to service instance `fabric:/app/users/foo`
- A request to `/api/users/bar` is routed to service instance `fabric:/app/users/bar`



## Send traffic to multiple stateful services

Similar to the stateless service example, an API Management operation can map requests to more than one **stateful** service instance, in which case you also may need to perform partition resolution for each stateful service instance.

To achieve this, an API Management operation contains an inbound processing policy with a Service Fabric back-end that maps to a stateful service instance in the Service Fabric back-end based on values retrieved from the incoming HTTP request. In addition to mapping a request to specific service instance, the request can also be mapped to a specific partition within the service instance, and optionally to either the primary replica or a random secondary replica within the partition.

### Example

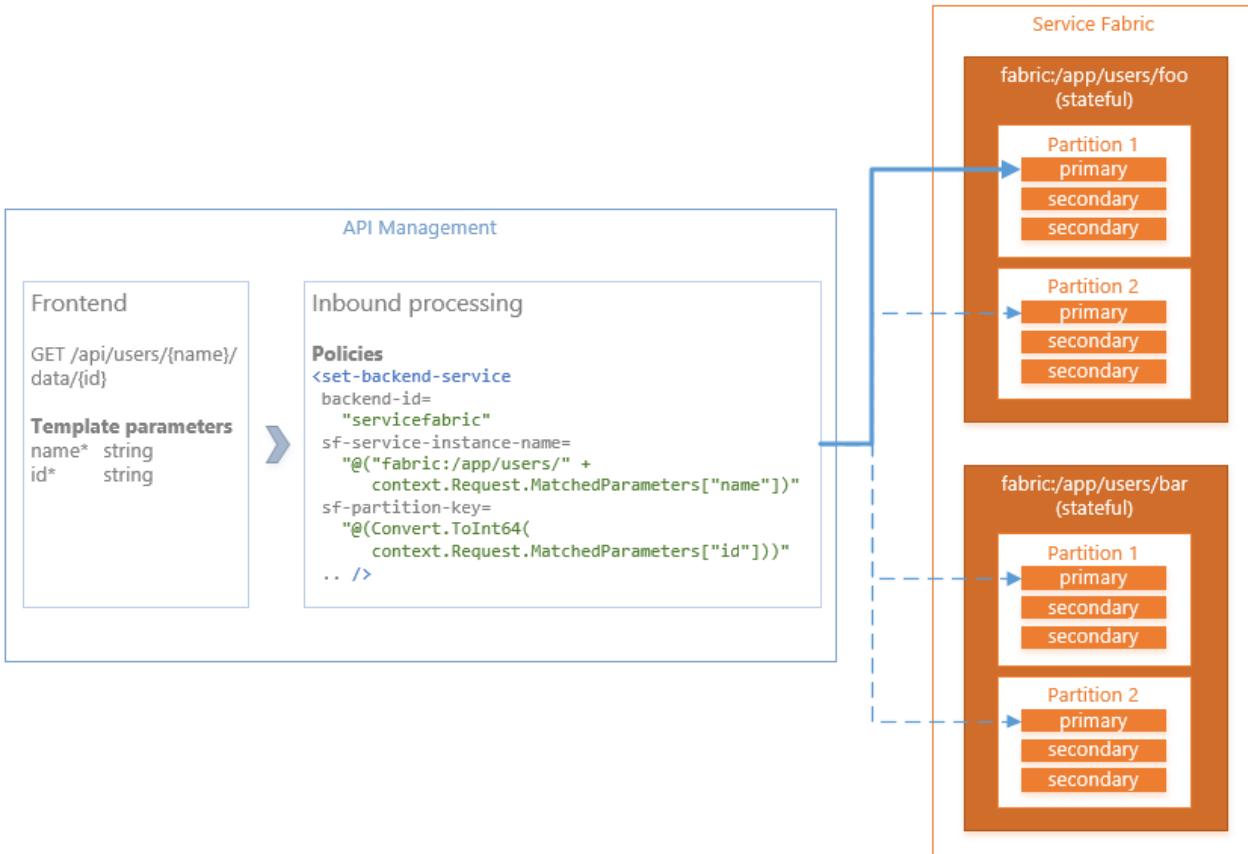
In this example, a new stateful service instance is created for each user of the application with a dynamically generated name using the following formula:

- `fabric:/app/users/<username>`

Each service has a unique name, but the names are not known up-front because the services are created in response to user or admin input and thus cannot be hard-coded into APIM policies or routing rules. Instead, the name of the service to which to send a request is generated in the back-end policy definition from the `name` value provided the URL request path. For example:

- A request to `/api/users/foo` is routed to service instance `fabric:/app/users/foo`
- A request to `/api/users/bar` is routed to service instance `fabric:/app/users/bar`

Each service instance is also partitioned using the Int64 partition scheme with two partitions and a key range that spans `Int64.MinValue` to `Int64.MaxValue`. The back-end policy computes a partition key within that range by converting the `id` value provided in the URL request path to a 64-bit integer, although any algorithm can be used here to compute the partition key.



## Next steps

Follow the [quick start guide](#) to set up your first Service Fabric cluster with API Management and flow requests through API Management to your services.

# Monitoring and diagnostics for Azure Service Fabric

7/20/2017 • 6 min to read • [Edit Online](#)

Monitoring and diagnostics are critical to developing, testing, and deploying applications and services in any environment. Service Fabric solutions work best when you plan and implement monitoring and diagnostics that help ensure applications and services are working as expected in a local development environment or in production.

The main goals of monitoring and diagnostics are to:

- Detect and diagnose hardware and infrastructure issues
- Detect software and app issues, reduce service downtime
- Understand resource consumption and help drive operations decisions
- Optimize application, service, and infrastructure performance
- Generate business insights and identify areas of improvement

The overall workflow of monitoring and diagnostics consists of three steps:

1. **Event generation:** this includes events (logs, traces, custom events) at the infrastructure (cluster), platform, and application / service level
2. **Event aggregation:** generated events need to be collected and aggregated before they can be displayed
3. **Analysis:** events need to be visualized and accessible in some format, to allow for analysis and display as needed

Multiple products are available that cover these three areas, and you are free to choose different technologies for each. It is important to make sure that the various pieces work together to deliver an end-to-end monitoring solution for your cluster.

## Event generation

The first step in the monitoring and diagnostics workflow is the creation and generation of events and logs. These events, logs, and traces are generated at two levels: the platform layer (including the cluster, the machines, or Service Fabric actions) or the application layer (any instrumentation added to apps and services deployed to the cluster). Events at each of these levels are customizable, though Service Fabric does provide some instrumentation by default.

Read more about [platform level events](#) and [application level events](#) to understand what is provided and how to add further instrumentation.

After making a decision on the logging provider you would like to use, you need to make sure your logs are being aggregated and stored correctly.

## Event aggregation

For collecting the logs and events being generated by your applications and your cluster, we typically recommend using [Azure Diagnostics](#) (more similar to agent-based log collection) or [EventFlow](#) (in-process log collection).

Collecting application logs using Azure Diagnostics extension is a good option for Service Fabric services if the set of log sources and destinations does not change often and there is a straightforward mapping between the sources and their destinations. The reason for this is configuring Azure Diagnostics happens at the Resource Manager layer, so making significant changes to the configuration requires updating/redeploying the cluster. Additionally, it is best

utilized in making sure your logs are being stored somewhere a little more permanent, from where they can be accessed by various analysis platforms. This means that it ends up being less efficient of a pipeline than going with an option like EventFlow.

Using [EventFlow](#) allows you to have services send their logs directly to an analysis and visualization platform, and/or to storage. Other libraries (ILogger, Serilog, etc.) might be used for the same purpose, but EventFlow has the benefit of having been designed specifically for in-process log collection and to support Service Fabric services. This tends to have several potential advantages:

- Easy configuration and deployment
  - The configuration of diagnostic data collection is just part of the service configuration. It is easy to always keep it "in sync" with the rest of the application
  - Per-application or per-service configuration is easily achievable
  - Configuring data destinations through EventFlow is just a matter of adding the appropriate NuGet package and changing the *eventFlowConfig.json* file
- Flexibility
  - The application can send the data wherever it needs to go, as long as there is a client library that supports the targeted data storage system. New destinations can be added as desired
  - Complex capture, filtering, and data-aggregation rules can be implemented
- Access to internal application data and context
  - The diagnostic subsystem running inside the application/service process can easily augment the traces with contextual information

One thing to note is that these two options are not mutually exclusive and while it is possible to get a similar job done with using one or the other, it could also make sense for you to set up both. In most situations, combining an agent with in-process collection could lead to a more reliable monitoring workflow. The Azure Diagnostics extension (agent) could be your chosen path for platform level logs while you could use EventFlow (in-process collection) for your application level logs. Once you have figured out what works best for you, it is time to think about how you want your data to be displayed and analyzed.

## Event analysis

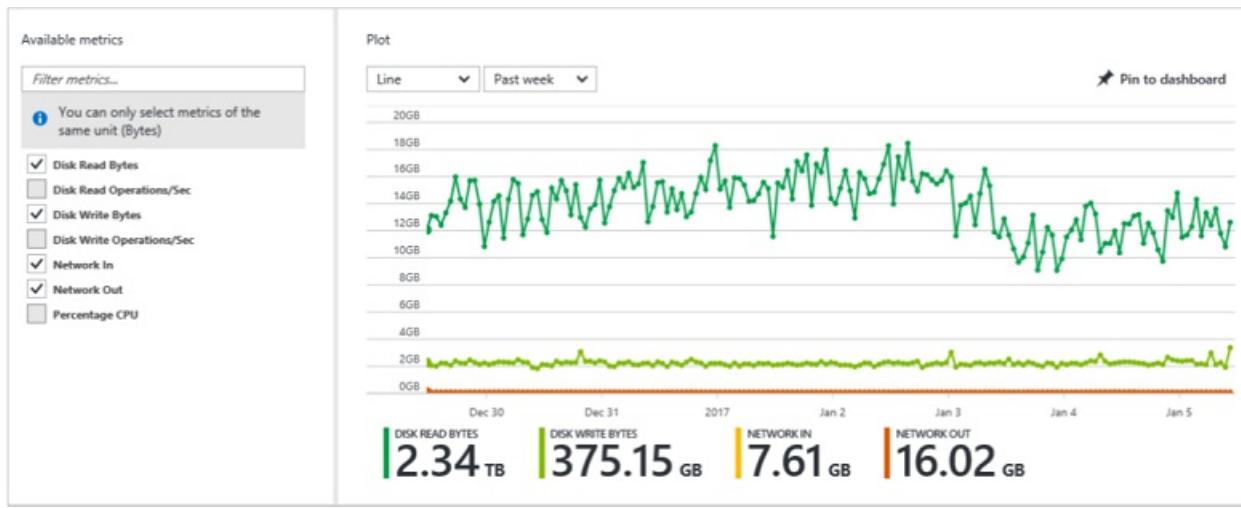
There are several great platforms that exist in the market when it comes to the analysis and visualization of monitoring and diagnostics data. The two that we recommend are [OMS](#) and [Application Insights](#) due to their better integration with Service Fabric, but you should also look into the [Elastic Stack](#) (especially if you are considering running a cluster in an offline environment), [Splunk](#), or any other platform of your preference.

The key points for any platform you choose should include how comfortable you are with the user interface and querying options, the ability to visualize data and create easily readable dashboards, and the additional tools they provide to enhance your monitoring, such as automated alerting.

In addition to the platform you choose, when you set up a Service Fabric cluster as an Azure resource, you also get access to Azure's out-of-the-box monitoring for machines, which can be useful for specific performance and metric monitoring.

### Azure Monitor

You can use [Azure Monitor](#) to monitor many of the Azure resources on which a Service Fabric cluster is built. A set of metrics for the [virtual machine scale set](#) and individual [virtual machines](#) is automatically collected and displayed in the Azure portal. To view the collected information, in the Azure portal, select the resource group that contains the Service Fabric cluster. Then, select the virtual machine scale set that you want to view. In the **Monitoring** section, select **Metrics** to view a graph of the values.



To customize the charts, follow the instructions in [Metrics in Microsoft Azure](#). You also can create alerts based on these metrics, as described in [Create alerts in Azure Monitor for Azure services](#). You can send alerts to a notification service by using web hooks, as described in [Configure a web hook on an Azure metric alert](#). Azure Monitor supports only one subscription. If you need to monitor multiple subscriptions, or if you need additional features, [Log Analytics](#), part of Microsoft Operations Management Suite, provides a holistic IT management solution both for on-premises and cloud-based infrastructure. You can route data from Azure Monitor directly to Log Analytics, so you can see metrics and logs for your entire environment in a single place.

## Next steps

### Watchdogs

A watchdog is a separate service that can watch health and load across services, and report health for anything in the health model hierarchy. This can help prevent errors that would not be detected based on the view of a single service. Watchdogs are also a good place to host code that performs remedial actions without user interaction (for example, cleaning up log files in storage at certain time intervals). You can find a sample watchdog service implementation [here](#).

Get started with understanding how events and logs get generated at the [platform level](#) and the [application level](#).

# Introduction to Service Fabric health monitoring

9/19/2017 • 20 min to read • [Edit Online](#)

Azure Service Fabric introduces a health model that provides rich, flexible, and extensible health evaluation and reporting. The model allows near-real-time monitoring of the state of the cluster and the services running in it. You can easily obtain health information and correct potential issues before they cascade and cause massive outages. In the typical model, services send reports based on their local views, and that information is aggregated to provide an overall cluster-level view.

Service Fabric components use this rich health model to report their current state. You can use the same mechanism to report health from your applications. If you invest in high-quality health reporting that captures your custom conditions, you can detect and fix issues for your running application much more easily.

The following Microsoft Virtual Academy video also describes the Service Fabric health model and how it's used:



## NOTE

We started the health subsystem to address a need for monitored upgrades. Service Fabric provides monitored application and cluster upgrades that ensure full availability, no downtime and minimal to no user intervention. To achieve these goals, the upgrade checks health based on configured upgrade policies. An upgrade can proceed only when health respects desired thresholds. Otherwise, the upgrade is either automatically rolled back or paused to give administrators a chance to fix the issues. To learn more about application upgrades, see [this article](#).

## Health store

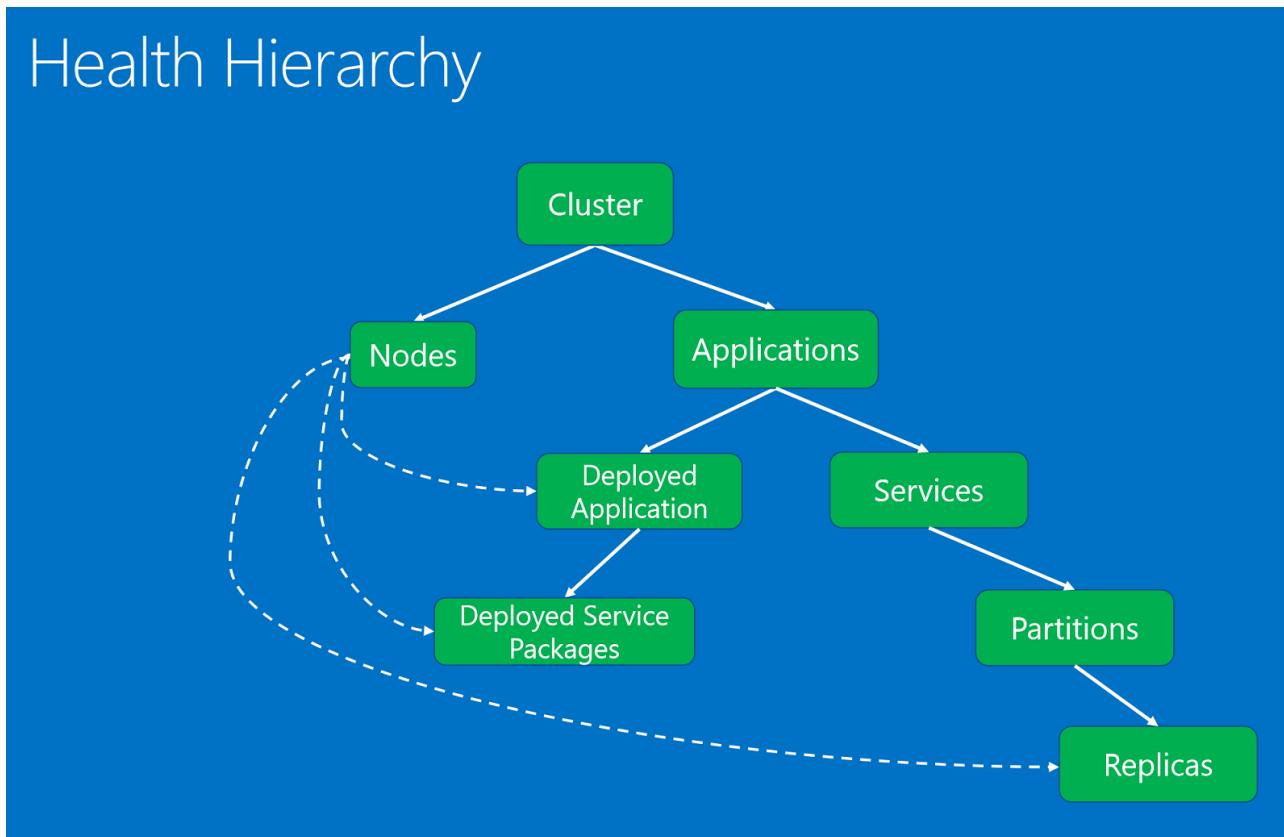
The health store keeps health-related information about entities in the cluster for easy retrieval and evaluation. It is implemented as a Service Fabric persisted stateful service to ensure high availability and scalability. The health store is part of the **fabric:/System** application, and it is available when the cluster is up and running.

## Health entities and hierarchy

The health entities are organized in a logical hierarchy that captures interactions and dependencies among different entities. The health store automatically builds health entities and hierarchy based on reports received from Service Fabric components.

The health entities mirror the Service Fabric entities. (For example, **health application entity** matches an application instance deployed in the cluster, while **health node entity** matches a Service Fabric cluster node.) The health hierarchy captures the interactions of the system entities, and it is the basis for advanced health evaluation. You can learn about key Service Fabric concepts in [Service Fabric technical overview](#). For more on application, see [Service Fabric application model](#).

The health entities and hierarchy allow the cluster and applications to be effectively reported, debugged, and monitored. The health model provides an accurate, *granular* representation of the health of the many moving pieces in the cluster.



The health entities, organized in a hierarchy based on parent-child relationships.

The health entities are:

- **Cluster.** Represents the health of a Service Fabric cluster. Cluster health reports describe conditions that affect the entire cluster. These conditions affect multiple entities in the cluster or the cluster itself. Based on the condition, the reporter can't narrow the issue down to one or more unhealthy children. Examples include the brain of the cluster splitting due to network partitioning or communication issues.
- **Node.** Represents the health of a Service Fabric node. Node health reports describe conditions that affect the node functionality. They typically affect all the deployed entities running on it. Examples include node out of disk space (or other machine-wide properties, such as memory, connections) and when a node is down. The node entity is identified by the node name (string).
- **Application.** Represents the health of an application instance running in the cluster. Application health reports describe conditions that affect the overall health of the application. They can't be narrowed down to individual children (services or deployed applications). Examples include the end-to-end interaction among different services in the application. The application entity is identified by the application name (URI).
- **Service.** Represents the health of a service running in the cluster. Service health reports describe conditions that affect the overall health of the service. The reporter can't narrow down the issue to an unhealthy partition or replica. Examples include a service configuration (such as port or external file share) that is causing issues for all partitions. The service entity is identified by the service name (URI).
- **Partition.** Represents the health of a service partition. Partition health reports describe conditions that affect the entire replica set. Examples include when the number of replicas is below target count and when a partition is in quorum loss. The partition entity is identified by the partition ID (GUID).
- **Replica.** Represents the health of a stateful service replica or a stateless service instance. The replica is the smallest unit that watchdogs and system components can report on for an application. For stateful services, examples include a primary replica that can't replicate operations to secondaries and slow replication. Also, a stateless instance can report when it is running out of resources or has connectivity issues. The replica entity is identified by the partition ID (GUID) and the replica or instance ID (long).

- **DeployedApplication.** Represents the health of an *application running on a node*. Deployed application health reports describe conditions specific to the application on the node that can't be narrowed down to service packages deployed on the same node. Examples include errors when application package can't be downloaded on that node and issues setting up application security principals on the node. The deployed application is identified by application name (URI) and node name (string).
- **DeployedServicePackage.** Represents the health of a service package running on a node in the cluster. It describes conditions specific to a service package that do not affect the other service packages on the same node for the same application. Examples include a code package in the service package that cannot be started and a configuration package that cannot be read. The deployed service package is identified by application name (URI), node name (string), service manifest name (string), and service package activation ID (string).

The granularity of the health model makes it easy to detect and correct issues. For example, if a service is not responding, it is feasible to report that the application instance is unhealthy. Reporting at that level is not ideal, however, because the issue might not be affecting all the services within that application. The report should be applied to the unhealthy service or to a specific child partition, if more information points to that partition. The data automatically surfaces through the hierarchy, and an unhealthy partition is made visible at service and application levels. This aggregation helps to pinpoint and resolve the root cause of the issue more quickly.

The health hierarchy is composed of parent-child relationships. A cluster is composed of nodes and applications. Applications have services and deployed applications. Deployed applications have deployed service packages. Services have partitions, and each partition has one or more replicas. There is a special relationship between nodes and deployed entities. An unhealthy node as reported by its authority system component, the Failover Manager service, affects the deployed applications, service packages, and replicas deployed on it.

The health hierarchy represents the latest state of the system based on the latest health reports, which is almost real-time information. Internal and external watchdogs can report on the same entities based on application-specific logic or custom monitored conditions. User reports coexist with the system reports.

Plan to invest in how to report and respond to health during the design of a large cloud service. This up-front investment makes the service easier to debug, monitor, and operate.

## Health states

Service Fabric uses three health states to describe whether an entity is healthy or not: OK, warning, and error. Any report sent to the health store must specify one of these states. The health evaluation result is one of these states.

The possible [health states](#) are:

- **OK.** The entity is healthy. There are no known issues reported on it or its children (when applicable).
- **Warning.** The entity has some issues, but it can still function correctly. For example, there are delays, but they do not cause any functional issues yet. In some cases, the warning condition may fix itself without external intervention. In these cases, health reports raise awareness and provide visibility into what is going on. In other cases, the warning condition may degrade into a severe problem without user intervention.
- **Error.** The entity is unhealthy. Action should be taken to fix the state of the entity, because it can't function properly.
- **Unknown.** The entity doesn't exist in the health store. This result can be obtained from the distributed queries that merge results from multiple components. For example, the get node list query goes to **FailoverManager**, **ClusterManager**, and **HealthManager**; get application list query goes to **ClusterManager** and **HealthManager**. These queries merge results from multiple system components. If another system component returns an entity that is not present in health store, the merged result has unknown health state. An entity is not in store because health reports have not yet been processed or the entity has been cleaned up after deletion.

## Health policies

The health store applies health policies to determine whether an entity is healthy based on its reports and its children.

#### NOTE

Health policies can be specified in the cluster manifest (for cluster and node health evaluation) or in the application manifest (for application evaluation and any of its children). Health evaluation requests can also pass in custom health evaluation policies, which are used only for that evaluation.

By default, Service Fabric applies strict rules (everything must be healthy) for the parent-child hierarchical relationship. If even one of the children has one unhealthy event, the parent is considered unhealthy.

#### Cluster health policy

The [cluster health policy](#) is used to evaluate the cluster health state and node health states. The policy can be defined in the cluster manifest. If it is not present, the default policy (zero tolerated failures) is used. The cluster health policy contains:

- [ConsiderWarningAsError](#). Specifies whether to treat warning health reports as errors during health evaluation. Default: false.
- [MaxPercentUnhealthyApplications](#). Specifies the maximum tolerated percentage of applications that can be unhealthy before the cluster is considered in error.
- [MaxPercentUnhealthyNodes](#). Specifies the maximum tolerated percentage of nodes that can be unhealthy before the cluster is considered in error. In large clusters, some nodes are always down or out for repairs, so this percentage should be configured to tolerate that.
- [ApplicationTypeHealthPolicyMap](#). The application type health policy map can be used during cluster health evaluation to describe special application types. By default, all applications are put into a pool and evaluated with MaxPercentUnhealthyApplications. If some application types should be treated differently, they can be taken out of the global pool. Instead, they are evaluated against the percentages associated with their application type name in the map. For example, in a cluster there are thousands of applications of different types, and a few control application instances of a special application type. The control applications should never be in error. You can specify global MaxPercentUnhealthyApplications to 20% to tolerate some failures, but for the application type "ControlApplicationType" set the MaxPercentUnhealthyApplications to 0. This way, if some of the many applications are unhealthy, but below the global unhealthy percentage, the cluster would be evaluated to Warning. A warning health state does not impact cluster upgrade or other monitoring triggered by Error health state. But even one control application in error would make cluster unhealthy, which triggers roll back or pauses the cluster upgrade, depending on the upgrade configuration. For the application types defined in the map, all application instances are taken out of the global pool of applications. They are evaluated based on the total number of applications of the application type, using the specific MaxPercentUnhealthyApplications from the map. All the rest of the applications remain in the global pool and are evaluated with MaxPercentUnhealthyApplications.

The following example is an excerpt from a cluster manifest. To define entries in the application type map, prefix the parameter name with "ApplicationTypeMaxPercentUnhealthyApplications-", followed by the application type name.

```
<FabricSettings>
  <Section Name="HealthManager/ClusterHealthPolicy">
    <Parameter Name="ConsiderWarningAsError" Value="False" />
    <Parameter Name="MaxPercentUnhealthyApplications" Value="20" />
    <Parameter Name="MaxPercentUnhealthyNodes" Value="20" />
    <Parameter Name="ApplicationTypeMaxPercentUnhealthyApplications-ControlApplicationType" Value="0" />
  </Section>
</FabricSettings>
```

#### Application health policy

The [application health policy](#) describes how the evaluation of events and child-states aggregation is done for applications and their children. It can be defined in the application manifest, **ApplicationManifest.xml**, in the application package. If no policies are specified, Service Fabric assumes that the entity is unhealthy if it has a health report or a child at the warning or error health state. The configurable policies are:

- [ConsiderWarningAsError](#). Specifies whether to treat warning health reports as errors during health evaluation. Default: false.
- [MaxPercentUnhealthyDeployedApplications](#). Specifies the maximum tolerated percentage of deployed applications that can be unhealthy before the application is considered in error. This percentage is calculated by dividing the number of unhealthy deployed applications over the number of nodes that the applications are currently deployed on in the cluster. The computation rounds up to tolerate one failure on small numbers of nodes. Default percentage: zero.
- [DefaultServiceTypeHealthPolicy](#). Specifies the default service type health policy, which replaces the default health policy for all service types in the application.
- [ServiceTypeHealthPolicyMap](#). Provides a map of service health policies per service type. These policies replace the default service type health policies for each specified service type. For example, if an application has a stateless gateway service type and a stateful engine service type, you can configure the health policies for their evaluation differently. When you specify policy per service type, you can gain more granular control of the health of the service.

## Service type health policy

The [service type health policy](#) specifies how to evaluate and aggregate the services and the children of services. The policy contains:

- [MaxPercentUnhealthyPartitionsPerService](#). Specifies the maximum tolerated percentage of unhealthy partitions before a service is considered unhealthy. Default percentage: zero.
- [MaxPercentUnhealthyReplicasPerPartition](#). Specifies the maximum tolerated percentage of unhealthy replicas before a partition is considered unhealthy. Default percentage: zero.
- [MaxPercentUnhealthyServices](#). Specifies the maximum tolerated percentage of unhealthy services before the application is considered unhealthy. Default percentage: zero.

The following example is an excerpt from an application manifest:

```
<Policies>
    <HealthPolicy ConsiderWarningAsError="true" MaxPercentUnhealthyDeployedApplications="20">
        <DefaultServiceTypeHealthPolicy>
            MaxPercentUnhealthyServices="0"
            MaxPercentUnhealthyPartitionsPerService="10"
            MaxPercentUnhealthyReplicasPerPartition="0"/>
        <ServiceTypeHealthPolicy ServiceTypeName="FrontEndServiceType">
            MaxPercentUnhealthyServices="0"
            MaxPercentUnhealthyPartitionsPerService="20"
            MaxPercentUnhealthyReplicasPerPartition="0"/>
        <ServiceTypeHealthPolicy ServiceTypeName="BackEndServiceType">
            MaxPercentUnhealthyServices="20"
            MaxPercentUnhealthyPartitionsPerService="0"
            MaxPercentUnhealthyReplicasPerPartition="0"/>
        </ServiceTypeHealthPolicy>
    </HealthPolicy>
</Policies>
```

## Health evaluation

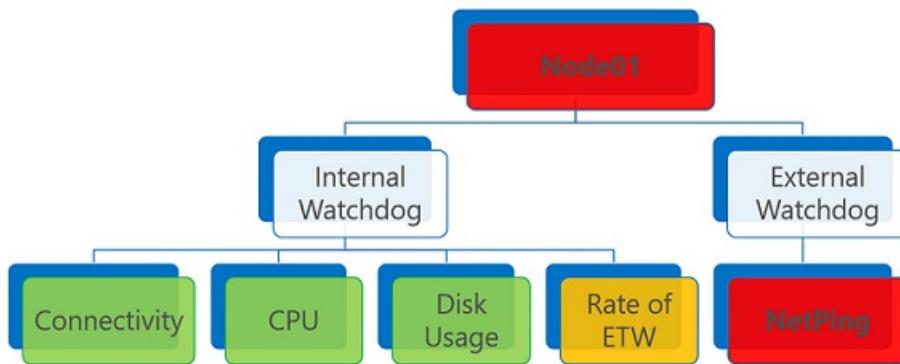
Users and automated services can evaluate health for any entity at any time. To evaluate an entity's health, the health store aggregates all health reports on the entity and evaluates all its children (when applicable). The health aggregation algorithm uses health policies that specify how to evaluate health reports and how to aggregate child

health states (when applicable).

### Health report aggregation

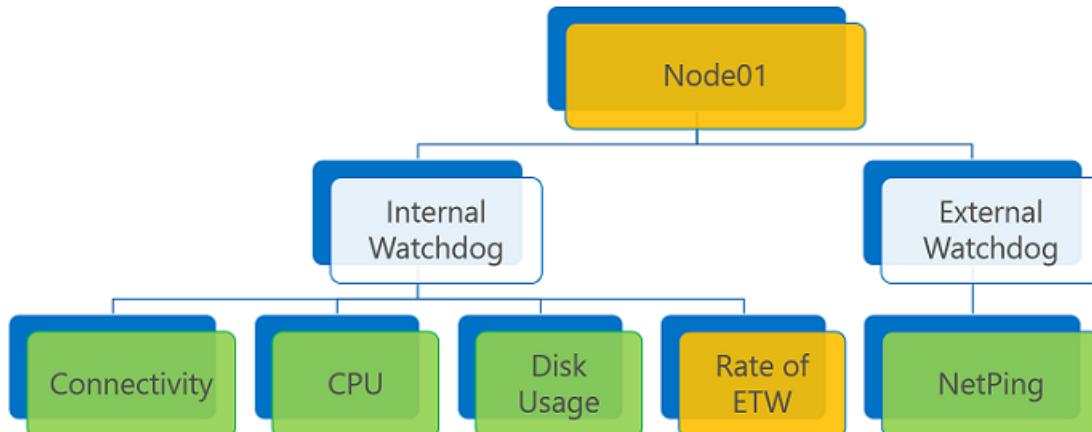
One entity can have multiple health reports sent by different reporters (system components or watchdogs) on different properties. The aggregation uses the associated health policies, in particular the ConsiderWarningAsError member of application or cluster health policy. ConsiderWarningAsError specifies how to evaluate warnings.

The aggregated health state is triggered by the *worst* health reports on the entity. If there is at least one error health report, the aggregated health state is an error.



A health entity that has one or more error health reports is evaluated as Error. The same is true for an expired health report, regardless of its health state.

If there are no error reports and one or more warnings, the aggregated health state is either warning or error, depending on the ConsiderWarningAsError policy flag.

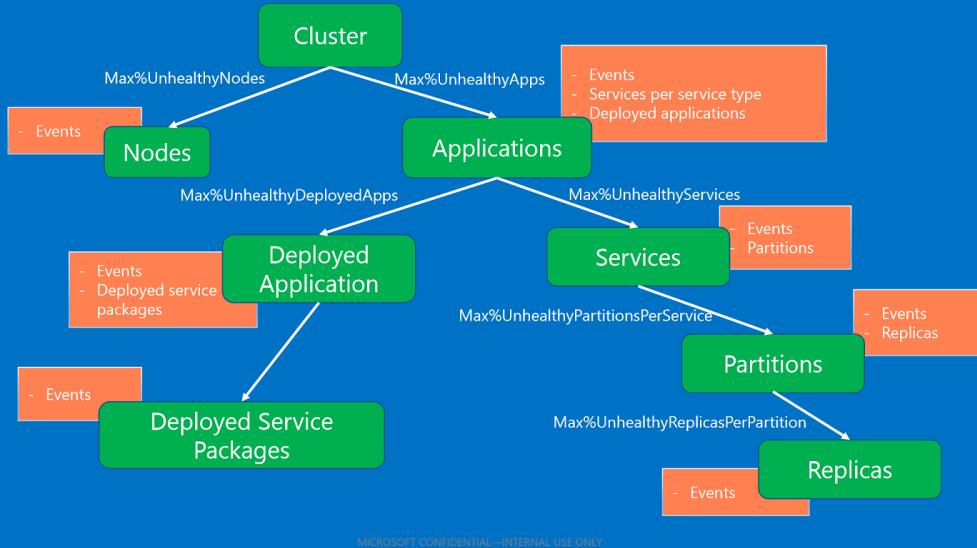


Health report aggregation with warning report and ConsiderWarningAsError set to false (default).

### Child health aggregation

The aggregated health state of an entity reflects the child health states (when applicable). The algorithm for aggregating child health states uses the health policies applicable based on the entity type.

# Entities Health Evaluation



Child aggregation based on health policies.

After the health store has evaluated all the children, it aggregates their health states based on the configured maximum percentage of unhealthy children. This percentage is taken from the policy based on the entity and child type.

- If all children have OK states, the child aggregated health state is OK.
- If children have both OK and warning states, the child aggregated health state is warning.
- If there are children with error states that do not respect the maximum allowed percentage of unhealthy children, the aggregated health state is an error.
- If the children with error states respect the maximum allowed percentage of unhealthy children, the aggregated health state is warning.

## Health reporting

System components, System Fabric applications, and internal/external watchdogs can report against Service Fabric entities. The reporters make *local* determinations of the health of the monitored entities, based on the conditions they are monitoring. They don't need to look at any global state or aggregate data. The desired behavior is to have simple reporters, and not complex organisms that need to look at many things to infer what information to send.

To send health data to the health store, a reporter needs to identify the affected entity and create a health report. To send the report, use the [FabricClient.HealthClient.ReportHealth](#) API, report health APIs exposed on the [Partition](#) or [CodePackageActivationContext](#) objects, PowerShell cmdlets, or REST.

### Health reports

The [health reports](#) for each of the entities in the cluster contain the following information:

- **Sourceld**. A string that uniquely identifies the reporter of the health event.
- **Entity identifier**. Identifies the entity where the report is applied. It differs based on the [entity type](#):
  - Cluster. None.
  - Node. Node name (string).
  - Application. Application name (URI). Represents the name of the application instance deployed in the cluster.
  - Service. Service name (URI). Represents the name of the service instance deployed in the cluster.

- Partition. Partition ID (GUID). Represents the partition unique identifier.
- Replica. The stateful service replica ID or the stateless service instance ID (INT64).
- DeployedApplication. Application name (URI) and node name (string).
- DeployedServicePackage. Application name (URI), node name (string), and service manifest name (string).
- **Property.** A *string* (not a fixed enumeration) that allows the reporter to categorize the health event for a specific property of the entity. For example, reporter A can report the health of the Node01 "Storage" property and reporter B can report the health of the Node01 "Connectivity" property. In the health store, these reports are treated as separate health events for the Node01 entity.
- **Description.** A string that allows a reporter to provide detailed information about the health event. **Sourceld**, **Property**, and **HealthState** should fully describe the report. The description adds human-readable information about the report. The text makes it easier for administrators and users to understand the health report.
- **HealthState.** An *enumeration* that describes the health state of the report. The accepted values are OK, Warning, and Error.
- **TimeToLive.** A timespan that indicates how long the health report is valid. Coupled with **RemoveWhenExpired**, it lets the health store know how to evaluate expired events. By default, the value is infinite, and the report is valid forever.
- **RemoveWhenExpired.** A Boolean. If set to true, the expired health report is automatically removed from the health store, and the report doesn't impact entity health evaluation. Used when the report is valid for a specified period of time only, and the reporter doesn't need to explicitly clear it out. It's also used to delete reports from the health store (for example, a watchdog is changed and stops sending reports with previous source and property). It can send a report with a brief TimeToLive along with RemoveWhenExpired to clear up any previous state from the health store. If the value is set to false, the expired report is treated as an error on the health evaluation. The false value signals to the health store that the source should report periodically on this property. If it doesn't, then there must be something wrong with the watchdog. The watchdog's health is captured by considering the event as an error.
- **SequenceNumber.** A positive integer that needs to be ever-increasing, it represents the order of the reports. It is used by the health store to detect stale reports that are received late because of network delays or other issues. A report is rejected if the sequence number is less than or equal to the most recently applied number for the same entity, source, and property. If it is not specified, the sequence number is generated automatically. It is necessary to put in the sequence number only when reporting on state transitions. In this situation, the source needs to remember which reports it sent and keep the information for recovery on failover.

These four pieces of information--Sourceld, entity identifier, Property, and HealthState--are required for every health report. The Sourceld string is not allowed to start with the prefix "**System.**", which is reserved for system reports. For the same entity, there is only one report for the same source and property. Multiple reports for the same source and property override each other, either on the health client side (if they are batched) or on the health store side. The replacement is based on sequence numbers; newer reports (with higher sequence numbers) replace older reports.

## Health events

Internally, the health store keeps [health events](#), which contain all the information from the reports, and additional metadata. The metadata includes the time the report was given to the health client and the time it was modified on the server side. The health events are returned by [health queries](#).

The added metadata contains:

- **SourceUtcTimestamp.** The time the report was given to the health client (Coordinated Universal Time).
- **LastModifiedUtcTimestamp.** The time the report was last modified on the server side (Coordinated Universal Time).
- **IsExpired.** A flag to indicate whether the report was expired when the query was executed by the health store. An event can be expired only if RemoveWhenExpired is false. Otherwise, the event is not returned by query and is removed from the store.

- **LastOkTransitionAt**, **LastWarningTransitionAt**, **LastErrorTransitionAt**. The last time for OK/warning/error transitions. These fields give the history of the health state transitions for the event.

The state transition fields can be used for smarter alerts or "historical" health event information. They enable scenarios such as:

- Alert when a property has been at warning/error for more than X minutes. Checking the condition for a period of time avoids alerts on temporary conditions. For example, an alert if the health state has been warning for more than five minutes can be translated into (`HealthState == Warning` and `Now - LastWarningTransitionTime > 5 minutes`).
- Alert only on conditions that have changed in the last X minutes. If a report was already at error before the specified time, it can be ignored because it was already signaled previously.
- If a property is toggling between warning and error, determine how long it has been unhealthy (that is, not OK). For example, an alert if the property hasn't been healthy for more than five minutes can be translated into (`HealthState != Ok` and `Now - LastOkTransitionTime > 5 minutes`).

## Example: Report and evaluate application health

The following example sends a health report through PowerShell on the application **fabric:/WordCount** from the source **MyWatchdog**. The health report contains information about the health property "availability" in an error health state, with infinite TimeToLive. Then it queries the application health, which returns aggregated health state errors and the reported health events in the list of health events.

```
PS C:\> Send-ServiceFabricApplicationHealthReport -ApplicationName fabric:/WordCount -SourceId "MyWatchdog" -HealthProperty "Availability" -HealthState Error
```

```
PS C:\> Get-ServiceFabricApplicationHealth fabric:/WordCount -ExcludeHealthStatistics
```

```
ApplicationName      : fabric:/WordCount
AggregatedHealthState : Error
UnhealthyEvaluations :
    :
        Error event: SourceId='MyWatchdog', Property='Availability'.
```

```
ServiceHealthStates :
    :
        ServiceName      : fabric:/WordCount/WordCountService
        AggregatedHealthState : Error
            :
                ServiceName      : fabric:/WordCount/WordCountWebService
                AggregatedHealthState : Ok
```

```
DeployedApplicationHealthStates :
    :
        ApplicationName      : fabric:/WordCount
        NodeName              : _Node_0
        AggregatedHealthState : Ok
            :
                ApplicationName      : fabric:/WordCount
                NodeName              : _Node_2
                AggregatedHealthState : Ok
                    :
                        ApplicationName      : fabric:/WordCount
                        NodeName              : _Node_3
                        AggregatedHealthState : Ok
                            :
                                ApplicationName      : fabric:/WordCount
                                NodeName              : _Node_4
                                AggregatedHealthState : Ok
                                    :
  ApplicationName      : fabric:/WordCount
  NodeName              : _Node_1
  AggregatedHealthState : Ok
```

```
HealthEvents :
    :
        SourceId      : System.CM
        Property       : State
        HealthState   : Ok
        SequenceNumber : 360
        SentAt        : 3/22/2016 7:56:53 PM
        ReceivedAt    : 3/22/2016 7:56:53 PM
        TTL           : Infinite
        Description    : Application has been created.
        RemoveWhenExpired : False
        IsExpired     : False
        Transitions   : Error->Ok = 3/22/2016 7:56:53 PM, LastWarning =
1/1/0001 12:00:00 AM
            :
                SourceId      : MyWatchdog
                Property       : Availability
                HealthState   : Error
                SequenceNumber : 131032204762818013
                SentAt        : 3/23/2016 3:27:56 PM
                ReceivedAt    : 3/23/2016 3:27:56 PM
                TTL           : Infinite
                Description    :
                RemoveWhenExpired : False
                IsExpired     : False
                Transitions   : Ok->Error = 3/23/2016 3:27:56 PM, LastWarning =
1/1/0001 12:00:00 AM
```

## Health model usage

The health model allows cloud services and the underlying Service Fabric platform to scale, because monitoring and health determinations are distributed among the different monitors within the cluster. Other systems have a single, centralized service at the cluster level that parses all the *potentially* useful information emitted by services. This approach hinders their scalability. It also doesn't allow them to collect specific information to help identify issues and potential issues as close to the root cause as possible.

The health model is used heavily for monitoring and diagnosis, for evaluating cluster and application health, and for monitored upgrades. Other services use health data to perform automatic repairs, build cluster health history, and issue alerts on certain conditions.

## Next steps

[View Service Fabric health reports](#)

[Use system health reports for troubleshooting](#)

[How to report and check service health](#)

[Add custom Service Fabric health reports](#)

[Monitor and diagnose services locally](#)

[Service Fabric application upgrade](#)

# Diagnostic functionality for Stateful Reliable Services

10/30/2017 • 1 min to read • [Edit Online](#)

The Azure Service Fabric Stateful Reliable Services StatefulServiceBase class emits [EventSource](#) events that can be used to debug the service, provide insights into how the runtime is operating, and help with troubleshooting.

## EventSource events

The EventSource name for the Stateful Reliable Services StatefulServiceBase class is "Microsoft-ServiceFabric-Services." Events from this event source appear in the [Diagnostics Events](#) window when the service is being [debugged in Visual Studio](#).

Examples of tools and technologies that help in collecting and/or viewing EventSource events are [PerfView](#), [Azure Diagnostics](#), and the [Microsoft TraceEvent Library](#).

## Events

Event Name	Event ID	Level	Event Description
StatefulRunAsyncInvocation	1	Informational	Emitted when the service RunAsync task is started
StatefulRunAsyncCancellation	2	Informational	Emitted when the service RunAsync task is canceled
StatefulRunAsyncCompletion	3	Informational	Emitted when the service RunAsync task is finished
StatefulRunAsyncSlowCancellation	4	Warning	Emitted when the service RunAsync task takes too long to complete cancellation
StatefulRunAsyncFailure	5	Error	Emitted when the service RunAsync task throws an exception

## Interpret events

StatefulRunAsyncInvocation, StatefulRunAsyncCompletion, and StatefulRunAsyncCancellation events are useful to the service writer to understand the lifecycle of a service, as well as the timing for when a service starts, cancels, or finishes. This information can be useful when debugging service issues or understanding the service lifecycle.

Service writers should pay close attention to StatefulRunAsyncSlowCancellation and StatefulRunAsyncFailure events because they indicate issues with the service.

StatefulRunAsyncFailure is emitted whenever the service RunAsync() task throws an exception. Typically, an exception thrown indicates an error or bug in the service. Additionally, the exception causes the service to fail, so it is moved to a different node. This operation can be expensive and can delay incoming requests while the service is moved. Service writers should determine the cause of the exception and, if possible, mitigate it.

StatefulRunAsyncSlowCancellation is emitted whenever a cancellation request for the RunAsync task takes longer

than four seconds. When a service takes too long to complete cancellation, it affects the ability of the service to be quickly restarted on another node. This scenario might affect the overall availability of the service.

## Next steps

[EventSource providers in PerfView](#)

# Diagnostics and performance monitoring for Reliable Actors

10/27/2017 • 8 min to read • [Edit Online](#)

The Reliable Actors runtime emits [EventSource](#) events and [performance counters](#). These provide insights into how the runtime is operating and help with troubleshooting and performance monitoring.

## EventSource events

The EventSource provider name for the Reliable Actors runtime is "Microsoft-ServiceFabric-Actors". Events from this event source appear in the [Diagnostics Events](#) window when the actor application is being [debugged in Visual Studio](#).

Examples of tools and technologies that help in collecting and/or viewing EventSource events are [PerfView](#), [Azure Diagnostics](#), [Semantic Logging](#), and the [Microsoft TraceEvent Library](#).

### Keywords

All events that belong to the Reliable Actors EventSource are associated with one or more keywords. This enables filtering of events that are collected. The following keyword bits are defined.

BIT	DESCRIPTION
0x1	Set of important events that summarize the operation of the Fabric Actors runtime.
0x2	Set of events that describe actor method calls. For more information, see the <a href="#">introductory topic on actors</a> .
0x4	Set of events related to actor state. For more information, see the topic on <a href="#">actor state management</a> .
0x8	Set of events related to turn-based concurrency in the actor. For more information, see the topic on <a href="#">concurrency</a> .

## Performance counters

The Reliable Actors runtime defines the following performance counter categories.

CATEGORY	DESCRIPTION
Service Fabric Actor	Counters specific to Azure Service Fabric actors, e.g. time taken to save actor state
Service Fabric Actor Method	Counters specific to methods implemented by Service Fabric actors, e.g. how often an actor method is invoked

Each of the above categories has one or more counters.

The [Windows Performance Monitor](#) application that is available by default in the Windows operating system can be used to collect and view performance counter data. [Azure Diagnostics](#) is another option for collecting performance

counter data and uploading it to Azure tables.

### Performance counter instance names

A cluster that has a large number of actor services or actor service partitions will have a large number of actor performance counter instances. The performance counter instance names can help in identifying the specific [partition](#) and actor method (if applicable) that the performance counter instance is associated with.

#### Service Fabric Actor category

For the category `Service Fabric Actor`, the counter instance names are in the following format:

`ServiceFabricPartitionID_ActorsRuntimeInternalID`

*ServiceFabricPartitionID* is the string representation of the Service Fabric partition ID that the performance counter instance is associated with. The partition ID is a GUID, and its string representation is generated through the [Guid.ToString](#) method with format specifier "D".

*ActorRuntimeInternalID* is the string representation of a 64-bit integer that is generated by the Fabric Actors runtime for its internal use. This is included in the performance counter instance name to ensure its uniqueness and avoid conflict with other performance counter instance names. Users should not try to interpret this portion of the performance counter instance name.

The following is an example of a counter instance name for a counter that belongs to the `Service Fabric Actor` category:

`2740af29-78aa-44bc-a20b-7e60fb783264_635650083799324046`

In the example above, `2740af29-78aa-44bc-a20b-7e60fb783264` is the string representation of the Service Fabric partition ID, and `635650083799324046` is the 64-bit ID that is generated for the runtime's internal use.

#### Service Fabric Actor Method category

For the category `Service Fabric Actor Method`, the counter instance names are in the following format:

`MethodName_ActorsRuntimeMethodId_ServiceFabricPartitionID_ActorsRuntimeInternalID`

*MethodName* is the name of the actor method that the performance counter instance is associated with. The format of the method name is determined based on some logic in the Fabric Actors runtime that balances the readability of the name with constraints on the maximum length of the performance counter instance names on Windows.

*ActorsRuntimeMethodId* is the string representation of a 32-bit integer that is generated by the Fabric Actors runtime for its internal use. This is included in the performance counter instance name to ensure its uniqueness and avoid conflict with other performance counter instance names. Users should not try to interpret this portion of the performance counter instance name.

*ServiceFabricPartitionID* is the string representation of the Service Fabric partition ID that the performance counter instance is associated with. The partition ID is a GUID, and its string representation is generated through the [Guid.ToString](#) method with format specifier "D".

*ActorRuntimeInternalID* is the string representation of a 64-bit integer that is generated by the Fabric Actors runtime for its internal use. This is included in the performance counter instance name to ensure its uniqueness and avoid conflict with other performance counter instance names. Users should not try to interpret this portion of the performance counter instance name.

The following is an example of a counter instance name for a counter that belongs to the `Service Fabric Actor Method` category:

`ivoicemailboxactor.leavemessageasync_2_89383d32-e57e-4a9b-a6ad-57c6792aa521_635650083804480486`

In the example above, `ivoicemailboxactor.leavemessageasync` is the method name, `2` is the 32-bit ID generated for the runtime's internal use, `89383d32-e57e-4a9b-a6ad-57c6792aa521` is the string representation of the Service Fabric

partition ID, and `635650083804480486` is the 64-bit ID generated for the runtime's internal use.

## List of events and performance counters

### Actor method events and performance counters

The Reliable Actors runtime emits the following events related to [actor methods](#).

Event Name	Event ID	Level	Keyword	Description
ActorMethodStart	7	Verbose	0x2	Actors runtime is about to invoke an actor method.
ActorMethodStop	8	Verbose	0x2	An actor method has finished executing. That is, the runtime's asynchronous call to the actor method has returned, and the task returned by the actor method has finished.
ActorMethodThrewException	9	Warning	0x3	An exception was thrown during the execution of an actor method, either during the runtime's asynchronous call to the actor method or during the execution of the task returned by the actor method. This event indicates some sort of failure in the actor code that needs investigation.

The Reliable Actors runtime publishes the following performance counters related to the execution of actor methods.

Category Name	Counter Name	Description
Service Fabric Actor Method	Invocations/Sec	Number of times that the actor service method is invoked per second
Service Fabric Actor Method	Average milliseconds per invocation	Time taken to execute the actor service method in milliseconds
Service Fabric Actor Method	Exceptions thrown/Sec	Number of times that the actor service method threw an exception per second

### Concurrency events and performance counters

The Reliable Actors runtime emits the following events related to [concurrency](#).

Event Name	Event ID	Level	Keyword	Description
ActorMethodCallsWaitingForLock	12	Verbose	0x8	This event is written at the start of each new turn in an actor. It contains the number of pending actor calls that are waiting to acquire the per-actor lock that enforces turn-based concurrency.

The Reliable Actors runtime publishes the following performance counters related to concurrency.

Category Name	Counter Name	Description
Service Fabric Actor	# of actor calls waiting for actor lock	Number of pending actor calls waiting to acquire the per-actor lock that enforces turn-based concurrency
Service Fabric Actor	Average milliseconds per lock wait	Time taken (in milliseconds) to acquire the per-actor lock that enforces turn-based concurrency
Service Fabric Actor	Average milliseconds actor lock held	Time (in milliseconds) for which the per-actor lock is held

## Actor state management events and performance counters

The Reliable Actors runtime emits the following events related to [actor state management](#).

Event Name	Event ID	Level	Keyword	Description
ActorSaveStateStart	10	Verbose	0x4	Actors runtime is about to save the actor state.
ActorSaveStateStop	11	Verbose	0x4	Actors runtime has finished saving the actor state.

The Reliable Actors runtime publishes the following performance counters related to actor state management.

Category Name	Counter Name	Description
Service Fabric Actor	Average milliseconds per save state operation	Time taken to save actor state in milliseconds
Service Fabric Actor	Average milliseconds per load state operation	Time taken to load actor state in milliseconds

## Events related to actor replicas

The Reliable Actors runtime emits the following events related to [actor replicas](#).

Event Name	Event ID	Level	Keyword	Description
ReplicaChangeRoleToPrimary	1	Informational	0x1	Actor replica changed role to Primary. This implies that the actors for this partition will be created inside this replica.
ReplicaChangeRoleFromPrimary	2	Informational	0x1	Actor replica changed role to non-Primary. This implies that the actors for this partition will no longer be created inside this replica. No new requests will be delivered to actors already created within this replica. The actors will be destroyed after any in-progress requests are completed.

### Actor activation and deactivation events and performance counters

The Reliable Actors runtime emits the following events related to [actor activation and deactivation](#).

Event Name	Event ID	Level	Keyword	Description
ActorActivated	5	Informational	0x1	An actor has been activated.
ActorDeactivated	6	Informational	0x1	An actor has been deactivated.

The Reliable Actors runtime publishes the following performance counters related to actor activation and deactivation.

Category Name	Counter Name	Description
Service Fabric Actor	Average OnActivateAsync milliseconds	Time taken to execute OnActivateAsync method in milliseconds

### Actor request processing performance counters

When a client invokes a method via an actor proxy object, it results in a request message being sent over the network to the actor service. The service processes the request message and sends a response back to the client. The Reliable Actors runtime publishes the following performance counters related to actor request processing.

Category Name	Counter Name	Description
Service Fabric Actor	# of outstanding requests	Number of requests being processed in the service
Service Fabric Actor	Average milliseconds per request	Time taken (in milliseconds) by the service to process a request

CATEGORY NAME	COUNTER NAME	DESCRIPTION
Service Fabric Actor	Average milliseconds for request deserialization	Time taken (in milliseconds) to deserialize actor request message when it is received at the service
Service Fabric Actor	Average milliseconds for response serialization	Time taken (in milliseconds) to serialize the actor response message at the service before the response is sent to the client

## Next steps

- [How Reliable Actors use the Service Fabric platform](#)
- [Actor API reference documentation](#)
- [Sample code](#)
- [EventSource providers in PerfView](#)

# Diagnostics and performance monitoring for Reliable Service Remoting

7/31/2017 • 4 min to read • [Edit Online](#)

The Reliable ServiceRemoting runtime emits [performance counters](#). These provide insights into how the ServiceRemoting is operating and help with troubleshooting and performance monitoring.

## Performance counters

The Reliable ServiceRemoting runtime defines the following performance counter categories:

CATEGORY	DESCRIPTION
Service Fabric Service	Counters specific to Azure Service Fabric Service Remoting , for example, average time taken to process request
Service Fabric Service Method	Counters specific to methods implemented by Service Fabric Remoting Service, for example, how often a service method is invoked

Each of the preceding categories has one or more counters.

The [Windows Performance Monitor](#) application that is available by default in the Windows operating system can be used to collect and view performance counter data. [Azure Diagnostics](#) is another option for collecting performance counter data and uploading it to Azure tables.

### Performance counter instance names

A cluster that has a large number of ServiceRemoting services or partitions have a large number of performance counter instances. The performance counter instance names can help in identifying the specific partition and Service method (if applicable) that the performance counter instance is associated with.

#### Service Fabric Service category

For the category `Service Fabric Service`, the counter instance names are in the following format:

`ServiceFabricPartitionID_ServiceReplicaOrInstanceId_ServiceRuntimeInternalID`

*ServiceFabricPartitionID* is the string representation of the Service Fabric partition ID that the performance counter instance is associated with. The partition ID is a GUID, and its string representation is generated through the `Guid.ToString` method with format specifier "D".

*ServiceReplicaOrInstanceId* is the string representation of the Service Fabric Replica/Instance ID that the performance counter instance is associated with.

*ServiceRuntimeInternalID* is the string representation of a 64-bit integer that is generated by the Fabric Service runtime for its internal use. This is included in the performance counter instance name to ensure its uniqueness and avoid conflict with other performance counter instance names. Users should not try to interpret this portion of the performance counter instance name.

The following is an example of a counter instance name for a counter that belongs to the `Service Fabric Service` category:

`2740af29-78aa-44bc-a20b-7e60fb783264_635650083799324046_5008379932`

In the preceding example, `2740af29-78aa-44bc-a20b-7e60fb783264` is the string representation of the Service Fabric partition ID, `635650083799324046` is string representation of Replica/InstanceID and `5008379932` is the 64-bit ID that is generated for the runtime's internal use.

#### Service Fabric Service Method category

For the category `Service Fabric Service Method`, the counter instance names are in the following format:

`MethodName_ServiceRuntimeMethodId_ServiceFabricPartitionID_ServiceReplicaOrInstanceId_ServiceRuntimeInternalID`

*MethodName* is the name of the service method that the performance counter instance is associated with. The format of the method name is determined based on some logic in the Fabric Service runtime that balances the readability of the name with constraints on the maximum length of the performance counter instance names on Windows.

*ServiceRuntimeMethodId* is the string representation of a 32-bit integer that is generated by the Fabric Service runtime for its internal use. This is included in the performance counter instance name to ensure its uniqueness and avoid conflict with other performance counter instance names. Users should not try to interpret this portion of the performance counter instance name.

*ServiceFabricPartitionID* is the string representation of the Service Fabric partition ID that the performance counter instance is associated with. The partition ID is a GUID, and its string representation is generated through the `Guid.ToString` method with format specifier "D".

*ServiceReplicaOrInstanceId* is the string representation of the Service Fabric Replica/Instance ID that the performance counter instance is associated with.

*ServiceRuntimeInternalID* is the string representation of a 64-bit integer that is generated by the Fabric Service runtime for its internal use. This is included in the performance counter instance name to ensure its uniqueness and avoid conflict with other performance counter instance names. Users should not try to interpret this portion of the performance counter instance name.

The following is an example of a counter instance name for a counter that belongs to the

`Service Fabric Service Method` category:

`ivoicemailboxservice.leavemessageasync_2_89383d32-e57e-4a9b-a6ad-57c6792aa521_635650083804480486_5008380`

In the preceding example, `ivoicemailboxservice.leavemessageasync` is the method name, `2` is the 32-bit ID generated for the runtime's internal use, `89383d32-e57e-4a9b-a6ad-57c6792aa521` is the string representation of the Service Fabric partition ID, `635650083804480486` is the string representation of the Service Fabric Replica/Instance ID and `5008380` is the 64-bit ID generated for the runtime's internal use.

## List of Performance counters

### Service method performance counters

The Reliable Service runtime publishes the following performance counters related to the execution of service methods.

CATEGORY NAME	COUNTER NAME	DESCRIPTION
Service Fabric Service Method	Invocations/Sec	Number of times that the service method is invoked per second
Service Fabric Service Method	Average milliseconds per invocation	Time taken to execute the service method in milliseconds

CATEGORY NAME	COUNTER NAME	DESCRIPTION
Service Fabric Service Method	Exceptions thrown/Sec	Number of times that the service method threw an exception per second

### Service request processing performance counters

When a client invokes a method via a service proxy object, it results in a request message being sent over the network to the remoting service. The service processes the request message and sends a response back to the client. The Reliable ServiceRemoting runtime publishes the following performance counters related to service request processing.

CATEGORY NAME	COUNTER NAME	DESCRIPTION
Service Fabric Service	# of outstanding requests	Number of requests being processed in the service
Service Fabric Service	Average milliseconds per request	Time taken (in milliseconds) by the service to process a request
Service Fabric Service	Average milliseconds for request deserialization	Time taken (in milliseconds) to deserialize service request message when it is received at the service
Service Fabric Service	Average milliseconds for response serialization	Time taken (in milliseconds) to serialize the service response message at the service before the response is sent to the client

## Next steps

- [Sample code](#)
- [EventSource providers in PerfView](#)

# Prepare your development environment on Windows

11/1/2017 • 2 min to read • [Edit Online](#)

To build and run [Azure Service Fabric applications](#) on your Windows development machine, install the runtime, SDK, and tools. You also need to enable execution of the Windows PowerShell scripts included in the SDK.

## Prerequisites

### Supported operating system versions

The following operating system versions are supported for development:

- Windows 7
- Windows 8/Windows 8.1
- Windows Server 2012 R2
- Windows Server 2016
- Windows 10

#### NOTE

Windows 7 only includes Windows PowerShell 2.0 by default. Service Fabric PowerShell cmdlets requires PowerShell 3.0 or higher. You can [download Windows PowerShell 5.0](#) from the Microsoft Download Center.

## Install the SDK and tools

### To use Visual Studio 2017

Service Fabric Tools are part of the Azure Development and Management workload in Visual Studio 2017. Enable this workload as part of your Visual Studio installation. In addition, you need to install the Microsoft Azure Service Fabric SDK, using Web Platform Installer.

- [Install the Microsoft Azure Service Fabric SDK](#)

### To use Visual Studio 2015 (requires Visual Studio 2015 Update 2 or later)

For Visual Studio 2015, Service Fabric tools are installed together with the SDK, using the Web Platform Installer:

- [Install the Microsoft Azure Service Fabric SDK and Tools](#)

### SDK installation only

If you only need the SDK, you can install this package:

- [Install the Microsoft Azure Service Fabric SDK](#)

The current versions are:

- Service Fabric SDK 2.8.219
- Service Fabric runtime 6.0.219
- Service Fabric Tools for Visual Studio 2015 1.8.50927.3

- Visual Studio 2017 Update 3 includes Service Fabric Tools for Visual Studio 1.7.20170817
- Visual Studio 2017 Update 4 Preview 1 (15.4.0 Preview 1.0) includes Service Fabric Tools for Visual Studio 1.7.20170721

For a list of supported versions, see [Service Fabric support](#)

## Enable PowerShell script execution

Service Fabric uses Windows PowerShell scripts for creating a local development cluster and for deploying applications from Visual Studio. By default, Windows blocks these scripts from running. To enable them, you must modify your PowerShell execution policy. Open PowerShell as an administrator and enter the following command:

```
Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Force -Scope CurrentUser
```

## Next steps

Now that you've finished setting up your development environment, start building and running apps.

- [Create your first Service Fabric application in Visual Studio](#)
- [Learn how to deploy and manage applications on your local cluster](#)
- [Learn about the programming models: Reliable Services and Reliable Actors](#)
- [Check out the Service Fabric code samples on GitHub](#)
- [Visualize your cluster by using Service Fabric Explorer](#)
- [Follow the Service Fabric learning path to get a broad introduction to the platform](#)
- Learn about [Service Fabric support options](#)

# Prepare your development environment on Linux

9/25/2017 • 5 min to read • [Edit Online](#)

To deploy and run [Azure Service Fabric applications](#) on your Linux development machine, install the runtime and common SDK. You can also install optional SDKs for Java and .NET Core development.

## Prerequisites

The following operating system versions are supported for development:

- Ubuntu 16.04 (Xenial Xerus)

## Installation Methods

### 1. Script installation

A script is provided for convenience for installing the Service Fabric runtime and the Service Fabric common SDK along with **sfctl** CLI. Run the manual installation steps in the next section to determine what is being installed and the licenses that are being agreed to. Running the script assumes you agree to the licenses for all the software that is being installed.

After the script is executed successfully, you can directly skip to [Set up a local cluster](#).

```
sudo curl -s https://raw.githubusercontent.com/Azure/service-fabric-scripts-and-templates/master/scripts/SetupServiceFabric/SetupServiceFabric.sh | sudo bash
```

### 2. Manual Installation

For manual installation of Service Fabric runtime and common SDK, follow the rest of this guide.

## Update your APT sources

To install the SDK and the associated runtime package via the apt-get command-line tool, you must first update your Advanced Packaging Tool (APT) sources.

1. Open a terminal.
2. Add the Service Fabric repo to your sources list.

```
sudo sh -c 'echo "deb [arch=amd64] http://apt-mo.trafficmanager.net/repos/servicefabric/ xenial main" > /etc/apt/sources.list.d/servicefabric.list'
```

3. Add the dotnet repo to your sources list.

```
sudo sh -c 'echo "deb [arch=amd64] https://apt-mo.trafficmanager.net/repos/dotnet-release/ xenial main" > /etc/apt/sources.list.d/dotnetdev.list'
```

4. Add the new Gnu Privacy Guard (GnuPG, or GPG) key to your APT keyring.

```
sudo apt-key adv --keyserver apt-mo.trafficmanager.net --recv-keys 417A0893  
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 417A0893
```

5. Add the official Docker GPG key to your APT keyring.

```
sudo apt-get install curl  
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

6. Set up the Docker repository.

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs)  
stable"
```

7. Refresh your package lists based on the newly added repositories.

```
sudo apt-get update
```

## Install and set up the Service Fabric SDK for local cluster setup

After you have updated your sources, you can install the SDK. Install the Service Fabric SDK package, confirm the installation, and agree to the license agreement.

```
sudo apt-get install servicefabricsdkcommon
```

### TIP

The following commands automate accepting the license for Service Fabric packages:

```
echo "servicefabric servicefabric/accepted-eula-ga select true" | sudo debconf-set-selections  
echo "servicefabricsdkcommon servicefabricsdkcommon/accepted-eula-ga select true" | sudo debconf-set-  
selections
```

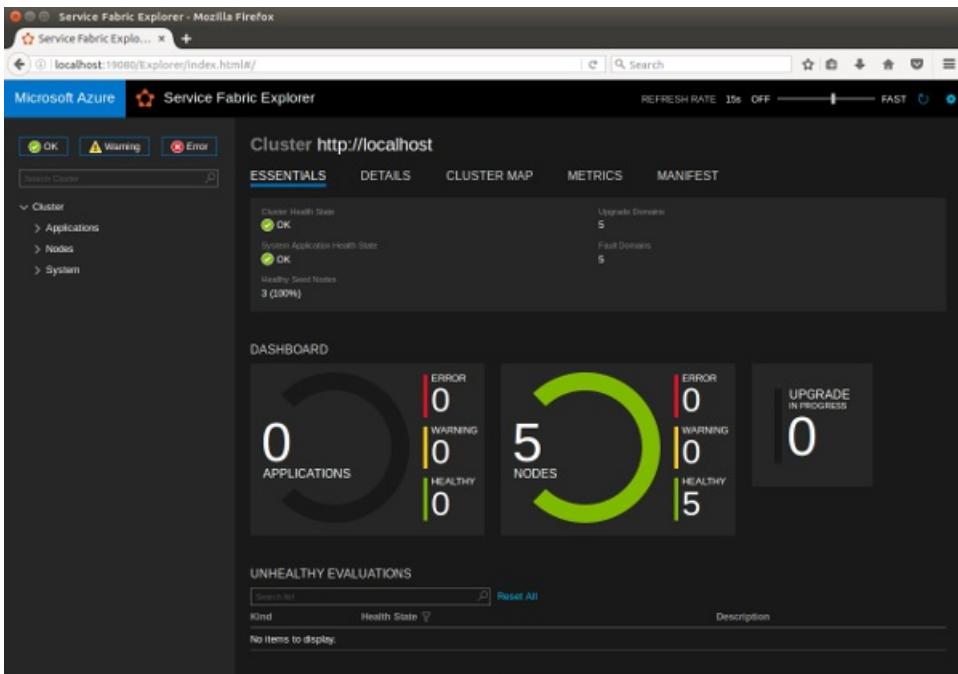
## Set up a local cluster

Once the installation completes, you should be able to start a local cluster.

1. Run the cluster setup script.

```
sudo /opt/microsoft/sdk/servicefabric/common/clustersetup/devclustersetup.sh
```

2. Open a web browser and go to [Service Fabric Explorer](#). If the cluster has started, you should see the Service Fabric Explorer dashboard.



At this point, you can deploy pre-built Service Fabric application packages or new ones based on guest containers or guest executables. To build new services by using the Java or .NET Core SDKs, follow the optional setup steps that are provided in subsequent sections.

#### NOTE

Standalone clusters aren't supported in Linux.

## Set up the Service Fabric CLI

The [Service Fabric CLI](#) has commands for interacting with Service Fabric entities, including clusters and applications. Please follow the instructions at [Service Fabric CLI](#) to install the CLI.

## Set up Yeoman generators for containers and guest executables

Service Fabric provides scaffolding tools which will help you create Service Fabric applications from a terminal using Yeoman template generators. Follow these steps to set up the Service Fabric Yeoman template generators:

1. Install nodejs and NPM on your machine

```
sudo apt-get install npm  
sudo apt install nodejs-legacy
```

2. Install [Yeoman](#) template generator on your machine from NPM

```
sudo npm install -g yo
```

3. Install the Service Fabric Yeo container generator and guest executable generator from NPM

```
sudo npm install -g generator-azuresfcontainer # for Service Fabric container application  
sudo npm install -g generator-azuresfguest # for Service Fabric guest executable application
```

After you have installed the generators, you should be able to create guest executable or container services by running `yo azuresfguest` or `yo azuresfcontainer`, respectively.

## Set up .NET Core 2.0 development

Install the [.NET Core 2.0 SDK for Ubuntu](#) to start [creating C# Service Fabric applications](#). Packages for .NET Core 2.0 Service Fabric applications are hosted on NuGet.org, currently in preview.

## Set up Java development

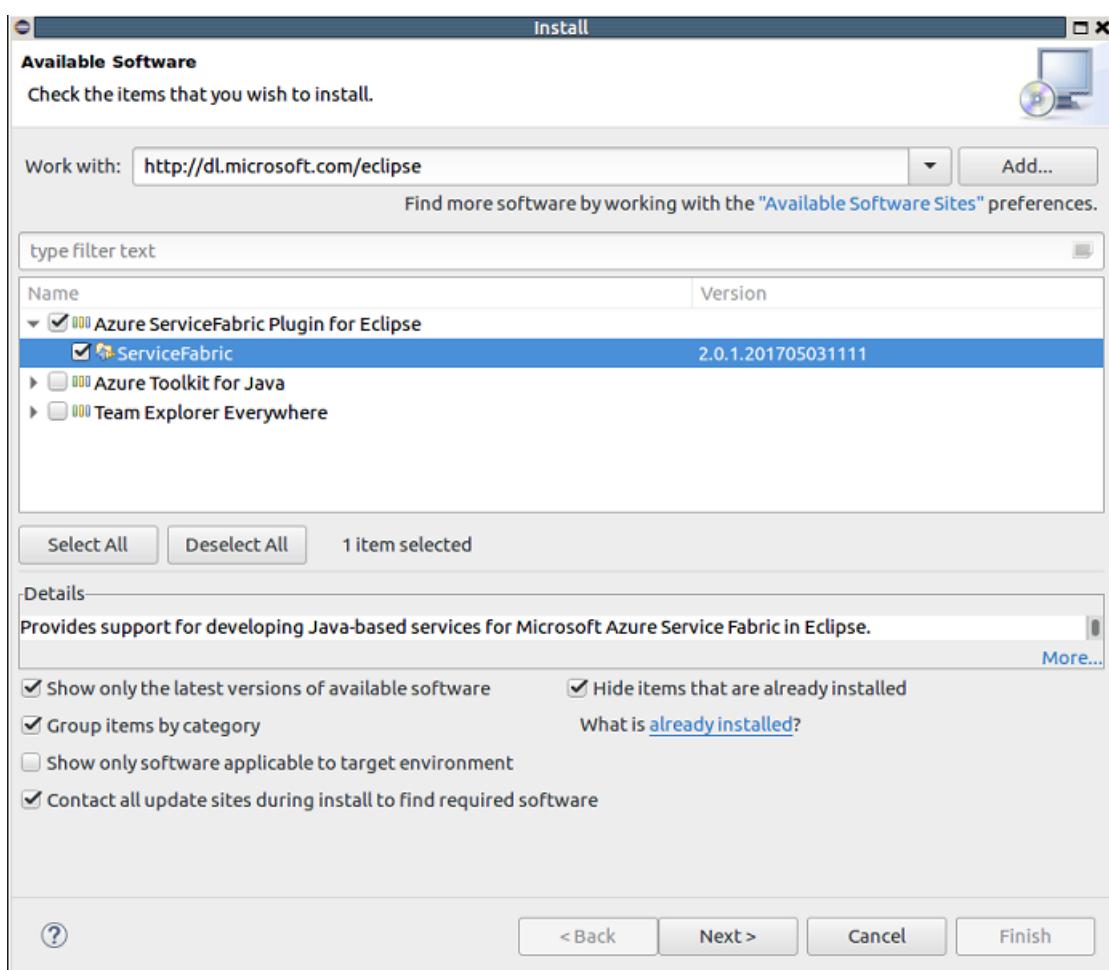
To build Service Fabric services using Java, install JDK 1.8 and Gradle to run build tasks. The following snippet installs Open JDK 1.8 along with Gradle. The Service Fabric Java libraries are pulled from Maven.

```
sudo apt-get install openjdk-8-jdk-headless  
sudo apt-get install gradle
```

## Install the Eclipse Neon plug-in (optional)

You can install the Eclipse plug-in for Service Fabric from within the Eclipse IDE for Java Developers. You can use Eclipse to create Service Fabric guest executable applications and container applications in addition to Service Fabric Java applications.

1. In Eclipse, ensure that you have latest Eclipse Neon and the latest Buildship version (1.0.17 or later) installed. You can check the versions of installed components by selecting **Help > Installation Details**. You can update Buildship by using the instructions at [Eclipse Buildship: Eclipse Plug-ins for Gradle](#).
2. To install the Service Fabric plug-in, select **Help > Install New Software**.
3. In the **Work with** box, type <http://dl.microsoft.com/eclipse>.
4. Click **Add**.



5. Select the **ServiceFabric** plug-in, and then click **Next**.
6. Complete the installation steps, and then accept the end-user license agreement.

If you already have the Service Fabric Eclipse plug-in installed, make sure that you have the latest version. You can check by selecting **Help > Installation Details** and then searching for Service Fabric in the list of installed plug-ins. If a newer version is available, select **Update**.

For more information, see [Service Fabric plug-in for Eclipse Java application development](#).

## Update the SDK and runtime

To update to the latest version of the SDK and runtime, run the following commands:

```
sudo apt-get update  
sudo apt-get install servicefabric servicefabricsdkcommon
```

To update the Java SDK binaries from Maven, you need to update the version details of the corresponding binary in the `build.gradle` file to point to the latest version. To know exactly where you need to update the version, you can refer to any `build.gradle` file in Service Fabric getting-started samples [here](#).

### NOTE

Updating the packages might cause your local development cluster to stop running. Restart your local cluster after an upgrade by following the instructions on this page.

## Remove the SDK

To remove the Service Fabric SDKs, run the following:

```
sudo apt-get remove servicefabric servicefabricsdkcommon  
sudo npm uninstall generator-azuresfcontainer  
sudo npm uninstall generator-azuresfguest  
sudo apt-get install -f
```

## Next steps

- [Create and deploy your first Service Fabric Java application on Linux by using Yeoman](#)
- [Create and deploy your first Service Fabric Java application on Linux by using Service Fabric Plugin for Eclipse](#)
- [Create your first CSharp application on Linux](#)
- [Prepare your development environment on OSX](#)
- [Use the Service Fabric CLI to manage your applications](#)
- [Service Fabric Windows/Linux differences](#)
- [Get started with Service Fabric CLI](#)

# Set up your development environment on Mac OS X

10/3/2017 • 4 min to read • [Edit Online](#)

You can build Service Fabric applications to run on Linux clusters using Mac OS X. This article covers how to set up your Mac for development.

## Prerequisites

Service Fabric does not run natively on OS X. To run a local Service Fabric cluster, we provide a pre-configured Ubuntu virtual machine using Vagrant and VirtualBox. Before you get started, you need:

- [Vagrant \(v1.8.4 or later\)](#)
- [VirtualBox](#)

### NOTE

You need to use mutually supported versions of Vagrant and VirtualBox. Vagrant might behave erratically on an unsupported VirtualBox version.

## Create the local VM

To create the local VM containing a 5-node Service Fabric cluster, perform the following steps:

1. Clone the `Vagrantfile` repo

```
git clone https://github.com/azure/service-fabric-linux-vagrant-onebox.git
```

This step downloads the file `Vagrantfile` containing the VM configuration along with the location the VM is downloaded from. The file points to a stock Ubuntu image.

2. Navigate to the local clone of the repo

```
cd service-fabric-linux-vagrant-onebox
```

3. (Optional) Modify the default VM settings

By default, the local VM is configured as follows:

- 3 GB of memory allocated
- Private host network configured at IP 192.168.50.50 enabling passthrough of traffic from the Mac host

You can change either of these settings or add other configuration to the VM in the `Vagrantfile`. See the [Vagrant documentation](#) for the full list of configuration options.

4. Create the VM

```
vagrant up
```

5. Log into the VM and install the Service Fabric SDK

```
vagrant ssh
```

Install the SDK as described in [SDK installation](#). The script below is provided for convenience for installing the Service Fabric runtime and the Service Fabric common SDK along with sfctl CLI. Running the script assumes you have read and agreed to the licenses for all the software that is being installed.

```
sudo curl -s https://raw.githubusercontent.com/Azure/service-fabric-scripts-and-templates/master/scripts/SetupServiceFabric/SetupServiceFabric.sh | sudo bash
```

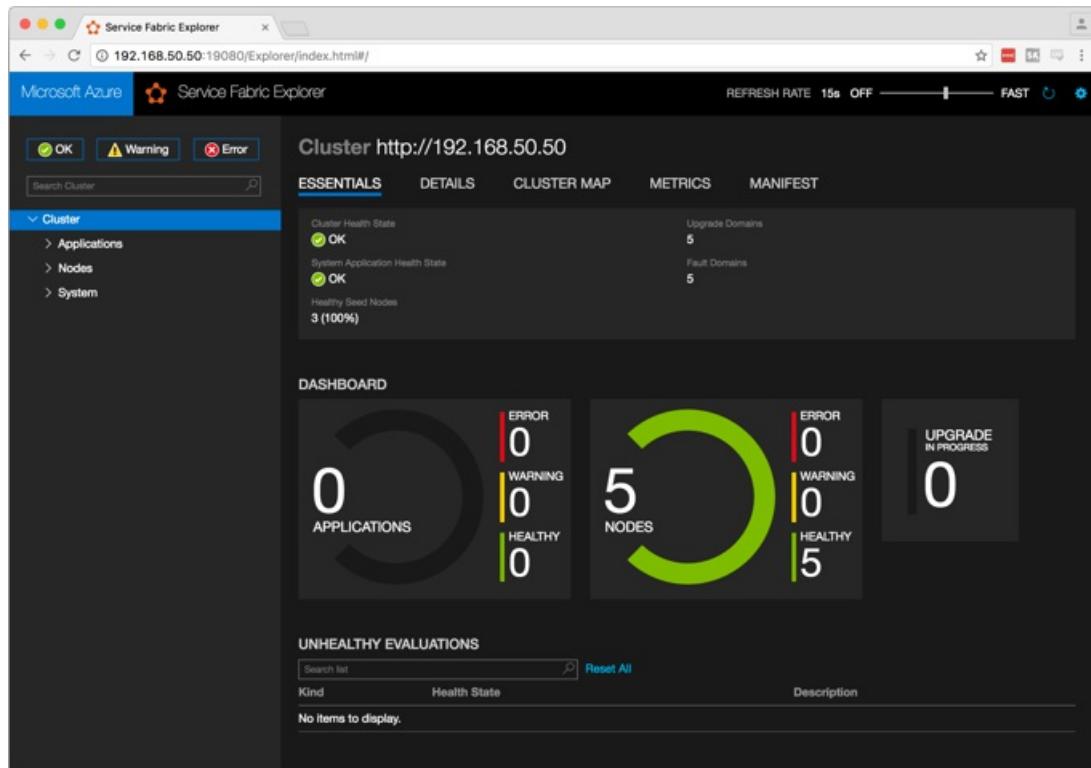
## 6. Start the Service Fabric cluster

```
sudo /opt/microsoft/sdk/servicefabric/common/clustersetup/devclustersetup.sh
```

### TIP

If the VM download is taking a long time, you can download it using wget or curl or through a browser by navigating to the link specified by **config.vm.box\_url** in the file `Vagrantfile`. After downloading it locally, edit `Vagrantfile` to point to the local path where you downloaded the image. For example if you downloaded the image to `/home/users/test/azureservicefabric.tp8.box`, then set **config.vm.box\_url** to that path.

## 7. Test that the cluster has been set up correctly by navigating to Service Fabric Explorer at <http://192.168.50.50:19080/Explorer> (assuming you kept the default private network IP).



## Install the necessary Java artifacts on Vagrant to use Service Fabric Java programming model

To build Service Fabric services using Java, ensure you have JDK 1.8 installed along with Gradle which is used for running build tasks. The following snippet installs Open JDK 1.8 along with Gradle. The Service Fabric Java libraries are pulled from Maven.

```
vagrant ssh  
sudo apt-get install openjdk-8-jdk-headless  
sudo apt-get install gradle
```

## Set up the Service Fabric CLI (sfctl) on your Mac

Follow the instructions at [Service Fabric CLI](#) to install the Service Fabric CLI (`sfctl`) on your Mac. The CLI commands for interacting with Service Fabric entities, including clusters, applications and services.

## Create application on you Mac using Yeoman

Service Fabric provides scaffolding tools which will help you create a Service Fabric application from terminal using Yeoman template generator. Please follow the steps below to ensure you have the Service Fabric yeoman template generator working on your machine.

1. You need to have Node.js and NPM installed on you mac. If not you can install Node.js and NPM using Homebrew using the following. To check the versions of Node.js and NPM installed on your Mac, you can use the `-v` option.

```
brew install node  
node -v  
npm -v
```

2. Install [Yeoman](#) template generator on your machine from NPM

```
npm install -g yo
```

3. Install the Yeoman generator you want to use, following the steps in the getting started [documentation](#). To create Service Fabric Applications using Yeoman, follow the steps -

```
npm install -g generator-azuresfjava      # for Service Fabric Java Applications  
npm install -g generator-azuresfguest    # for Service Fabric Guest executables  
npm install -g generator-azuresfcontainer # for Service Fabric Container Applications
```

4. To build a Service Fabric Java application on Mac, you would need - JDK 1.8 and Gradle installed on the machine.

## Set up .NET Core 2.0 development

Install the [.NET Core 2.0 SDK for Mac](#) to start [creating C# Service Fabric applications](#). Packages for .NET Core 2.0 Service Fabric applications are hosted on NuGet.org, currently in preview.

## Install the Service Fabric plugin for Eclipse Neon

Service Fabric provides a plugin for the [Eclipse Neon for Java IDE](#) that can simplify the process of creating, building, and deploying Java services. You can follow the installation steps mentioned in this general [documentation](#) about installing or updating Service Fabric Eclipse plugin.

**TIP**

By default we support the default IP as mentioned in the `Vagrantfile` in the `Local.json` of the generated application. In case you change that and deploy Vagrant with a different IP, please update the corresponding IP in `Local.json` of your application as well.

## Next steps

- [Create and deploy your first Service Fabric Java application on Linux using Yeoman](#)
- [Create and deploy your first Service Fabric Java application on Linux using Service Fabric Plugin for Eclipse](#)
- [Create a Service Fabric cluster in the Azure portal](#)
- [Create a Service Fabric cluster using the Azure Resource Manager](#)
- [Understand the Service Fabric application model](#)
- [Use the Service Fabric CLI to manage your applications](#)

# Azure Service Fabric CLI

10/20/2017 • 5 min to read • [Edit Online](#)

The Azure Service Fabric command-line interface (CLI) is a command-line utility for interacting with and managing Service Fabric entities. The Service Fabric CLI can be used with either Windows or Linux clusters. The Service Fabric CLI runs on any platform where Python is supported.

## IMPORTANT

There are two CLI utilities used to interact with Service Fabric. [Azure CLI](#) is used to manage Azure resources, such as an Azure-hosted Service Fabric cluster. [Service Fabric CLI](#) is used to directly connect to the Service Fabric cluster (regardless of where it's hosted) and manage the cluster, applications, and services.

## Prerequisites

Prior to installation, make sure your environment has both Python and pip installed. For more information, see the [pip quickstart documentation](#) and the official [Python installation documentation](#).

The CLI supports Python versions 2.7, 3.5 and 3.6. Python 3.6 is the recommended version, since Python 2.7 will reach end of support soon.

### Service Fabric target runtime

The Service Fabric CLI is meant to support the latest runtime version of the Service Fabric SDK. Use the following table to determine which version of CLI to install:

CLI VERSION	SUPPORTED RUNTIME VERSION
Latest (~=3)	Latest (~=6.0)
1.1.0	5.6, 5.7

You can optionally specify a target version of the CLI to install by suffixing the `pip install` command with `==<version>`. For example, for version 1.1.0 the syntax would be:

```
pip install -I sfctl==1.1.0
```

Replace the following `pip install` command with the previously mentioned command when necessary.

For more information on Service Fabric CLI releases, see the [GitHub documentation](#).

## Install pip, Python, and the Service Fabric CLI

There are many ways to install pip and Python on your platform. Here are some steps to get major operating systems set up quickly with Python 3 and pip.

### Windows

For Windows 10, Windows Server 2016, and Windows Server 2012 R2, use the standard official installation instructions. The Python installer also installs pip by default.

1. Go to the official [Python downloads page](#), and download the latest release of Python 3.6.

2. Start the installer.
3. At the bottom of the prompt, select **Add Python 3.6 to PATH**.
4. Select **Install Now**, and finish the installation.

Now you can open a new command window and get the version of both Python and pip.

```
python --version  
pip --version
```

Then run the following command to install the Service Fabric CLI:

```
pip install sfctl  
sfctl -h
```

### Ubuntu and Windows subsystem for Linux

To install the Service Fabric CLI, run the following commands:

```
sudo apt-get install python3  
sudo apt-get install python3-pip  
pip3 install sfctl
```

Then you can test the installation with:

```
sfctl -h
```

If you receive a command not found error such as:

```
sfctl: command not found
```

Be sure that `~/.local/bin` is accessible from the `$PATH`:

```
export PATH=$PATH:~//.local/bin  
echo "export PATH=$PATH:~//.local/bin" >> .bashrc
```

If the installation on Windows subsystem for Linux fails with incorrect folder permissions, it may be necessary to try again with elevated permissions:

```
sudo pip3 install sfctl
```

### MacOS

For MacOS, we recommend that you use the [HomeBrew package manager](#). If HomeBrew is not already installed, install it by running the following command:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Then from the terminal, install Python 3.6, pip, and the Service Fabric CLI by running the following commands:

```
brew install python3  
pip3 install sfctl  
sfctl -h
```

## CLI syntax

Commands are always prefixed with `sfctl`. For general information about all the commands you can use, use `sfctl -h`. For help with a single command, use `sfctl <command> -h`.

Commands follow a repeatable structure, with the target of the command preceding the verb or the action.

```
sfctl <object> <action>
```

In this example, `<object>` is the target for `<action>`.

## Select a cluster

Before you perform any operations, you must select a cluster to connect to. For example, to select and connect to the cluster with the name `testcluster.com`, run the following command:

### WARNING

Do not use unsecured Service Fabric clusters in a production environment.

```
sfctl cluster select --endpoint http://testcluster.com:19080
```

The cluster endpoint must be prefixed by `http` or `https`. It must include the port for the HTTP gateway. The port and address are the same as the Service Fabric Explorer URL.

For clusters that are secured with a certificate, you can specify a PEM-encoded certificate. The certificate can be specified as a single file or as a cert and a key pair. If it is a self-signed certificate that is not CA signed, you can pass the `--no-verify` option to bypass CA verification.

```
sfctl cluster select --endpoint https://testsecurecluster.com:19080 --pem ./client.pem --no-verify
```

For more information, see [Connect to a secure Azure Service Fabric cluster](#).

## Basic operations

Cluster connection information persists across multiple Service Fabric CLI sessions. After you select a Service Fabric cluster, you can run any Service Fabric command on the cluster.

For example, to get the Service Fabric cluster health state, use the following command:

```
sfctl cluster health
```

The command results in the following output:

```
{  
    "aggregatedHealthState": "Ok",  
    "applicationHealthStates": [  
        {  
            "aggregatedHealthState": "Ok",  
            "name": "fabric:/System"  
        }  
    ],  
    "healthEvents": [],  
    "nodeHealthStates": [  
        {  
            "aggregatedHealthState": "Ok",  
            "id": {  
                "id": "66aa824a642124089ee474b398d06a57"  
            },  
            "name": "_Test_0"  
        }  
    ],  
    "unhealthyEvaluations": []  
}
```

## Tips and troubleshooting

Here are some suggestions and tips for solving common problems.

### Convert a certificate from PFX to PEM format

The Service Fabric CLI supports client-side certificates as PEM (.pem extension) files. If you use PFX files from Windows, you must convert those certificates to PEM format. To convert a PFX file to a PEM file, use the following command:

```
openssl pkcs12 -in certificate.pfx -out mycert.pem -nodes
```

Similarly, to convert from a PEM file to a PFX file, you can use the following command (no password is being provided here):

```
openssl pkcs12 -export -out Certificates.pfx -inkey Certificates.pem -in Certificates.pem -passout pass:''
```

For more information, see the [OpenSSL documentation](#).

### Connection problems

Some operations might generate the following message:

```
Failed to establish a new connection: [Errno 8] nodename nor servname provided, or not known
```

Verify that the specified cluster endpoint is available and listening. Also, verify that the Service Fabric Explorer UI is available at that host and port. To update the endpoint, use `sfctl cluster select`.

### Detailed logs

Detailed logs often are helpful when you debug or report a problem. A global `--debug` flag increases the verbosity of log files.

### Command help and syntax

For help with a specific command or a group of commands, use the `-h` flag.

```
sfctl application -h
```

Here is another example:

```
sfctl application create -h
```

## Updating the Service Fabric CLI

To update the Service Fabric CLI, run the following commands (replace `pip` with `pip3` depending on what you chose during your original install):

```
pip uninstall sfctl  
pip install sfctl
```

## Next steps

- [Deploy an application with the Azure Service Fabric CLI](#)
- [Get started with Service Fabric on Linux](#)

# Create your first C# Service Fabric stateful reliable services application

10/5/2017 • 3 min to read • [Edit Online](#)

Learn how to deploy your first Service Fabric application for .NET on Windows in just a few minutes. When you're finished, you'll have a local cluster running with a reliable service application.

## Prerequisites

Before you get started, make sure that you have [set up your development environment](#). This includes installing the Service Fabric SDK and Visual Studio 2017 or 2015.

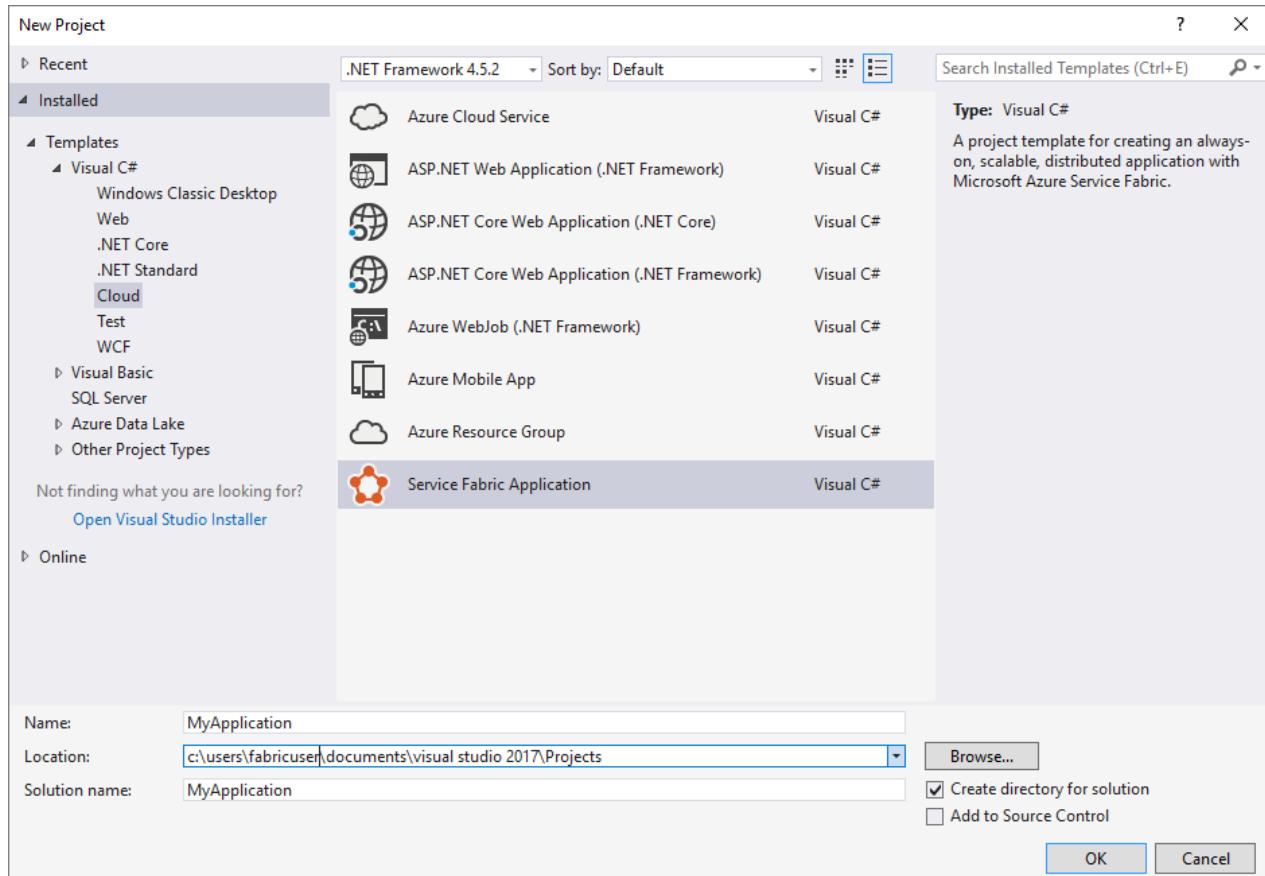
## Create the application

Launch Visual Studio as an **administrator**.

Create a project with **CTRL + SHIFT + N**

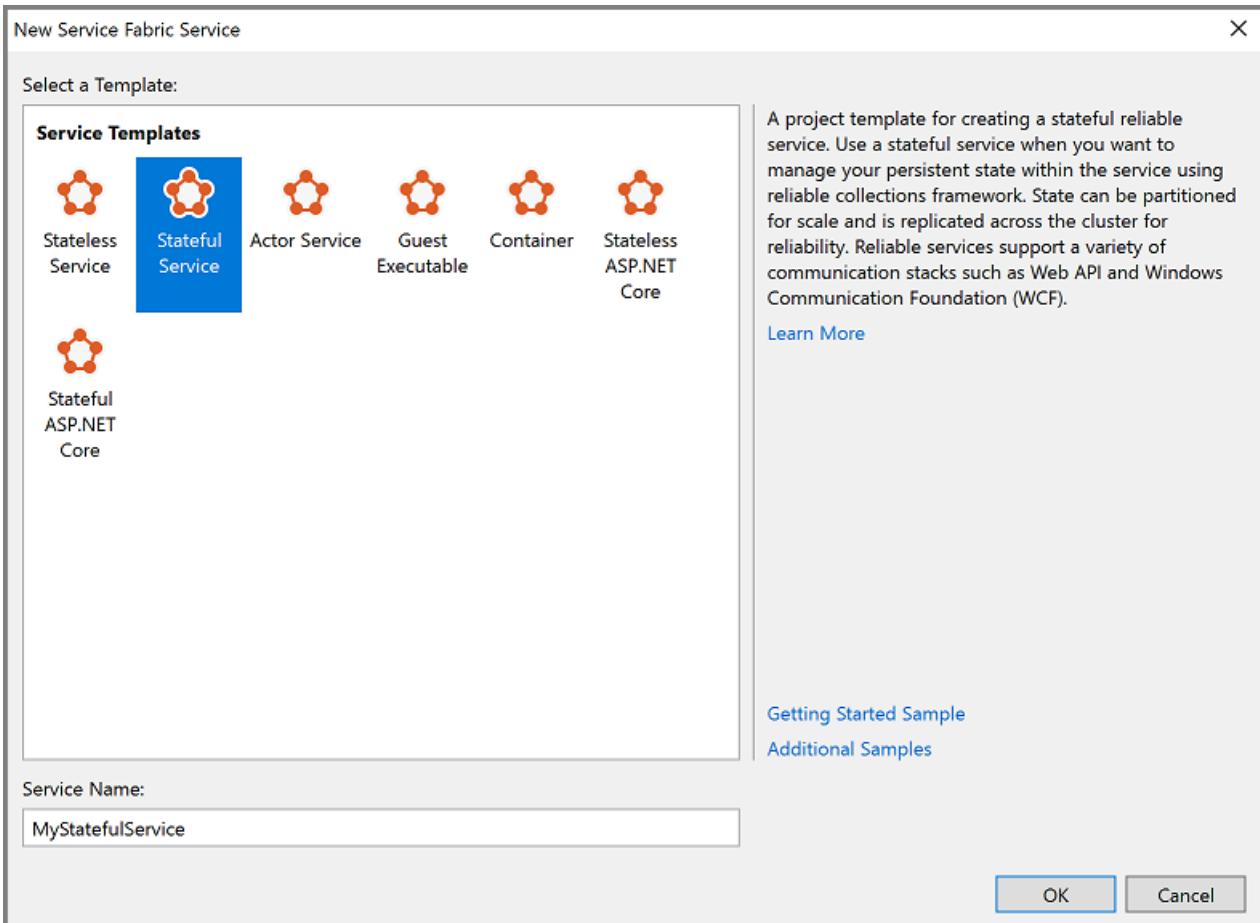
In the **New Project** dialog, choose **Cloud > Service Fabric Application**.

Name the application **MyApplication** and press **OK**.

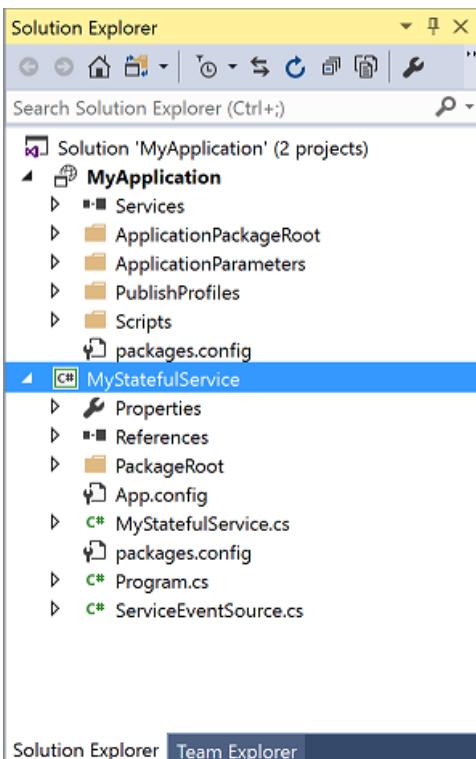


You can create any type of Service Fabric application from the next dialog. For this Quickstart, choose **Stateful Service**.

Name the service **MyStatefulService** and press **OK**.



Visual Studio creates the application project and the stateful service project and displays them in Solution Explorer.



The application project (**MyApplication**) does not contain any code directly. Instead, it references a set of service projects. In addition, it contains three other types of content:

- **Publish profiles**

Profiles for deploying to different environments.

- **Scripts**

PowerShell script for deploying/upgrading your application.

- **Application definition**

Includes the ApplicationManifest.xml file under *ApplicationPackageRoot* which describes your application's composition. Associated application parameter files are under *ApplicationParameters*, which can be used to specify environment-specific parameters. Visual Studio selects an application parameter file that's specified in the associated publish profile during deployment to a specific environment.

For an overview of the contents of the service project, see [Getting started with Reliable Services](#).

## Deploy and debug the application

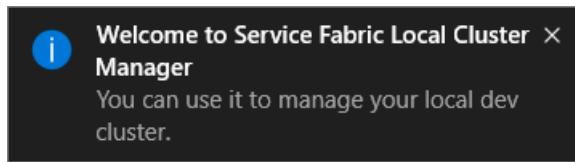
Now that you have an application, run it.

In Visual Studio, press **F5** to deploy the application for debugging.

**NOTE**

The first time you run and deploy the application locally, Visual Studio creates a local cluster for debugging. This may take some time. The cluster creation status is displayed in the Visual Studio output window.

When the cluster is ready, you get a notification from the local cluster system tray manager application included with the SDK.



Once the application starts, Visual Studio automatically brings up the **Diagnostics Event Viewer**, where you can see trace output from your services.

A screenshot of the Visual Studio Diagnostic Events window. The title bar says "Diagnostic Events" and "MyStatefulService.cs". The main area is a table with three columns: "Timestamp", "Event Name", and "Message". The table shows 60 events of type "ServiceMessage" with the following messages: "Current Counter Value: 52", "Current Counter Value: 51", "Current Counter Value: 50", "Current Counter Value: 49", "Current Counter Value: 48", "Current Counter Value: 47", "Current Counter Value: 46", "Current Counter Value: 45", "Current Counter Value: 44", and "Current Counter Value: 43". At the bottom of the table, there is a status bar that says "Listening... (60 of 60 events shown)" and a "Clear Filter" button.

Timestamp	Event Name	Message
▶ 22:54:19.188	ServiceMessage	Current Counter Value: 52
▶ 22:54:18.167	ServiceMessage	Current Counter Value: 51
▶ 22:54:17.114	ServiceMessage	Current Counter Value: 50
▶ 22:54:16.093	ServiceMessage	Current Counter Value: 49
▶ 22:54:15.023	ServiceMessage	Current Counter Value: 48
▶ 22:54:13.963	ServiceMessage	Current Counter Value: 47
▶ 22:54:12.901	ServiceMessage	Current Counter Value: 46
▶ 22:54:11.839	ServiceMessage	Current Counter Value: 45
▶ 22:54:10.768	ServiceMessage	Current Counter Value: 44
▶ 22:54:09.739	ServiceMessage	Current Counter Value: 43
▶		
Listening... (60 of 60 events shown) <a href="#">Clear Filter</a>		

The stateful service template we used simply shows a counter value incrementing in the **RunAsync** method of **MyStatefulService.cs**.

Expand one of the events to see more details, including the node where the code is running. In this case, it is `_Node_2`, though it may differ on your machine.

Diagnostic Events MyStatefulService.cs

Filter Events

Timestamp	Event Name	Message
22:57:52.049	ServiceMessage	Current Counter Value: 254
22:57:50.977	ServiceMessage	Current Counter Value: 253
	{	
	"Timestamp": "2017-06-28T22:57:50.9773879-07:00",	
	"ProviderName": "MyCompany-MyApplication-MyStatefulService",	
	"Id": 2,	
	"Message": "Current Counter Value: 253",	
	"ProcessId": 25152,	
	"Level": "Informational",	
	"Keywords": "0x0000F00000000000",	
	"EventName": "ServiceMessage",	
	"ActivityID": null,	
	"RelatedActivityID": null,	
	"Payload": {	
	"serviceName": "fabric:/MyApplication/MyStatefulService",	
	"serviceTypeName": "MyStatefulServiceType",	
	"replicaOrInstanceId": 131431891899140045,	
	"partitionId": "afbcf88c-93cd-4267-a9de-0efa532bbc4d",	
	"applicationName": "fabric:/MyApplication",	
	"applicationTypeName": "MyApplicationType",	
	"nodeName": "_Node_0",	
	"message": "Current Counter Value: 253"	
	}	
	}	
▷ 22:57:49.946	ServiceMessage	Current Counter Value: 252
Paused (262 of 262 events shown) <a href="#">Clear Filter</a>		

The local cluster contains five nodes hosted on a single machine. In a production environment, each node is hosted on a distinct physical or virtual machine. To simulate the loss of a machine while exercising the Visual Studio debugger at the same time, let's take down one of the nodes on the local cluster.

In the **Solution Explorer** window, open **MyStatefulService.cs**.

Find the `RunAsync` method and set a breakpoint on the first line of the method.

```

0 references
protected override async Task RunAsync(CancellationToken cancellationToken)
{
    // TODO: Replace the following sample code with your own logic
    //       or remove this RunAsync override if it's not needed in your service.

    var myDictionary = await this.StateManager.GetOrAddAsync<IReliableDictionary<string, long>>("myDictionary");

    while (true)
    {
        cancellationToken.ThrowIfCancellationRequested();

        using (var tx = this.StateManager.CreateTransaction())
        {
            var result = await myDictionary.TryGetValueAsync(tx, "Counter");

            ServiceEventSource.Current.ServiceMessage(this, "Current Counter Value: {0}",
                result.HasValue ? result.Value.ToString() : "Value does not exist.");

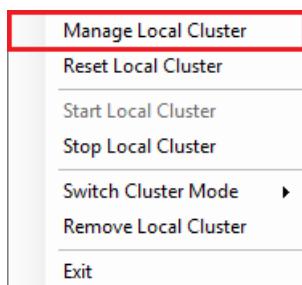
            await myDictionary.AddOrUpdateAsync(tx, "Counter", 0, (key, value) => ++value);

            // If an exception is thrown before calling CommitAsync, the transaction aborts, all changes are
            // discarded, and nothing is saved to the secondary replicas.
            await tx.CommitAsync();
        }

        await Task.Delay(TimeSpan.FromSeconds(1), cancellationToken);
    }
}

```

Launch the **Service Fabric Explorer** tool by right-clicking on the **Local Cluster Manager** system tray application and choose **Manage Local Cluster**.



**Service Fabric Explorer** offers a visual representation of a cluster. It includes the set of applications deployed to it and the set of physical nodes that make it up.

In the left pane, expand **Cluster > Nodes** and find the node where your code is running.

Click **Actions > Deactivate (Restart)** to simulate a machine restarting.

ESSENTIALS	DETAILS
Name _Node_0	Upgrade Domain 0
Health State <span style="color: green;">OK</span>	Fault Domain fd:/0
Status Up	IP Address or Domain Name localhost
Type NodeType0	Is Seed Node true

Momentarily, you should see your breakpoint hit in Visual Studio as the computation you were doing on one node seamlessly fails over to another.

Next, return to the Diagnostic Events Viewer and observe the messages. The counter has continued incrementing, even though the events are actually coming from a different node.

The screenshot shows the Diagnostic Events window in Visual Studio. The title bar says "Diagnostic Events" and "MyStatefulService.cs". The main area displays a table with three columns: "Timestamp", "Event Name", and "Message". There are two entries:

Timestamp	Event Name	Message
23:04:53.651	ServiceMessage	Current Counter Value: 646
23:04:52.627	ServiceMessage	Current Counter Value: 645

The third entry is expanded, showing a JSON payload. The "nodeName" field is highlighted with a red rectangle.

```
{  
    "Timestamp": "2017-06-28T23:04:52.6279215-07:00",  
    "ProviderName": "MyCompany-MyApplication-MyStatefulService",  
    "Id": 2,  
    "Message": "Current Counter Value: 645",  
    "ProcessId": 3976,  
    "Level": "Informational",  
    "Keywords": "0x0000F00000000000",  
    "EventName": "ServiceMessage",  
    "ActivityID": null,  
    "RelatedActivityID": null,  
    "Payload": {  
        "serviceName": "fabric:/MyApplication/MyStatefulService",  
        "serviceTypeName": "MyStatefulServiceType",  
        "replicaOrInstanceId": 131431891964379560,  
        "partitionId": "afbcf88c-93cd-4267-a9de-0efa532bbc4d",  
        "applicationName": "fabric:/MyApplication",  
        "applicationTypeName": "MyApplicationType",  
        "nodeName": "_Node_1",  
        "message": "Current Counter Value: 645"  
    }  
}  
  
▷ 23:04:51.597 ServiceMessage Current Counter Value: 644
```

At the bottom, it says "Paused (657 of 657 events shown) [Clear Filter](#)".

## Cleaning up the local cluster (optional)

Remember, this local cluster is real. Stopping the debugger removes your application instance and unregisters the application type. However, the cluster continues to run in the background. When you're ready to stop the local cluster, there are a couple options.

### Keep application and trace data

Shut down the cluster by right-clicking on the **Local Cluster Manager** system tray application and then choose **Stop Local Cluster**.

### Delete the cluster and all data

Remove the cluster by right-clicking on the **Local Cluster Manager** system tray application and then choose **Remove Local Cluster**.

If you choose this option, Visual Studio will redeploy the cluster the next time you run the application. Choose this option if you don't intend to use the local cluster for some time or if you need to reclaim resources.

## Next steps

Read more about [reliable services](#).

# Host a Node.js application on Azure Service Fabric

10/13/2017 • 3 min to read • [Edit Online](#)

This quickstart helps you deploy an existing application (Node.js in this example) to a Service Fabric cluster running on Azure.

## Prerequisites

Before you get started, make sure that you have [set up your development environment](#). Which includes installing the Service Fabric SDK and Visual Studio 2017 or 2015.

You also need to have an existing Node.js application for deployment. This quickstart uses a simple Node.js website that can be downloaded [here](#). Extract this file to your

<path-to-project>\ApplicationPackageRoot\<package-name>\Code\ folder after you create the project in the next step.

If you don't have an Azure subscription, create a [free account](#).

## Create the service

Launch Visual Studio as an **administrator**.

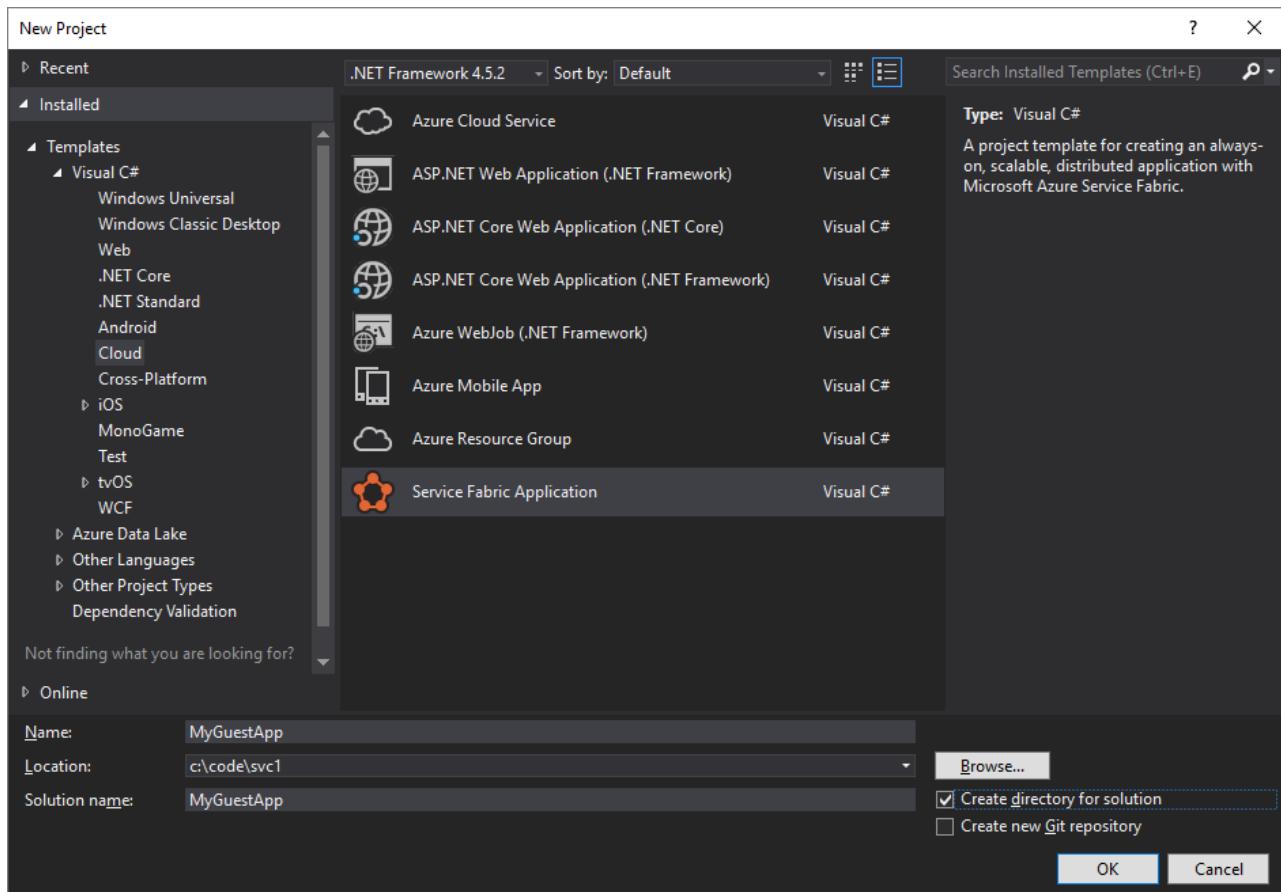
Create a project with **CTRL + SHIFT + N**

In the **New Project** dialog, choose **Cloud > Service Fabric Application**.

Name the application **MyGuestApp** and press **OK**.

### IMPORTANT

Node.js can easily break the 260 character limit for paths that windows has. Use a short path for the project itself such as **c:\code\svc1**. Optionally, you can follow [these instructions](#) to enable long file paths in Windows 10.

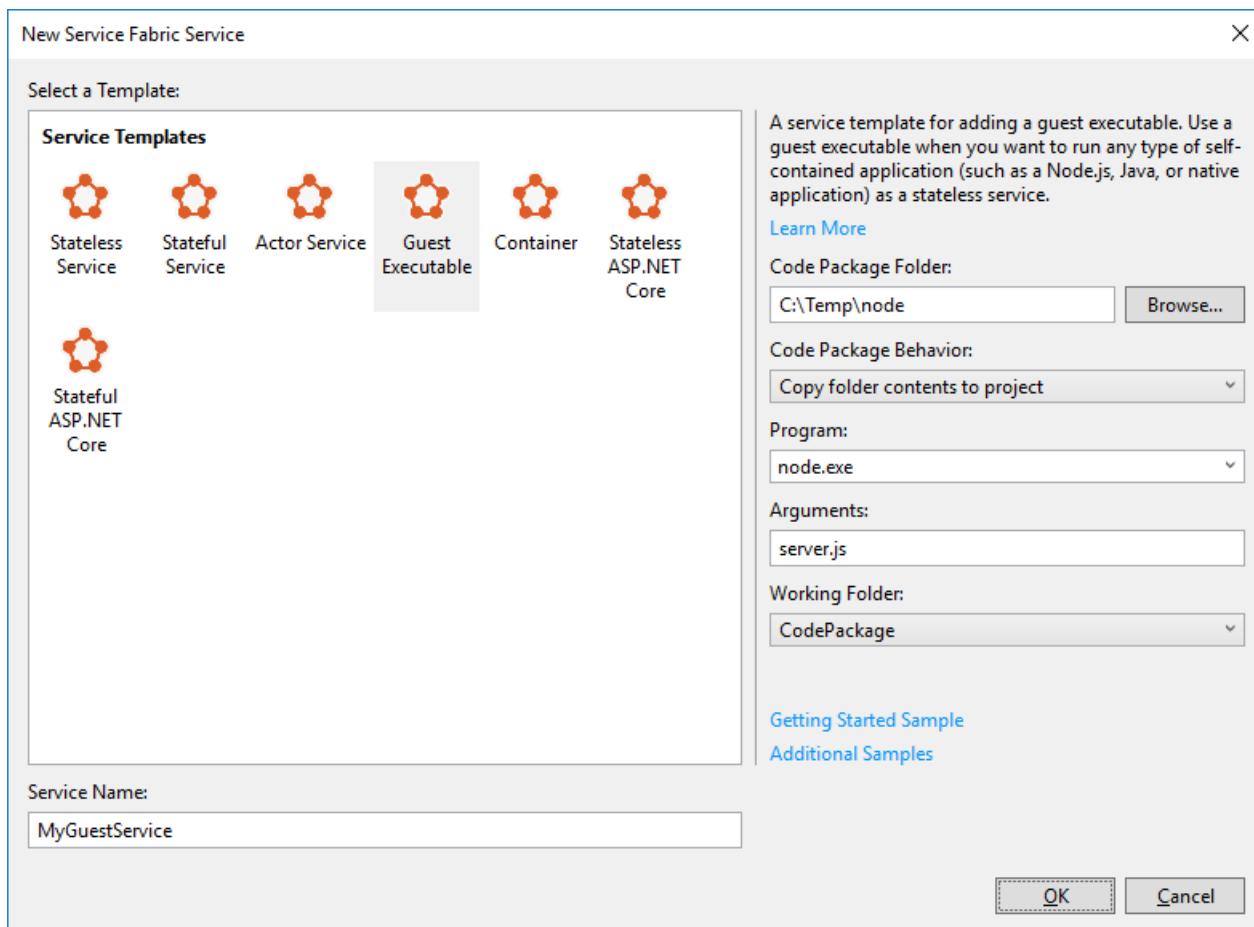


You can create any type of Service Fabric service from the next dialog. For this quickstart, choose **Guest Executable**.

Name the service **MyGuestService** and set the options on the right to the following values:

SETTING	VALUE
Code Package Folder	<the folder with your Node.js app>
Code Package Behavior	Copy folder contents to project
Program	node.exe
Arguments	server.js
Working Folder	CodePackage

Press **OK**.



Visual Studio creates the application project and the actor service project and displays them in Solution Explorer.

The application project (**MyGuestApp**) does not contain any code directly. Instead, it references a set of service projects. In addition, it contains three other types of content:

- **Publish profiles**

Tooling preferences for different environments.

- **Scripts**

PowerShell script for deploying/upgrading your application.

- **Application definition**

Includes the application manifest under *ApplicationPackageRoot*. Associated application parameter files are under *ApplicationParameters*, which define the application and allow you to configure it specifically for a given environment.

For an overview of the contents of the service project, see [Getting started with Reliable Services](#).

## Set up networking

The example Node.js app we're deploying uses port **80** and we need to tell Service Fabric that we need that port exposed.

Open the **ServiceManifest.xml** file in the project. At the bottom of the manifest, there is a

`<Resources> \ <Endpoints>` with an entry already defined. Modify that entry to add `Port`, `Protocol`, and `Type`.

```

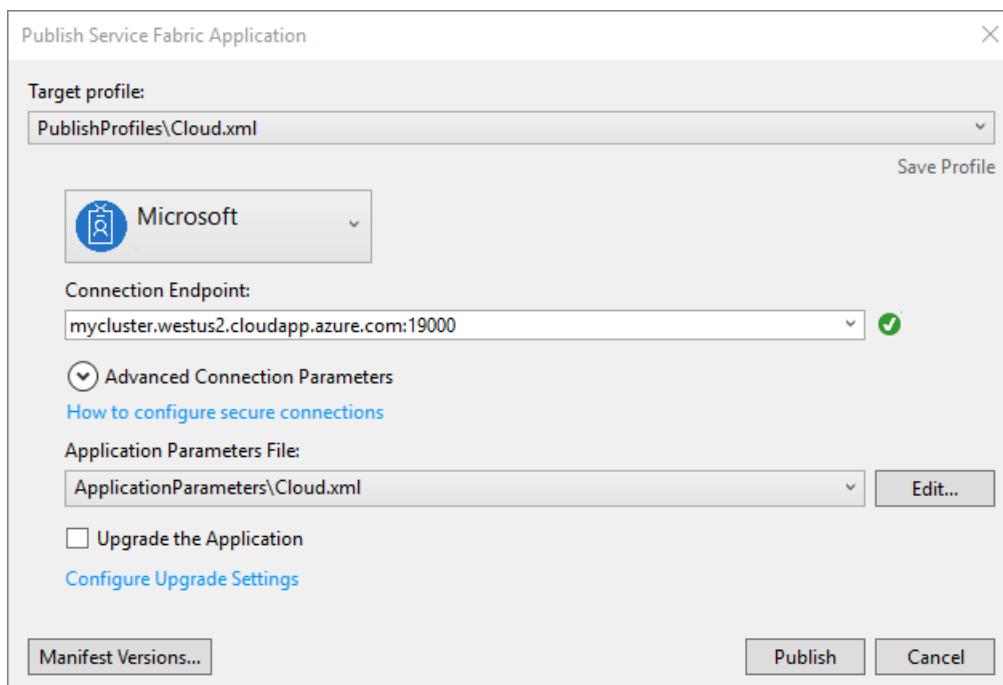
<Resources>
  <Endpoints>
    <!-- This endpoint is used by the communication listener to obtain the port on which to
        listen. Please note that if your service is partitioned, this port is shared with
        replicas of different partitions that are placed in your code. -->
    <Endpoint Name="MyGuestAppServiceTypeEndpoint" Port="80" Protocol="http" Type="Input" />
  </Endpoints>
</Resources>

```

## Deploy to Azure

If you press **F5** and run the project, it is deployed to the local cluster. However, let's deploy to Azure instead.

Right-click on the project and choose **Publish...** which opens a dialog to publish to Azure.

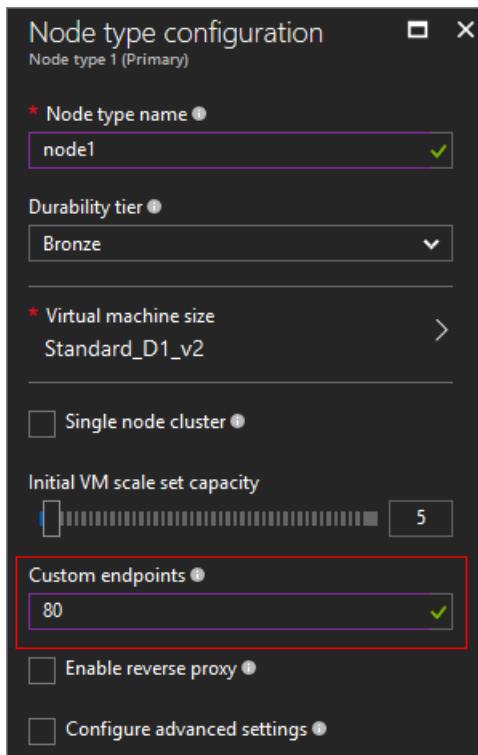


Select the **PublishProfiles\Cloud.xml** target profile.

If you haven't previously, choose an Azure account to deploy to. If you don't have one yet, [sign-up for one](#).

Under **Connection Endpoint**, select the Service Fabric cluster to deploy to. If you do not have one, select [\*\*<Create New Cluster...>\*\*](#) which opens up web browser window to the Azure portal. For more information, see [create a cluster in the portal](#).

When you create the Service Fabric cluster, make sure to set the **Custom endpoints** setting to **80**.



Creating a new Service Fabric cluster takes some time to complete. Once it has been created, go back to the publish dialog and select <Refresh>. The new cluster is listed in the drop-down box; select it.

Press **Publish** and wait for the deployment to finish.

This may take a few minutes. After it completes, it may take a few more minutes for the application to be fully available.

## Test the website

After your service has been published, test it in a web browser.

First, open the Azure portal and find your Service Fabric service.

Check the overview blade of the service address. Use the domain name from the *Client connection endpoint* property. For example, <http://mysvcfab1.westus2.cloudapp.azure.com>.

Resource group (change)	Service Fabric version
mysvcfab1rg	5.6.210.9494
Status	Client connection endpoint
Baseline upgrade	mysvcfab1.westus2.cloudapp.azure.com:19000
Location	Node count
West US 2	5

Navigate to this address where you will see the `HELLO WORLD` response.

## Delete the cluster

Do not forget to delete all of the resources you've created for this quickstart, as you are charged for those resources.

## Next steps

Read more about [guest executables](#).

# Deploy a guest executable to Service Fabric

9/25/2017 • 14 min to read • [Edit Online](#)

You can run any type of code, such as Node.js, Java, or C++ in Azure Service Fabric as a service. Service Fabric refers to these types of services as guest executables.

Guest executables are treated by Service Fabric like stateless services. As a result, they are placed on nodes in a cluster, based on availability and other metrics. This article describes how to package and deploy a guest executable to a Service Fabric cluster, by using Visual Studio or a command-line utility.

In this article, we cover the steps to package a guest executable and deploy it to Service Fabric.

## Benefits of running a guest executable in Service Fabric

There are several advantages to running a guest executable in a Service Fabric cluster:

- High availability. Applications that run in Service Fabric are made highly available. Service Fabric ensures that instances of an application are running.
- Health monitoring. Service Fabric health monitoring detects if an application is running, and provides diagnostic information if there is a failure.
- Application lifecycle management. Besides providing upgrades with no downtime, Service Fabric provides automatic rollback to the previous version if there is a bad health event reported during an upgrade.
- Density. You can run multiple applications in a cluster, which eliminates the need for each application to run on its own hardware.
- Discoverability: Using REST you can call the Service Fabric Naming service to find other services in the cluster.

## Samples

- [Sample for packaging and deploying a guest executable](#)
- [Sample of two guest executables \(C# and nodejs\) communicating via the Naming service using REST](#)

## Overview of application and service manifest files

As part of deploying a guest executable, it is useful to understand the Service Fabric packaging and deployment model as described in [application model](#). The Service Fabric packaging model relies on two XML files: the application and service manifests. The schema definition for the ApplicationManifest.xml and ServiceManifest.xml files is installed with the Service Fabric SDK into `C:\Program Files\Microsoft SDKs\Service Fabric\schemas\ServiceFabricServiceModel.xsd`.

- **Application manifest** The application manifest is used to describe the application. It lists the services that compose it, and other parameters that are used to define how one or more services should be deployed, such as the number of instances.

In Service Fabric, an application is a unit of deployment and upgrade. An application can be upgraded as a single unit where potential failures and potential rollbacks are managed. Service Fabric guarantees that the upgrade process is either successful, or, if the upgrade fails, does not leave the application in an unknown or unstable state.

- **Service manifest** The service manifest describes the components of a service. It includes data, such as the name and type of service, and its code and configuration. The service manifest also includes some additional

parameters that can be used to configure the service once it is deployed.

## Application package file structure

To deploy an application to Service Fabric, the application should follow a predefined directory structure. The following is an example of that structure.

```
|-- ApplicationPackageRoot  
  |-- GuestService1Pkg  
    |-- Code  
      |-- existingapp.exe  
    |-- Config  
      |-- Settings.xml  
    |-- Data  
      |-- ServiceManifest.xml  
  |-- ApplicationManifest.xml
```

The ApplicationPackageRoot contains the ApplicationManifest.xml file that defines the application. A subdirectory for each service included in the application is used to contain all the artifacts that the service requires. These subdirectories are the ServiceManifest.xml and, typically, the following:

- *Code*. This directory contains the service code.
- *Config*. This directory contains a Settings.xml file (and other files if necessary) that the service can access at runtime to retrieve specific configuration settings.
- *Data*. This is an additional directory to store additional local data that the service may need. Data should be used to store only ephemeral data. Service Fabric does not copy or replicate changes to the data directory if the service needs to be relocated (for example, during failover).

### NOTE

You don't have to create the `config` and `data` directories if you don't need them.

## Package an existing executable

When packaging a guest executable, you can choose either to use a Visual Studio project template or to [create the application package manually](#). Using Visual Studio, the application package structure and manifest files are created by the new project template for you.

### TIP

The easiest way to package an existing Windows executable into a service is to use Visual Studio and on Linux to use Yeoman

## Use Visual Studio to package and deploy an existing executable

Visual Studio provides a Service Fabric service template to help you deploy a guest executable to a Service Fabric cluster.

1. Choose **File > New Project**, and create a Service Fabric application.
2. Choose **Guest Executable** as the service template.
3. Click **Browse** to select the folder with your executable and fill in the rest of the parameters to create the service.
  - *Code Package Behavior*. Can be set to copy all the content of your folder to the Visual Studio Project,

which is useful if the executable does not change. If you expect the executable to change and want the ability to pick up new builds dynamically, you can choose to link to the folder instead. You can use linked folders when creating the application project in Visual Studio. This links to the source location from within the project, making it possible for you to update the guest executable in its source destination. Those updates become part of the application package on build.

- *Program* specifies the executable that should be run to start the service.
- *Arguments* specifies the arguments that should be passed to the executable. It can be a list of parameters with arguments.
- *WorkingFolder* specifies the working directory for the process that is going to be started. You can specify three values:
  - `CodeBase` specifies that the working directory is going to be set to the code directory in the application package (`Code` directory shown in the preceding file structure).
  - `CodePackage` specifies that the working directory is going to be set to the root of the application package (`GuestService1Pkg` shown in the preceding file structure).
  - `Work` specifies that the files are placed in a subdirectory called `work`.

4. Give your service a name, and click **OK**.
5. If your service needs an endpoint for communication, you can now add the protocol, port, and type to the `ServiceManifest.xml` file. For example:

```
<Endpoint Name="NodeAppTypeEndpoint" Protocol="http" Port="3000" UriScheme="http" PathSuffix="/myapp/" Type="Input" />
```

6. You can now use the package and publish action against your local cluster by debugging the solution in Visual Studio. When ready, you can publish the application to a remote cluster or check in the solution to source control.
7. Go to the end of this article to see how to view your guest executable service running in Service Fabric Explorer.

## Use Yeoman to package and deploy an existing executable on Linux

The procedure for creating and deploying a guest executable on Linux is the same as deploying a csharp or java application.

1. In a terminal, type `yo azuresfguest`.
2. Name your application.
3. Name your service, and provide the details including path of the executable and the parameters it must be invoked with.

Yeoman creates an application package with the appropriate application and manifest files along with install and uninstall scripts.

## Manually package and deploy an existing executable

The process of manually packaging a guest executable is based on the following general steps:

1. Create the package directory structure.
2. Add the application's code and configuration files.
3. Edit the service manifest file.
4. Edit the application manifest file.

### Create the package directory structure

You can start by creating the directory structure, as described in the preceding section, "Application package file structure."

## Add the application's code and configuration files

After you have created the directory structure, you can add the application's code and configuration files under the code and config directories. You can also create additional directories or subdirectories under the code or config directories.

Service Fabric does an `xcopy` of the content of the application root directory, so there is no predefined structure to use other than creating two top directories, code and settings. (You can pick different names if you want. More details are in the next section.)

### NOTE

Make sure that you include all the files and dependencies that the application needs. Service Fabric copies the content of the application package on all nodes in the cluster where the application's services are going to be deployed. The package should contain all the code that the application needs to run. Do not assume that the dependencies are already installed.

## Edit the service manifest file

The next step is to edit the service manifest file to include the following information:

- The name of the service type. This is an ID that Service Fabric uses to identify a service.
- The command to use to launch the application (ExeHost).
- Any script that needs to be run to set up the application (SetupEntryPoint).

The following is an example of a `ServiceManifest.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceManifest xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Name="NodeApp" Version="1.0.0.0" xmlns="http://schemas.microsoft.com/2011/01/fabric">
  <ServiceTypes>
    <StatelessServiceType ServiceTypeName="NodeApp" UseImplicitHost="true"/>
  </ServiceTypes>
  <CodePackage Name="code" Version="1.0.0.0">
    <SetupEntryPoint>
      <ExeHost>
        <Program>scripts\launchConfig.cmd</Program>
      </ExeHost>
    </SetupEntryPoint>
    <EntryPoint>
      <ExeHost>
        <Program>node.exe</Program>
        <Arguments>bin/www</Arguments>
        <WorkingFolder>CodePackage</WorkingFolder>
      </ExeHost>
    </EntryPoint>
  </CodePackage>
  <Resources>
    <Endpoints>
      <Endpoint Name="NodeAppTypeEndpoint" Protocol="http" Port="3000" Type="Input" />
    </Endpoints>
  </Resources>
</ServiceManifest>
```

The following sections go over the different parts of the file that you need to update.

### Update ServiceTypes

```
<ServiceTypes>
  <StatelessServiceType ServiceTypeName="NodeApp" UseImplicitHost="true" />
</ServiceTypes>
```

- You can pick any name that you want for `ServiceTypeName`. The value is used in the `ApplicationManifest.xml` file to identify the service.
- Specify `UseImplicitHost="true"`. This attribute tells Service Fabric that the service is based on a self-contained app, so all Service Fabric needs to do is to launch it as a process and monitor its health.

### **Update CodePackage**

The `CodePackage` element specifies the location (and version) of the service's code.

```
<CodePackage Name="Code" Version="1.0.0.0">
```

The `Name` element is used to specify the name of the directory in the application package that contains the service's code. `CodePackage` also has the `version` attribute. This can be used to specify the version of the code, and can also potentially be used to upgrade the service's code by using the application lifecycle management infrastructure in Service Fabric.

### **Optional: Update SetupEntryPoint**

```
<SetupEntryPoint>
  <ExeHost>
    <Program>scripts\launchConfig.cmd</Program>
  </ExeHost>
</SetupEntryPoint>
```

The `SetupEntryPoint` element is used to specify any executable or batch file that should be executed before the service's code is launched. It is an optional step, so it does not need to be included if there is no initialization required. The `SetupEntryPoint` is executed every time the service is restarted.

There is only one `SetupEntryPoint`, so setup scripts need to be grouped in a single batch file if the application's setup requires multiple scripts. The `SetupEntryPoint` can execute any type of file: executable files, batch files, and PowerShell cmdlets. For more details, see [Configure SetupEntryPoint](#).

In the preceding example, the `SetupEntryPoint` runs a batch file called `LaunchConfig.cmd` that is located in the `scripts` subdirectory of the code directory (assuming the `WorkingFolder` element is set to `CodeBase`).

### **Update EntryPoint**

```
<EntryPoint>
  <ExeHost>
    <Program>node.exe</Program>
    <Arguments>bin/www</Arguments>
    <WorkingFolder>CodeBase</WorkingFolder>
  </ExeHost>
</EntryPoint>
```

The `EntryPoint` element in the service manifest file is used to specify how to launch the service. The `ExeHost` element specifies the executable (and arguments) that should be used to launch the service.

- `Program` specifies the name of the executable that should start the service.
- `Arguments` specifies the arguments that should be passed to the executable. It can be a list of parameters with arguments.
- `WorkingFolder` specifies the working directory for the process that is going to be started. You can specify three values:
  - `CodeBase` specifies that the working directory is going to be set to the code directory in the application package (`Code` directory in the preceding file structure).
  - `CodePackage` specifies that the working directory is going to be set to the root of the application

package (`GuestService1Pkg` in the preceding file structure).

- `Work` specifies that the files are placed in a subdirectory called work.

The WorkingFolder is useful to set the correct working directory so that relative paths can be used by either the application or initialization scripts.

#### Update Endpoints and register with Naming Service for communication

```
<Endpoints>
  <Endpoint Name="NodeAppTypeEndpoint" Protocol="http" Port="3000" Type="Input" />
</Endpoints>
```

In the preceding example, the `Endpoint` element specifies the endpoints that the application can listen on. In this example, the Node.js application listens on http on port 3000.

Furthermore you can ask Service Fabric to publish this endpoint to the Naming Service so other services can discover the endpoint address to this service. This enables you to be able to communicate between services that are guest executables. The published endpoint address is of the form

`UriScheme://IPAddressOrFQDN:Port/PathSuffix`. `UriScheme` and `PathSuffix` are optional attributes.

`IPAddressOrFQDN` is the IP address or fully qualified domain name of the node this executable gets placed on, and it is calculated for you.

In the following example, once the service is deployed, in Service Fabric Explorer you see an endpoint similar to

`http://10.1.4.92:3000/myapp/` published for the service instance. Or if this is a local machine, you see

`http://localhost:3000/myapp/`.

```
<Endpoints>
  <Endpoint Name="NodeAppTypeEndpoint" Protocol="http" Port="3000" UriScheme="http" PathSuffix="myapp/"
    Type="Input" />
</Endpoints>
```

You can use these addresses with [reverse proxy](#) to communicate between services.

#### Edit the application manifest file

Once you have configured the `Servicemanifest.xml` file, you need to make some changes to the

`ApplicationManifest.xml` file to ensure that the correct service type and name are used.

```
<?xml version="1.0" encoding="utf-8"?>
<ApplicationManifest xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ApplicationTypeName="NodeAppType"
  ApplicationTypeVersion="1.0" xmlns="http://schemas.microsoft.com/2011/01/fabric">
  <ServiceManifestImport>
    <ServiceManifestRef ServiceManifestName="NodeApp" ServiceManifestVersion="1.0.0.0" />
  </ServiceManifestImport>
</ApplicationManifest>
```

#### ServiceManifestImport

In the `ServiceManifestImport` element, you can specify one or more services that you want to include in the app.

Services are referenced with `ServiceManifestName`, which specifies the name of the directory where the

`ServiceManifest.xml` file is located.

```
<ServiceManifestImport>
  <ServiceManifestRef ServiceManifestName="NodeApp" ServiceManifestVersion="1.0.0.0" />
</ServiceManifestImport>
```

# Set up logging

For guest executables, it is useful to be able to see console logs to find out if the application and configuration scripts show any errors. Console redirection can be configured in the `ServiceManifest.xml` file using the `ConsoleRedirection` element.

## WARNING

Never use the console redirection policy in an application that is deployed in production because this can affect the application failover. *Only* use this for local development and debugging purposes.

```
<EntryPoint>
  <ExeHost>
    <Program>node.exe</Program>
    <Arguments>bin/www</Arguments>
    <WorkingFolder>CodeBase</WorkingFolder>
    <ConsoleRedirection FileRetentionCount="5" FileMaxSizeInKb="2048"/>
  </ExeHost>
</EntryPoint>
```

`ConsoleRedirection` can be used to redirect console output (both `stdout` and `stderr`) to a working directory. This provides the ability to verify that there are no errors during the setup or execution of the application in the Service Fabric cluster.

`FileRetentionCount` determines how many files are saved in the working directory. A value of 5, for example, means that the log files for the previous five executions are stored in the working directory.

`FileMaxSizeInKb` specifies the maximum size of the log files.

Log files are saved in one of the service's working directories. To determine where the files are located, use Service Fabric Explorer to determine which node the service is running on, and which working directory is being used. This process is covered later in this article.

# Deployment

The last step is to [deploy your application](#). The following PowerShell script shows how to deploy your application to the local development cluster, and start a new Service Fabric service.

```
Connect-ServiceFabricCluster localhost:19000

Write-Host 'Copying application package...'
Copy-ServiceFabricApplicationPackage -ApplicationPackagePath 'C:\Dev\MultipleApplications' -
ImageStoreConnectionString 'file:C:\SfDevCluster\Data\ImageStoreShare' -ApplicationPackagePathInImageStore
'nodeapp'

Write-Host 'Registering application type...'
Register-ServiceFabricApplicationType -ApplicationPathInImageStore 'nodeapp'

New-ServiceFabricApplication -ApplicationName 'fabric:/nodeapp' -ApplicationTypeName 'NodeAppType' -
ApplicationTypeVersion 1.0

New-ServiceFabricService -ApplicationName 'fabric:/nodeapp' -ServiceName 'fabric:/nodeapp/nodeappservice' -
ServiceTypeName 'NodeApp' -Stateless -PartitionSchemeSingleton -InstanceCount 1
```

## TIP

Compress the package before copying to the image store if the package is large or has many files. Read more [here](#).

A Service Fabric service can be deployed in various "configurations." For example, it can be deployed as single or multiple instances, or it can be deployed in such a way that there is one instance of the service on each node of the Service Fabric cluster.

The `InstanceCount` parameter of the `New-ServiceFabricService` cmdlet is used to specify how many instances of the service should be launched in the Service Fabric cluster. You can set the `InstanceCount` value, depending on the type of application that you are deploying. The two most common scenarios are:

- `InstanceCount = "1"`. In this case, only one instance of the service is deployed in the cluster. Service Fabric's scheduler determines which node the service is going to be deployed on.
- `InstanceCount = "-1"`. In this case, one instance of the service is deployed on every node in the Service Fabric cluster. The result is having one (and only one) instance of the service for each node in the cluster.

This is a useful configuration for front-end applications (for example, a REST endpoint), because client applications need to "connect" to any of the nodes in the cluster to use the endpoint. This configuration can also be used when, for example, all nodes of the Service Fabric cluster are connected to a load balancer. Client traffic can then be distributed across the service that is running on all nodes in the cluster.

## Check your running application

In Service Fabric Explorer, identify the node where the service is running. In this example, it runs on Node1:

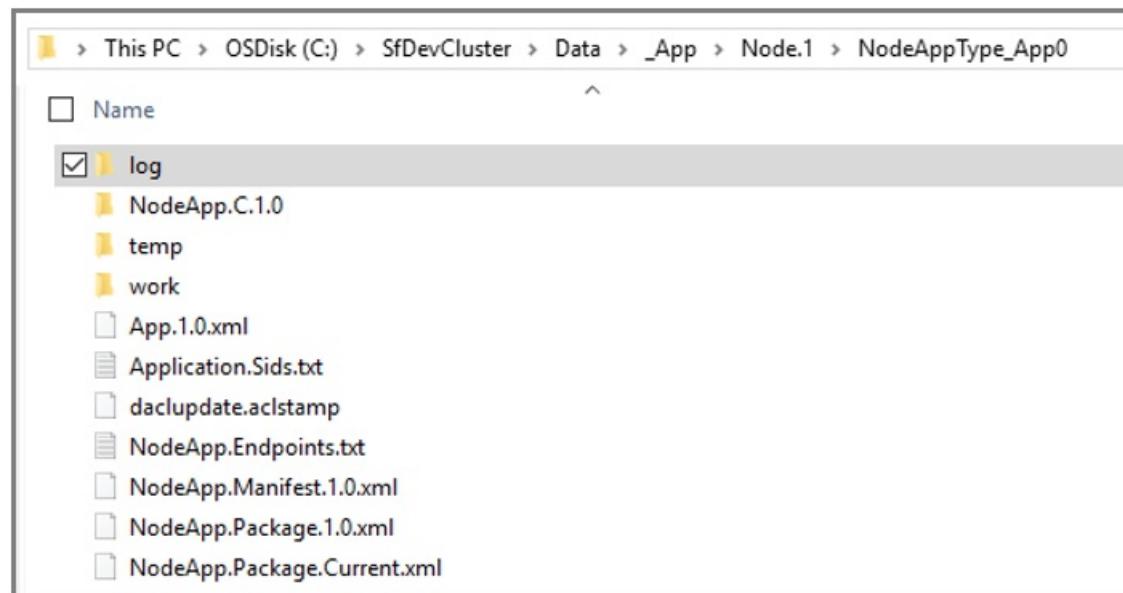
The screenshot shows the Microsoft Azure Service Fabric Explorer interface. The left sidebar displays a tree view of the cluster structure, including the Cluster, Applications (with System and NodeAppType), and Nodes (with \_Node\_0, \_Node\_1, \_Node\_2, \_Node\_3, and \_Node\_4). The main pane is titled 'Application fabric:/nodeapp' and contains three tabs: ESSENTIALS, DETAILS, and MANIFEST. The ESSENTIALS tab is selected, showing the application name 'fabric:/nodeapp', health state 'OK', status 'Ready', and version '1.0'. It also includes sections for 'UNHEALTHY EVALUATIONS' (empty) and 'SERVICES'. The SERVICES table lists a single service entry:

Name	Service Type	Version	Service Kind	Health State	Status
fabric:/nodeapp/nodeappservice	NodeApp	1.0.0	Stateless	OK	Active

If you navigate to the node and browse to the application, you see the essential node information, including its location on disk.

The screenshot shows the Microsoft Azure Service Fabric Explorer interface. On the left, there's a navigation pane with sections for Cluster, Applications (System, NodeAppType, fabric:/nodeapp), and Nodes. Under fabric:/nodeapp, there's a sub-node \_Node\_0, which is expanded to show its children: fabric:/nodeapp, NodeApp, Code Packages, and Replicas. The NodeApp item is selected. On the right, the main area is titled "Deployed Application fabric:/nodeapp". It has tabs for ESSENTIALS and DETAILS. In the ESSENTIALS tab, it shows the Name (fabric:/nodeapp), Status (Active), Application Type (NodeAppType), and Disk Location (C:\SfDevCluster\Data\\_App\\_Node.0\NodeAppType.App0). Below this, there's a section for "UNHEALTHY EVALUATIONS" with a search bar and a table header for Kind, Health State, and Description, but it says "No items to display". At the bottom, there's a section for "DEPLOYED SERVICE PACKAGES" with a search bar and a table header for Name, Version, and Status, showing one entry: NodeApp (Version 1.0.0, Status Active).

If you browse to the directory by using Server Explorer, you can find the working directory and the service's log folder, as shown in the following screenshot:



## Next steps

In this article, you have learned how to package a guest executable and deploy it to Service Fabric. See the following articles for related information and tasks.

- [Sample for packaging and deploying a guest executable](#), including a link to the prerelease of the packaging tool
- [Sample of two guest executables \(C# and nodejs\) communicating via the Naming service using REST](#)
- [Deploy multiple guest executables](#)
- [Create your first Service Fabric application using Visual Studio](#)

# Deploy multiple guest executables

10/5/2017 • 6 min to read • [Edit Online](#)

This article shows how to package and deploy multiple guest executables to Azure Service Fabric. For building and deploying a single Service Fabric package read how to [deploy a guest executable to Service Fabric](#).

While this walkthrough shows how to deploy an application with a Node.js front end that uses MongoDB as the data store, you can apply the steps to any application that has dependencies on another application.

You can use Visual Studio to produce the application package that contains multiple guest executables. See [Using Visual Studio to package an existing application](#). After you have added the first guest executable, right click on the application project and select the **Add->New Service Fabric service** to add the second guest executable project to the solution. Note: If you choose to link the source in the Visual Studio project, building the Visual Studio solution, will make sure that your application package is up to date with changes in the source.

## Samples

- [Sample for packaging and deploying a guest executable](#)
- [Sample of two guest executables \(C# and nodejs\) communicating via the Naming service using REST](#)

## Manually package the multiple guest executable application

Alternatively you can manually package the guest executable. For the manual packaging, this article uses the Service Fabric packaging tool, which is available at <http://aka.ms/servicefabricpacktool>.

### Packaging the Node.js application

This article assumes that Node.js is not installed on the nodes in the Service Fabric cluster. As a consequence, you need to add Node.exe to the root directory of your node application before packaging. The directory structure of the Node.js application (using Express web framework and Jade template engine) should look similar to the one below:

```
|-- NodeApplication
|-- bin
|   |-- www
|-- node_modules
|   |-- .bin
|   |-- express
|   |-- jade
|   |-- etc.
|-- public
|   |-- images
|   |-- etc.
|-- routes
|   |-- index.js
|   |-- users.js
|-- views
|   |-- index.jade
|   |-- etc.
|-- app.js
|-- package.json
|-- node.exe
```

As a next step, you create an application package for the Node.js application. The code below creates a Service Fabric application package that contains the Node.js application.

```
.\ServiceFabricAppPackageUtil.exe /source:'[yourdirectory]\MyNodeApplication' /target:'[yourtargetdirectory]  
/appname:NodeService /exe:'node.exe' /ma:'bin/www' /AppType:NodeAppType
```

Below is a description of the parameters that are being used:

- **/source** points to the directory of the application that should be packaged.
- **/target** defines the directory in which the package should be created. This directory has to be different from the source directory.
- **/appname** defines the application name of the existing application. It's important to understand that this translates to the service name in the manifest, and not to the Service Fabric application name.
- **/exe** defines the executable that Service Fabric is supposed to launch, in this case `node.exe`.
- **/ma** defines the argument that is being used to launch the executable. As Node.js is not installed, Service Fabric needs to launch the Node.js web server by executing `node.exe bin/www`. `/ma:'bin/www'` tells the packaging tool to use `bin/www` as the argument for `node.exe`.
- **/AppType** defines the Service Fabric application type name.

If you browse to the directory that was specified in the /target parameter, you can see that the tool has created a fully functioning Service Fabric package as shown below:

```
|--[yourtargetdirectory]  
|-- NodeApplication  
|-- C  
|   |-- bin  
|   |-- data  
|   |-- node_modules  
|   |-- public  
|   |-- routes  
|   |-- views  
|   |-- app.js  
|   |-- package.json  
|   |-- node.exe  
|-- config  
|   |--Settings.xml  
|-- ServiceManifest.xml  
|-- ApplicationManifest.xml
```

The generated ServiceManifest.xml now has a section that describes how the Node.js web server should be launched, as shown in the code snippet below:

```
<CodePackage Name="C" Version="1.0">  
  <EntryPoint>  
    <ExeHost>  
      <Program>node.exe</Program>  
      <Arguments>'bin/www'</Arguments>  
      <WorkingFolder>CodePackage</WorkingFolder>  
    </ExeHost>  
  </EntryPoint>  
</CodePackage>
```

In this sample, the Node.js web server listens to port 3000, so you need to update the endpoint information in the ServiceManifest.xml file as shown below.

```
<Resources>
  <Endpoints>
    <Endpoint Name="NodeServiceEndpoint" Protocol="http" Port="3000" Type="Input" />
  </Endpoints>
</Resources>
```

## Packaging the MongoDB application

Now that you have packaged the Node.js application, you can go ahead and package MongoDB. As mentioned before, the steps that you go through now are not specific to Node.js and MongoDB. In fact, they apply to all applications that are meant to be packaged together as one Service Fabric application.

To package MongoDB, you want to make sure you package Mongod.exe and Mongo.exe. Both binaries are located in the `bin` directory of your MongoDB installation directory. The directory structure looks similar to the one below.

```
|-- MongoDB
|-- bin
|-- mongod.exe
|-- mongo.exe
|-- anybinary.exe
```

Service Fabric needs to start MongoDB with a command similar to the one below, so you need to use the `/ma` parameter when packaging MongoDB.

```
mongod.exe --dbpath [path to data]
```

### NOTE

The data is not being preserved in the case of a node failure if you put the MongoDB data directory on the local directory of the node. You should either use durable storage or implement a MongoDB replica set in order to prevent data loss.

In PowerShell or the command shell, we run the packaging tool with the following parameters:

```
.\ServiceFabricAppPackageUtil.exe /source: [yourdirectory]\MongoDB' /target:'[yourtargetdirectory]'
```

```
/appname:MongoDB /exe:'bin\mongod.exe' /ma:'--dbpath [path to data]' /AppType:NodeAppType
```

In order to add MongoDB to your Service Fabric application package, you need to make sure that the `/target` parameter points to the same directory that already contains the application manifest along with the Node.js application. You also need to make sure that you are using the same ApplicationType name.

Let's browse to the directory and examine what the tool has created.

```
|--[yourtargetdirectory]
|-- MyNodeApplication
|-- MongoDB
|   |-- C
|   |   |-- bin
|   |   |   |-- mongod.exe
|   |   |   |-- mongo.exe
|   |   |   |-- etc.
|   |-- config
|   |   |-- Settings.xml
|   |-- ServiceManifest.xml
|-- ApplicationManifest.xml
```

As you can see, the tool added a new folder, MongoDB, to the directory that contains the MongoDB binaries. If you open the `ApplicationManifest.xml` file, you can see that the package now contains both the Nodejs application and MongoDB. The code below shows the content of the application manifest.

```
<ApplicationManifest xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ApplicationTypeName="MyNodeApp" ApplicationTypeVersion="1.0" xmlns="http://schemas.microsoft.com/2011/01/fabric">
    <ServiceManifestImport>
        <ServiceManifestRef ServiceManifestName="MongoDB" ServiceManifestVersion="1.0" />
    </ServiceManifestImport>
    <ServiceManifestImport>
        <ServiceManifestRef ServiceManifestName="NodeService" ServiceManifestVersion="1.0" />
    </ServiceManifestImport>
    <DefaultServices>
        <Service Name="MongoDBService">
            <StatelessService ServiceTypeName="MongoDB">
                <SingletonPartition />
            </StatelessService>
        </Service>
        <Service Name="NodeServiceService">
            <StatelessService ServiceTypeName="NodeService">
                <SingletonPartition />
            </StatelessService>
        </Service>
    </DefaultServices>
</ApplicationManifest>
```

## Publishing the application

The last step is to publish the application to the local Service Fabric cluster by using the PowerShell scripts below:

```
Connect-ServiceFabricCluster localhost:19000

Write-Host 'Copying application package...'
Copy-ServiceFabricApplicationPackage -ApplicationPackagePath '[yourtargetdirectory]' -
ImageStoreConnectionString 'file:C:\SfDevCluster\Data\ImageStoreShare' -ApplicationPackagePathInImageStore
'NodeAppType'

Write-Host 'Registering application type...'
Register-ServiceFabricApplicationType -ApplicationPathInImageStore 'NodeAppType'

New-ServiceFabricApplication -ApplicationName 'fabric:/NodeApp' -ApplicationTypeName 'NodeAppType' -
ApplicationTypeVersion 1.0
```

Once the application is successfully published to the local cluster, you can access the Node.js application on the port that we have entered in the service manifest of the Node.js application--for example <http://localhost:3000>.

In this tutorial, you have seen how to easily package two existing applications as one Service Fabric application. You have also learned how to deploy it to Service Fabric so that it can benefit from some of the Service Fabric features, such as high availability and health system integration.

## Adding more guest executables to an existing application using Yeoman on Linux

To add another service to an application already created using `yo`, perform the following steps:

1. Change directory to the root of the existing application. For example, `cd ~/YeomanSamples/MyApplication`, if `MyApplication` is the application created by Yeoman.
2. Run `yo azuresfguest:AddService` and provide the necessary details.

## Next steps

- Learn about deploying containers with [Service Fabric and containers overview](#)
- [Sample for packaging and deploying a guest executable](#)
- [Sample of two guest executables \(C# and nodejs\) communicating via the Naming service using REST](#)

# Create your first Service Fabric container application on Windows

8/15/2017 • 12 min to read • [Edit Online](#)

Running an existing application in a Windows container on a Service Fabric cluster doesn't require any changes to your application. This article walks you through creating a Docker image containing a Python [Flask](#) web application and deploying it to a Service Fabric cluster. You will also share your containerized application through [Azure Container Registry](#). This article assumes a basic understanding of Docker. You can learn about Docker by reading the [Docker Overview](#).

## Prerequisites

A development computer running:

- Visual Studio 2015 or Visual Studio 2017.
- [Service Fabric SDK and tools](#).
- Docker for Windows. [Get Docker CE for Windows \(stable\)](#). After installing and starting Docker, right-click on the tray icon and select **Switch to Windows containers**. This is required to run Docker images based on Windows.

A Windows cluster with three or more nodes running on Windows Server 2016 with Containers- [Create a cluster](#) or [try Service Fabric for free](#).

A registry in Azure Container Registry - [Create a container registry](#) in your Azure subscription.

## Define the Docker container

Build an image based on the [Python image](#) located on Docker Hub.

Define your Docker container in a Dockerfile. The Dockerfile contains instructions for setting up the environment inside your container, loading the application you want to run, and mapping ports. The Dockerfile is the input to the `docker build` command, which creates the image.

Create an empty directory and create the file *Dockerfile* (with no file extension). Add the following to *Dockerfile* and save your changes:

```
# Use an official Python runtime as a base image
FROM python:2.7-windowsservercore

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
ADD . /app

# Install any needed packages specified in requirements.txt
RUN pip install -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

Read the [Dockerfile reference](#) for more information.

## Create a simple web application

Create a Flask web application listening on port 80 that returns "Hello World!". In the same directory, create the file *requirements.txt*. Add the following and save your changes:

```
Flask
```

Also create the *app.py* file and add the following:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello():

    return 'Hello World!'

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80)
```

## Build the image

Run the `docker build` command to create the image that runs your web application. Open a PowerShell window and navigate to the directory containing the Dockerfile. Run the following command:

```
docker build -t helloworldapp .
```

This command builds the new image using the instructions in your Dockerfile, naming (-t tagging) the image "helloworldapp". Building an image pulls the base image down from Docker Hub and creates a new image that adds your application on top of the base image.

Once the build command completes, run the `docker images` command to see information on the new image:

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
helloworldapp	latest	8ce25f5d6a79	2 minutes ago	10.4 GB

## Run the application locally

Verify your image locally before pushing it to the container registry.

Run the application:

```
docker run -d --name my-web-site helloworldapp
```

*name* gives a name to the running container (instead of the container ID).

Once the container starts, find its IP address so that you can connect to your running container from a browser:

```
docker inspect -f "{{ .NetworkSettings.Networks.nat.IPAddress }}" my-web-site
```

Connect to the running container. Open a web browser pointing to the IP address returned, for example "<http://172.31.194.61>". You should see the heading "Hello World!" display in the browser.

To stop your container, run:

```
docker stop my-web-site
```

Delete the container from your development machine:

```
docker rm my-web-site
```

## Push the image to the container registry

After you verify that the container runs on your development machine, push the image to your registry in Azure Container Registry.

Run `docker login` to log in to your container registry with your [registry credentials](#).

The following example passes the ID and password of an Azure Active Directory [service principal](#). For example, you might have assigned a service principal to your registry for an automation scenario. Or, you could login using your registry username and password.

```
docker login myregistry.azurecr.io -u xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx -p myPassword
```

The following command creates a tag, or alias, of the image, with a fully qualified path to your registry. This example places the image in the `samples` namespace to avoid clutter in the root of the registry.

```
docker tag helloworldapp myregistry.azurecr.io/samples/helloworldapp
```

Push the image to your container registry:

```
docker push myregistry.azurecr.io/samples/helloworldapp
```

## Create the containerized service in Visual Studio

The Service Fabric SDK and tools provide a service template to help you create a containerized application.

1. Start Visual Studio. Select **File > New > Project**.
2. Select **Service Fabric application**, name it "MyFirstContainer", and click **OK**.
3. Select **Guest Container** from the list of **service templates**.
4. In **Image Name** enter "myregistry.azurecr.io/samples/helloworldapp", the image you pushed to your container repository.
5. Give your service a name, and click **OK**.

## Configure communication

The containerized service needs an endpoint for communication. Add an `Endpoint` element with the protocol, port, and type to the ServiceManifest.xml file. For this article, the containerized service listens on port 8081. In this example, a fixed port 8081 is used. If no port is specified, a random port from the application port range is chosen.

```
<Resources>
  <Endpoints>
    <Endpoint Name="Guest1TypeEndpoint" UriScheme="http" Port="8081" Protocol="http"/>
  </Endpoints>
</Resources>
```

By defining an endpoint, Service Fabric publishes the endpoint to the Naming service. Other services running in the cluster can resolve this container. You can also perform container-to-container communication using the [reverse proxy](#). Communication is performed by providing the reverse proxy HTTP listening port and the name of the services that you want to communicate with as environment variables.

## Configure and set environment variables

Environment variables can be specified for each code package in the service manifest. This feature is available for all services irrespective of whether they are deployed as containers or processes or guest executables. You can override environment variable values in the application manifest or specify them during deployment as application parameters.

The following service manifest XML snippet shows an example of how to specify environment variables for a code package:

```
<CodePackage Name="Code" Version="1.0.0">
  ...
  <EnvironmentVariables>
    <EnvironmentVariable Name="HttpGatewayPort" Value="" />
  </EnvironmentVariables>
</CodePackage>
```

These environment variables can be overridden in the application manifest:

```
<ServiceManifestImport>
  <ServiceManifestRef ServiceManifestName="Guest1Pkg" ServiceManifestVersion="1.0.0" />
    <EnvironmentVariable Name="HttpGatewayPort" Value="19080"/>
  </EnvironmentOverrides>
  ...
</ServiceManifestImport>
```

## Configure container port-to-host port mapping and container-to-container discovery

Configure a host port used to communicate with the container. The port binding maps the port on which the service is listening inside the container to a port on the host. Add a `PortBinding` element in `ContainerHostPolicies` element of the ApplicationManifest.xml file. For this article, `ContainerPort` is 80 (the container exposes port 80, as specified in the Dockerfile) and `EndpointRef` is "Guest1TypeEndpoint" (the endpoint previously defined in the service manifest). Incoming requests to the service on port 8081 are mapped to port 80 on the container.

```
<Policies>
  <ContainerHostPolicies CodePackageRef="Code">
    <PortBinding ContainerPort="80" EndpointRef="Guest1TypeEndpoint"/>
  </ContainerHostPolicies>
</Policies>
```

## Configure container registry authentication

Configure container registry authentication by adding `RepositoryCredentials` to `ContainerHostPolicies` of the ApplicationManifest.xml file. Add the account and password for the myregistry.azurecr.io container registry, which allows the service to download the container image from the repository.

```
<Policies>
  <ContainerHostPolicies CodePackageRef="Code">
    <RepositoryCredentials AccountName="myregistry" Password="=P==/==/8=/=+u4lyOB=+=nWzEeRff=" PasswordEncrypted="false"/>
    <PortBinding ContainerPort="80" EndpointRef="Guest1TypeEndpoint"/>
  </ContainerHostPolicies>
</Policies>
```

We recommend that you encrypt the repository password by using an encipherment certificate that's deployed to all nodes of the cluster. When Service Fabric deploys the service package to the cluster, the encipherment certificate is used to decrypt the cipher text. The `Invoke-ServiceFabricEncryptText` cmdlet is used to create the cipher text for the password, which is added to the ApplicationManifest.xml file.

The following script creates a new self-signed certificate and exports it to a PFX file. The certificate is imported into an existing key vault and then deployed to the Service Fabric cluster.

```

# Variables.
$certpwd = ConvertTo-SecureString -String "Pa$$word321!" -Force -AsPlainText
$filepath = "C:\MyCertificates\dataenciphermentcert.pfx"
$subjectname = "dataencipherment"
$vaultname = "mykeyvault"
$certificateName = "dataenciphermentcert"
$groupname="myclustergroup"
$clustername = "mycluster"

$subscriptionId = "subscription ID"

Login-AzureRmAccount

Select-AzureRmSubscription -SubscriptionId $subscriptionId

# Create a self signed cert, export to PFX file.
New-SelfSignedCertificate -Type DocumentEncryptionCert -KeyUsage DataEncipherment -Subject $subjectname -Provider 'Microsoft Enhanced Cryptographic Provider v1.0' `| Export-PfxCertificate -FilePath $filepath -Password $certpwd

# Import the certificate to an existing key vault. The key vault must be enabled for deployment.
$cer = Import-AzureKeyVaultCertificate -VaultName $vaultName -Name $certificateName -FilePath $filepath -Password $certpwd

Set-AzureRmKeyVaultAccessPolicy -VaultName $vaultName -ResourceGroupName $groupname -EnabledForDeployment

# Add the certificate to all the VMs in the cluster.
Add-AzureRmServiceFabricApplicationCertificate -ResourceGroupName $groupname -Name $clustername -SecretIdentifier $cer.SecretId

```

Encrypt the password using the [Invoke-ServiceFabricEncryptText](#) cmdlet.

```

$text = "=P==/==/8=/+u4lyOB=+=nWzEeRFF="
Invoke-ServiceFabricEncryptText -CertStore -CertThumbprint $cer.Thumbprint -Text $text -StoreLocation Local -StoreName My

```

Replace the password with the cipher text returned by the [Invoke-ServiceFabricEncryptText](#) cmdlet and set

`PasswordEncrypted` to "true".

```

<Policies>
  <ContainerHostPolicies CodePackageRef="Code">
    <RepositoryCredentials AccountName="myregistry" Password="MIIB6QYJKoZIhvNAQcDoIIB2jCCAdYCAQAxggFRMIIBTQIBADA1MCExHzAdBgNVBAMMFnJ5YW53aWRhdGF1bmNpcGhlcm1lbnQC
      EFfyjOX/17S6RIOsJA6UZ1QwDQYJKoZIhvcaNAQEhMAAEg gEAS7oqxvoz8i6+8zULhDzFpBpOTLU+c2mhBdqXpkLwVfcmwUNA82rEWG57V11jZXe7J9BkW91y4xhU8BbARkZHLEuKqg0saTrTHsMBQ6KMQDo
      tSDu8m8Y2BR5Y100wRjvVx3y5+iNYuy/Jm
      gSrNyyMQ/45HfMuVb5B4rwnuP8PAkXNT9VLbPeqAfxsMkYg+vGCDEtd8m+bX/7Xgp/kfwxymOuUCrq/YmSwe9QTG3pBri7Hq1K3zEpX4FH/7W2
      Zb4o3fBAQ+FuxH4nFjFNoYG29inL0bKEct
      yNZNKrvhdM3n1Uk/8W2Hr62FQ33HgeFR1yxQjLsUu800PrYcR5tLfytB8BqkqhkiG9w0BBwEwHQYJYIZIAWDBAEqBBBybgM5NUV8BeetUbMR8
      mJhgFBvVSUsnp9B8Ryebmtgu36dZiS0bDsI
      NtTv1zhk11LIlae/5kjPv95r3lw6DHmV4kXLwiCNlcWPYIWBGIuspwyG+28EWsrHmN7Dt2WqeNQ==" PasswordEncrypted="true"/>
    <PortBinding ContainerPort="80" EndpointRef="Guest1TypeEndpoint"/>
  </ContainerHostPolicies>
</Policies>

```

## Configure isolation mode

Windows supports two isolation modes for containers: process and Hyper-V. With the process isolation mode, all the containers running on the same host machine share the kernel with the host. With the Hyper-V isolation mode, the kernels are isolated between each Hyper-V container and the container host. The isolation mode is specified in

the `ContainerHostPolicies` element in the application manifest file. The isolation modes that can be specified are `process`, `hyperv`, and `default`. The default isolation mode defaults to `process` on Windows Server hosts, and defaults to `hyperv` on Windows 10 hosts. The following snippet shows how the isolation mode is specified in the application manifest file.

```
<ContainerHostPolicies CodePackageRef="Code" Isolation="hyperv">
```

## Configure resource governance

[Resource governance](#) restricts the resources that the container can use on the host. The `ResourceGovernancePolicy` element, which is specified in the application manifest, is used to declare resource limits for a service code package. Resource limits can be set for the following resources: Memory, MemorySwap, CpuShares (CPU relative weight), MemoryReservationInMB, BlkioWeight (BlockIO relative weight). In this example, service package `Guest1Pkg` gets one core on the cluster nodes where it is placed. Memory limits are absolute, so the code package is limited to 1024 MB of memory (and a soft-guarantee reservation of the same). Code packages (containers or processes) are not able to allocate more memory than this limit, and attempting to do so results in an out-of-memory exception. For resource limit enforcement to work, all code packages within a service package should have memory limits specified.

```
<ServiceManifestImport>
  <ServiceManifestRef ServiceManifestName="Guest1Pkg" ServiceManifestVersion="1.0.0" />
  <Policies>
    <ServicePackageResourceGovernancePolicy CpuCores="1"/>
    <ResourceGovernancePolicy CodePackageRef="Code" MemoryInMB="1024" />
  </Policies>
</ServiceManifestImport>
```

## Deploy the container application

Save all your changes and build the application. To publish your application, right-click on **MyFirstContainer** in Solution Explorer and select **Publish**.

In **Connection Endpoint**, enter the management endpoint for the cluster. For example, "containercluster.westus2.cloudapp.azure.com:19000". You can find the client connection endpoint in the Overview blade for your cluster in the [Azure portal](#).

Click **Publish**.

[Service Fabric Explorer](#) is a web-based tool for inspecting and managing applications and nodes in a Service Fabric cluster. Open a browser and navigate to <http://containercluster.westus2.cloudapp.azure.com:19080/Explorer/> and follow the application deployment. The application deploys but is in an error state until the image is downloaded on the cluster nodes (which can take some time, depending on the image size):

Name	Application Type	Version	Health State	Status
fabric/MyFirstContainer	MyFirstContainerType	1.0.0	Error	Ready

The application is ready when it's in `Ready` state:

The screenshot shows the Microsoft Azure Service Fabric Explorer interface. On the left, there's a navigation tree with nodes like 'Cluster', 'Applications' (under 'MyFirstContainerType'), 'Nodes' (under '\_nodetype1\_0'), and 'Instances'. A specific instance ID '131380355716907738' is selected. The main pane displays the 'ESSENTIALS' tab for this instance. It shows the 'Id' as '131380355716907738', 'Service Kind' as 'Stateless', and 'Status' as 'Ready'. The 'Service Name' is listed as 'fabric:/MyFirstContainer/Guest1'. Below this, the 'ADDRESS' section shows an 'Endpoints' table with one entry: 'Guest 1 Type Endpoint' and its URL 'http://10.0.0.4:80/'. There are also 'DETAILS' and 'Logs' tabs at the top right.

Open a browser and navigate to <http://containercluster.westus2.cloudapp.azure.com:8081>. You should see the heading "Hello World!" display in the browser.

## Clean up

You continue to incur charges while the cluster is running, consider [deleting your cluster](#). [Party clusters](#) are automatically deleted after a few hours.

After you push the image to the container registry you can delete the local image from your development computer:

```
docker rmi helloworldapp
docker rmi myregistry.azurecr.io/samples/helloworldapp
```

## Complete example Service Fabric application and service manifests

Here are the complete service and application manifests used in this article.

### ServiceManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<ServiceManifest Name="Guest1Pkg"
    Version="1.0.0"
    xmlns="http://schemas.microsoft.com/2011/01/fabric"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <ServiceTypes>
        <!-- This is the name of your ServiceType.
            The UseImplicitHost attribute indicates this is a guest service. -->
        <StatelessServiceType ServiceTypeName="Guest1Type" UseImplicitHost="true" />
    </ServiceTypes>

        <!-- Code package is your service executable. -->
    <CodePackage Name="Code" Version="1.0.0">
        <EntryPoint>
            <!-- Follow this link for more information about deploying Windows containers to Service Fabric:
                https://aka.ms/sfguestcontainers -->
            <ContainerHost>
                <ImageName>myregistry.azurecr.io/samples/helloworldapp</ImageName>
            </ContainerHost>
        </EntryPoint>
        <!-- Pass environment variables to your container: -->
        <EnvironmentVariables>
            <EnvironmentVariable Name="HttpGatewayPort" Value="" />
            <EnvironmentVariable Name="BackendServiceName" Value="" />
        </EnvironmentVariables>
    </CodePackage>

        <!-- Config package is the contents of the Config directory under PackageRoot that contains an
            independently-updateable and versioned set of custom configuration settings for your service. -->
    <ConfigPackage Name="Config" Version="1.0.0" />

    <Resources>
        <Endpoints>
            <!-- This endpoint is used by the communication listener to obtain the port on which to
                listen. Please note that if your service is partitioned, this port is shared with
                replicas of different partitions that are placed in your code. -->
            <Endpoint Name="Guest1TypeEndpoint" UriScheme="http" Port="8081" Protocol="http"/>
        </Endpoints>
    </Resources>
</ServiceManifest>

```

## ApplicationManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<ApplicationManifest ApplicationTypeName="MyFirstContainerType"
    ApplicationTypeVersion="1.0.0"
    xmlns="http://schemas.microsoft.com/2011/01/fabric"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <Parameters>
        <Parameter Name="Guest1_InstanceCount" DefaultValue="-1" />
    </Parameters>
    <!-- Import the ServiceManifest from the ServicePackage. The ServiceManifestName and ServiceManifestVersion
        should match the Name and Version attributes of the ServiceManifest element defined in the
        ServiceManifest.xml file. -->
    <ServiceManifestImport>
        <ServiceManifestRef ServiceManifestName="Guest1Pkg" ServiceManifestVersion="1.0.0" />
        <EnvironmentOverrides CodePackageRef="FrontendService.Code">
            <EnvironmentVariable Name="HttpGatewayPort" Value="19080"/>
        </EnvironmentOverrides>
        <ConfigOverrides />
        <Policies>
            <ContainerHostPolicies CodePackageRef="Code">
                <RepositoryCredentials AccountName="myregistry"
Password="MIIB6QYJKoZIhvcNAQcDoIIB2jCCAdYCAQAxggFRMIIBTQIBADA1MCExHzAdBgNVBAMMFnJ5YW53aWRhdGF1bmNpcGhlcmlbnQC
EFfyjOX/17S6RtOsja6UZ1QwDQYJKoZIhvcNAQEHMAAEg
gEAS7oqxvoz8i6+8zULhDzFpBpOTLU+c2mhBdqXpkLwVfcmlWUNA82rEWG57Vl1jZXe7J9BkW9ly4xhU8BbARKZHLEuKqg0saTrTHsMBQ6KMQDo
tSDu8m8Y2BR5Y100wRjvVx3y5+iNYuy/JmM
gSrNyyMQ/45HfMuVb5B4rwnuP8PAkXNT9VLbPeqAfxsMkYg+vGCDEtd8m+bX/7Xgp/kfwxymOuUCrq/YmSwe9QTG3pBri7Hq1K3zEpX4FH/7W2
Zb4o3fBAQ+FuxH4nFjFNoYG29inL0bKEct
yNZNKrvhdM3n1Uk/8W2Hr62FQ33HgeFR1yxQjLsUu800PrYcR5tLfytB8Bqkqhkig9w0BBwEwHQYJYIZIAWUDBAEqBBBybgM5NUV8BeetUbMR8
mJhgFBvVSUsnp9B8Ryebmtgu36dZiSoBdsI
NtTv1zhk11LIlae/5kjPv95r3lw6DHmV4kXLwiCNlcWPYIWBGItuspwg+28EWsrHmN7Dt2WqeNQ==" PasswordEncrypted="true"/>
                <PortBinding ContainerPort="80" EndpointRef="Guest1TypeEndpoint"/>
            </ContainerHostPolicies>
            <ServicePackageResourceGovernancePolicy CpuCores="1"/>
            <ResourceGovernancePolicy CodePackageRef="Code" MemoryInMB="1024" />
        </Policies>
    </ServiceManifestImport>
    <DefaultServices>
        <!-- The section below creates instances of service types, when an instance of this
            application type is created. You can also create one or more instances of service type using the
            ServiceFabric PowerShell module.

            The attribute ServiceTypeName below must match the name defined in the imported ServiceManifest.xml
            file. -->
        <Service Name="Guest1">
            <StatelessService ServiceTypeName="Guest1Type" InstanceCount="[Guest1_InstanceCount]">
                <SingletonPartition />
            </StatelessService>
        </Service>
    </DefaultServices>
</ApplicationManifest>

```

## Configure time interval before container is force terminated

You can configure a time interval for the runtime to wait before the container is removed after the service deletion (or a move to another node) has started. Configuring the time interval sends the `docker stop <time in seconds>` command to the container. For more detail, see [docker stop](#). The time interval to wait is specified under the `Hosting` section. The following cluster manifest snippet shows how to set the wait interval:

```
{  
    "name": "Hosting",  
    "parameters": [  
        {  
            "ContainerDeactivationTimeout": "10",  
            ...  
        }  
    ]  
}
```

The default time interval is set to 10 seconds. Since this configuration is dynamic, a config only upgrade on the cluster updates the timeout.

## Configure the runtime to remove unused container images

You can configure the Service Fabric cluster to remove unused container images from the node. This configuration allows disk space to be recaptured if too many container images are present on the node. To enable this feature, update the `Hosting` section in the cluster manifest as shown in the following snippet:

```
{  
    "name": "Hosting",  
    "parameters": [  
        {  
            "PruneContainerImages": "True",  
            "ContainerImagesToSkip": "microsoft/windowsservercore|microsoft/nanoserver|...",  
            ...  
        }  
    ]  
}
```

For images that should not be deleted, you can specify them under the `ContainerImagesToSkip` parameter.

## Next steps

- Learn more about running [containers on Service Fabric](#).
- Read the [Deploy a .NET application in a container](#) tutorial.
- Learn about the Service Fabric [application life-cycle](#).
- Checkout the [Service Fabric container code samples](#) on GitHub.

# Create your first Service Fabric container application on Linux

10/5/2017 • 9 min to read • [Edit Online](#)

Running an existing application in a Linux container on a Service Fabric cluster doesn't require any changes to your application. This article walks you through creating a Docker image containing a Python [Flask](#) web application and deploying it to a Service Fabric cluster. You will also share your containerized application through [Azure Container Registry](#). This article assumes a basic understanding of Docker. You can learn about Docker by reading the [Docker Overview](#).

## Prerequisites

- A development computer running:
  - [Service Fabric SDK and tools](#).
  - [Docker CE for Linux](#).
  - [Service Fabric CLI](#)
- A registry in Azure Container Registry - [Create a container registry](#) in your Azure subscription.

## Define the Docker container

Build an image based on the [Python image](#) located on Docker Hub.

Define your Docker container in a Dockerfile. The Dockerfile contains instructions for setting up the environment inside your container, loading the application you want to run, and mapping ports. The Dockerfile is the input to the `docker build` command, which creates the image.

Create an empty directory and create the file *Dockerfile* (with no file extension). Add the following to *Dockerfile* and save your changes:

```
# Use an official Python runtime as a base image
FROM python:2.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
ADD . /app

# Install any needed packages specified in requirements.txt
RUN pip install -r requirements.txt

# Make port 80 available outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

Read the [Dockerfile reference](#) for more information.

## Create a simple web application

Create a Flask web application listening on port 80 that returns "Hello World!". In the same directory, create the file *requirements.txt*. Add the following and save your changes:

```
Flask
```

Also create the *app.py* file and add the following:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello():

    return 'Hello World!'

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80)
```

## Build the image

Run the `docker build` command to create the image that runs your web application. Open a PowerShell window and navigate to `c:\temp\helloworldapp`. Run the following command:

```
docker build -t helloworldapp .
```

This command builds the new image using the instructions in your Dockerfile, naming (-t tagging) the image "helloworldapp". Building an image pulls the base image down from Docker Hub and creates a new image that adds your application on top of the base image.

Once the build command completes, run the `docker images` command to see information on the new image:

```
$ docker images

REPOSITORY          TAG      IMAGE ID   CREATED        SIZE
helloworldapp       latest   86838648aab6  2 minutes ago  194 MB
```

## Run the application locally

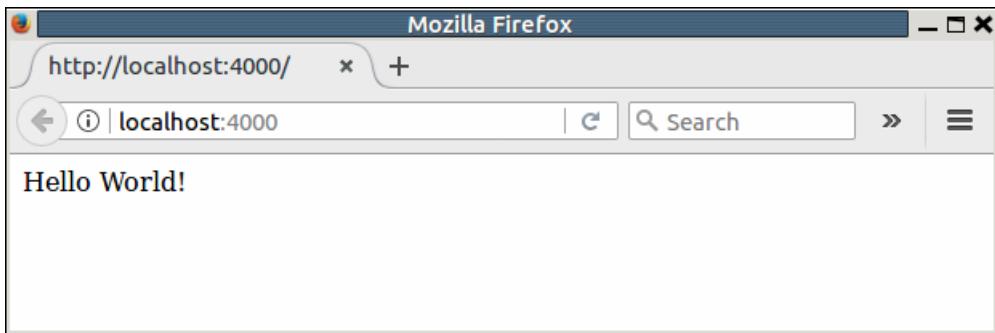
Verify that your containerized application runs locally before pushing it to the container registry.

Run the application, mapping your computer's port 4000 to the container's exposed port 80:

```
docker run -d -p 4000:80 --name my-web-site helloworldapp
```

*name* gives a name to the running container (instead of the container ID).

Connect to the running container. Open a web browser pointing to the IP address returned on port 4000, for example "<http://localhost:4000>". You should see the heading "Hello World!" display in the browser.



To stop your container, run:

```
docker stop my-web-site
```

Delete the container from your development machine:

```
docker rm my-web-site
```

## Push the image to the container registry

After you verify that the application runs in Docker, push the image to your registry in Azure Container Registry.

Run `docker login` to log in to your container registry with your [registry credentials](#).

The following example passes the ID and password of an Azure Active Directory [service principal](#). For example, you might have assigned a service principal to your registry for an automation scenario. Or, you could login using your registry username and password.

```
docker login myregistry.azurecr.io -u xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx -p myPassword
```

The following command creates a tag, or alias, of the image, with a fully qualified path to your registry. This example places the image in the `samples` namespace to avoid clutter in the root of the registry.

```
docker tag helloworldapp myregistry.azurecr.io/samples/helloworldapp
```

Push the image to your container registry:

```
docker push myregistry.azurecr.io/samples/helloworldapp
```

## Package the Docker image with Yeoman

The Service Fabric SDK for Linux includes a [Yeoman](#) generator that makes it easy to create your application and add a container image. Let's use Yeoman to create an application with a single Docker container called `SimpleContainerApp`.

To create a Service Fabric container application, open a terminal window and run `yo azuresfcontainer`.

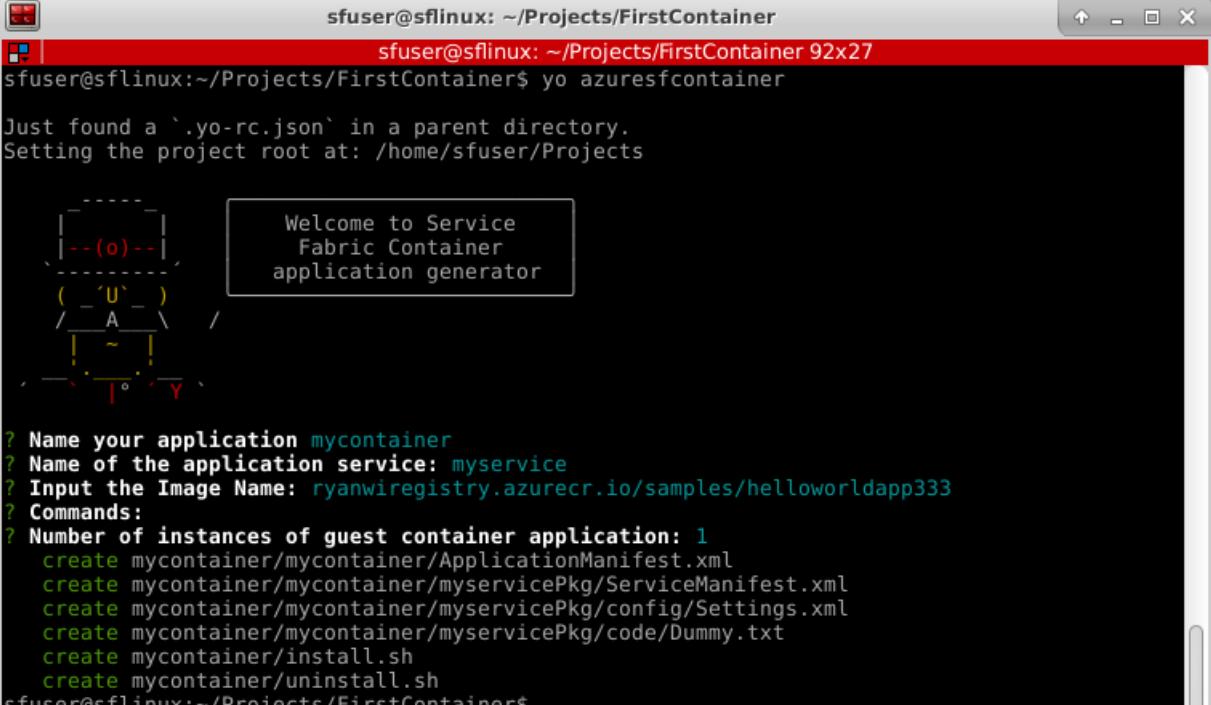
Name your application (for example, "mycontainer") and name the application service (for example, "myservice").

For the image name, provide the URL for the container image in a container registry (for example, "myregistry.azurecr.io/samples/helloworldapp").

Since this image has a workload entry-point defined, you don't need to explicitly specify input commands

(commands run inside the container, which will keep the container running after startup).

Specify an instance count of "1".



```
sfuser@sflinux: ~/Projects/FirstContainer
sfuser@sflinux: ~/Projects/FirstContainer 92x27
sfuser@sflinux:~/Projects/FirstContainer$ yo azuresfcontainer

Just found a `yo-rc.json` in a parent directory.
Setting the project root at: /home/sfuser/Projects

  [---]
  |--(o)---|
  ( - U - )
  / A \ /
  | ~ |
  . Y .

? Name your application mycontainer
? Name of the application service: myservice
? Input the Image Name: ryanwireregistry.azurecr.io/samples/helloworldapp333
? Commands:
? Number of instances of guest container application: 1
  create mycontainer/mycontainer/ApplicationManifest.xml
  create mycontainer/mycontainer/myservicePkg/ServiceManifest.xml
  create mycontainer/mycontainer/myservicePkg/config/Settings.xml
  create mycontainer/mycontainer/myservicePkg/code/Dummy.txt
  create mycontainer/install.sh
  create mycontainer/uninstall.sh
sfuser@sflinux:~/Projects/FirstContainers$
```

## Configure port mapping and container repository authentication

Your containerized service needs an endpoint for communication. Now add the protocol, port, and type to an `Endpoint` in the `ServiceManifest.xml` file under the 'Resources' tag. For this article, the containerized service listens on port 4000:

```
<Resources>
  <Endpoints>
    <!-- This endpoint is used by the communication listener to obtain the port on which to
        listen. Please note that if your service is partitioned, this port is shared with
        replicas of different partitions that are placed in your code. -->
    <Endpoint Name="myServiceTypeEndpoint" UriScheme="http" Port="4000" Protocol="http"/>
  </Endpoints>
</Resources>
```

Providing the `UriScheme` automatically registers the container endpoint with the Service Fabric Naming service for discoverability. A full `ServiceManifest.xml` example file is provided at the end of this article.

Configure the container port-to-host port mapping by specifying a `PortBinding` policy in `ContainerHostPolicies` of the `ApplicationManifest.xml` file. For this article, `ContainerPort` is 80 (the container exposes port 80, as specified in the Dockerfile) and `EndpointRef` is "myServiceTypeEndpoint" (the endpoint defined in the service manifest). Incoming requests to the service on port 4000 are mapped to port 80 on the container. If your container needs to authenticate with a private repository, then add `RepositoryCredentials`. For this article, add the account name and password for the `myregistry.azurecr.io` container registry. Ensure the policy is added under the '`ServiceManifestImport`' tag corresponding to the right service package.

```
<ServiceManifestImport>
  <ServiceManifestRef ServiceManifestName="MyServicePkg" ServiceManifestVersion="1.0.0" />
<Policies>
  <ContainerHostPolicies CodePackageRef="Code">
    <RepositoryCredentials AccountName="myregistry" Password="=P==/==/=8=/=+u4lyOB=+=nWzEeRfF="
    PasswordEncrypted="false"/>
    <PortBinding ContainerPort="80" EndpointRef="myServiceTypeEndpoint"/>
  </ContainerHostPolicies>
</Policies>
</ServiceManifestImport>
```

## Build and package the Service Fabric application

The Service Fabric Yeoman templates include a build script for [Gradle](#), which you can use to build the application from the terminal. To build and package the application, run the following:

```
cd mycontainer
gradle
```

## Deploy the application

Once the application is built, you can deploy it to the local cluster using the Service Fabric CLI.

Connect to the local Service Fabric cluster.

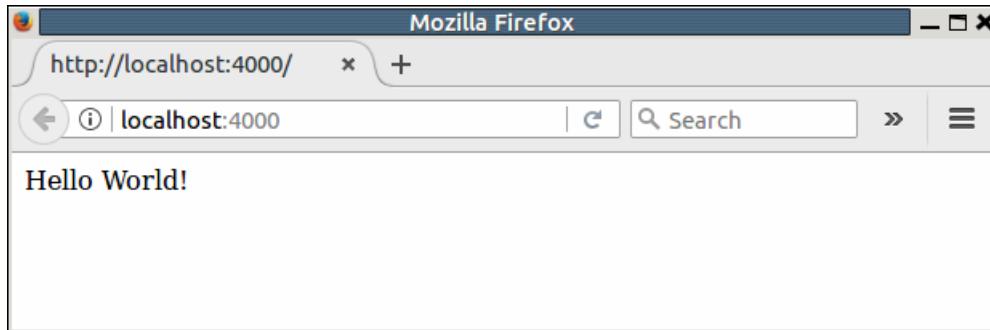
```
sfctl cluster select --endpoint http://localhost:19080
```

Use the install script provided in the template to copy the application package to the cluster's image store, register the application type, and create an instance of the application.

```
./install.sh
```

Open a browser and navigate to Service Fabric Explorer at <http://localhost:19080/Explorer> (replace localhost with the private IP of the VM if using Vagrant on Mac OS X). Expand the Applications node and note that there is now an entry for your application type and another for the first instance of that type.

Connect to the running container. Open a web browser pointing to the IP address returned on port 4000, for example "<http://localhost:4000>". You should see the heading "Hello World!" display in the browser.



## Clean up

Use the uninstall script provided in the template to delete the application instance from the local development cluster and unregister the application type.

```
./uninstall.sh
```

After you push the image to the container registry you can delete the local image from your development computer:

```
docker rmi helloworldapp
docker rmi myregistry.azurecr.io/samples/helloworldapp
```

## Complete example Service Fabric application and service manifests

Here are the complete service and application manifests used in this article.

### ServiceManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceManifest Name="myservicePkg"
    Version="1.0.0"
    xmlns="http://schemas.microsoft.com/2011/01/fabric"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <ServiceTypes>
        <!-- This is the name of your ServiceType.
            The UseImplicitHost attribute indicates this is a guest service. -->
        <StatelessServiceType ServiceTypeName="myserviceType" UseImplicitHost="true" />
    </ServiceTypes>

        <!-- Code package is your service executable. -->
    <CodePackage Name="Code" Version="1.0.0">
        <EntryPoint>
            <!-- Follow this link for more information about deploying Windows containers
                to Service Fabric: https://aka.ms/sfguestcontainers -->
            <ContainerHost>
                <ImageName>myregistry.azurecr.io/samples/helloworldapp</ImageName>
                <Commands></Commands>
            </ContainerHost>
        </EntryPoint>
        <!-- Pass environment variables to your container: -->

        <EnvironmentVariables>
            <!--
                <EnvironmentVariable Name="VariableName" Value="VariableValue"/>
            -->
        </EnvironmentVariables>
    </CodePackage>

    <Resources>
        <Endpoints>
            <!-- This endpoint is used by the communication listener to obtain the port on which to
                listen. Please note that if your service is partitioned, this port is shared with
                replicas of different partitions that are placed in your code. -->
            <Endpoint Name="myServiceTypeEndpoint" UriScheme="http" Port="4000" Protocol="http"/>
        </Endpoints>
    </Resources>
</ServiceManifest>
```

### ApplicationManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<ApplicationManifest ApplicationTypeName="mycontainerType"
    ApplicationTypeVersion="1.0.0"
    xmlns="http://schemas.microsoft.com/2011/01/fabric"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <!-- Import the ServiceManifest from the ServicePackage. The ServiceManifestName and ServiceManifestVersion
        should match the Name and Version attributes of the ServiceManifest element defined in the
        ServiceManifest.xml file. -->
    <ServiceManifestImport>
        <ServiceManifestRef ServiceManifestName="myservicePkg" ServiceManifestVersion="1.0.0" />
        <ConfigOverrides />
        <Policies>
            <ContainerHostPolicies CodePackageRef="Code">
                <RepositoryCredentials AccountName="myregistry" Password="=P==/==/=8=/=+u4lyOB=+=nWzEeRff="
                    PasswordEncrypted="false"/>
                <PortBinding ContainerPort="80" EndpointRef="myServiceTypeEndpoint"/>
            </ContainerHostPolicies>
        </Policies>
    </ServiceManifestImport>
    <DefaultServices>
        <!-- The section below creates instances of service types, when an instance of this
            application type is created. You can also create one or more instances of service type using the
            ServiceFabric PowerShell module.

            The attribute ServiceTypeName below must match the name defined in the imported ServiceManifest.xml
            file. -->
        <Service Name="myservice">
            <!-- On a local development cluster, set InstanceCount to 1. On a multi-node production
                cluster, set InstanceCount to -1 for the container service to run on every node in
                the cluster.
            -->
            <StatelessService ServiceTypeName="myserviceType" InstanceCount="1">
                <SingletonPartition />
            </StatelessService>
        </Service>
    </DefaultServices>
</ApplicationManifest>

```

## Adding more services to an existing application

To add another container service to an application already created using yeoman, perform the following steps:

1. Change directory to the root of the existing application. For example, `cd ~/YeomanSamples/MyApplication`, if `MyApplication` is the application created by Yeoman.
2. Run `yo azuresfcontainer:AddService`

## Configure time interval before container is force terminated

You can configure a time interval for the runtime to wait before the container is removed after the service deletion (or a move to another node) has started. Configuring the time interval sends the `docker stop <time in seconds>` command to the container. For more detail, see [docker stop](#). The time interval to wait is specified under the `Hosting` section. The following cluster manifest snippet shows how to set the wait interval:

```
{  
    "name": "Hosting",  
    "parameters": [  
        {  
            "ContainerDeactivationTimeout": "10",  
            ...  
        }  
    ]  
}
```

The default time interval is set to 10 seconds. Since this configuration is dynamic, a config only upgrade on the cluster updates the timeout.

## Configure the runtime to remove unused container images

You can configure the Service Fabric cluster to remove unused container images from the node. This configuration allows disk space to be recaptured if too many container images are present on the node. To enable this feature, update the `Hosting` section in the cluster manifest as shown in the following snippet:

```
{  
    "name": "Hosting",  
    "parameters": [  
        {  
            "PruneContainerImages": "True",  
            "ContainerImagesToSkip": "microsoft/windowsservercore|microsoft/nanoserver|...",  
            ...  
        }  
    ]  
}
```

For images that should not be deleted, you can specify them under the `ContainerImagesToSkip` parameter.

## Next steps

- Learn more about running [containers on Service Fabric](#).
- Read the [Deploy a .NET application in a container](#) tutorial.
- Learn about the Service Fabric [application life-cycle](#).
- Checkout the [Service Fabric container code samples](#) on GitHub.

# Container security

9/25/2017 • 2 min to read • [Edit Online](#)

Service Fabric provides a mechanism for services inside a container to access a certificate that is installed on the nodes in a Windows or Linux cluster (version 5.7 or higher). In addition, Service Fabric also supports gMSA (group Managed Service Accounts) for Windows containers.

## Certificate management for containers

You can secure your container services by specifying a certificate. The certificate must be installed in LocalMachine on all nodes of the cluster. The certificate information is provided in the application manifest under the `ContainerHostPolicies` tag as the following snippet shows:

```
<ContainerHostPolicies CodePackageRef="NodeContainerService.Code">
  <CertificateRef Name="MyCert1" X509StoreName="My" X509FindValue="[Thumbprint1]"/>
  <CertificateRef Name="MyCert2" X509FindValue="[Thumbprint2]"/>
```

For windows clusters, when starting the application, the runtime reads the certificates and generates a PFX file and password for each certificate. This PFX file and password are accessible inside the container using the following environment variables:

- **`Certificate_ServicePackageName_CodePackageName_CertName_PFX`**
- **`Certificate_ServicePackageName_CodePackageName_CertName_Password`**

For Linux clusters, the certificates(PEM) are simply copied over from the store specified by `X509StoreName` onto the container. The corresponding environment variables on linux are:

- **`Certificate_ServicePackageName_CodePackageName_CertName_PEM`**
- **`Certificate_ServicePackageName_CodePackageName_CertName_PrivateKey`**

Alternatively, if you already have the certificates in the required form and would simply want to access it inside the container, you can create a data package inside your app package and specify the following inside your application manifest:

```
<ContainerHostPolicies CodePackageRef="NodeContainerService.Code">
  <CertificateRef Name="MyCert1" DataPackageRef="[DataPackageName]" DataPackageVersion="[Version]"
    RelativePath="[Relative Path to certificate inside DataPackage]" Password="[password]" IsPasswordEncrypted="
    [true/false]"/>
```

The container service or process is responsible for importing the certificate files into the container. To import the certificate, you can use `setupentrypoint.sh` scripts or execute custom code within the container process. Sample code in C# for importing the PFX file follows:

```

string certificateFilePath =
Environment.GetEnvironmentVariable("Certificate_MyServicePackage_NodeContainerService.Code_MyCert1_PFX");
string passwordFilePath =
Environment.GetEnvironmentVariable("Certificate_MyServicePackage_NodeContainerService.Code_MyCert1_Password");
X509Store store = new X509Store(StoreName.My, StoreLocation.CurrentUser);
string password = File.ReadAllLines(passwordFilePath, Encoding.Default)[0];
password = password.Replace("\0", string.Empty);
X509Certificate2 cert = new X509Certificate2(certificateFilePath, password,
X509KeyStorageFlags.MachineKeySet | X509KeyStorageFlags.PersistKeySet);
store.Open(OpenFlags.ReadWrite);
store.Add(cert);
store.Close();

```

This PFX certificate can be used for authenticating the application or service or secure communication with other services. By default, the files are ACLed only to SYSTEM. You can ACL it to other accounts as required by the service.

## Set up gMSA for Windows containers

To set up gMSA (group Managed Service Accounts), a credential specification file (`credspec`) is placed on all nodes in the cluster. The file can be copied on all nodes using a VM extension. The `credspec` file must contain the gMSA account information. For more information on the `credspec` file, see [Service Accounts](#). The credential specification and the `Hostname` tag are specified in the application manifest. The `Hostname` tag must match the gMSA account name that the container runs under. The `Hostname` tag allows the container to authenticate itself to other services in the domain using Kerberos authentication. A sample for specifying the `Hostname` and the `credspec` in the application manifest is shown in the following snippet:

```

<Policies>
  <ContainerHostPolicies CodePackageRef="NodeService.Code" Isolation="process" Hostname="gMSAAccountName">
    <SecurityOption Value="credentialspec=file://WebApplication1.json"/>
  </ContainerHostPolicies>
</Policies>

```

## Next steps

- [Deploy a Windows container to Service Fabric on Windows Server 2016](#)
- [Deploy a Docker container to Service Fabric on Linux](#)

# Docker Compose deployment support in Azure Service Fabric (Preview)

10/3/2017 • 4 min to read • [Edit Online](#)

Docker uses the `docker-compose.yml` file for defining multi-container applications. To make it easy for customers familiar with Docker to orchestrate existing container applications on Azure Service Fabric, we have included preview support for Docker Compose deployment natively in the platform. Service Fabric can accept version 3 and later of `docker-compose.yml` files.

Because this support is in preview, only a subset of Compose directives is supported. For example, application upgrades are not supported. However, you can always remove and deploy applications instead of upgrading them.

To use this preview, create your cluster with version 5.7 or greater of the Service Fabric runtime through the Azure portal along with the corresponding SDK.

## NOTE

This feature is in preview and is not supported in production. The examples below are based on runtime version 6.0 and SDK version 2.8.

## Deploy a Docker Compose file on Service Fabric

The following commands create a Service Fabric application (named `fabric:/TestContainerApp`), which you can monitor and manage like any other Service Fabric application. You can use the specified application name for health queries. Service Fabric recognizes "DeploymentName" as the identifier of the Compose deployment.

### Use PowerShell

Create a Service Fabric Compose deployment from a `docker-compose.yml` file by running the following command in PowerShell:

```
New-ServiceFabricComposeDeployment -DeploymentName TestContainerApp -Compose docker-compose.yml [-RegistryUserName <>] [-RegistryPassword <>] [-PasswordEncrypted]
```

`RegistryUserName` and `RegistryPassword` refer to the container registry username and password. After you've completed the deployment, you can check its status by using the following command:

```
Get-ServiceFabricComposeDeploymentStatus -DeploymentName TestContainerApp
```

To delete the Compose deployment through PowerShell, use the following command:

```
Remove-ServiceFabricComposeDeployment -DeploymentName TestContainerApp
```

To start a Compose deployment upgrade through PowerShell, use the following command:

```
Start-ServiceFabricComposeDeploymentUpgrade -DeploymentName TestContainerApp -Compose docker-compose-v2.yml -Monitored -FailureAction Rollback
```

After upgrade is accepted, the upgrade progress could be tracked using the following command:

```
Get-ServiceFabricComposeDeploymentUpgrade -Deployment TestContainerApp
```

## Use Azure Service Fabric CLI (sfctl)

Alternatively, you can use the following Service Fabric CLI command:

```
sfctl compose create --deployment-name TestContainerApp --file-path docker-compose.yml [ [ --user --encrypted-pass ] | [ --user --has-pass ] ] [ --timeout ]
```

After you've created the deployment, you can check its status by using the following command:

```
sfctl compose status --deployment-name TestContainerApp [ --timeout ]
```

To delete the compose deployment, use the following command:

```
sfctl compose remove --deployment-name TestContainerApp [ --timeout ]
```

To start a Compose deployment upgrade, use the following command:

```
sfctl compose upgrade --deployment-name TestContainerApp --file-path docker-compose-v2.yml [ [ --user --encrypted-pass ] | [ --user --has-pass ] ] [ --upgrade-mode Monitored ] [ --failure-action Rollback ] [ --timeout ]
```

After upgrade is accepted, the upgrade progress could be tracked using the following command:

```
sfctl compose upgrade-status --deployment-name TestContainerApp
```

## Supported Compose directives

This preview supports a subset of the configuration options from the Compose version 3 format, including the following primitives:

- Services > Deploy > Replicas
- Services > Deploy > Placement > Constraints
- Services > Deploy > Resources > Limits
  - -cpu-shares
  - -memory
  - -memory-swap
- Services > Commands
- Services > Environment
- Services > Ports
- Services > Image
- Services > Isolation (only for Windows)
- Services > Logging > Driver
- Services > Logging > Driver > Options
- Volume & Deploy > Volume

Set up the cluster for enforcing resource limits, as described in [Service Fabric resource governance](#). All other Docker Compose directives are unsupported for this preview.

## ServiceDnsName computation

If the service name that you specify in a Compose file is a fully qualified domain name (that is, it contains a dot [.]), the DNS name registered by Service Fabric is `<serviceName>` (including the dot). If not, each path segment in the application name becomes a domain label in the service DNS name, with the first path segment becoming the top-level domain label.

For example, if the specified application name is `fabric:/SampleApp/MyComposeApp`, `<ServiceName>.MyComposeApp.SampleApp` would be the registered DNS name.

## Compose deployment (instance definition) versus Service Fabric app model (type definition)

A docker-compose.yml file describes a deployable set of containers, including their properties and configurations. For example, the file can contain environment variables and ports. You can also specify deployment parameters, such as placement constraints, resource limits, and DNS names, in the docker-compose.yml file.

The [Service Fabric application model](#) uses service types and application types, where you can have many application instances of the same type. For example, you can have one application instance per customer. This type-based model supports multiple versions of the same application type that's registered with the runtime.

For example, customer A can have an application instantiated with type 1.0 of AppTypeA, and customer B can have another application instantiated with the same type and version. You define the application types in the application manifests, and you specify the application name and deployment parameters when you create the application.

Although this model offers flexibility, we are also planning to support a simpler, instance-based deployment model where types are implicit from the manifest file. In this model, each application gets its own independent manifest. We are previewing this effort by adding support for docker-compose.yml, which is an instance-based deployment format.

## Next steps

- Read up on the [Service Fabric application model](#)
- [Get started with Service Fabric CLI](#)

# Resource governance

10/10/2017 • 7 min to read • [Edit Online](#)

When you're running multiple services on the same node or cluster, it's possible that one service might consume more resources, starving other services in the process. This problem is referred to as the "noisy neighbor" problem. Azure Service Fabric enables the developer to specify reservations and limits per service to guarantee resources and limit resource usage.

Before you proceed with this article, we recommend that you get familiar with the [Service Fabric application model](#) and the [Service Fabric hosting model](#).

## Resource governance metrics

Resource governance is supported in Service Fabric in accordance with the [service package](#). The resources that are assigned to the service package can be further divided between code packages. The resource limits that are specified also mean the reservation of the resources. Service Fabric supports specifying CPU and memory per service package, with two built-in [metrics](#):

- *CPU* (metric name `servicefabric:/_CpuCores`): A logical core that's available on the host machine. All cores across all nodes are weighted the same.
- *Memory* (metric name `servicefabric:/_MemoryInMB`): Memory is expressed in megabytes, and it maps to physical memory that is available on the machine.

For these two metrics, [Cluster Resource Manager](#) tracks total cluster capacity, the load on each node in the cluster, and the remaining resources in the cluster. These two metrics are equivalent to any other user or custom metric. All existing features can be used with them:

- The cluster can be [balanced](#) according to these two metrics (default behavior).
- The cluster can be [defragmented](#) according to these two metrics.
- When [describing a cluster](#), buffered capacity can be set for these two metrics.

[Dynamic load reporting](#) is not supported for these metrics, and loads for these metrics are defined at creation time.

## Resource governance mechanism

The Service Fabric runtime currently does not provide reservation for resources. When a process or a container is opened, the runtime sets the resource limits to the loads that were defined at creation time. Furthermore, the runtime rejects the opening of new service packages that are available when resources are exceeded. To better understand how the process works, let's take an example of a node with two CPU cores (mechanism for memory governance is equivalent):

1. First, a container is placed on the node, requesting one CPU core. The runtime opens the container and sets the CPU limit to one core. The container won't be able to use more than one core.
2. Then, a replica of a service is placed on the node, and the corresponding service package specifies a limit of one CPU core. The runtime opens the code package and sets its CPU limit to one core.

At this point, the sum of limits is equal to the capacity of the node. A process and a container are running with one core each and not interfering with each other. Service Fabric doesn't place any more containers or replicas when they are specifying the CPU limit.

However, there are two situations in which other processes might contend for CPU. In these situations, a process and a container from our example might experience the noisy neighbor problem:

- *Mixing governed and non-governed services and containers:* If a user creates a service without any resource governance specified, the runtime sees it as consuming no resources, and can place it on the node in our example. In this case, this new process effectively consumes some CPU at the expense of the services that are already running on the node. There are two solution to this problem. Either don't mix governed and non-governed services on the same cluster, or use [placement constraints](#) so that these two types of services don't end up on the same set of nodes.
- *When another process is started on the node, outside Service Fabric (for example, an OS service):* In this situation, the process outside Service Fabric also contends for CPU with existing services. The solution to this problem is to set up node capacities correctly to account for OS overhead, as shown in the next section.

## Cluster setup for enabling resource governance

When a node starts and joins the cluster, Service Fabric detects the available amount of memory and the available number of cores, and then sets the node capacities for those two resources.

To leave buffer space for the operating system, and for other processes might be running on the node, Service Fabric uses only 80% of the available resources on the node. This percentage is configurable, and can be changed in the cluster manifest.

Here is an example of how to instruct Service Fabric to use 50% of available CPU and 70% of available memory:

```
<Section Name="PlacementAndLoadBalancing">
    <!-- 0.0 means 0%, and 1.0 means 100%-->
    <Parameter Name="CpuPercentageNodeCapacity" Value="0.5" />
    <Parameter Name="MemoryPercentageNodeCapacity" Value="0.7" />
</Section>
```

If you need full manual setup of node capacities, you can use the regular mechanism for describing the nodes in the cluster. Here is an example of how to set up the node with four cores and 2 GB of memory:

```
<NodeType Name="MyNodeType">
    <Capacities>
        <Capacity Name="servicefabric:/_CpuCores" Value="4"/>
        <Capacity Name="servicefabric:/_MemoryInMB" Value="2048"/>
    </Capacities>
</NodeType>
```

When auto-detection of available resources is enabled, and node capacities are manually defined in the cluster manifest, Service Fabric checks that the node has enough resources to support the capacity that the user has defined:

- If node capacities that are defined in the manifest are less than or equal to the available resources on the node, then Service Fabric uses the capacities that are specified in the manifest.
- If node capacities that are defined in the manifest are greater than available resources, Service Fabric uses the available resources as node capacities.

Auto-detection of available resources can be turned off if it is not required. To turn it off, change the following setting:

```

<Section Name="PlacementAndLoadBalancing">
    <Parameter Name="AutoDetectAvailableResources" Value="false" />
</Section>

```

For optimal performance, the following setting should also be turned on in the cluster manifest:

```

<Section Name="PlacementAndLoadBalancing">
    <Parameter Name="PreventTransientOvercommit" Value="true" />
    <Parameter Name="AllowConstraintCheckFixesDuringApplicationUpgrade" Value="true" />
</Section>

```

## Specify resource governance

Resource governance limits are specified in the application manifest (ServiceManifestImport section) as shown in the following example:

```

<?xml version='1.0' encoding='UTF-8'?>
<ApplicationManifest ApplicationTypeName='TestAppTC1' ApplicationTypeVersion='vTC1'
xsi:schemaLocation='http://schemas.microsoft.com/2011/01/fabric ServiceFabricServiceModel.xsd'
xmlns='http://schemas.microsoft.com/2011/01/fabric' xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'>
    <Parameters>
    </Parameters>
    <!--
        ServicePackageA has the number of CPU cores defined, but doesn't have the MemoryInMB defined.
        In this case, Service Fabric sums the limits on code packages and uses the sum as
        the overall ServicePackage limit.
    -->
    <ServiceManifestImport>
        <ServiceManifestRef ServiceManifestName='ServicePackageA' ServiceManifestVersion='v1'/>
        <Policies>
            <ServicePackageResourceGovernancePolicy CpuCores="1"/>
            <ResourceGovernancePolicy CodePackageRef="CodeA1" CpuShares="512" MemoryInMB="1000" />
            <ResourceGovernancePolicy CodePackageRef="CodeA2" CpuShares="256" MemoryInMB="1000" />
        </Policies>
    </ServiceManifestImport>

```

In this example, the service package called **ServicePackageA** gets one core on the nodes where it is placed. This service package contains two code packages (**CodeA1** and **CodeA2**), and both specify the `CpuShares` parameter. The proportion of CpuShares 512:256 divides the core across the two code packages.

Thus, in this example, CodeA1 gets two-thirds of a core, and CodeA2 gets one-third of a core (and a soft-guarantee reservation of the same). If CpuShares are not specified for code packages, Service Fabric divides the cores equally among them.

Memory limits are absolute, so both code packages are limited to 1024 MB of memory (and a soft-guarantee reservation of the same). Code packages (containers or processes) can't allocate more memory than this limit, and attempting to do so results in an out-of-memory exception. For resource limit enforcement to work, all code packages within a service package should have memory limits specified.

## Other resources for containers

Besides CPU and memory, it's possible to specify other resource limits for containers. These limits are specified at the code-package level and are applied when the container is started. Unlike with CPU and memory, Cluster Resource Manager isn't aware of these resources, and won't do any capacity checks or load balancing for them.

- `MemorySwapInMB`: The amount of swap memory that a container can use.
- `MemoryReservationInMB`: The soft limit for memory governance that is enforced only when memory

contention is detected on the node.

- *CpuPercent*: The percentage of CPU that the container can use. If CPU limits are specified for the service package, this parameter is effectively ignored.
- *MaximumIOPS*: The maximum IOPS that a container can use (read and write).
- *MaximumIOBytesps*: The maximum IO (bytes per second) that a container can use (read and write).
- *BlockIOWeight*: The block IO weight for relative to other containers.

These resources can be combined with CPU and memory. Here is an example of how to specify additional resources for containers:

```
<ServiceManifestImport>
    <ServiceManifestRef ServiceManifestName="FrontendServicePackage" ServiceManifestVersion="1.0"/>
    <Policies>
        <ResourceGovernancePolicy CodePackageRef="FrontendService.Code" CpuPercent="5"
            MemorySwapInMB="4084" MemoryReservationInMB="1024" MaximumIOPS="20" />
    </Policies>
</ServiceManifestImport>
```

## Next steps

- To learn more about Cluster Resource Manager, read [Introducing the Service Fabric cluster resource manager](#).
- To learn more about the application model, service packages, and code packages--and how replicas map to them--read [Model an application in Service Fabric](#).

# Using volume plugins and logging drivers in your container

10/12/2017 • 1 min to read • [Edit Online](#)

Service Fabric supports specifying [Docker volume plugins](#) and [Docker logging drivers](#) for your container service.

## Install volume/logging driver

If the Docker volume/logging driver is not installed on the machine, install it manually through RDP/SSH-ing into the machine or through a VMSS start-up script. For instance, in order to install the Docker Volume Driver, SSH into the machine and execute:

```
docker plugin install --alias azure --grant-all-permissions docker4x/17.09.0-ce-azure1 \
  CLOUD_PLATFORM=AZURE \
  AZURE_STORAGE_ACCOUNT="[MY-STORAGE-ACCOUNT-NAME]" \
  AZURE_STORAGE_ACCOUNT_KEY="[MY-STORAGE-ACCOUNT-KEY]" \
  DEBUG=1
```

## Specify the plugin or driver in the manifest

The plugins are specified in the application manifest as shown in the following manifest:

```
?xml version="1.0" encoding="UTF-8"?>
<ApplicationManifest ApplicationTypeName="WinNodeJsApp" ApplicationTypeVersion="1.0"
  xmlns="http://schemas.microsoft.com/2011/01/fabric" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Description>Calculator Application</Description>
  <Parameters>
    <Parameter Name="ServiceInstanceCount" DefaultValue="3"></Parameter>
    <Parameter Name="MyCpuShares" DefaultValue="3"></Parameter>
  </Parameters>
  <ServiceManifestImport>
    <ServiceManifestRef ServiceManifestName="NodeServicePackage" ServiceManifestVersion="1.0"/>
  </ServiceManifestImport>
  <Policies>
    <ContainerHostPolicies CodePackageRef="NodeService.Code" Isolation="hyperv">
      <PortBinding ContainerPort="8905" EndpointRef="Endpoint1"/>
      <RepositoryCredentials PasswordEncrypted="false" Password="*****" AccountName="test"/>
      <LogConfig Driver="etwlogs" >
        <DriverOption Name="test" Value="vale"/>
      </LogConfig>
      <Volume Source="c:\workspace" Destination="c:\testmountlocation1" IsReadOnly="false"></Volume>
      <Volume Source="d:\myfolder" Destination="c:\testmountlocation2" IsReadOnly="true"> </Volume>
      <Volume Source="myvolume1" Destination="c:\testmountlocation2" Driver="azure" IsReadOnly="true">
        <DriverOption Name="share" Value="models"/>
      </Volume>
    </ContainerHostPolicies>
  </Policies>
  </ServiceManifestImport>
  <ServiceTemplates>
    <StatelessService ServiceTypeName="StatelessNodeService" InstanceCount="5">
      <SingletonPartition></SingletonPartition>
    </StatelessService>
  </ServiceTemplates>
</ApplicationManifest>
```

In the preceding example, the `Source` tag for the `Volume` refers to the source folder. The source folder could be a

folder in the VM that hosts the containers or a persistent remote store. The `Destination` tag is the location that the `Source` is mapped to within the running container.

When specifying a volume plugin, Service Fabric automatically creates the volume using the parameters specified. The `Source` tag is the name of the volume, and the `Driver` tag specifies the volume driver plugin. Options can be specified using the `DriverOption` tag as shown in the following snippet:

```
<Volume Source="myvolume1" Destination="c:\testmountlocation4" Driver="azurefile" IsReadOnly="true">
    <DriverOption Name="share" Value="models"/>
</Volume>
```

If a Docker log driver is specified, it is necessary to deploy agents (or containers) to handle the logs in the cluster. The `DriverOption` tag can be used to specify log driver options as well.

Refer to the following articles to deploy containers to a Service Fabric cluster:

[Deploy a container on Service Fabric](#)

# How to containerize your Service Fabric Reliable Services and Reliable Actors (Preview)

8/11/2017 • 2 min to read • [Edit Online](#)

Service Fabric supports containerizing Service Fabric microservices (Reliable Services, and Reliable Actor based services). For more information, see [service fabric containers](#).

This feature is in preview and this article provides the various steps to get your service running inside a container.

## NOTE

This feature is in preview and is not supported in production. Currently this feature only works for Windows.

## Steps to containerize your Service Fabric Application

1. Open your Service Fabric application in Visual Studio.
2. Add class [SFBinaryLoader.cs](#) to your project. The code in this class is a helper to correctly load the Service Fabric runtime binaries inside your application when running inside a container.
3. For each code package you would like to containerize, initialize the loader at the program entry point. Add the static constructor shown in the following code snippet to your program entry point file.

```
namespace MyApplication
{
    internal static class Program
    {
        static Program()
        {
            SFBinaryLoader.Initialize();
        }

        /// <summary>
        /// This is the entry point of the service host process.
        /// </summary>
        private static void Main()
    }
}
```

4. Build and [package](#) your project. To build and create a package, right-click the application project in Solution Explorer and choose the **Package** command.
5. For every code package you need to containerize, run the PowerShell script [CreateDockerPackage.ps1](#). The usage is as follows:

```
$codePackagePath = 'Path to the code package to containerize.'
$dockerPackageOutputDirectoryPath = 'Output path for the generated docker folder.'
$applicationExeName = 'Name of the ode package executable.'
CreateDockerPackage.ps1 -CodePackageDirectoryPath $codePackagePath -DockerPackageOutputDirectoryPath
$dockerPackageOutputDirectoryPath -ApplicationExeName $applicationExeName
```

The script creates a folder with Docker artifacts at \$dockerPackageOutputDirectoryPath. Modify the generated Dockerfile to expose any ports, run setup scripts etc. based on your needs.

6. Next you need to [build](#) and [push](#) your Docker container package to your repository.
7. Modify the ApplicationManifest.xml and ServiceManifest.xml to add your container image, repository information, registry authentication, and port-to-host mapping. For modifying the manifests, see [Create an Azure Service Fabric container application](#). The code package definition in the service manifest needs to be replaced with corresponding container image. Make sure to change the EntryPoint to a ContainerHost type.

```
<!-- Code package is your service executable. -->
<CodePackage Name="Code" Version="1.0.0">
<EntryPoint>
    <!-- Follow this link for more information about deploying Windows containers to Service Fabric:
        https://aka.ms/sfguestcontainers -->
    <ContainerHost>
        <ImageName>myregistry.azurecr.io/samples/helloworldapp</ImageName>
    </ContainerHost>
</EntryPoint>
<!-- Pass environment variables to your container: -->
</CodePackage>
```

8. Add the port-to-host mapping for your replicator and service endpoint. Since both these ports are assigned at runtime by Service Fabric, the ContainerPort is set to zero to use the assigned port for mapping.

```
<Policies>
<ContainerHostPolicies CodePackageRef="Code">
    <PortBinding ContainerPort="0" EndpointRef="ServiceEndpoint"/>
    <PortBinding ContainerPort="0" EndpointRef="ReplicatorEndpoint"/>
</ContainerHostPolicies>
</Policies>
```

9. To test this application, you need to deploy it to a cluster that is running version 5.7 or higher. In addition, you need to edit and update the cluster settings to enable this preview feature. Follow the steps in this [article](#) to add the setting shown next.

```
{
    "name": "Hosting",
    "parameters": [
        {
            "name": "FabricContainerAppsEnabled",
            "value": "true"
        }
    ]
}
```

10. Next [deploy](#) the edited application package to this cluster.

You should now have a containerized Service Fabric application running your cluster.

## Next steps

- Learn more about running [containers on Service Fabric](#).
- Learn about the Service Fabric [application life-cycle](#).

# Service Fabric container networking modes

9/25/2017 • 3 min to read • [Edit Online](#)

The default networking mode offered in the Service Fabric cluster for container services is the `nat` networking mode. With the `nat` networking mode, having more than one containers service listening to the same port results in deployment errors. For running several services that listen on the same port, Service Fabric supports the `open` networking mode (version 5.7 or higher). With the `open` networking mode, each container service gets a dynamically assigned IP address internally allowing multiple services to listen to the same port.

Thus, with a single service type with a static endpoint defined in the service manifest, new services may be created and deleted without deployment errors using the `open` networking mode. Similarly, one can use the same `docker-compose.yml` file with static port mappings for creating multiple services.

Using the dynamically assigned IP to discover services is not advisable since the IP address changes when the service restarts or moves to another node. Only use the **Service Fabric Naming Service** or the **DNS Service** for service discovery.

## WARNING

Only a total of 4096 IPs are allowed per vNET in Azure. Thus, the sum of the number of nodes and the number of container service instances (with `open` networking) cannot exceed 4096 within a vNET. For such high-density scenarios, the `nat` networking mode is recommended.

## Setting up open networking mode

1. Set up the Azure Resource Manager template by enabling DNS Service and the IP Provider under `fabricSettings`.

```

"fabricSettings": [
    {
        "name": "DnsService",
        "parameters": [
            {
                "name": "IsEnabled",
                "value": "true"
            }
        ]
    },
    {
        "name": "Hosting",
        "parameters": [
            {
                "name": "IPProviderEnabled",
                "value": "true"
            }
        ]
    },
    {
        "name": "Trace/Etw",
        "parameters": [
            {
                "name": "Level",
                "value": "5"
            }
        ]
    },
    {
        "name": "Setup",
        "parameters": [
            {
                "name": "ContainerNetworkSetup",
                "value": "true"
            }
        ]
    }
],

```

2. Set up the network profile section to allow multiple IP addresses to be configured on each node of the cluster. The following example sets up five IP addresses per node (thus you can have five service instances listening to the port on each node) for a Windows/Linux Service Fabric cluster.

```

"variables": {
    "nicName": "NIC",
    "vmName": "vm",
    "virtualNetworkName": "VNet",
    "vnetID": "[resourceId('Microsoft.Network/virtualNetworks',variables('virtualNetworkName'))]",
    "vmNodeType0Name": "[toLower(concat('NT1', variables('vmName')))]",
    "subnet0Name": "Subnet-0",
    "subnet0Prefix": "10.0.0.0/24",
    "subnet0Ref": "[concat(variables('vnetID'), '/subnets/', variables('subnet0Name'))]",
    "lbID0": "[resourceId('Microsoft.Network/loadBalancers', concat('LB', '-', parameters('clusterName'), '-', variables('vmNodeType0Name')))]",
    "lbIPConfig0": "[concat(variables('lbID0'), '/frontendIPConfigurations/LoadBalancerIPConfig')]",
    "lbPoolID0": "[concat(variables('lbID0'), '/backendAddressPools/LoadBalancerBEAddressPool')]",
    "lbProbeID0": "[concat(variables('lbID0'), '/probes/FabricGatewayProbe')]",
    "lbHttpProbeID0": "[concat(variables('lbID0'), '/probes/FabricHttpGatewayProbe')]",
    "lbNatPoolID0": "[concat(variables('lbID0'), '/inboundNatPools/LoadBalancerBEAddressNatPool')]"
}
"networkProfile": {
    "networkInterfaceConfigurations": [
        {
            "name": "[concat(parameters('nicName'), '-0')]",
            "properties": {
                "ipConfigurations": [

```

```

"ipConfigurations": [
    {
        "name": "[concat(parameters('nicName'), '-', 0)]",
        "properties": {
            "primary": "true",
            "loadBalancerBackendAddressPools": [
                {
                    "id": "[variables('lbPoolID0')]"
                }
            ],
            "loadBalancerInboundNatPools": [
                {
                    "id": "[variables('lbNatPoolID0')]"
                }
            ],
            "subnet": {
                "id": "[variables('subnet0Ref')]"
            }
        }
    },
    {
        "name": "[concat(parameters('nicName'), '-', 1)]",
        "properties": {
            "primary": "false",
            "subnet": {
                "id": "[variables('subnet0Ref')]"
            }
        }
    },
    {
        "name": "[concat(parameters('nicName'), '-', 2)]",
        "properties": {
            "primary": "false",
            "subnet": {
                "id": "[variables('subnet0Ref')]"
            }
        }
    },
    {
        "name": "[concat(parameters('nicName'), '-', 3)]",
        "properties": {
            "primary": "false",
            "subnet": {
                "id": "[variables('subnet0Ref')]"
            }
        }
    },
    {
        "name": "[concat(parameters('nicName'), '-', 4)]",
        "properties": {
            "primary": "false",
            "subnet": {
                "id": "[variables('subnet0Ref')]"
            }
        }
    },
    {
        "name": "[concat(parameters('nicName'), '-', 5)]",
        "properties": {
            "primary": "false",
            "subnet": {
                "id": "[variables('subnet0Ref')]"
            }
        }
    }
],
"primary": true
}

```

```
        ]  
    }
```

3. For Windows clusters only, set up an NSG rule opening up port UDP/53 for the vNET with the following values:

PRIORITY	NAME	SOURCE	DESTINATION	SERVICE	ACTION
2000	Custom_Dns	VirtualNetwork	VirtualNetwork	DNS (UDP/53)	Allow

4. Specify the networking mode in the app manifest for each service `<NetworkConfig NetworkType="open">`. The mode `open` results in the service getting a dedicated IP address. If a mode isn't specified, it defaults to the basic `nat` mode. Thus, in the following manifest example, `NodeContainerServicePackage1` and `NodeContainerServicePackage2` can each listen to the same port (both services are listening on `Endpoint1`).

```
<?xml version="1.0" encoding="UTF-8"?>  
<ApplicationManifest ApplicationTypeName="NodeJsApp" ApplicationTypeVersion="1.0"  
xmlns="http://schemas.microsoft.com/2011/01/fabric" xmlns:xsi="http://www.w3.org/2001/XMLSchema-  
instance">  
    <Description>Calculator Application</Description>  
    <Parameters>  
        <Parameter Name="ServiceInstanceCount" DefaultValue="3"></Parameter>  
        <Parameter Name="MyCpuShares" DefaultValue="3"></Parameter>  
    </Parameters>  
    <ServiceManifestImport>  
        <ServiceManifestRef ServiceManifestName="NodeContainerServicePackage1"  
ServiceManifestVersion="1.0"/>  
        <Policies>  
            <ContainerHostPolicies CodePackageRef="NodeContainerService1.Code" Isolation="hyperv">  
                <NetworkConfig NetworkType="open"/>  
                <PortBinding ContainerPort="8905" EndpointRef="Endpoint1"/>  
            </ContainerHostPolicies>  
        </Policies>  
    </ServiceManifestImport>  
    <ServiceManifestImport>  
        <ServiceManifestRef ServiceManifestName="NodeContainerServicePackage2"  
ServiceManifestVersion="1.0"/>  
        <Policies>  
            <ContainerHostPolicies CodePackageRef="NodeContainerService2.Code" Isolation="default">  
                <NetworkConfig NetworkType="open"/>  
                <PortBinding ContainerPort="8910" EndpointRef="Endpoint1"/>  
            </ContainerHostPolicies>  
        </Policies>  
    </ServiceManifestImport>  
</ApplicationManifest>
```

You can mix and match different networking modes across services within an application for a Windows cluster. Thus, you can have some services on `open` mode and some on `nat` networking mode. When a service is configured with `nat`, the port it is listening to must be unique. Mixing networking modes for different services isn't supported on Linux clusters.

## Next steps

In this article, you learned about networking modes offered by Service Fabric.

- [Service Fabric application model](#)
- [Service Fabric service manifest resources](#)
- [Deploy a Windows container to Service Fabric on Windows Server 2016](#)
- [Deploy a Docker container to Service Fabric on Linux](#)

# Reliable Services overview

8/15/2017 • 9 min to read • [Edit Online](#)

Azure Service Fabric simplifies writing and managing stateless and stateful Reliable Services. This topic covers:

- The Reliable Services programming model for stateless and stateful services.
- The choices you have to make when writing a Reliable Service.
- Some scenarios and examples of when to use Reliable Services and how they are written.

Reliable Services is one of the programming models available on Service Fabric. The other is the Reliable Actor programming model, which provides a virtual Actor programming model on top of the Reliable Services model. For more information on the Reliable Actors programming model, see [Introduction to Service Fabric Reliable Actors](#).

Service Fabric manages the lifetime of services, from provisioning and deployment through upgrade and deletion, via [Service Fabric application management](#).

## What are Reliable Services?

Reliable Services gives you a simple, powerful, top-level programming model to help you express what is important to your application. With the Reliable Services programming model, you get:

- Access to the rest of the Service Fabric programming APIs. Unlike Service Fabric Services modeled as [Guest Executables](#), Reliable Services get to use the rest of the Service Fabric APIs directly. This allows services to:
  - query the system
  - report health about entities in the cluster
  - receive notifications about configuration and code changes
  - find and communicate with other services,
  - (optionally) use the [Reliable Collections](#)
  - ...and giving them access to many other capabilities, all from a first class programming model in several programming languages.
- A simple model for running your own code that looks like programming models you are used to. Your code has a well-defined entry point and easily managed lifecycle.
- A pluggable communication model. Use the transport of your choice, such as HTTP with [Web API](#), WebSockets, custom TCP protocols, or anything else. Reliable Services provide some great out-of-the-box options you can use, or you can provide your own.
- For stateful services, the Reliable Services programming model allows you to consistently and reliably store your state right inside your service by using [Reliable Collections](#). Reliable Collections are a simple set of highly available and reliable collection classes that will be familiar to anyone who has used C# collections. Traditionally, services needed external systems for Reliable state management. With Reliable Collections, you can store your state next to your compute with the same high availability and reliability you've come to expect from highly available external stores. This model also improves latency because you are co-locating the compute and state it needs to function.

Watch this Microsoft Virtual Academy video for an overview of Reliable services:



## What makes Reliable Services different?

Reliable Services in Service Fabric are different from services you may have written before. Service Fabric provides reliability, availability, consistency, and scalability.

- **Reliability** - Your service stays up even in unreliable environments where your machines fail or hit network issues, or in cases where the services themselves encounter errors and crash or fail. For stateful services, your state is preserved even in the presence of network or other failures.
- **Availability** - Your service is reachable and responsive. Service Fabric maintains your desired number of running copies.
- **Scalability** - Services are decoupled from specific hardware, and they can grow or shrink as necessary through the addition or removal of hardware or other resources. Services are easily partitioned (especially in the stateful case) to ensure that the service can scale and handle partial failures. Services can be created and deleted dynamically via code, enabling more instances to be spun up as necessary, say in response to customer requests. Finally, Service Fabric encourages services to be lightweight. Service Fabric allows thousands of services to be provisioned within a single process, rather than requiring or dedicating entire OS instances or processes to a single instance of a service.
- **Consistency** - Any information stored in this service can be guaranteed to be consistent. This is true even across multiple reliable collections within a service. Changes across collections within a service can be made in a transactionally atomic manner.

## Service lifecycle

Whether your service is stateful or stateless, Reliable Services provide a simple lifecycle that lets you quickly plug in your code and get started. There are just one or two methods that you need to implement to get your service up and running.

- **CreateServiceReplicaListeners/CreateServiceInstanceListeners** - This method is where the service defines the communication stack(s) that it wants to use. The communication stack, such as [Web API](#), is what defines the listening endpoint or endpoints for the service (how clients reach the service). It also defines how the messages that appear interact with the rest of the service code.
- **RunAsync** - This method is where your service runs its business logic, and where it would kick off any background tasks that should run for the lifetime of the service. The cancellation token that is provided is a signal for when that work should stop. For example, if the service needs to pull messages out of a Reliable Queue and process them, this is where that work happens.

If you're learning about reliable services for the first time, read on! If you're looking for a detailed walkthrough of the lifecycle of reliable services, you can head over to [this article](#).

## Example services

Knowing this programming model, let's take a quick look at two different services to see how these pieces fit together.

### Stateless Reliable Services

A stateless service is one where there is no state maintained within the service across calls. Any state that is present is entirely disposable and doesn't require synchronization, replication, persistence, or high availability.

For example, consider a calculator that has no memory and receives all terms and operations to perform at once.

In this case, the `RunAsync()` (C#) or `runAsync()` (Java) of the service can be empty, since there is no background task-processing that the service needs to do. When the calculator service is created, it returns an `ICommunicationListener` (C#) or `CommunicationListener` (Java) (for example [Web API](#)) that opens up a listening endpoint on some port. This listening endpoint hooks up to the different calculation methods (example: "Add(n1, n2)") that define the calculator's public API.

When a call is made from a client, the appropriate method is invoked, and the calculator service performs the operations on the data provided and returns the result. It doesn't store any state.

Not storing any internal state makes this example calculator simple. But most services aren't truly stateless. Instead, they externalize their state to some other store. (For example, any web app that relies on keeping session state in a backing store or cache is not stateless.)

A common example of how stateless services are used in Service Fabric is as a front-end that exposes the public-facing API for a web application. The front-end service then talks to stateful services to complete a user request. In this case, calls from clients are directed to a known port, such as 80, where the stateless service is listening. This stateless service receives the call and determines whether the call is from a trusted party and which service it's destined for. Then, the stateless service forwards the call to the correct partition of the stateful service and waits for a response. When the stateless service receives a response, it replies to the original client. An example of such a service is in our samples [C# / Java](#). This is only one example of this pattern in the samples, there are others in other samples as well.

## Stateful Reliable Services

A stateful service is one that must have some portion of state kept consistent and present in order for the service to function. Consider a service that constantly computes a rolling average of some value based on updates it receives. To do this, it must have the current set of incoming requests it needs to process and the current average. Any service that retrieves, processes, and stores information in an external store (such as an Azure blob or table store today) is stateful. It just keeps its state in the external state store.

Most services today store their state externally, since the external store is what provides reliability, availability, scalability, and consistency for that state. In Service Fabric, services aren't required to store their state externally. Service Fabric takes care of these requirements for both the service code and the service state.

### NOTE

Support for Stateful Reliable Services is not available on Linux yet (for C# or Java).

Let's say we want to write a service that processes images. To do this, the service takes in an image and the series of conversions to perform on that image. This service returns a communication listener (let's suppose it's a WebAPI) that exposes an API like `ConvertImage(Image i, IList<Conversion> conversions)`. When it receives a request, the service stores it in a `IReliableQueue`, and returns some id to the client so it can track the request.

In this service, `RunAsync()` could be more complex. The service has a loop inside its `RunAsync()` that pulls requests out of `IReliableQueue` and performs the conversions requested. The results get stored in an `IReliableDictionary` so that when the client comes back they can get their converted images. To ensure that even if something fails the image isn't lost, this Reliable Service would pull out of the queue, perform the conversions, and store the result all in a single transaction. In this case, the message is removed from the queue and the results are stored in the result dictionary only when the conversions are complete. Alternatively, the service could pull the image out of the queue and immediately store it in a remote store. This reduces the

amount of state the service has to manage, but increases complexity since the service has to keep the necessary metadata to manage the remote store. With either approach, if something failed in the middle the request remains in the queue waiting to be processed.

One thing to note about this service is that it sounds like a normal .NET service! The only difference is that the data structures being used (`IReliableQueue` and `IReliableDictionary`) are provided by Service Fabric, and are highly reliable, available, and consistent.

## When to use Reliable Services APIs

If any of the following characterize your application service needs, then you should consider Reliable Services APIs:

- You want your service's code (and optionally state) to be highly available and reliable
- You need transactional guarantees across multiple units of state (for example, orders and order line items).
- Your application's state can be naturally modeled as Reliable Dictionaries and Queues.
- Your applications code or state needs to be highly available with low latency reads and writes.
- Your application needs to control the concurrency or granularity of transacted operations across one or more Reliable Collections.
- You want to manage the communications or control the partitioning scheme for your service.
- Your code needs a free-threaded runtime environment.
- Your application needs to dynamically create or destroy Reliable Dictionaries or Queues or whole Services at runtime.
- You need to programmatically control Service Fabric-provided backup and restore features for your service's state.
- Your application needs to maintain change history for its units of state.
- You want to develop or consume third-party-developed, custom state providers.

## Next steps

- [Reliable Services quick start](#)
- [Reliable Services advanced usage](#)
- [The Reliable Actors programming model](#)

# Reliable Services lifecycle overview

10/16/2017 • 8 min to read • [Edit Online](#)

When you're thinking about the lifecycles of Azure Service Fabric Reliable Services, the basics of the lifecycle are the most important. In general, the lifecycle includes the following:

- During startup:
  - Services are constructed.
  - The services have an opportunity to construct and return zero or more listeners.
  - Any returned listeners are opened, allowing communication with the service.
  - The service's **RunAsync** method is called, allowing the service to do long-running tasks or background work.
- During shutdown:
  - The cancellation token passed to **RunAsync** is canceled, and the listeners are closed.
  - After the listeners close, the service object itself is destructed.

There are details around the exact ordering of these events. The order of events can change slightly depending on whether the Reliable Service is stateless or stateful. In addition, for stateful services, we must deal with the Primary swap scenario. During this sequence, the role of Primary is transferred to another replica (or comes back) without the service shutting down. Finally, we must think about error or failure conditions.

## Stateless service startup

The lifecycle of a stateless service is straightforward. Here's the order of events:

1. The service is constructed.
2. Then, in parallel, two things happen:
  - `StatelessService.CreateServiceInstanceListeners()` is invoked and any returned listeners are opened.  
`ICommunicationListener.OpenAsync()` is called on each listener.
  - The service's `StatelessService.RunAsync()` method is called.
3. If present, the service's `StatelessService.OnOpenAsync()` method is called. This call is an uncommon override, but it is available.

Keep in mind that there is no ordering between the calls to create and open the listeners and **RunAsync**. The listeners can open before **RunAsync** is started. Similarly, you can invoke **RunAsync** before the communication listeners are open or even constructed. If any synchronization is required, it is left as an exercise to the implementer. Here are some common solutions:

- Sometimes listeners can't function until some other information is created or work is done. For stateless services, that work can usually be done in other locations, such as the following:
  - In the service's constructor.
  - During the `CreateServiceInstanceListeners()` call.
  - As a part of the construction of the listener itself.
- Sometimes the code in **RunAsync** doesn't start until the listeners are open. In this case, additional coordination is necessary. One common solution is that there is a flag within the listeners that indicates when they have finished. This flag is then checked in **RunAsync** before continuing to actual work.

## Stateless service shutdown

For shutting down a stateless service, the same pattern is followed, just in reverse:

1. In parallel:
  - Any open listeners are closed. `ICommunicationListener.CloseAsync()` is called on each listener.
  - The cancellation token passed to `RunAsync()` is canceled. A check of the cancellation token's `IsCancellationRequested` property returns true, and if called, the token's `ThrowIfCancellationRequested` method throws an `OperationCanceledException`.
2. After `CloseAsync()` finishes on each listener and `RunAsync()` also finishes, the service's `StatelessService.OnCloseAsync()` method is called, if present. It is uncommon to override `StatelessService.OnCloseAsync()`.
3. After `StatelessService.OnCloseAsync()` finishes, the service object is destructed.

## Stateful service startup

Stateful services have a similar pattern to stateless services, with a few changes. For starting up a stateful service, the order of events is as follows:

1. The service is constructed.
2. `StatefulServiceBase.OnOpenAsync()` is called. This call is not commonly overridden in the service.
3. The following things happen in parallel:
  - `StatefulServiceBase.CreateServiceReplicaListeners()` is invoked.
    - If the service is a Primary service, all returned listeners are opened. `ICommunicationListener.OpenAsync()` is called on each listener.
    - If the service is a Secondary service, only those listeners marked as `ListenOnSecondary = true` are opened. Having listeners that are open on secondaries is less common.
  - If the service is currently a Primary, the service's `StatefulServiceBase.RunAsync()` method is called.
4. After all the replica listener's `OpenAsync()` calls finish and `RunAsync()` is called, `StatefulServiceBase.OnChangeRoleAsync()` is called. This call is not commonly overridden in the service.

Similar to stateless services, there's no coordination between the order in which the listeners are created and opened and when **RunAsync** is called. If you need coordination, the solutions are much the same. There is one additional case for stateful service. Say that the calls that arrive at the communication listeners require information kept inside some [Reliable Collections](#). Because the communication listeners could open before the reliable collections are readable or writeable, and before **RunAsync** could start, some additional coordination is necessary. The simplest and most common solution is for the communication listeners to return an error code that the client uses to retry the request.

## Stateful service shutdown

Like stateless services, the lifecycle events during shutdown are the same as during startup, but reversed. When a stateful service is being shut down, the following events occur:

1. In parallel:
  - Any open listeners are closed. `ICommunicationListener.CloseAsync()` is called on each listener.
  - The cancellation token passed to `RunAsync()` is canceled. A check of the cancellation token's `IsCancellationRequested` property returns true, and if called, the token's `ThrowIfCancellationRequested` method throws an `OperationCanceledException`.
2. After `CloseAsync()` finishes on each listener and `RunAsync()` also finishes, the service's `StatefulServiceBase.OnChangeRoleAsync()` is called. This call is not commonly overridden in the service.

#### NOTE

The need to wait for **RunAsync** to finish is only necessary if this replica is a Primary replica.

3. After the `StatefulServiceBase.OnChangeRoleAsync()` method finishes, the `StatefulServiceBase.OnCloseAsync()` method is called. This call is an uncommon override, but it is available.
4. After `StatefulServiceBase.OnCloseAsync()` finishes, the service object is destructed.

## Stateful service Primary swaps

While a stateful service is running, only the Primary replicas of that stateful services have their communication listeners opened and their **RunAsync** method called. Secondary replicas are constructed, but see no further calls. While a stateful service is running, the replica that's currently the Primary can change. What does this mean in terms of the lifecycle events that a replica can see? The behavior the stateful replica sees depends on whether it is the replica being demoted or promoted during the swap.

### For the Primary that's demoted

For the Primary replica that's demoted, Service Fabric needs this replica to stop processing messages and quit any background work it is doing. As a result, this step looks like it did when the service is shut down. One difference is that the service isn't destructed or closed because it remains as a Secondary. The following APIs are called:

1. In parallel:
  - Any open listeners are closed. `ICommunicationListener.CloseAsync()` is called on each listener.
  - The cancellation token passed to `RunAsync()` is canceled. A check of the cancellation token's `IsCancellationRequested` property returns true, and if called, the token's `ThrowIfCancellationRequested` method throws an `OperationCanceledException`.
2. After `CloseAsync()` finishes on each listener and `RunAsync()` also finishes, the service's `StatefulServiceBase.OnChangeRoleAsync()` is called. This call is not commonly overridden in the service.

### For the Secondary that's promoted

Similarly, Service Fabric needs the Secondary replica that's promoted to start listening for messages on the wire and start any background tasks it needs to complete. As a result, this process looks like it did when the service is created, except that the replica itself already exists. The following APIs are called:

1. In parallel:
  - `StatefulServiceBase.CreateServiceReplicaListeners()` is invoked and any returned listeners are opened. `ICommunicationListener.OpenAsync()` is called on each listener.
  - The service's `StatefulServiceBase.RunAsync()` method is called.
2. After all the replica listener's `OpenAsync()` calls finish and `RunAsync()` is called, `StatefulServiceBase.OnChangeRoleAsync()` is called. This call is not commonly overridden in the service.

### Common issues during stateful service shutdown and Primary demotion

Service Fabric changes the Primary of a stateful service for a variety of reasons. The most common are [cluster rebalancing](#) and [application upgrade](#). During these operations (as well as during normal service shutdown, like you'd see if the service was deleted), it is important that the service respect the `CancellationToken`.

Services that do not handle cancellation cleanly can experience several issues. These operations are slow because Service Fabric waits for the services to stop gracefully. This can ultimately lead to failed upgrades that time out and roll back. Failure to honor the cancellation token can also cause imbalanced clusters. Clusters become unbalanced because nodes get hot, but the services can't be rebalanced because it takes too long to move them elsewhere.

Because the services are stateful, it is also likely that they use the [Reliable Collections](#). In Service Fabric, when a Primary is demoted, one of the first things that happens is that write access to the underlying state is revoked. This leads to a second set of issues that can affect the service lifecycle. The collections return exceptions based on the timing and whether the replica is being moved or shut down. These exceptions should be handled correctly. Exceptions thrown by Service Fabric fall into permanent ([FabricException](#)) and transient ([FabricTransientException](#)) categories. Permanent exceptions should be logged and thrown while the transient exceptions can be retried based on some retry logic.

Handling the exceptions that come from use of the [ReliableCollections](#) in conjunction with service lifecycle events is an important part of testing and validating a Reliable Service. We recommend that you always run your service under load while performing upgrades and [chaos testing](#) before deploying to production. These basic steps help ensure that your service is correctly implemented and handles lifecycle events correctly.

## Notes on the service lifecycle

- Both the [RunAsync\(\)](#) method and the [CreateServiceReplicaListeners/CreateServiceInstanceListeners](#) calls are optional. A service can have one of them, both, or neither. For example, if the service does all its work in response to user calls, there is no need for it to implement [RunAsync\(\)](#). Only the communication listeners and their associated code are necessary. Similarly, creating and returning communication listeners is optional, as the service can have only background work to do, and so only needs to implement [RunAsync\(\)](#).
- It is valid for a service to complete [RunAsync\(\)](#) successfully and return from it. Completing is not a failure condition. Completing [RunAsync\(\)](#) indicates that the background work of the service has finished. For stateful reliable services, [RunAsync\(\)](#) is called again if the replica is demoted from Primary to Secondary and then promoted back to Primary.
- If a service exits from [RunAsync\(\)](#) by throwing some unexpected exception, this constitutes a failure. The service object is shut down and a health error is reported.
- Although there is no time limit on returning from these methods, you immediately lose the ability to write to Reliable Collections, and therefore, cannot complete any real work. We recommended that you return as quickly as possible upon receiving the cancellation request. If your service does not respond to these API calls in a reasonable amount of time, Service Fabric can forcibly terminate your service. Usually this only happens during application upgrades or when a service is being deleted. This timeout is 15 minutes by default.
- Failures in the [OnCloseAsync\(\)](#) path result in [OnAbort\(\)](#) being called, which is a last-chance best-effort opportunity for the service to clean up and release any resources that they have claimed.

## Next steps

- [Introduction to Reliable Services](#)
- [Reliable Services quick start](#)
- [Reliable Services advanced usage](#)
- [Replicas and instances](#)

# Reliable services lifecycle overview

8/18/2017 • 3 min to read • [Edit Online](#)

When thinking about the lifecycles of Reliable Services, the basics of the lifecycle are the most important. In general:

- During Startup
  - Services are constructed
  - They have an opportunity to construct and return zero or more listeners
  - Any returned listeners are opened, allowing communication with the service
  - The Service's runAsync method is called, allowing the service to do long running or background work
- During shutdown
  - The cancellation token passed to runAsync is canceled, and the listeners are closed
  - Once that is complete, the service object itself is destructed

There are details around the exact ordering of these events. In particular, the order of events may change slightly depending on whether the Reliable Service is Stateless or Stateful. In addition, for stateful services, we have to deal with the Primary swap scenario. During this sequence, the role of Primary is transferred to another replica (or comes back) without the service shutting down. Finally, we have to think about error or failure conditions.

## Stateless service startup

The lifecycle of a stateless service is fairly straightforward. Here's the order of events:

1. The Service is constructed
2. Then, in parallel two things happen:
  - `StatelessService.createServiceInstanceListeners()` is invoked and any returned listeners are Opened (`CommunicationListener.openAsync()` is called on each listener)
  - The service's runAsync method (`StatelessService.runAsync()`) is called
3. If present, the service's own onOpenAsync method is called (Specifically, `StatelessService.onOpenAsync()` is called. This is an uncommon override but it is available).

It is important to note that there is no ordering between the calls to create and open the listeners and runAsync. The listeners may open before runAsync is started. Similarly, runAsync may end up invoked before the communication listeners are open or have even been constructed. If any synchronization is required, it is left as an exercise to the implementer. Common solutions:

- Sometimes listeners can't function until some other information is created or work done. For stateless services that work can usually be done in the service's constructor, during the `createServiceInstanceListeners()` call, or as a part of the construction of the listener itself.
- Sometimes the code in runAsync does not want to start until the listeners are open. In this case additional coordination is necessary. One common solution is some flag within the listeners indicating when they have completed, which is checked in runAsync before continuing to actual work.

## Stateless service shutdown

When shutting down a stateless service, the same pattern is followed, just in reverse:

1. In parallel

- Any open listeners are Closed (`CommunicationListener.closeAsync()` is called on each listener)
  - The cancellation token passed to `runAsync()` is canceled (checking the cancellation token's `isCancelled` property returns true, and if called the token's `throwIfCancellationRequested` method throws a `CancellationException`)
2. Once `closeAsync()` completes on each listener and `runAsync()` also completes, the service's `StatelessService.onCloseAsync()` method is called, if present (again this is an uncommon override).
  3. After `StatelessService.onCloseAsync()` completes, the service object is destructed

## Notes on service lifecycle

- Both the `runAsync()` method and the `createServiceInstanceListeners` calls are optional. A service may have one of them, both, or neither. For example, if the service does all its work in response to user calls, there is no need for it to implement `runAsync()`. Only the communication listeners and their associated code are necessary. Similarly, creating and returning communication listeners is optional, as the service may have only background work to do, and so only needs to implement `runAsync()`
- It is valid for a service to complete `runAsync()` successfully and return from it. This is not considered a failure condition and would represent the background work of the service completing. For stateful reliable services `runAsync()` would be called again if the service were demoted from primary and then promoted back to primary.
- If a service exits from `runAsync()` by throwing some unexpected exception, this is a failure and the service object is shut down and a health error reported.
- While there is no time limit on returning from these methods, you immediately lose the ability to write and therefore cannot complete any real work. It is recommended that you return as quickly as possible upon receiving the cancellation request. If your service does not respond to these API calls in a reasonable amount of time Service Fabric may forcibly terminate your service. Usually this only happens during application upgrades or when a service is being deleted. This timeout is 15 minutes by default.
- Failures in the `onCloseAsync()` path result in `onAbort()` being called which is a last-chance best-effort opportunity for the service to clean up and release any resources that they have claimed.

### NOTE

Stateful reliable services are not supported in java yet.

## Next steps

- [Introduction to Reliable Services](#)
- [Reliable Services quick start](#)
- [Reliable Services advanced usage](#)

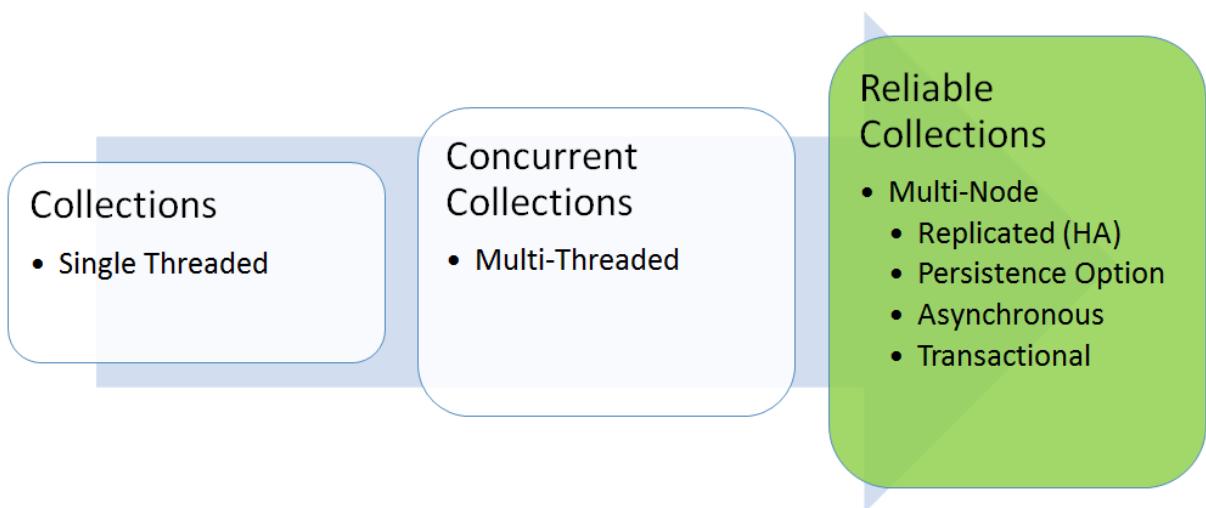
# Introduction to Reliable Collections in Azure Service Fabric stateful services

6/27/2017 • 2 min to read • [Edit Online](#)

Reliable Collections enable you to write highly available, scalable, and low-latency cloud applications as though you were writing single computer applications. The classes in the **Microsoft.ServiceFabric.Data.Collections** namespace provide a set of collections that automatically make your state highly available. Developers need to program only to the Reliable Collection APIs and let Reliable Collections manage the replicated and local state.

The key difference between Reliable Collections and other high-availability technologies (such as Redis, Azure Table service, and Azure Queue service) is that the state is kept locally in the service instance while also being made highly available. This means that:

- All reads are local, which results in low latency and high-throughput reads.
- All writes incur the minimum number of network IOs, which results in low latency and high-throughput writes.



Reliable Collections can be thought of as the natural evolution of the **System.Collections** classes: a new set of collections that are designed for the cloud and multi-computer applications without increasing complexity for the developer. As such, Reliable Collections are:

- Replicated: State changes are replicated for high availability.
- Persisted: Data is persisted to disk for durability against large-scale outages (for example, a datacenter power outage).
- Asynchronous: APIs are asynchronous to ensure that threads are not blocked when incurring IO.
- Transactional: APIs utilize the abstraction of transactions so you can manage multiple Reliable Collections within a service easily.

Reliable Collections provide strong consistency guarantees out of the box to make reasoning about application state easier. Strong consistency is achieved by ensuring transaction commits finish only after the entire transaction has been logged on a majority quorum of replicas, including the primary. To achieve weaker consistency, applications can acknowledge back to the client/requester before the asynchronous commit returns.

The Reliable Collections APIs are an evolution of concurrent collections APIs (found in the **System.Collections.Concurrent** namespace):

- Asynchronous: Returns a task since, unlike concurrent collections, the operations are replicated and persisted.
- No out parameters: Uses `ConditionalValue<T>` to return a bool and a value instead of out parameters.  
`ConditionalValue<T>` is like `Nullable<T>` but does not require T to be a struct.
- Transactions: Uses a transaction object to enable the user to group actions on multiple Reliable Collections in a transaction.

Today, **Microsoft.ServiceFabric.Data.Collections** contains three collections:

- **Reliable Dictionary**: Represents a replicated, transactional, and asynchronous collection of key/value pairs. Similar to **ConcurrentDictionary**, both the key and the value can be of any type.
- **Reliable Queue**: Represents a replicated, transactional, and asynchronous strict first-in, first-out (FIFO) queue. Similar to **ConcurrentQueue**, the value can be of any type.
- **Reliable Concurrent Queue**: Represents a replicated, transactional, and asynchronous best effort ordering queue for high throughput. Similar to the **ConcurrentQueue**, the value can be of any type.

## Next steps

- [Reliable Collection Guidelines & Recommendations](#)
- [Working with Reliable Collections](#)
- [Transactions and Locks](#)
- [Reliable State Manager and Collection Internals](#)
- [Managing Data](#)
  - [Backup and Restore](#)
  - [Notifications](#)
  - [Reliable Collection serialization](#)
  - [Serialization and Upgrade](#)
  - [Reliable State Manager configuration](#)
- [Others](#)
  - [Reliable Services quick start](#)
  - [Developer reference for Reliable Collections](#)

# Guidelines and recommendations for Reliable Collections in Azure Service Fabric

6/27/2017 • 2 min to read • [Edit Online](#)

This section provides guidelines for using Reliable State Manager and Reliable Collections. The goal is to help users avoid common pitfalls.

The guidelines are organized as simple recommendations prefixed with the terms *Do*, *Consider*, *Avoid* and *Do not*.

- Do not modify an object of custom type returned by read operations (for example, `TryPeekAsync` or `TryGetValueAsync`). Reliable Collections, just like Concurrent Collections, return a reference to the objects and not a copy.
- Do deep copy the returned object of a custom type before modifying it. Since structs and built-in types are pass-by-value, you do not need to do a deep copy on them.
- Do not use `TimeSpan.MaxValue` for time-outs. Time-outs should be used to detect deadlocks.
- Do not use a transaction after it has been committed, aborted, or disposed.
- Do not use an enumeration outside of the transaction scope it was created in.
- Do not create a transaction within another transaction's `using` statement because it can cause deadlocks.
- Do ensure that your `IComparable< TKey >` implementation is correct. The system takes dependency on `IComparable< TKey >` for merging checkpoints and rows.
- Do use Update lock when reading an item with an intention to update it to prevent a certain class of deadlocks.
- Consider keeping your items (for example, `TKey + TValue` for Reliable Dictionary) below 80 KBytes: smaller the better. This reduces the amount of Large Object Heap usage as well as disk and network IO requirements. Often, it reduces replicating duplicate data when only one small part of the value is being updated. Common way to achieve this in Reliable Dictionary, is to break your rows in to multiple rows.
- Consider using backup and restore functionality to have disaster recovery.
- Avoid mixing single entity operations and multi-entity operations (e.g. `GetCountAsync`, `CreateEnumerableAsync`) in the same transaction due to the different isolation levels.
- Do handle `InvalidOperationException`. User transactions can be aborted by the system for variety of reasons. For example, when the Reliable State Manager is changing its role out of Primary or when a long-running transaction is blocking truncation of the transactional log. In such cases, user may receive `InvalidOperationException` indicating that their transaction has already been terminated. Assuming, the termination of the transaction was not requested by the user, best way to handle this exception is to dispose the transaction, check if the cancellation token has been signaled (or the role of the replica has been changed), and if not create a new transaction and retry.

Here are some things to keep in mind:

- The default time-out is four seconds for all the Reliable Collection APIs. Most users should use the default time-out.
- The default cancellation token is `cancellationToken.None` in all Reliable Collections APIs.
- The key type parameter (`TKey`) for a Reliable Dictionary must correctly implement `GetHashCode()` and `Equals()`. Keys must be immutable.
- To achieve high availability for the Reliable Collections, each service should have at least a target and minimum replica set size of 3.
- Read operations on the secondary may read versions that are not quorum committed. This means that a version of data that is read from a single secondary might be false progressed. Reads from Primary are always stable:

can never be false progressed.

## Next steps

- [Working with Reliable Collections](#)
- [Transactions and Locks](#)
- [Reliable State Manager and Collection Internals](#)
- Managing Data
  - [Backup and Restore](#)
  - [Notifications](#)
  - [Serialization and Upgrade](#)
  - [Reliable State Manager configuration](#)
- Others
  - [Reliable Services quick start](#)
  - [Developer reference for Reliable Collections](#)

# Working with Reliable Collections

6/27/2017 • 10 min to read • [Edit Online](#)

Service Fabric offers a stateful programming model available to .NET developers via Reliable Collections. Specifically, Service Fabric provides reliable dictionary and reliable queue classes. When you use these classes, your state is partitioned (for scalability), replicated (for availability), and transacted within a partition (for ACID semantics). Let's look at a typical usage of a reliable dictionary object and see what it's actually doing.

```
///retry:  
  
try {  
    // Create a new Transaction object for this partition  
    using (ITransaction tx = base.StateManager.CreateTransaction()) {  
        // AddAsync takes key's write lock; if >4 secs, TimeoutException  
        // Key & value put in temp dictionary (read your own writes),  
        // serialized, redo/undo record is logged & sent to  
        // secondary replicas  
        await m_dic.AddAsync(tx, key, value, cancellationToken);  
  
        // CommitAsync sends Commit record to log & secondary replicas  
        // After quorum responds, all locks released  
        await tx.CommitAsync();  
    }  
    // If CommitAsync not called, Dispose sends Abort  
    // record to log & all locks released  
}  
catch (TimeoutException) {  
    await Task.Delay(100, cancellationToken); goto retry;  
}
```

All operations on reliable dictionary objects (except for ClearAsync which is not undoable), require an ITransaction object. This object has associated with it any and all changes you're attempting to make to any reliable dictionary and/or reliable queue objects within a single partition. You acquire an ITransaction object by calling the partition's StateManager's CreateTransaction method.

In the code above, the ITransaction object is passed to a reliable dictionary's AddAsync method. Internally, dictionary methods that accept a key take a reader/writer lock associated with the key. If the method modifies the key's value, the method takes a write lock on the key and if the method only reads from the key's value, then a read lock is taken on the key. Since AddAsync modifies the key's value to the new, passed-in value, the key's write lock is taken. So, if 2 (or more) threads attempt to add values with the same key at the same time, one thread will acquire the write lock and the other threads will block. By default, methods block for up to 4 seconds to acquire the lock; after 4 seconds, the methods throw a TimeoutException. Method overloads exist allowing you to pass an explicit timeout value if you'd prefer.

Usually, you write your code to react to a TimeoutException by catching it and retrying the entire operation (as shown in the code above). In my simple code, I'm just calling Task.Delay passing 100 milliseconds each time. But, in reality, you might be better off using some kind of exponential back-off delay instead.

Once the lock is acquired, AddAsync adds the key and value object references to an internal temporary dictionary associated with the ITransaction object. This is done to provide you with read-your-own-writes semantics. That is, after you call AddAsync, a later call to TryGetValueAsync (using the same ITransaction object) will return the value even if you have not yet committed the transaction. Next, AddAsync serializes your key and value objects to byte arrays and appends these byte arrays to a log file on the local node. Finally, AddAsync sends the byte arrays to all

the secondary replicas so they have the same key/value information. Even though the key/value information has been written to a log file, the information is not considered part of the dictionary until the transaction that they are associated with has been committed.

In the code above, the call to CommitAsync commits all of the transaction's operations. Specifically, it appends commit information to the log file on the local node and also sends the commit record to all the secondary replicas. Once a quorum (majority) of the replicas has replied, all data changes are considered permanent and any locks associated with keys that were manipulated via the ITransaction object are released so other threads/transactions can manipulate the same keys and their values.

If CommitAsync is not called (usually due to an exception being thrown), then the ITransaction object gets disposed. When disposing an uncommitted ITransaction object, Service Fabric appends abort information to the local node's log file and nothing needs to be sent to any of the secondary replicas. And then, any locks associated with keys that were manipulated via the transaction are released.

## Common pitfalls and how to avoid them

Now that you understand how the reliable collections work internally, let's take a look at some common misuses of them. See the code below:

```
using (ITransaction tx = StateManager.CreateTransaction()) {
    // AddAsync serializes the name/user, logs the bytes,
    // & sends the bytes to the secondary replicas.
    await m_dic.AddAsync(tx, name, user);

    // The line below updates the property's value in memory only; the
    // new value is NOT serialized, logged, & sent to secondary replicas.
    user.LastLogin = DateTime.UtcNow; // Corruption!

    await tx.CommitAsync();
}
```

When working with a regular .NET dictionary, you can add a key/value to the dictionary and then change the value of a property (such as LastLogin). However, this code will not work correctly with a reliable dictionary. Remember from the earlier discussion, the call to AddAsync serializes the key/value objects to byte arrays and then saves the arrays to a local file and also sends them to the secondary replicas. If you later change a property, this changes the property's value in memory only; it does not impact the local file or the data sent to the replicas. If the process crashes, what's in memory is thrown away. When a new process starts or if another replica becomes primary, then the old property value is what is available.

I cannot stress enough how easy it is to make the kind of mistake shown above. And, you will only learn about the mistake if/when the process goes down. The correct way to write the code is simply to reverse the two lines:

```
using (ITransaction tx = StateManager.CreateTransaction()) {
    user.LastLogin = DateTime.UtcNow; // Do this BEFORE calling AddAsync
    await m_dic.AddAsync(tx, name, user);
    await tx.CommitAsync();
}
```

Here is another example showing a common mistake:

```

using (ITransaction tx = StateManager.CreateTransaction()) {
    // Use the user's name to look up their data
    ConditionalValue<User> user =
        await m_dic.TryGetValueAsync(tx, name);

    // The user exists in the dictionary, update one of their properties.
    if (user.HasValue) {
        // The line below updates the property's value in memory only; the
        // new value is NOT serialized, logged, & sent to secondary replicas.
        user.Value.LastLogin = DateTime.UtcNow; // Corruption!
        await tx.CommitAsync();
    }
}

```

Again, with regular .NET dictionaries, the code above works fine and is a common pattern: the developer uses a key to look up a value. If the value exists, the developer changes a property's value. However, with reliable collections, this code exhibits the same problem as already discussed: **you MUST not modify an object once you have given it to a reliable collection.**

The correct way to update a value in a reliable collection, is to get a reference to the existing value and consider the object referred to by this reference immutable. Then, create a new object which is an exact copy of the original object. Now, you can modify the state of this new object and write the new object into the collection so that it gets serialized to byte arrays, appended to the local file and sent to the replicas. After committing the change(s), the in-memory objects, the local file, and all the replicas have the same exact state. All is good!

The code below shows the correct way to update a value in a reliable collection:

```

using (ITransaction tx = StateManager.CreateTransaction()) {
    // Use the user's name to look up their data
    ConditionalValue<User> currentUser =
        await m_dic.TryGetValueAsync(tx, name);

    // The user exists in the dictionary, update one of their properties.
    if (currentUser.HasValue) {
        // Create new user object with the same state as the current user object.
        // NOTE: This must be a deep copy; not a shallow copy. Specifically, only
        // immutable state can be shared by currentUser & updatedUser object graphs.
        User updatedUser = new User(currentUser);

        // In the new object, modify any properties you desire
        updatedUser.LastLogin = DateTime.UtcNow;

        // Update the key's value to the updateUser info
        await m_dic.SetValue(tx, name, updatedUser);

        await tx.CommitAsync();
    }
}

```

## Define immutable data types to prevent programmer error

Ideally, we'd like the compiler to report errors when you accidentally produce code that mutates state of an object that you are supposed to consider immutable. But, the C# compiler does not have the ability to do this. So, to avoid potential programmer bugs, we highly recommend that you define the types you use with reliable collections to be immutable types. Specifically, this means that you stick to core value types (such as numbers [Int32, UInt64, etc.], DateTime, Guid, TimeSpan, and the like). And, of course, you can also use String. It is best to avoid collection properties as serializing and deserializing them can frequently hurt performance. However, if

you want to use collection properties, we highly recommend the use of .NET's immutable collections library ([System.Collections.Immutable](#)). This library is available for download from <http://nuget.org>. We also recommend sealing your classes and making fields read-only whenever possible.

The UserInfo type below demonstrates how to define an immutable type taking advantage of aforementioned recommendations.

```
[DataContract]
// If you don't seal, you must ensure that any derived classes are also immutable
public sealed class UserInfo {
    private static readonly IEnumerable<ItemId> NoBids = ImmutableList<ItemId>.Empty;

    public UserInfo(String email, IEnumerable<ItemId> itemsBidding = null) {
        Email = email;
        ItemsBidding = (itemsBidding == null) ? NoBids : itemsBidding.ToImmutableList();
    }

    [OnDeserialized]
    private void OnDeserialized(StreamingContext context) {
        // Convert the serialized collection to an immutable collection
        ItemsBidding = ItemsBidding.ToImmutableList();
    }

    [DataMember]
    public readonly String Email;

    // Ideally, this would be a readonly field but it can't be because OnDeserialized
    // has to set it. So instead, the getter is public and the setter is private.
    [DataMember]
    public IEnumerable<ItemId> ItemsBidding { get; private set; }

    // Since each UserInfo object is immutable, we add a new ItemId to the ItemsBidding
    // collection by creating a new immutable UserInfo object with the added ItemId.
    public UserInfo AddItemBidding(ItemId itemId) {
        return new UserInfo(Email, ((ImmutableList<ItemId>)ItemsBidding).Add(itemId));
    }
}
```

The ItemId type is also an immutable type as shown here:

```
[DataContract]
public struct ItemId {

    [DataMember] public readonly String Seller;
    [DataMember] public readonly String ItemName;
    public ItemId(String seller, String itemName) {
        Seller = seller;
        ItemName = itemName;
    }
}
```

## Schema versioning (upgrades)

Internally, Reliable Collections serialize your objects using .NET's DataContractSerializer. The serialized objects are persisted to the primary replica's local disk and are also transmitted to the secondary replicas. As your service matures, it's likely you'll want to change the kind of data (schema) your service requires. You must approach versioning of your data with great care. First and foremost, you must always be able to deserialize old data. Specifically, this means your deserialization code must be infinitely backward compatible: Version 333 of your service code must be able to operate on data placed in a reliable collection by version 1 of your service code 5

years ago.

Furthermore, service code is upgraded one upgrade domain at a time. So, during an upgrade, you have two different versions of your service code running simultaneously. You must avoid having the new version of your service code use the new schema as old versions of your service code might not be able to handle the new schema. When possible, you should design each version of your service to be forward compatible by 1 version. Specifically, this means that V1 of your service code should be able to simply ignore any schema elements it does not explicitly handle. However, it must be able to save any data it doesn't explicitly know about and simply write it back out when updating a dictionary key or value.

#### **WARNING**

While you can modify the schema of a key, you must ensure that your key's hash code and equals algorithms are stable. If you change how either of these algorithms operate, you will not be able to look up the key within the reliable dictionary ever again.

Alternatively, you can perform what is typically referred to as a 2-phase upgrade. With a 2-phase upgrade, you upgrade your service from V1 to V2: V2 contains the code that knows how to deal with the new schema change but this code doesn't execute. When the V2 code reads V1 data, it operates on it and writes V1 data. Then, after the upgrade is complete across all upgrade domains, you can somehow signal to the running V2 instances that the upgrade is complete. (One way to signal this is to roll out a configuration upgrade; this is what makes this a 2-phase upgrade.) Now, the V2 instances can read V1 data, convert it to V2 data, operate on it, and write it out as V2 data. When other instances read V2 data, they do not need to convert it, they just operate on it, and write out V2 data.

## Next Steps

To learn about creating forward compatible data contracts, see [Forward-Compatible Data Contracts](#).

To learn best practices on versioning data contracts, see [Data Contract Versioning](#).

To learn how to implement version tolerant data contracts, see [Version-Tolerant Serialization Callbacks](#).

To learn how to provide a data structure that can interoperate across multiple versions, see [IExtensibleDataObject](#).

# Transactions and lock modes in Azure Service Fabric Reliable Collections

6/27/2017 • 3 min to read • [Edit Online](#)

## Transaction

A transaction is a sequence of operations performed as a single logical unit of work. A transaction must exhibit the following ACID properties. (see: <https://technet.microsoft.com/en-us/library/ms190612>)

- **Atomicity:** A transaction must be an atomic unit of work. In other words, either all its data modifications are performed, or none of them is performed.
- **Consistency:** When completed, a transaction must leave all data in a consistent state. All internal data structures must be correct at the end of the transaction.
- **Isolation:** Modifications made by concurrent transactions must be isolated from the modifications made by any other concurrent transactions. The isolation level used for an operation within an ITransaction is determined by the IReliableState performing the operation.
- **Durability:** After a transaction has completed, its effects are permanently in place in the system. The modifications persist even in the event of a system failure.

### Isolation levels

Isolation level defines the degree to which the transaction must be isolated from modifications made by other transactions. There are two isolation levels that are supported in Reliable Collections:

- **Repeatable Read:** Specifies that statements cannot read data that has been modified but not yet committed by other transactions and that no other transactions can modify data that has been read by the current transaction until the current transaction finishes. For more details, see <https://msdn.microsoft.com/library/ms173763.aspx>.
- **Snapshot:** Specifies that data read by any statement in a transaction is the transactionally consistent version of the data that existed at the start of the transaction. The transaction can recognize only data modifications that were committed before the start of the transaction. Data modifications made by other transactions after the start of the current transaction are not visible to statements executing in the current transaction. The effect is as if the statements in a transaction get a snapshot of the committed data as it existed at the start of the transaction. Snapshots are consistent across Reliable Collections. For more details, see <https://msdn.microsoft.com/library/ms173763.aspx>.

Reliable Collections automatically choose the isolation level to use for a given read operation depending on the operation and the role of the replica at the time of transaction's creation. Following is the table that depicts isolation level defaults for Reliable Dictionary and Queue operations.

OPERATION \ ROLE	PRIMARY	SECONDARY
Single Entity Read	Repeatable Read	Snapshot
Enumeration, Count	Snapshot	Snapshot

## NOTE

Common examples for Single Entity Operations are `IReliableDictionary.TryGetValueAsync`,  
`IReliableQueue.TryPeekAsync`.

Both the Reliable Dictionary and the Reliable Queue support Read Your Writes. In other words, any write within a transaction will be visible to a following read that belongs to the same transaction.

## Locks

In Reliable Collections, all transactions implement rigorous two phase locking: a transaction does not release the locks it has acquired until the transaction terminates with either an abort or a commit.

Reliable Dictionary uses row level locking for all single entity operations. Reliable Queue trades off concurrency for strict transactional FIFO property. Reliable Queue uses operation level locks allowing one transaction with `TryPeekAsync` and/or `TryDequeueAsync` and one transaction with `EnqueueAsync` at a time. Note that to preserve FIFO, if a `TryPeekAsync` or `TryDequeueAsync` ever observes that the Reliable Queue is empty, they will also lock `EnqueueAsync`.

Write operations always take Exclusive locks. For read operations, the locking depends on a couple of factors. Any read operation done using Snapshot isolation is lock free. Any Repeatable Read operation by default takes Shared locks. However, for any read operation that supports Repeatable Read, the user can ask for an Update lock instead of the Shared lock. An Update lock is an asymmetric lock used to prevent a common form of deadlock that occurs when multiple transactions lock resources for potential updates at a later time.

The lock compatibility matrix can be found in the following table:

REQUEST \ GRANTED	NONE	SHARED	UPDATE	EXCLUSIVE
Shared	No conflict	No conflict	Conflict	Conflict
Update	No conflict	No conflict	Conflict	Conflict
Exclusive	No conflict	Conflict	Conflict	Conflict

Time-out argument in the Reliable Collections APIs is used for deadlock detection. For example, two transactions (T1 and T2) are trying to read and update K1. It is possible for them to deadlock, because they both end up having the Shared lock. In this case, one or both of the operations will time out.

This deadlock scenario is a great example of how an Update lock can prevent deadlocks.

## Next steps

- [Working with Reliable Collections](#)
- [Reliable Services notifications](#)
- [Reliable Services backup and restore \(disaster recovery\)](#)
- [Reliable State Manager configuration](#)
- [Developer reference for Reliable Collections](#)

# Introduction to ReliableConcurrentQueue in Azure Service Fabric

6/27/2017 • 8 min to read • [Edit Online](#)

Reliable Concurrent Queue is an asynchronous, transactional, and replicated queue which features high concurrency for enqueue and dequeue operations. It is designed to deliver high throughput and low latency by relaxing the strict FIFO ordering provided by [Reliable Queue](#) and instead provides a best-effort ordering.

## APIs

CONCURRENT QUEUE	RELIABLE CONCURRENT QUEUE
void Enqueue(T item)	Task EnqueueAsync(ITransaction tx, T item)
bool TryDequeue(out T result)	Task< ConditionalValue < T > > TryDequeueAsync(ITransaction tx)
int Count()	long Count()

## Comparison with Reliable Queue

Reliable Concurrent Queue is offered as an alternative to [Reliable Queue](#). It should be used in cases where strict FIFO ordering is not required, as guaranteeing FIFO requires a tradeoff with concurrency. [Reliable Queue](#) uses locks to enforce FIFO ordering, with at most one transaction allowed to enqueue and at most one transaction allowed to dequeue at a time. In comparison, Reliable Concurrent Queue relaxes the ordering constraint and allows any number concurrent transactions to interleave their enqueue and dequeue operations. Best-effort ordering is provided, however the relative ordering of two values in a Reliable Concurrent Queue can never be guaranteed.

Reliable Concurrent Queue provides higher throughput and lower latency than [Reliable Queue](#) whenever there are multiple concurrent transactions performing enqueues and/or dequeues.

A sample use case for the ReliableConcurrentQueue is the [Message Queue](#) scenario. In this scenario, one or more message producers create and add items to the queue, and one or more message consumers pull messages from the queue and process them. Multiple producers and consumers can work independently, using concurrent transactions in order to process the queue.

## Usage Guidelines

- The queue expects that the items in the queue have a low retention period. That is, the items would not stay in the queue for a long time.
- The queue does not guarantee strict FIFO ordering.
- The queue does not read its own writes. If an item is enqueued within a transaction, it will not be visible to a dequeuer within the same transaction.
- Dequeues are not isolated from each other. If item A is dequeued in transaction  $txnA$ , even though  $txnA$  is not committed, item A would not be visible to a concurrent transaction  $txnB$ . If  $txnA$  aborts, A will become visible to  $txnB$  immediately.
- *TryPeekAsync* behavior can be implemented by using a *TryDequeueAsync* and then aborting the transaction. An example of this can be found in the Programming Patterns section.

- Count is non-transactional. It can be used to get an idea of the number of elements in the queue, but represents a point-in-time and cannot be relied upon.
- Expensive processing on the dequeued items should not be performed while the transaction is active, to avoid long-running transactions which may have a performance impact on the system.

## Code Snippets

Let us look at a few code snippets and their expected outputs. Exception handling is ignored in this section.

### EnqueueAsync

Here are a few code snippets for using EnqueueAsync followed by their expected outputs.

- *Case 1: Single Enqueue Task*

```
using (var txn = this.StateManager.CreateTransaction())
{
    await this.Queue.EnqueueAsync(txn, 10, cancellationToken);
    await this.Queue.EnqueueAsync(txn, 20, cancellationToken);

    await txn.CommitAsync();
}
```

Assume that the task completed successfully, and that there were no concurrent transactions modifying the queue.

The user can expect the queue to contain the items in any of the following orders:

10, 20

20, 10

- *Case 2: Parallel Enqueue Task*

```
// Parallel Task 1
using (var txn = this.StateManager.CreateTransaction())
{
    await this.Queue.EnqueueAsync(txn, 10, cancellationToken);
    await this.Queue.EnqueueAsync(txn, 20, cancellationToken);

    await txn.CommitAsync();
}

// Parallel Task 2
using (var txn = this.StateManager.CreateTransaction())
{
    await this.Queue.EnqueueAsync(txn, 30, cancellationToken);
    await this.Queue.EnqueueAsync(txn, 40, cancellationToken);

    await txn.CommitAsync();
}
```

Assume that the tasks completed successfully, that the tasks ran in parallel, and that there were no other concurrent transactions modifying the queue. No inference can be made about the order of items in the queue. For this code snippet, the items may appear in any of the  $4!$  possible orderings. The queue will attempt to keep the items in the original (enqueued) order, but may be forced to reorder them due to concurrent operations or faults.

### DequeueAsync

Here are a few code snippets for using TryDequeueAsync followed by the expected outputs. Assume that the queue is already populated with the following items in the queue:

- Case 1: Single Dequeue Task

```
using (var txn = this.StateManager.CreateTransaction())
{
    await this.Queue.TryDequeueAsync(txn, cancellationToken);
    await this.Queue.TryDequeueAsync(txn, cancellationToken);
    await this.Queue.TryDequeueAsync(txn, cancellationToken);

    await txn.CommitAsync();
}
```

Assume that the task completed successfully, and that there were no concurrent transactions modifying the queue. Since no inference can be made about the order of the items in the queue, any three of the items may be dequeued, in any order. The queue will attempt to keep the items in the original (enqueued) order, but may be forced to reorder them due to concurrent operations or faults.

- Case 2: Parallel Dequeue Task

```
// Parallel Task 1
List<int> dequeue1;
using (var txn = this.StateManager.CreateTransaction())
{
    dequeue1.Add(await this.Queue.TryDequeueAsync(txn, cancellationToken)).val;
    dequeue1.Add(await this.Queue.TryDequeueAsync(txn, cancellationToken)).val;

    await txn.CommitAsync();
}

// Parallel Task 2
List<int> dequeue2;
using (var txn = this.StateManager.CreateTransaction())
{
    dequeue2.Add(await this.Queue.TryDequeueAsync(txn, cancellationToken)).val;
    dequeue2.Add(await this.Queue.TryDequeueAsync(txn, cancellationToken)).val;

    await txn.CommitAsync();
}
```

Assume that the tasks completed successfully, that the tasks ran in parallel, and that there were no other concurrent transactions modifying the queue. Since no inference can be made about the order of the items in the queue, the lists `dequeue1` and `dequeue2` will each contain any two items, in any order.

The same item will *not* appear in both lists. Hence, if `dequeue1` has 10, 30, then `dequeue2` would have 20, 40.

- Case 3: Dequeue Ordering With Transaction Abort

Aborting a transaction with in-flight dequeues puts the items back on the head of the queue. The order in which the items are put back on the head of the queue is not guaranteed. Let us look at the following code:

```
using (var txn = this.StateManager.CreateTransaction())
{
    await this.Queue.TryDequeueAsync(txn, cancellationToken);
    await this.Queue.TryDequeueAsync(txn, cancellationToken);

    // Abort the transaction
    await txn.AbortAsync();
}
```

Assume that the items were dequeued in the following order:

10, 20

When we abort the transaction, the items would be added back to the head of the queue in any of the following orders:

10, 20

20, 10

The same is true for all cases where the transaction was not successfully *Committed*.

## Programming Patterns

In this section, let us look at a few programming patterns that might be helpful in using ReliableConcurrentQueue.

### Batch Dequeues

A recommended programming pattern is for the consumer task to batch its dequeues instead of performing one dequeue at a time. The user can choose to throttle delays between every batch or the batch size. The following code snippet shows this programming model. Note that in this example, the processing is done after the transaction is committed, so if a fault were to occur while processing, the unprocessed items will be lost without having been processed. Alternatively, the processing can be done within the transaction's scope, however this may have a negative impact on performance and requires handling of the items already processed.

```

int batchSize = 5;
long delayMs = 100;

while(!cancellationToken.IsCancellationRequested)
{
    // Buffer for dequeued items
    List<int> processItems = new List<int>();

    using (var txn = this.StateManager.CreateTransaction())
    {
        ConditionalValue<int> ret;

        for(int i = 0; i < batchSize; ++i)
        {
            ret = await this.Queue.TryDequeueAsync(txn, cancellationToken);

            if (ret.HasValue)
            {
                // If an item was dequeued, add to the buffer for processing
                processItems.Add(ret.Value);
            }
            else
            {
                // else break the for loop
                break;
            }
        }

        await txn.CommitAsync();
    }

    // Process the dequeues
    for (int i = 0; i < processItems.Count; ++i)
    {
        Console.WriteLine("Value : " + processItems[i]);
    }

    int delayFactor = batchSize - processItems.Count;
    await Task.Delay(TimeSpan.FromMilliseconds(delayMs * delayFactor), cancellationToken);
}

```

## Best-Effort Notification-Based Processing

Another interesting programming pattern uses the Count API. Here, we can implement best-effort notification-based processing for the queue. The queue Count can be used to throttle an enqueue or a dequeue task. Note that as in the previous example, since the processing occurs outside the transaction, unprocessed items may be lost if a fault occurs during processing.

```

int threshold = 5;
long delayMs = 1000;

while(!cancellationToken.IsCancellationRequested)
{
    while (this.Queue.Count < threshold)
    {
        cancellationToken.ThrowIfCancellationRequested();

        // If the queue does not have the threshold number of items, delay the task and check again
        await Task.Delay(TimeSpan.FromMilliseconds(delayMs), cancellationToken);
    }

    // If there are approximately threshold number of items, try and process the queue

    // Buffer for dequeued items
    List<int> processItems = new List<int>();

    using (var txn = this.StateManager.CreateTransaction())
    {
        ConditionalValue<int> ret;

        do
        {
            ret = await this.Queue.TryDequeueAsync(txn, cancellationToken);

            if (ret.HasValue)
            {
                // If an item was dequeued, add to the buffer for processing
                processItems.Add(ret.Value);
            }
        } while (processItems.Count < threshold && ret.HasValue);

        await txn.CommitAsync();
    }

    // Process the dequeues
    for (int i = 0; i < processItems.Count; ++i)
    {
        Console.WriteLine("Value : " + processItems[i]);
    }
}

```

## Best-Effort Drain

A drain of the queue cannot be guaranteed due to the concurrent nature of the data structure. It is possible that, even if no user operations on the queue are in-flight, a particular call to TryDequeueAsync may not return an item which was previously enqueued and committed. The enqueued item is guaranteed to *eventually* become visible to dequeue, however without an out-of-band communication mechanism, an independent consumer cannot know that the queue has reached a steady-state even if all producers have been stopped and no new enqueue operations are allowed. Thus, the drain operation is best-effort as implemented below.

The user should stop all further producer and consumer tasks, and wait for any in-flight transactions to commit or abort, before attempting to drain the queue. If the user knows the expected number of items in the queue, they can set up a notification which signals that all items have been dequeued.

```

int numItemsDequeued;
int batchSize = 5;

ConditionalValue ret;

do
{
    List<int> processItems = new List<int>();

    using (var txn = this.StateManager.CreateTransaction())
    {
        do
        {
            ret = await this.Queue.TryDequeueAsync(txn, cancellationToken);

            if(ret.HasValue)
            {
                // Buffer the dequeues
                processItems.Add(ret.Value);
            }
        } while (ret.HasValue && processItems.Count < batchSize);

        await txn.CommitAsync();
    }

    // Process the dequeues
    for (int i = 0; i < processItems.Count; ++i)
    {
        Console.WriteLine("Value : " + processItems[i]);
    }
} while (ret.HasValue);

```

## Peek

ReliableConcurrentQueue does not provide the *TryPeekAsync* api. Users can get the peek semantic by using a *TryDequeueAsync* and then aborting the transaction. In this example, dequeues are processed only if the item's value is greater than 10.

```

using (var txn = this.StateManager.CreateTransaction())
{
    ConditionalValue ret = await this.Queue.TryDequeueAsync(txn, cancellationToken);
    bool valueProcessed = false;

    if (ret.HasValue)
    {
        if (ret.Value > 10)
        {
            // Process the item
            Console.WriteLine("Value : " + ret.Value);
            valueProcessed = true;
        }
    }

    if (valueProcessed)
    {
        await txn.CommitAsync();
    }
    else
    {
        await txn.AbortAsync();
    }
}

```

## Must Read

- [Reliable Services Quick Start](#)
- [Working with Reliable Collections](#)
- [Reliable Services notifications](#)
- [Reliable Services Backup and Restore \(Disaster Recovery\)](#)
- [Reliable State Manager Configuration](#)
- [Getting Started with Service Fabric Web API Services](#)
- [Advanced Usage of the Reliable Services Programming Model](#)
- [Developer Reference for Reliable Collections](#)

# Reliable Collection object serialization in Azure Service Fabric

6/27/2017 • 4 min to read • [Edit Online](#)

Reliable Collections' replicate and persist their items to make sure they are durable across machine failures and power outages. Both to replicate and to persist items, Reliable Collections' need to serialize them.

Reliable Collections' get the appropriate serializer for a given type from Reliable State Manager. Reliable State Manager contains built-in serializers and allows custom serializers to be registered for a given type.

## Built-in Serializers

Reliable State Manager includes built-in serializer for some common types, so that they can be serialized efficiently by default. For other types, Reliable State Manager falls back to use the [DataContractSerializer](#). Built-in serializers are more efficient since they know their types cannot change and they do not need to include information about the type like its type name.

Reliable State Manager has built-in serializer for following types:

- Guid
- bool
- byte
- sbyte
- byte[]
- char
- string
- decimal
- double
- float
- int
- uint
- long
- ulong
- short
- ushort

## Custom Serialization

Custom serializers are commonly used to increase performance or to encrypt the data over the wire and on disk. Among other reasons, custom serializers are commonly more efficient than generic serializer since they don't need to serialize information about the type.

[IReliableStateManager.TryAddStateSerializer](#) is used to register a custom serializer for the given type T. This registration should happen in the construction of the StatefulServiceBase to ensure that before recovery starts, all Reliable Collections have access to the relevant serializer to read their persisted data.

```
public StatefulBackendService(StatefulServiceContext context)
    : base(context)
{
    if (!this.StateManager.TryAddStateSerializer(new OrderKeySerializer()))
    {
        throw new InvalidOperationException("Failed to set OrderKey custom serializer");
    }
}
```

#### NOTE

Custom serializers are given precedence over built-in serializers. For example, when a custom serializer for int is registered, it is used to serialize integers instead of the built-in serializer for int.

### How to implement a custom serializer

A custom serializer needs to implement the [IStateSerializer](#) interface.

#### NOTE

[IStateSerializer](#) includes an overload for Write and Read that takes in an additional T called base value. This API is for differential serialization. Currently differential serialization feature is not exposed. Hence, these two overloads are not called until differential serialization is exposed and enabled.

Following is an example custom type called OrderKey that contains four properties

```
public class OrderKey : IComparable<OrderKey>, IEquatable<OrderKey>
{
    public byte Warehouse { get; set; }

    public short District { get; set; }

    public int Customer { get; set; }

    public long Order { get; set; }

    #region Object Overrides for GetHashCode, CompareTo and Equals
    #endregion
}
```

Following is an example implementation of [IStateSerializer](#). Note that Read and Write overloads that take in baseValue, call their respective overload for forwards compatibility.

```

public class OrderKeySerializer : IStateSerializer<OrderKey>
{
    OrderKey IStateSerializer<OrderKey>.Read(BinaryReader reader)
    {
        var value = new OrderKey();
        value.Warehouse = reader.ReadByte();
        value.District = reader.ReadInt16();
        value.Customer = reader.ReadInt32();
        value.Order = reader.ReadInt64();

        return value;
    }

    void IStateSerializer<OrderKey>.Write(OrderKey value, BinaryWriter writer)
    {
        writer.Write(value.Warehouse);
        writer.Write(value.District);
        writer.Write(value.Customer);
        writer.Write(value.Order);
    }

    // Read overload for differential de-serialization
    OrderKey IStateSerializer<OrderKey>.Read(OrderKey baseValue, BinaryReader reader)
    {
        return ((IStateSerializer<OrderKey>)this).Read(reader);
    }

    // Write overload for differential serialization
    void IStateSerializer<OrderKey>.Write(OrderKey baseValue, OrderKey newValue, BinaryWriter writer)
    {
        ((IStateSerializer<OrderKey>)this).Write(newValue, writer);
    }
}

```

## Upgradability

In a [rolling application upgrade](#), the upgrade is applied to a subset of nodes, one upgrade domain at a time. During this process, some upgrade domains will be on the newer version of your application, and some upgrade domains will be on the older version of your application. During the rollout, the new version of your application must be able to read the old version of your data, and the old version of your application must be able to read the new version of your data. If the data format is not forward and backward compatible, the upgrade may fail, or worse, data may be lost or corrupted.

If you are using built-in serializer, you do not have to worry about compatibility. However, if you are using a custom serializer or the `DataContractSerializer`, the data have to be infinitely backwards and forwards compatible. In other words, each version of serializer needs to be able to serialize and de-serialize any version of the type.

Data Contract users should follow the well-defined versioning rules for adding, removing, and changing fields. Data Contract also has support for dealing with unknown fields, hooking into the serialization and deserialization process, and dealing with class inheritance. For more information, see [Using Data Contract](#).

Custom serializer users should adhere to the guidelines of the serializer they are using to make sure it is backwards and forwards compatible. Common way of supporting all versions is adding size information at the beginning and only adding optional properties. This way each version can read as much it can and jump over the remaining part of the stream.

## Next steps

- [Serialization and upgrade](#)
- [Developer reference for Reliable Collections](#)

- [Upgrading your Application Using Visual Studio](#) walks you through an application upgrade using Visual Studio.
- [Upgrading your Application Using Powershell](#) walks you through an application upgrade using PowerShell.
- Control how your application upgrades by using [Upgrade Parameters](#).
- Learn how to use advanced functionality while upgrading your application by referring to [Advanced Topics](#).
- Fix common problems in application upgrades by referring to the steps in [Troubleshooting Application Upgrades](#).

# Azure Service Fabric Reliable State Manager and Reliable Collection internals

6/27/2017 • 1 min to read • [Edit Online](#)

This document delves inside Reliable State Manager and Reliable Collections to see how core components work behind the scenes.

## NOTE

This document is work in-progress. Add comments to this article to tell us what topic you would like to learn more about.

## Local persistence model: log and checkpoint

The Reliable State Manager and Reliable Collections follow a persistence model that is called Log and Checkpoint. In this model, each state change is logged on disk first and then applied in memory. The complete state itself is persisted only occasionally (a.k.a. Checkpoint). The benefit is that deltas are turned into sequential append-only writes on disk for improved performance.

To better understand the Log and Checkpoint model, let's first look at the infinite disk scenario. The Reliable State Manager logs every operation before it is replicated. Logging allows the Reliable Collections to apply the operation only in memory. Since logs are persisted, even when the replica fails and needs to be restarted, the Reliable State Manager has enough information in its log to replay all the operations the replica has lost. As the disk is infinite, log records never need to be removed and the Reliable Collection needs to manage only the in-memory state.

Now let's look at the finite disk scenario. As log records accumulate, the Reliable State Manager will run out of disk space. Before that happens, the Reliable State Manager needs to truncate its log to make room for the newer records. Reliable State Manager requests the Reliable Collections to checkpoint their in-memory state to disk. At this point, the Reliable Collections' would persist its in-memory state. Once the Reliable Collections complete their checkpoints, the Reliable State Manager can truncate the log to free up disk space. When the replica needs to be restarted, Reliable Collections recover their checkpointed state, and the Reliable State Manager recovers and plays back all the state changes that occurred since the last checkpoint.

Another value add of checkpointing is that it improves recovery times in common scenarios. Log contains all operations that have happened since the last checkpoint. So it may include multiple versions of an item like multiple values for a given row in Reliable Dictionary. In contrast, a Reliable Collection checkpoints only the latest version of each value for a key.

## Next steps

- [Transactions and Locks](#)

# Get started with Reliable Services

6/27/2017 • 9 min to read • [Edit Online](#)

An Azure Service Fabric application contains one or more services that run your code. This guide shows you how to create both stateless and stateful Service Fabric applications with [Reliable Services](#). This Microsoft Virtual Academy video also shows you how to create a stateless Reliable service:



## Basic concepts

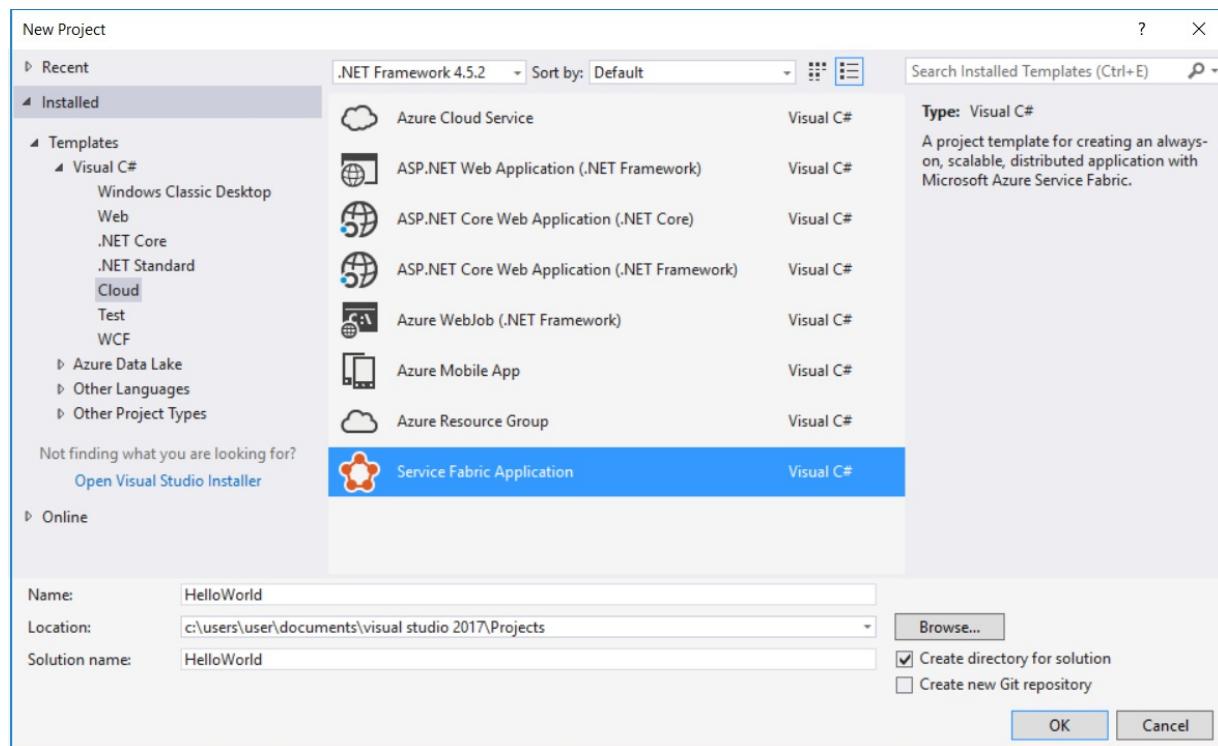
To get started with Reliable Services, you only need to understand a few basic concepts:

- **Service type:** This is your service implementation. It is defined by the class you write that extends `StatelessService` and any other code or dependencies used therein, along with a name and a version number.
- **Named service instance:** To run your service, you create named instances of your service type, much like you create object instances of a class type. A service instance has a name in the form of a URI using the "fabric://" scheme, such as "fabric:/MyApp/MyService".
- **Service host:** The named service instances you create need to run inside a host process. The service host is just a process where instances of your service can run.
- **Service registration:** Registration brings everything together. The service type must be registered with the Service Fabric runtime in a service host to allow Service Fabric to create instances of it to run.

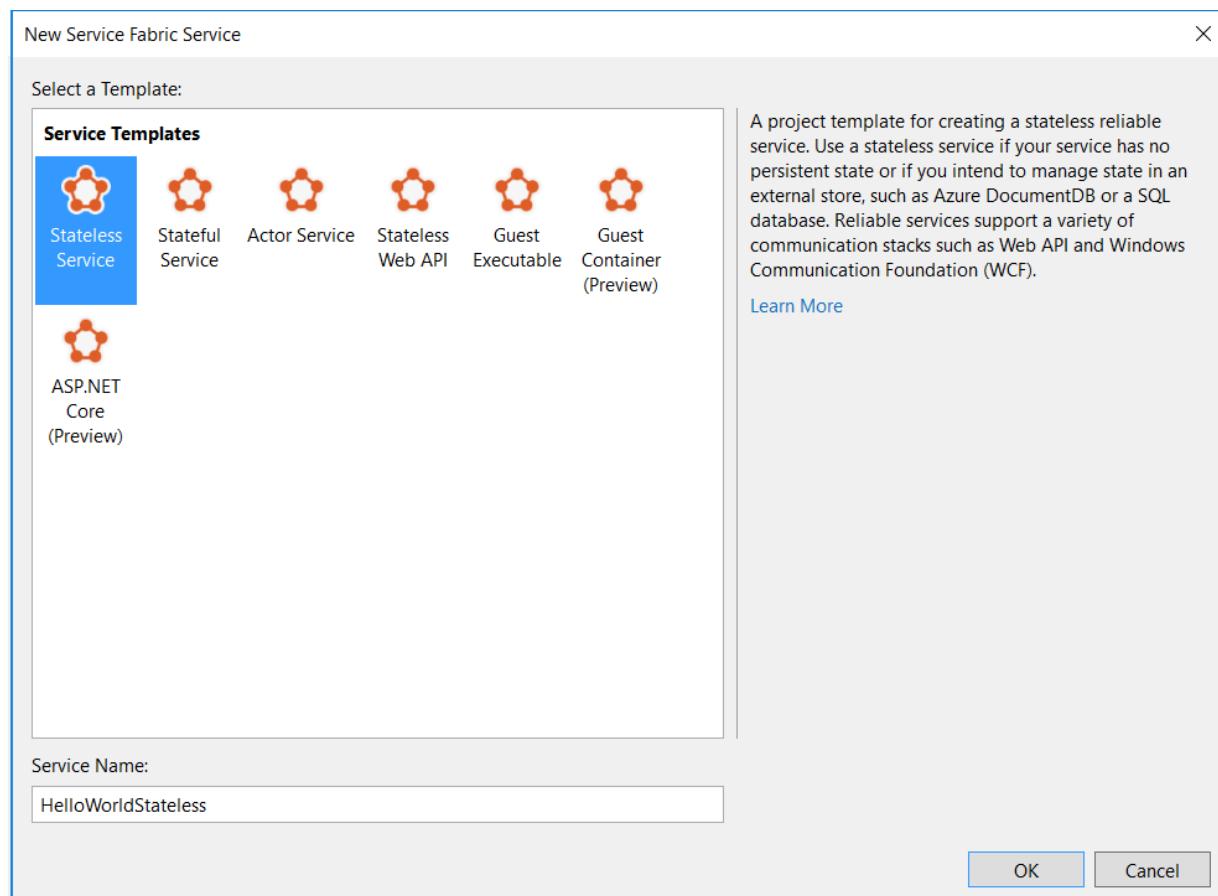
## Create a stateless service

A stateless service is a type of service that is currently the norm in cloud applications. It is considered stateless because the service itself does not contain data that needs to be stored reliably or made highly available. If an instance of a stateless service shuts down, all of its internal state is lost. In this type of service, state must be persisted to an external store, such as Azure Tables or a SQL database, for it to be made highly available and reliable.

Launch Visual Studio 2015 or Visual Studio 2017 as an administrator, and create a new Service Fabric application project named *HelloWorld*:



Then create a stateless service project named *HelloWorldStateless*:



Your solution now contains two projects:

- *HelloWorld*. This is the *application* project that contains your *services*. It also contains the application manifest that describes the application, as well as a number of PowerShell scripts that help you to deploy your application.
- *HelloWorldStateless*. This is the service project. It contains the stateless service implementation.

## Implement the service

Open the **HelloWorldStateless.cs** file in the service project. In Service Fabric, a service can run any business logic. The service API provides two entry points for your code:

- An open-ended entry point method, called *RunAsync*, where you can begin executing any workloads, including long-running compute workloads.

```
protected override async Task RunAsync(CancellationToken cancellationToken)
{
    ...
}
```

- A communication entry point where you can plug in your communication stack of choice, such as ASP.NET Core. This is where you can start receiving requests from users and other services.

```
protected override IEnumerable<ServiceInstanceListener> CreateServiceInstanceListeners()
{
    ...
}
```

In this tutorial, we will focus on the `RunAsync()` entry point method. This is where you can immediately start running your code. The project template includes a sample implementation of `RunAsync()` that increments a rolling count.

#### NOTE

For details about how to work with a communication stack, see [Service Fabric Web API services with OWIN self-hosting](#)

## RunAsync

```
protected override async Task RunAsync(CancellationToken cancellationToken)
{
    // TODO: Replace the following sample code with your own logic
    //       or remove this RunAsync override if it's not needed in your service.

    long iterations = 0;

    while (true)
    {
        cancellationToken.ThrowIfCancellationRequested();

        ServiceEventSource.Current.ServiceMessage(this, "Working-{0}", ++iterations);

        await Task.Delay(TimeSpan.FromSeconds(1), cancellationToken);
    }
}
```

The platform calls this method when an instance of a service is placed and ready to execute. For a stateless service, that simply means when the service instance is opened. A cancellation token is provided to coordinate when your service instance needs to be closed. In Service Fabric, this open/close cycle of a service instance can occur many times over the lifetime of the service as a whole. This can happen for various reasons, including:

- The system moves your service instances for resource balancing.
- Faults occur in your code.
- The application or system is upgraded.
- The underlying hardware experiences an outage.

This orchestration is managed by the system to keep your service highly available and properly balanced.

`RunAsync()` should not block synchronously. Your implementation of `RunAsync` should return a `Task` or await on any long-running or blocking operations to allow the runtime to continue. Note in the `while(true)` loop in the previous example, a `Task`-returning `await Task.Delay()` is used. If your workload must block synchronously, you should schedule a new `Task` with `Task.Run()` in your `RunAsync` implementation.

Cancellation of your workload is a cooperative effort orchestrated by the provided cancellation token. The system will wait for your task to end (by successful completion, cancellation, or fault) before it moves on. It is important to honor the cancellation token, finish any work, and exit `RunAsync()` as quickly as possible when the system requests cancellation.

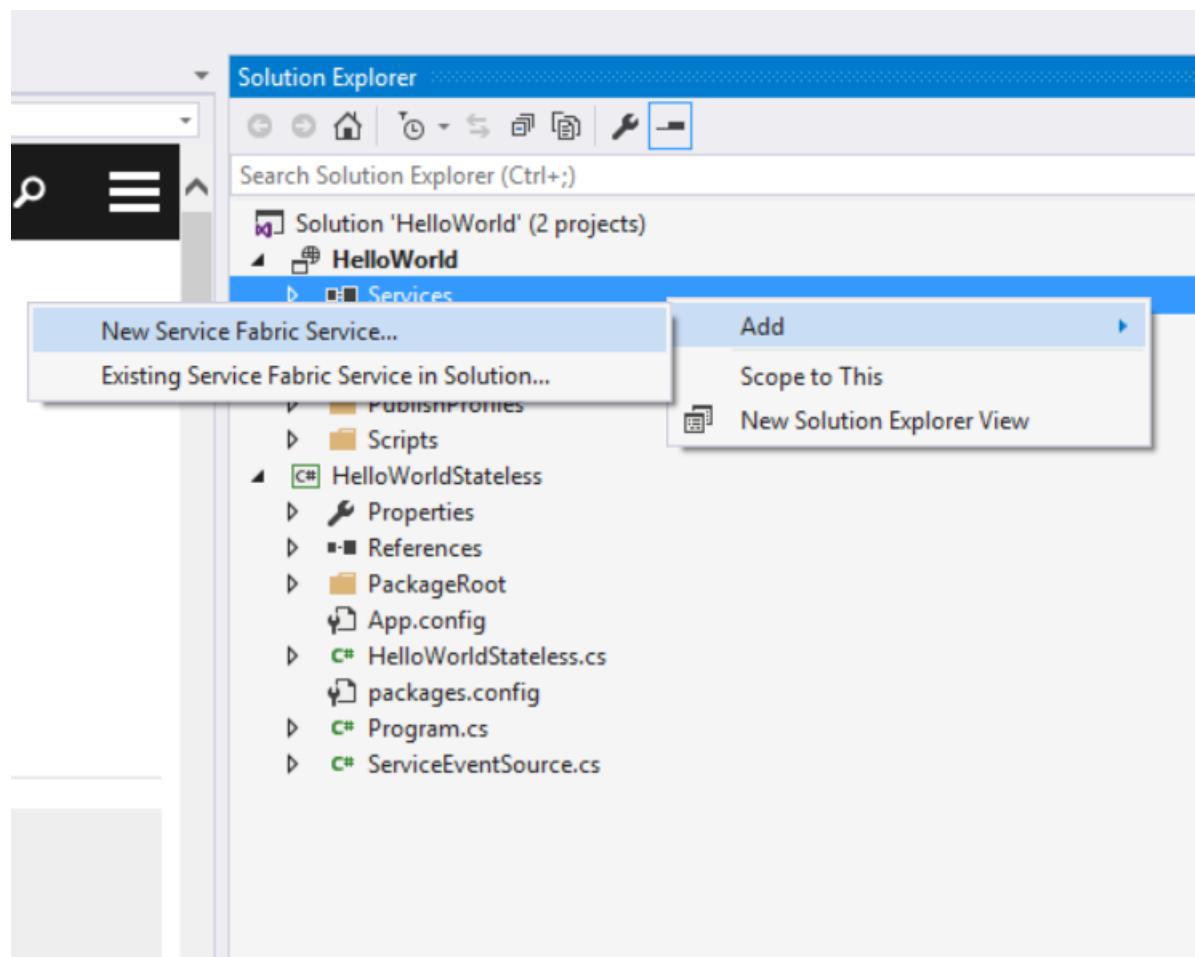
In this stateless service example, the count is stored in a local variable. But because this is a stateless service, the value that's stored exists only for the current lifecycle of its service instance. When the service moves or restarts, the value is lost.

## Create a stateful service

Service Fabric introduces a new kind of service that is stateful. A stateful service can maintain state reliably within the service itself, co-located with the code that's using it. State is made highly available by Service Fabric without the need to persist state to an external store.

To convert a counter value from stateless to highly available and persistent, even when the service moves or restarts, you need a stateful service.

In the same *HelloWorld* application, you can add a new service by right-clicking on the Services references in the application project and selecting **Add -> New Service Fabric Service**.



Select **Stateful Service** and name it *HelloWorldStateful*. Click **OK**.

Select a Template:

**Service Templates**

Stateless Service



Stateful Service



Actor Service



Stateless Web API



Guest Executable

Guest Container  
(Preview)ASP.NET  
Core  
(Preview)

A project template for creating a stateful reliable service. Use a stateful service when you want to manage your persistent state within the service using reliable collections framework. State can be partitioned for scale and is replicated across the cluster for reliability. Reliable services support a variety of communication stacks such as Web API and Windows Communication Foundation (WCF).

[Learn More](#)

Service Name:

Your application should now have two services: the stateless service *HelloWorldStateless* and the stateful service *HelloWorldStateful*.

A stateful service has the same entry points as a stateless service. The main difference is the availability of a *state provider* that can store state reliably. Service Fabric comes with a state provider implementation called [Reliable Collections](#), which lets you create replicated data structures through the Reliable State Manager. A stateful Reliable Service uses this state provider by default.

Open **HelloWorldStateful.cs** in *HelloWorldStateful*, which contains the following RunAsync method:

```

protected override async Task RunAsync(CancellationToken cancellationToken)
{
    // TODO: Replace the following sample code with your own logic
    //       or remove this RunAsync override if it's not needed in your service.

    var myDictionary = await this.StateManager.GetOrAddAsync<IReliableDictionary<string, long>>
        ("myDictionary");

    while (true)
    {
        cancellationToken.ThrowIfCancellationRequested();

        using (var tx = this.StateManager.CreateTransaction())
        {
            var result = await myDictionary.TryGetValueAsync(tx, "Counter");

            ServiceEventSource.Current.ServiceMessage(this, "Current Counter Value: {0}",
                result.HasValue ? result.Value.ToString() : "Value does not exist.");

            await myDictionary.AddOrUpdateAsync(tx, "Counter", 0, (key, value) => ++value);

            // If an exception is thrown before calling CommitAsync, the transaction aborts, all changes
            // are discarded, and nothing is saved to the secondary replicas.
            await tx.CommitAsync();
        }

        await Task.Delay(TimeSpan.FromSeconds(1), cancellationToken);
    }
}

```

## RunAsync

`RunAsync()` operates similarly in stateful and stateless services. However, in a stateful service, the platform performs additional work on your behalf before it executes `RunAsync()`. This work can include ensuring that the Reliable State Manager and Reliable Collections are ready to use.

## Reliable Collections and the Reliable State Manager

```

var myDictionary = await this.StateManager.GetOrAddAsync<IReliableDictionary<string, long>>
    ("myDictionary");

```

[IReliableDictionary](#) is a dictionary implementation that you can use to reliably store state in the service. With Service Fabric and Reliable Collections, you can store data directly in your service without the need for an external persistent store. Reliable Collections make your data highly available. Service Fabric accomplishes this by creating and managing multiple *replicas* of your service for you. It also provides an API that abstracts away the complexities of managing those replicas and their state transitions.

Reliable Collections can store any .NET type, including your custom types, with a couple of caveats:

- Service Fabric makes your state highly available by *replicating* state across nodes, and Reliable Collections store your data to local disk on each replica. This means that everything that is stored in Reliable Collections must be *serializable*. By default, Reliable Collections use [DataContract](#) for serialization, so it's important to make sure that your types are [supported by the Data Contract Serializer](#) when you use the default serializer.
- Objects are replicated for high availability when you commit transactions on Reliable Collections. Objects stored in Reliable Collections are kept in local memory in your service. This means that you have a local reference to the object.

It is important that you do not mutate local instances of those objects without performing an update operation on the reliable collection in a transaction. This is because changes to local instances of objects

will not be replicated automatically. You must re-insert the object back into the dictionary or use one of the *update* methods on the dictionary.

The Reliable State Manager manages Reliable Collections for you. You can simply ask the Reliable State Manager for a reliable collection by name at any time and at any place in your service. The Reliable State Manager ensures that you get a reference back. We don't recommend that you save references to reliable collection instances in class member variables or properties. Special care must be taken to ensure that the reference is set to an instance at all times in the service lifecycle. The Reliable State Manager handles this work for you, and it's optimized for repeat visits.

### Transactional and asynchronous operations

```
using (ITransaction tx = this.StateManager.CreateTransaction())
{
    var result = await myDictionary.TryGetValueAsync(tx, "Counter-1");

    await myDictionary.AddOrUpdateAsync(tx, "Counter-1", 0, (k, v) => ++v);

    await tx.CommitAsync();
}
```

Reliable Collections have many of the same operations that their `System.Collections.Generic` and `System.Collections.Concurrent` counterparts do, except LINQ. Operations on Reliable Collections are asynchronous. This is because write operations with Reliable Collections perform I/O operations to replicate and persist data to disk.

Reliable Collection operations are *transactional*, so that you can keep state consistent across multiple Reliable Collections and operations. For example, you may dequeue a work item from a Reliable Queue, perform an operation on it, and save the result in a Reliable Dictionary, all within a single transaction. This is treated as an atomic operation, and it guarantees that either the entire operation will succeed or the entire operation will roll back. If an error occurs after you dequeue the item but before you save the result, the entire transaction is rolled back and the item remains in the queue for processing.

## Run the application

We now return to the *HelloWorld* application. You can now build and deploy your services. When you press **F5**, your application will be built and deployed to your local cluster.

After the services start running, you can view the generated Event Tracing for Windows (ETW) events in a **Diagnostic Events** window. Note that the events displayed are from both the stateless service and the stateful service in the application. You can pause the stream by clicking the **Pause** button. You can then examine the details of a message by expanding that message.

#### NOTE

Before you run the application, make sure that you have a local development cluster running. Check out the [getting started guide](#) for information on setting up your local environment.

Diagnostic Events		
Filter Events		Configure
Timestamp	Event Name	Message
21:57:06.800	ServiceMessage	Working-17
21:57:05.784	ServiceMessage	Working-16
21:57:05.380	ServiceTypeRegistered	Service host process 4984 registered service type HelloWorldStateful
21:57:05.368	ServiceTypeRegistered	Service host process 5636 registered service type HelloWorldStateful
21:57:04.769	ServiceMessage	Working-15
21:57:03.750	ServiceMessage	Working-14
21:57:02.534	ServiceMessage	Working-13
21:57:01.519	ServiceMessage	Working-12
21:57:00.503	ServiceMessage	Working-11
21:56:59.488	ServiceMessage	Working-10
21:56:58.486	ServiceMessage	Working-9
21:56:57.484	ServiceMessage	Working-8
21:56:56.468	ServiceMessage	Working-7
21:56:55.452	ServiceMessage	Working-6
21:56:54.437	ServiceMessage	Working-5
21:56:53.421	ServiceMessage	Working-4
21:56:52.419	ServiceMessage	Working-3
21:56:51.405	ServiceMessage	Working-2
21:56:50.387	ServiceMessage	Working-1
21:56:49.378	ServiceMessage	Working-0
21:56:49.351	StatelessRunAsyncInvocation RunAsync	has been invoked for a stateless service instance. Application Type Name: HelloWorldType, Application Name: fabric:/HelloWorld, Service Type Name: HelloWorldStateless
21:56:48.563	ServiceTypeRegistered	Service host process 3904 registered service type HelloWorldStateful
21:56:48.090	ServiceTypeRegistered	Service host process 5416 registered service type HelloWorldStateless
21:56:44.865	CM	Application created: Application fabric:/HelloWorld Created: ApplicationType = HelloWorldType ApplicationTypeVersion = 1.0.0.0.
21:56:44.706	FM	Service Created: Service fabric:/HelloWorld/HelloWorldStateless partition 0d53bbe5-f098-4b49-b24f-6b4d86fcfc3a of ServiceType HelloWorldStatelessType created in Application fabr
21:56:44.644	FM	Service Created: Service fabric:/HelloWorld/HelloWorldStateful partition b8b172f0-bd95-49ca-97c9-e71d40f33b56 of ServiceType HelloWorldStatefulType created in Application fabric

## Next steps

[Debug your Service Fabric application in Visual Studio](#)

[Get started: Service Fabric Web API services with OWIN self-hosting](#)

[Learn more about Reliable Collections](#)

[Deploy an application](#)

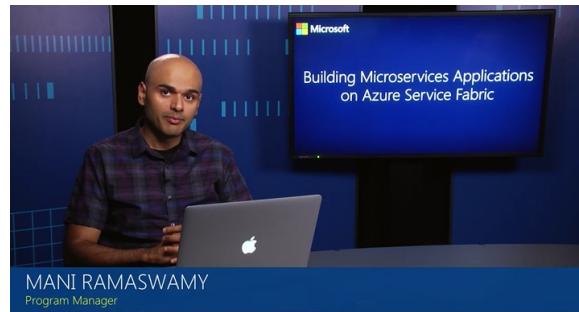
[Application upgrade](#)

[Developer reference for Reliable Services](#)

# Get started with Reliable Services

8/23/2017 • 4 min to read • [Edit Online](#)

This article explains the basics of Azure Service Fabric Reliable Services and walks you through creating and deploying a simple Reliable Service application written in Java. This Microsoft Virtual Academy video also shows you how to create a stateless Reliable service:



## Installation and setup

Before you start, make sure you have the Service Fabric development environment set up on your machine. If you need to set it up, go to [getting started on Mac](#) or [getting started on Linux](#).

## Basic concepts

To get started with Reliable Services, you only need to understand a few basic concepts:

- **Service type:** This is your service implementation. It is defined by the class you write that extends `StatelessService` and any other code or dependencies used therein, along with a name and a version number.
- **Named service instance:** To run your service, you create named instances of your service type, much like you create object instances of a class type. Service instances are in fact object instantiations of your service class that you write.
- **Service host:** The named service instances you create need to run inside a host. The service host is just a process where instances of your service can run.
- **Service registration:** Registration brings everything together. The service type must be registered with the Service Fabric runtime in a service host to allow Service Fabric to create instances of it to run.

## Create a stateless service

Start by creating a Service Fabric application. The Service Fabric SDK for Linux includes a Yeoman generator to provide the scaffolding for a Service Fabric application with a stateless service. Start by running the following Yeoman command:

```
$ yo azuresfjava
```

Follow the instructions to create a **Reliable Stateless Service**. For this tutorial, name the application "HelloWorldApplication" and the service "HelloWorld". The result includes directories for the `HelloWorldApplication` and `HelloWorld`.

```
HelloWorldApplication/
├── build.gradle
└── HelloWorld
    ├── build.gradle
    └── src
        └── statelessservice
            ├── HelloWorldServiceHost.java
            └── HelloWorldService.java
└── HelloWorldApplication
    ├── ApplicationManifest.xml
    └── HelloWorldPkg
        ├── Code
        │   ├── entryPoint.sh
        │   └── _readme.txt
        ├── Config
        │   └── _readme.txt
        ├── Data
        │   └── _readme.txt
        └── ServiceManifest.xml
├── install.sh
└── settings.gradle
└── uninstall.sh
```

## Implement the service

Open **HelloWorldApplication>HelloWorld/src/statelessservice>HelloWorldService.java**. This class defines the service type, and can run any code. The service API provides two entry points for your code:

- An open-ended entry point method, called `runAsync()`, where you can begin executing any workloads, including long-running compute workloads.

```
@Override
protected CompletableFuture<?> runAsync(CancellationToken cancellationToken) {
    ...
}
```

- A communication entry point where you can plug in your communication stack of choice. This is where you can start receiving requests from users and other services.

```
@Override
protected List<ServiceInstanceListener> createServiceInstanceListeners() {
    ...
}
```

In this tutorial, we focus on the `runAsync()` entry point method. This is where you can immediately start running your code.

### RunAsync

The platform calls this method when an instance of a service is placed and ready to execute. For a stateless service, that simply means when the service instance is opened. A cancellation token is provided to coordinate when your service instance needs to be closed. In Service Fabric, this open/close cycle of a service instance can occur many times over the lifetime of the service as a whole. This can happen for various reasons, including:

- The system moves your service instances for resource balancing.
- Faults occur in your code.
- The application or system is upgraded.
- The underlying hardware experiences an outage.

This orchestration is managed by Service Fabric to keep your service highly available and properly balanced.

`runAsync()` should not block synchronously. Your implementation of `runAsync` should return a `CompletableFuture` to allow the runtime to continue. If your workload needs to implement a long running task that should be done inside the `CompletableFuture`.

#### Cancellation

Cancellation of your workload is a cooperative effort orchestrated by the provided cancellation token. The system waits for your task to end (by successful completion, cancellation, or fault) before it moves on. It is important to honor the cancellation token, finish any work, and exit `runAsync()` as quickly as possible when the system requests cancellation. The following example demonstrates how to handle a cancellation event:

```
@Override  
protected CompletableFuture<?> runAsync(CancellationToken cancellationToken) {  
  
    // TODO: Replace the following sample code with your own logic  
    // or remove this runAsync override if it's not needed in your service.  
  
    CompletableFuture.runAsync(() -> {  
        long iterations = 0;  
        while(true)  
        {  
            cancellationToken.throwIfCancellationRequested();  
            logger.log(Level.INFO, "Working-{0}", ++iterations);  
  
            try  
            {  
                Thread.sleep(1000);  
            }  
            catch (IOException ex) {}  
        }  
    });  
}
```

#### Service registration

Service types must be registered with the Service Fabric runtime. The service type is defined in the `ServiceManifest.xml` and your service class that implements `StatelessService`. Service registration is performed in the process main entry point. In this example, the process main entry point is `HelloWorldServiceHost.java`:

```
public static void main(String[] args) throws Exception {  
    try {  
        ServiceRuntime.registerStatelessServiceAsync("HelloWorldType", (context) -> new HelloWorldService(),  
Duration.ofSeconds(10));  
        logger.log(Level.INFO, "Registered stateless service type HelloWorldType.");  
        Thread.sleep(Long.MAX_VALUE);  
    }  
    catch (Exception ex) {  
        logger.log(Level.SEVERE, "Exception in registration:", ex);  
        throw ex;  
    }  
}
```

## Run the application

The Yeoman scaffolding includes a gradle script to build the application and bash scripts to deploy and remove the application. To run the application, first build the application with gradle:

```
$ gradle
```

This produces a Service Fabric application package that can be deployed using Service Fabric CLI.

## Deploy with Service Fabric CLI

The install.sh script contains the necessary Service Fabric CLI commands to deploy the application package. Run the install.sh script to deploy the application.

```
$ ./install.sh
```

## Next steps

- [Getting started with Service Fabric CLI](#)

# Create your first Azure Service Fabric application

9/25/2017 • 3 min to read • [Edit Online](#)

Service Fabric provides SDKs for building services on Linux in both .NET Core and Java. In this tutorial, we look at how to create an application for Linux and build a service using C# on .NET Core 2.0.

## Prerequisites

Before you get started, make sure that you have [set up your Linux development environment](#). If you are using Mac OS X, you can [set up a Linux one-box environment in a virtual machine using Vagrant](#).

You will also want to install the [Service Fabric CLI](#)

### Install and set up the generators for C#

Service Fabric provides scaffolding tools which help you create Service Fabric applications from a terminal using Yeoman template generators. Follow these steps to set up the Service Fabric Yeoman template generators for C#:

1. Install nodejs and NPM on your machine

```
sudo apt-get install npm  
sudo apt install nodejs-legacy
```

2. Install [Yeoman](#) template generator on your machine from NPM

```
sudo npm install -g yo
```

3. Install the Service Fabric Yeo Java application generator from NPM

```
sudo npm install -g generator-azuresfcsharp
```

## Create the application

A Service Fabric application can contain one or more services, each with a specific role in delivering the application's functionality. The Service Fabric [Yeoman](#) generator for C#, which you installed in last step, makes it easy to create your first service and to add more later. Let's use Yeoman to create an application with a single service.

1. In a terminal, type the following command to start building the scaffolding: `yo azuresfcsharp`
2. Name your application.
3. Choose the type of your first service and name it. For the purposes of this tutorial, we choose a Reliable Actor Service.

```

$ yo azuresfcsharp

Welcome to Service
Fabric generator for
CSharp

? Name your application myapp
? Choose a framework for your service Reliable Actor Service
? Enter the name of actor service : myactorserv
  create myapp/myapp/MyactorservPkg/ServiceManifest.xml
  create myapp/myapp/ApplicationManifest.xml
  create myapp/myapp/MyactorservPkg/Code/entryPoint.sh
  create myapp/myapp/MyactorservPkg/Config/Settings.xml
  create myapp/src/myapp/Myactorserv.Interfaces/IMyactorserv.cs
  create myapp/src/myapp/Myactorserv.Interfaces/Myactorserv.Interfaces.csproj
  create myapp/src/myapp/MyactorservService/Myactorserv.cs
  create myapp/src/myapp/MyactorservService/MyactorservService.csproj
  create myapp/src/myapp/MyactorservService/Program.cs
  create myapp/src/myapp/MyactorservTestClient/MyactorservTestClient.csproj
  create myapp/src/myapp/MyactorservTestClient/Program.cs
  create myapp/src/myapp/MyactorservService/ActorEventListener.cs
  create myapp/src/myapp/MyactorservService/ActorEventSource.cs
  create myapp/MyactorservServiceTestClient/testclient.sh
  create myapp/install.sh
  create myapp/uninstall.sh
  create myapp/build.sh
  create myapp/myapp/MyactorservPkg/Config/_readme.txt
  create myapp/myapp/MyactorservPkg/Data/_readme.txt

```

#### NOTE

For more information about the options, see [Service Fabric programming model overview](#).

## Build the application

The Service Fabric Yeoman templates include a build script that you can use to build the app from the terminal (after navigating to the application folder).

```

cd myapp
./build.sh

```

## Deploy the application

Once the application is built, you can deploy it to the local cluster.

1. Connect to the local Service Fabric cluster.

```

sfctl cluster select --endpoint http://localhost:19080

```

2. Run the install script provided in the template to copy the application package to the cluster's image store, register the application type, and create an instance of the application.

```

./install.sh

```

Deploying the built application is the same as any other Service Fabric application. See the documentation on [managing a Service Fabric application with the Service Fabric CLI](#) for detailed instructions.

Parameters to these commands can be found in the generated manifests inside the application package.

Once the application has been deployed, open a browser and navigate to [Service Fabric Explorer](http://localhost:19080/Explorer) at <http://localhost:19080/Explorer>. Then, expand the **Applications** node and note that there is now an entry for your application type and another for the first instance of that type.

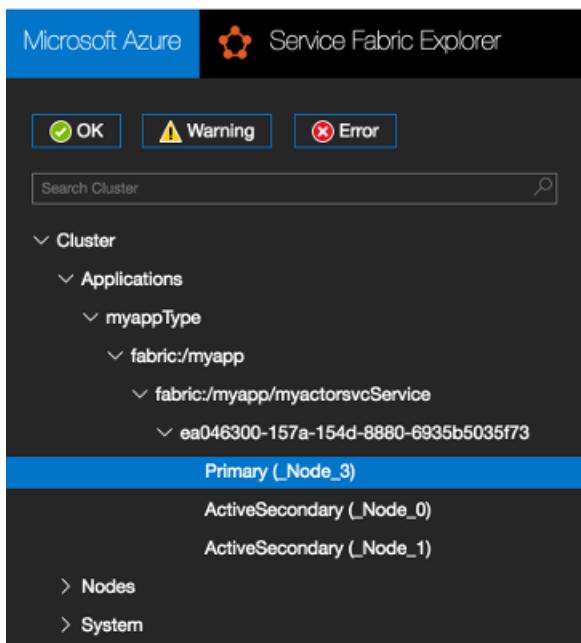
## Start the test client and perform a failover

Actor projects do not do anything on their own. They require another service or client to send them messages. The actor template includes a simple test script that you can use to interact with the actor service.

1. Run the script using the watch utility to see the output of the actor service.

```
cd myactorsvcTestClient  
watch -n 1 ./testclient.sh
```

2. In Service Fabric Explorer, locate node hosting the primary replica for the actor service. In the screenshot below, it is node 3.



3. Click the node you found in the previous step, then select **Deactivate (restart)** from the Actions menu. This action restarts one node in your local cluster forcing a failover to a secondary replica running on another node. As you perform this action, pay attention to the output from the test client and note that the counter continues to increment despite the failover.

## Adding more services to an existing application

To add another service to an application already created using `yo`, perform the following steps:

1. Change directory to the root of the existing application. For example, `cd ~/YeomanSamples/MyApplication`, if `MyApplication` is the application created by Yeoman.
2. Run `yo azuresfcsharp:AddService`

## Migrating from project.json to .csproj

1. Running 'dotnet migrate' in project root directory will migrate all the project.json to csproj format.
2. Update the project references accordingly to csproj files in project files.
3. Update the project file names to csproj files in build.sh.

## Next steps

- [Interacting with Service Fabric clusters using the Service Fabric CLI](#)
- [Learn about Service Fabric support options](#)
- [Getting started with Service Fabric CLI](#)

# How to use the Reliable Services communication APIs

6/27/2017 • 10 min to read • [Edit Online](#)

Azure Service Fabric as a platform is completely agnostic about communication between services. All protocols and stacks are acceptable, from UDP to HTTP. It's up to the service developer to choose how services should communicate. The Reliable Services application framework provides built-in communication stacks as well as APIs that you can use to build your custom communication components.

## Set up service communication

The Reliable Services API uses a simple interface for service communication. To open an endpoint for your service, simply implement this interface:

```
public interface ICommunicationListener
{
    Task<string> OpenAsync(CancellationToken cancellationToken);

    Task CloseAsync(CancellationToken cancellationToken);

    void Abort();
}
```

```
public interface CommunicationListener {
    CompletableFuture<String> openAsync(CancellationToken cancellationToken);

    CompletableFuture<?> closeAsync(CancellationToken cancellationToken);

    void abort();
}
```

You can then add your communication listener implementation by returning it in a service-based class method override.

For stateless services:

```
class MyStatelessService : StatelessService
{
    protected override IEnumerable<ServiceInstanceListener> CreateServiceInstanceListeners()
    {
        ...
    }
    ...
}
```

```

public class MyStatelessService extends StatelessService {

    @Override
    protected List<ServiceInstanceListener> createServiceInstanceListeners() {
        ...
    }
    ...
}

```

For stateful services:

#### NOTE

Stateful reliable services are not supported in Java yet.

```

class MyStatefulService : StatefulService
{
    protected override IEnumerable<ServiceReplicaListener> CreateServiceReplicaListeners()
    {
        ...
    }
    ...
}

```

In both cases, you return a collection of listeners. This allows your service to listen on multiple endpoints, potentially using different protocols, by using multiple listeners. For example, you may have an HTTP listener and a separate WebSocket listener. Each listener gets a name, and the resulting collection of *name : address* pairs are represented as a JSON object when a client requests the listening addresses for a service instance or a partition.

In a stateless service, the override returns a collection of ServiceInstanceListeners. A `ServiceInstanceListener` contains a function to create an `ICommunicationListener(C#) / CommunicationListener(Java)` and gives it a name. For stateful services, the override returns a collection of ServiceReplicaListeners. This is slightly different from its stateless counterpart, because a `ServiceReplicaListener` has an option to open an `ICommunicationListener` on secondary replicas. Not only can you use multiple communication listeners in a service, but you can also specify which listeners accept requests on secondary replicas and which ones listen only on primary replicas.

For example, you can have a ServiceRemotingListener that takes RPC calls only on primary replicas, and a second, custom listener that takes read requests on secondary replicas over HTTP:

```

protected override IEnumerable<ServiceReplicaListener> CreateServiceReplicaListeners()
{
    return new[]
    {
        new ServiceReplicaListener(context =>
            new MyCustomHttpListener(context),
            "HTTPReadonlyEndpoint",
            true),

        new ServiceReplicaListener(context =>
            this.CreateServiceRemotingListener(context),
            "rpcPrimaryEndpoint",
            false)
    };
}

```

#### NOTE

When creating multiple listeners for a service, each listener **must** be given a unique name.

Finally, describe the endpoints that are required for the service in the [service manifest](#) under the section on endpoints.

```
<Resources>
  <Endpoints>
    <Endpoint Name="WebServiceEndpoint" Protocol="http" Port="80" />
    <Endpoint Name="OtherServiceEndpoint" Protocol="tcp" Port="8505" />
  </Endpoints>
</Resources>
```

The communication listener can access the endpoint resources allocated to it from the `CodePackageActivationContext` in the `ServiceContext`. The listener can then start listening for requests when it is opened.

```
var codePackageActivationContext = serviceContext.CodePackageActivationContext;
var port = codePackageActivationContext.GetEndpoint("ServiceEndpoint").Port;
```

```
CodePackageActivationContext codePackageActivationContext = serviceContext.getCodePackageActivationContext();
int port = codePackageActivationContext.getEndpoint("ServiceEndpoint").getPort();
```

#### NOTE

Endpoint resources are common to the entire service package, and they are allocated by Service Fabric when the service package is activated. Multiple service replicas hosted in the same ServiceHost may share the same port. This means that the communication listener should support port sharing. The recommended way of doing this is for the communication listener to use the partition ID and replica-instance ID when it generates the listen address.

## Service address registration

A system service called the *Naming Service* runs on Service Fabric clusters. The Naming Service is a registrar for services and their addresses that each instance or replica of the service is listening on. When the `OpenAsync(C#) / openAsync(Java)` method of an `ICommunicationListener(C#) / CommunicationListener(Java)` completes, its return value gets registered in the Naming Service. This return value that gets published in the Naming Service is a string whose value can be anything at all. This string value is what clients see when they ask for an address for the service from the Naming Service.

```

public Task<string> OpenAsync(CancellationToken cancellationToken)
{
    EndpointResourceDescription serviceEndpoint =
    serviceContext.CodePackageActivationContext.GetEndpoint("ServiceEndpoint");
    int port = serviceEndpoint.Port;

    this.listeningAddress = string.Format(
        CultureInfo.InvariantCulture,
        "http://+:{0}/",
        port);

    this.publishAddress = this.listeningAddress.Replace("+", FabricRuntime.GetNodeContext().IPAddressOrFQDN);

    this.webApp = WebApp.Start(this.listeningAddress, appBuilder => this.startup.Invoke(appBuilder));

    // the string returned here will be published in the Naming Service.
    return Task.FromResult(this.publishAddress);
}

```

```

public CompletableFuture<String> openAsync(CancellationToken cancellationToken)
{
    EndpointResourceDescription serviceEndpoint =
    serviceContext.getCodePackageActivationContext.getEndpoint("ServiceEndpoint");
    int port = serviceEndpoint.getPort();

    this.publishAddress = String.format("http://%s:%d/", FabricRuntime.getNodeContext().getIpAddressOrFQDN(), port);

    this.webApp = new WebApp(port);
    this.webApp.start();

    /* the string returned here will be published in the Naming Service.
     */
    return CompletableFuture.completedFuture(this.publishAddress);
}

```

Service Fabric provides APIs that allow clients and other services to then ask for this address by service name. This is important because the service address is not static. Services are moved around in the cluster for resource balancing and availability purposes. This is the mechanism that allow clients to resolve the listening address for a service.

#### **NOTE**

For a complete walk-through of how to write a communication listener, see [Service Fabric Web API services with OWIN self-hosting](#) for C#, whereas for Java you can write your own HTTP server implementation, see EchoServer application example at <https://github.com/Azure-Samples/service-fabric-java-getting-started>.

## Communicating with a service

The Reliable Services API provides the following libraries to write clients that communicate with services.

### Service endpoint resolution

The first step to communication with a service is to resolve an endpoint address of the partition or instance of the service you want to talk to. The [ServicePartitionResolver\(C#\) / FabricServicePartitionResolver\(Java\)](#) utility class is a basic primitive that helps clients determine the endpoint of a service at runtime. In Service Fabric terminology, the process of determining the endpoint of a service is referred to as the *service endpoint resolution*.

To connect to services within a cluster, ServicePartitionResolver can be created using default settings. This is the

recommended usage for most situations:

```
ServicePartitionResolver resolver = ServicePartitionResolver.GetDefault();
```

```
FabricServicePartitionResolver resolver = FabricServicePartitionResolver.getDefault();
```

To connect to services in a different cluster, a ServicePartitionResolver can be created with a set of cluster gateway endpoints. Note that gateway endpoints are just different endpoints for connecting to the same cluster. For example:

```
ServicePartitionResolver resolver = new ServicePartitionResolver("mycluster.cloudapp.azure.com:19000",
    "mycluster.cloudapp.azure.com:19001");
```

```
FabricServicePartitionResolver resolver = new
FabricServicePartitionResolver("mycluster.cloudapp.azure.com:19000", "mycluster.cloudapp.azure.com:19001");
```

Alternatively, `ServicePartitionResolver` can be given a function for creating a `FabricClient` to use internally:

```
public delegate FabricClient CreateFabricClientDelegate();
```

```
public FabricServicePartitionResolver(CreateFabricClient createFabricClient) {
    ...
}

public interface CreateFabricClient {
    public FabricClient getFabricClient();
}
```

`FabricClient` is the object that is used to communicate with the Service Fabric cluster for various management operations on the cluster. This is useful when you want more control over how a service partition resolver interacts with your cluster. `FabricClient` performs caching internally and is generally expensive to create, so it is important to reuse `FabricClient` instances as much as possible.

```
ServicePartitionResolver resolver = new ServicePartitionResolver(() => CreateMyFabricClient());
```

```
FabricServicePartitionResolver resolver = new FabricServicePartitionResolver(() -> new
CreateFabricClientImpl());
```

A resolve method is then used to retrieve the address of a service or a service partition for partitioned services.

```
ServicePartitionResolver resolver = ServicePartitionResolver.GetDefault();

ResolvedServicePartition partition =
    await resolver.ResolveAsync(new Uri("fabric:/MyApp/MyService"), new ServicePartitionKey(),
cancellationToken);
```

```

FabricServicePartitionResolver resolver = FabricServicePartitionResolver.getDefault();

CompletableFuture<ResolvedServicePartition> partition =
    resolver.resolveAsync(new URI("fabric:/MyApp/MyService"), new ServicePartitionKey());

```

A service address can be resolved easily using a ServicePartitionResolver, but more work is required to ensure the resolved address can be used correctly. Your client needs to detect whether the connection attempt failed because of a transient error and can be retried (e.g., service moved or is temporarily unavailable), or a permanent error (e.g., service was deleted or the requested resource no longer exists). Service instances or replicas can move around from node to node at any time for multiple reasons. The service address resolved through ServicePartitionResolver may be stale by the time your client code attempts to connect. In that case again the client needs to re-resolve the address. Providing the previous `ResolvedServicePartition` indicates that the resolver needs to try again rather than simply retrieve a cached address.

Typically, the client code need not work with the ServicePartitionResolver directly. It is created and passed on to communication client factories in the Reliable Services API. The factories use the resolver internally to generate a client object that can be used to communicate with services.

### Communication clients and factories

The communication factory library implements a typical fault-handling retry pattern that makes retrying connections to resolved service endpoints easier. The factory library provides the retry mechanism while you provide the error handlers.

`ICommunicationClientFactory(C#) / CommunicationClientFactory(Java)` defines the base interface implemented by a communication client factory that produces clients that can talk to a Service Fabric service. The implementation of the `CommunicationClientFactory` depends on the communication stack used by the Service Fabric service where the client wants to communicate. The Reliable Services API provides a `CommunicationClientFactoryBase<TCommunicationClient>`. This provides a base implementation of the `CommunicationClientFactory` interface and performs tasks that are common to all the communication stacks. (These tasks include using a `ServicePartitionResolver` to determine the service endpoint). Clients usually implement the abstract `CommunicationClientFactoryBase` class to handle logic that is specific to the communication stack.

The communication client just receives an address and uses it to connect to a service. The client can use whatever protocol it wants.

```

class MyCommunicationClient : ICommunicationClient
{
    public ResolvedServiceEndpoint Endpoint { get; set; }

    public string ListenerName { get; set; }

    public ResolvedServicePartition ResolvedServicePartition { get; set; }
}

```

```

public class MyCommunicationClient implements CommunicationClient {

    private ResolvedServicePartition resolvedServicePartition;
    private String listenerName;
    private ResolvedServiceEndpoint endPoint;

    /*
     * Getters and Setters
     */
}

```

The client factory is primarily responsible for creating communication clients. For clients that don't maintain a persistent connection, such as an HTTP client, the factory only needs to create and return the client. Other protocols that maintain a persistent connection, such as some binary protocols, should also be validated by the factory to determine whether the connection needs to be re-created.

```
public class MyCommunicationClientFactory : CommunicationClientFactoryBase<MyCommunicationClient>
{
    protected override void AbortClient(MyCommunicationClient client)
    {
    }

    protected override Task<MyCommunicationClient> CreateClientAsync(string endpoint, CancellationToken cancellationToken)
    {
    }

    protected override bool ValidateClient(MyCommunicationClient clientChannel)
    {
    }

    protected override bool ValidateClient(string endpoint, MyCommunicationClient client)
    {
    }
}
```

```
public class MyCommunicationClientFactory extends CommunicationClientFactoryBase<MyCommunicationClient> {

    @Override
    protected boolean validateClient(MyCommunicationClient clientChannel) {
    }

    @Override
    protected boolean validateClient(String endpoint, MyCommunicationClient client) {
    }

    @Override
    protected CompletableFuture<MyCommunicationClient> createClientAsync(String endpoint) {
    }

    @Override
    protected void abortClient(MyCommunicationClient client) {
    }
}
```

Finally, an exception handler is responsible for determining what action to take when an exception occurs. Exceptions are categorized into **retryable** and **non retryable**.

- **Non retryable** exceptions simply get rethrown back to the caller.
- **retryable** exceptions are further categorized into **transient** and **non-transient**.
  - **Transient** exceptions are those that can simply be retried without re-resolving the service endpoint address. These will include transient network problems or service error responses other than those that indicate the service endpoint address does not exist.
  - **Non-transient** exceptions are those that require the service endpoint address to be re-resolved. These include exceptions that indicate the service endpoint could not be reached, indicating the service has moved to a different node.

The `TryHandleException` makes a decision about a given exception. If it **does not know** what decisions to make about an exception, it should return **false**. If it **does know** what decision to make, it should set the result accordingly and return **true**.

```

class MyExceptionHandler : IExceptionHandler
{
    public bool TryHandleException(ExceptionInformation exceptionInformation, OperationRetrySettings
retrySettings, out ExceptionHandlingResult result)
    {
        // if exceptionInformation.Exception is known and is transient (can be retried without re-resolving)
        result = new ExceptionHandlingRetryResult(exceptionInformation.Exception, true, retrySettings,
retrySettings.DefaultMaxRetryCount);
        return true;

        // if exceptionInformation.Exception is known and is not transient (indicates a new service endpoint
address must be resolved)
        result = new ExceptionHandlingRetryResult(exceptionInformation.Exception, false, retrySettings,
retrySettings.DefaultMaxRetryCount);
        return true;

        // if exceptionInformation.Exception is unknown (let the next IExceptionHandler attempt to handle it)
        result = null;
        return false;
    }
}

```

```

public class MyExceptionHandler implements ExceptionHandler {

    @Override
    public ExceptionHandlingResult handleException(ExceptionInformation exceptionInformation,
OperationRetrySettings retrySettings) {

        /* if exceptionInformation.getException() is known and is transient (can be retried without re-
resolving)
         */
        result = new ExceptionHandlingRetryResult(exceptionInformation.getException(), true, retrySettings,
retrySettings.getDefaultMaxRetryCount());
        return true;

        /* if exceptionInformation.getException() is known and is not transient (indicates a new service
endpoint address must be resolved)
         */
        result = new ExceptionHandlingRetryResult(exceptionInformation.getException(), false, retrySettings,
retrySettings.getDefaultMaxRetryCount());
        return true;

        /* if exceptionInformation.getException() is unknown (let the next ExceptionHandler attempt to handle
it)
         */
        result = null;
        return false;
    }
}

```

## Putting it all together

With an `ICommunicationClient(C#) / CommunicationClient(Java)`,  
`ICommunicationClientFactory(C#) / CommunicationClientFactory(Java)`, and  
`IExceptionHandler(C#) / ExceptionHandler(Java)` built around a communication protocol, a  
`ServicePartitionClient(C#) / FabricServicePartitionClient(Java)` wraps it all together and provides the fault-  
handling and service partition address resolution loop around these components.

```

private MyCommunicationClientFactory myCommunicationClientFactory;
private Uri myServiceUri;

var myServicePartitionClient = new ServicePartitionClient<MyCommunicationClient>(
    this.myCommunicationClientFactory,
    this.myServiceUri,
    myPartitionKey);

var result = await myServicePartitionClient.InvokeWithRetryAsync(async (client) =>
{
    // Communicate with the service using the client.
},
CancellationToken.None);

```

```

private MyCommunicationClientFactory myCommunicationClientFactory;
private URI myServiceUri;

FabricServicePartitionClient myServicePartitionClient = new
FabricServicePartitionClient<MyCommunicationClient>(
    this.myCommunicationClientFactory,
    this.myServiceUri,
    myPartitionKey);

CompletableFuture<?> result = myServicePartitionClient.invokeWithRetryAsync(client -> {
    /* Communicate with the service using the client.
     */
});

```

## Next steps

- See an example of HTTP communication between services in a [C# sample project on GitHub](#) or [Java sample project on GitHub](#).
- [Remote procedure calls with Reliable Services remoting](#)
- [Web API that uses OWIN in Reliable Services](#)
- [WCF communication by using Reliable Services](#)

# Service Remoting with Reliable Services

9/27/2017 • 7 min to read • [Edit Online](#)

For services that are not tied to a particular communication protocol or stack, such as WebAPI, Windows Communication Foundation (WCF), or others, the Reliable Services framework provides a remoting mechanism to quickly and easily set up remote procedure call for services.

## Set up Remoting on a Service

Setting up remoting for a service is done in two simple steps:

1. Create an interface for your service to implement. This interface defines the methods that are available for a remote procedure call on your service. The methods must be task-returning asynchronous methods. The interface must implement `Microsoft.ServiceFabric.Services.Remoting.IService` to signal that the service has a remoting interface.
2. Use a remoting listener in your service. `RemotingListener` is an `ICommunicationListener` implementation that provides remoting capabilities. The `Microsoft.ServiceFabric.Services.Remoting.Runtime` namespace contains an extension method, `CreateServiceRemotingListener` for both stateless and stateful services that can be used to create a remoting listener using the default remoting transport protocol.

### NOTE

The `Remoting` namespace is available as a separate NuGet package called  
`Microsoft.ServiceFabric.Services.Remoting`

For example, the following stateless service exposes a single method to get "Hello World" over a remote procedure call.

```

using Microsoft.ServiceFabric.Services.Communication.Runtime;
using Microsoft.ServiceFabric.Services.Remoting;
using Microsoft.ServiceFabric.Services.Remoting.Runtime;
using Microsoft.ServiceFabric.Services.Runtime;

public interface IMyService : IService
{
    Task<string> HelloWorldAsync();
}

class MyService : StatelessService, IMyService
{
    public MyService(StatelessServiceContext context)
        : base (context)
    {
    }

    public Task HelloWorldAsync()
    {
        return Task.FromResult("Hello!");
    }

    protected override IEnumerable<ServiceInstanceListener> CreateServiceInstanceListeners()
    {
        return new[] { new ServiceInstanceListener(context =>
this.CreateServiceRemotingListener(context)) };
    }
}

```

#### **NOTE**

The arguments and the return types in the service interface can be any simple, complex, or custom types, but they must be serializable by the .NET [DataContractSerializer](#).

## Call Remote Service Methods

Calling methods on a service by using the remoting stack is done by using a local proxy to the service through the `Microsoft.ServiceFabric.Services.Remoting.Client.ServiceProxy` class. The `ServiceProxy` method creates a local proxy by using the same interface that the service implements. With that proxy, you can call methods on the interface remotely.

```

IMyService helloWorldClient = ServiceProxy.Create<IMyService>(new
Uri("fabric:/MyApplication/MyHelloWorldService"));

string message = await helloWorldClient.HelloWorldAsync();

```

The remoting framework propagates exceptions thrown at the service to the client. So exception-handling logic at the client by using `ServiceProxy` can directly handle exceptions that the service throws.

## Service Proxy Lifetime

ServiceProxy creation is a lightweight operation, so users can create as many as they need it. Service Proxy can be reused as long as users need it. If Remote Api throws Exception, users can still reuse the same proxy. Each ServiceProxy contains communication client used to send messages over the wire. While invoking API, we have internal check to see if communication client used is valid. Based on that result, we re-create the communication client. Hence if Exception occurs, users do not need to recreate serviceproxy.

## ServiceProxyFactory Lifetime

ServiceProxyFactory is a factory that creates proxy for different remoting interfaces. If you use API ServiceProxy.Create for creating proxy, then framework creates the singleton ServiceProxyFactory. It is useful to create one manually when you need to override [IServiceRemotingClientFactory](#) properties. Factory is an expensive operation. ServiceProxyFactory maintains cache of communication client. Best practice is to cache ServiceProxyFactory for as long as possible.

## Remoting Exception Handling

All the remote exception thrown by service API, are sent back to the client as AggregateException. RemoteExceptions should be DataContract Serializable otherwise [ServiceException](#) is thrown to the proxy API with the serialization error in it.

ServiceProxy does handle all Failover Exception for the service partition it is created for. It re-resolves the endpoints if there is Failover Exceptions(Non-Transient Exceptions) and retries the call with the correct endpoint. Number of retries for failover Exception is indefinite. If Transient Exceptions occurs, proxy retries the call.

Default retry parameters are provied by [OperationRetrySettings]. (<https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicefabric.services.communication.client.operationretrysettings>) User can configure these values by passing OperationRetrySettings object to ServiceProxyFactory constructor.

## How to use Remoting V2 stack

With 2.8 NuGet Remoting package, you have the option to use Remoting V2 stack. Remoting V2 stack is more performant and provides features like custom serializable and more pluggable Api's. By default, if you don't do following changes, it continues to use Remoting V1 Stack. Remoting V2 is not compatible with V1(previous Remoting stack), so follow below article on how to upgrade from V1 to V2 without impacting service availability.

### Using Assembly Attribute to use V2 stack.

Here are the steps to follow to change to V2 Stack.

1. Add an Endpoint Resource with name as "ServiceEndpointV2" in the service manifest.

```
<Resources>
  <Endpoints>
    <Endpoint Name="ServiceEndpointV2" />
  </Endpoints>
</Resources>
```

2. Use Remoting Extension Method to create Remoting Listener.

```
protected override IEnumerable<ServiceInstanceListener> CreateServiceInstanceListeners()
{
    return this.CreateServiceRemotingInstanceListeners();
}
```

3. Add Assembly Attribute on Remoting Interfaces.

```
[assembly: FabricTransportServiceRemotingProvider(RemotingListener = RemotingListener.V2Listener,
RemotingClient = RemotingClient.V2Client)]
```

No Changes are required in Client Project. Build the Client assembly with interface assembly, to makes sure that above assembly attribute is being used.

### Using Explicit V2 Classes to create Listener/ ClientFactory

Here are the steps to follow.

1. Add an Endpoint Resource with name as "ServiceEndpointV2" in the service manifest.

```
<Resources>
<Endpoints>
    <Endpoint Name="ServiceEndpointV2" />
</Endpoints>
</Resources>
```

2. Use [Remoting V2Listener](#). Default Service Endpoint Resource name used is "ServiceEndpointV2" and must be defined in Service Manifest.

```
protected override IEnumerable<ServiceInstanceListener> CreateServiceInstanceListeners()
{
    return new[]
    {
        new ServiceInstanceListener((c) =>
        {
            return new FabricTransportServiceRemotingListener(c, this);
        })
    };
}
```

3. Use V2 [Client Factory](#).

```
var proxyFactory = new ServiceProxyFactory((c) =>
{
    return new FabricTransportServiceRemotingClientFactory();
});
```

## How to upgrade from Remoting V1 to Remoting V2.

In order to upgrade from V1 to V2, 2-step upgrades are required. Following steps to be executed in the sequence listed.

1. Upgrade the V1 Service to V2 Service by using CompactListener Attribute. This change makes sure that service is listening on V1 and V2 Listener.

- a) Add an Endpoint Resource with name as "ServiceEndpointV2" in the service manifest.

```
<Resources>
<Endpoints>
    <Endpoint Name="ServiceEndpointV2" />
</Endpoints>
</Resources>
```

- b) Use Following Extension Method to Create Remoting Listener.

```
protected override IEnumerable<ServiceInstanceListener> CreateServiceInstanceListeners()
{
    return this.CreateServiceRemotingInstanceListeners();
}
```

- c) Add Assembly Attribute on Remoting Interfaces to use CompatListener and V2 Client.

```
[assembly: FabricTransportServiceRemotingProvider(RemotingListener = RemotingListener.CompatListener,
RemotingClient = RemotingClient.V2Client)]
```

2. Upgrade the V1 Client to V2 Client by using V2 Client Attribute. This step makes sure Client is using V2 stack. No Change in Client Project/Service is required. Building Client projects with updated interface assembly is sufficient.
3. This step is optional. Use V2Listener Attribute and then Upgrade the V2 Service. This step makes sure that service is listening only on V2 Listener.

```
[assembly: FabricTransportServiceRemotingProvider(RemotingListener = RemotingListener.V2Listener,
RemotingClient = RemotingClient.V2Client)]
```

## How to use Custom Serialization with Remoting V2.

Following example uses Json Serialization with Remoting V2.

1. Implement IServiceRemotingMessageSerializationProvider interface to provide implementation for custom serialization. Here is the code-snippet on how implementation looks like.

```
public class ServiceRemotingJsonSerializationProvider : IServiceRemotingMessageSerializationProvider
{
    public IServiceRemotingRequestMessageBodySerializer CreateRequestMessageSerializer(Type
serviceInterfaceType,
        IEnumerable<Type> requestBodyTypes)
    {
        return new ServiceRemotingRequestJsonMessageBodySerializer(serviceInterfaceType,
requestBodyTypes);
    }

    public IServiceRemotingResponseMessageBodySerializer CreateResponseMessageSerializer(Type
serviceInterfaceType,
        IEnumerable<Type> responseBodyTypes)
    {
        return new ServiceRemotingResponseJsonMessageBodySerializer(serviceInterfaceType,
responseBodyTypes);
    }

    public IServiceRemotingMessageBodyFactory CreateMessageBodyFactory()
    {
        return new JsonMessageFactory();
    }
}

class JsonMessageFactory: IServiceRemotingMessageBodyFactory
{
    public IServiceRemotingRequestMessageBody CreateRequest(string interfaceName, string methodName,
        int numberOfParameters)
    {
        return new JsonRemotingRequestBody(new JObject());
    }

    public IServiceRemotingResponseMessageBody CreateResponse(string interfaceName, string methodName)
    {
        return new JsonRemotingResponseBody();
    }
}

class ServiceRemotingRequestJsonMessageBodySerializer: IServiceRemotingRequestMessageBodySerializer
{
    public ServiceRemotingRequestJsonMessageBodySerializer(Type serviceInterfaceType,
        IEnumerable<Type> parameterInfo)
```

```

    }

    public OutgoingMessageBody Serialize(IServiceRemotingRequestMessageBody
serviceRemotingRequestMessageBody)
{
    if (serviceRemotingRequestMessageBody == null)
    {
        return null;
    }

    var json = serviceRemotingRequestMessageBody.ToString();
    var bytes = Encoding.UTF8.GetBytes(json);
    var segment = new ArraySegment<byte>(bytes);
    var segments = new List<ArraySegment<byte>> {segment};
    return new OutgoingMessageBody(segments);
}

public IServiceRemotingRequestMessageBody Deserialize(IncomingMessageBody messageBody)
{
    using (var sr = new StreamReader(messageBody.GetReceivedBuffer()))

        using (JsonReader reader = new JsonTextReader(sr))
    {
        var serializer = new JsonSerializer();
        var ob = serializer.Deserialize< JObject >(reader);
        var ob2 = new JsonRemotingRequestBody(ob);
        return ob2;
    }
}
}

class ServiceRemotingResponseJsonMessageBodySerializer: IServiceRemotingResponseMessageBodySerializer
{

    public ServiceRemotingResponseJsonMessageBodySerializer(Type serviceInterfaceType,
        IEnumerable< Type > parameterInfo)
    {
    }

    public OutgoingMessageBody Serialize(IServiceRemotingResponseMessageBody responseMessageBody)
    {
        var json = JsonConvert.SerializeObject(responseMessageBody, new JsonSerializerSettings()
        {
            TypeNameHandling = TypeNameHandling.All
        });
        var bytes = Encoding.UTF8.GetBytes(json);
        var segment = new ArraySegment<byte>(bytes);
        var list = new List<ArraySegment<byte>> {segment};
        return new OutgoingMessageBody(list);
    }

    public IServiceRemotingResponseMessageBody Deserialize(IncomingMessageBody messageBody)
    {
        using (var sr = new StreamReader(messageBody.GetReceivedBuffer()))

            using (var reader = new JsonTextReader(sr))
        {
            var serializer = JsonSerializer.Create(new JsonSerializerSettings()
            {
                TypeNameHandling = TypeNameHandling.All
            });
            return serializer.Deserialize< JsonRemotingResponseBody >(reader);
        }
    }
}

internal class JsonRemotingResponseBody: IServiceRemotingResponseMessageBody
{
    public object Value
    {

```

```

    public object Value,
    public void Set(object response)
    {
        this.Value = response;
    }

    public object Get(Type paramType)
    {
        return this.Value;
    }
}

class JsonRemotingRequestBody: IServiceRemotingRequestMessageBody
{
    private readonly JObject jobject;

    public JsonRemotingRequestBody(JObject ob)
    {
        this.jobject = ob;
    }

    public void SetParameter(int position, string paramName, object parameter)
    {
        this.jobject.Add(paramName, JToken.FromObject(parameter));
    }

    public object GetParameter(int position, string paramName, Type paramType)
    {
        var ob = this.jobject[paramName];
        return ob.ToObject(paramType);
    }

    public override string ToString()
    {
        return this.jobject.ToString();
    }
}

```

## 2. Override Default Serialization Provider with JsonSerializerProvider for Remoting Listener.

```

protected override IEnumerable<ServiceInstanceListener> CreateServiceInstanceListeners()
{
    return new[]
    {
        new ServiceInstanceListener((c) =>
        {
            return new FabricTransportServiceRemotingListener(c, this,
                new ServiceRemotingJsonSerializationProvider());
        })
    };
}

```

## 3. Override Default Serialization Provider with JsonSerializerProvider for Remoting Client Factory.

```

var proxyFactory = new ServiceProxyFactory((c) =>
{
    return new FabricTransportServiceRemotingClientFactory(
        serializationProvider: new ServiceRemotingJsonSerializationProvider());
});

```

## Next steps

- [Web API with OWIN in Reliable Services](#)
- [WCF communication with Reliable Services](#)
- [Securing communication for Reliable Services](#)

# Service remoting with Reliable Services

6/30/2017 • 2 min to read • [Edit Online](#)

The Reliable Services framework provides a remoting mechanism to quickly and easily set up remote procedure call for services.

## Set up remoting on a service

Setting up remoting for a service is done in two simple steps:

1. Create an interface for your service to implement. This interface defines the methods that are available for a remote procedure call on your service. The methods must be task-returning asynchronous methods. The interface must implement `microsoft.serviceFabric.services.remoting.Service` to signal that the service has a remoting interface.
2. Use a remoting listener in your service. This is an `CommunicationListener` implementation that provides remoting capabilities. `FabricTransportServiceRemotingListener` can be used to create a remoting listener using the default remoting transport protocol.

For example, the following stateless service exposes a single method to get "Hello World" over a remote procedure call.

```
import java.util.ArrayList;
import java.util.concurrent.CompletableFuture;
import java.util.List;
import microsoft.servicefabric.services.communication.runtime.ServiceInstanceListener;
import microsoft.servicefabric.services.remoting.Service;
import microsoft.servicefabric.services.runtime.StatelessService;

public interface MyService extends Service {
    CompletableFuture<String> helloWorldAsync();
}

class MyServiceImpl extends StatelessService implements MyService {
    public MyServiceImpl(StatelessServiceContext context) {
        super(context);
    }

    public CompletableFuture<String> helloWorldAsync() {
        return CompletableFuture.completedFuture("Hello!");
    }

    @Override
    protected List<ServiceInstanceListener> createServiceInstanceListeners() {
        ArrayList<ServiceInstanceListener> listeners = new ArrayList<>();
        listeners.add(new ServiceInstanceListener((context) -> {
            return new FabricTransportServiceRemotingListener(context,this);
        }));
        return listeners;
    }
}
```

### NOTE

The arguments and the return types in the service interface can be any simple, complex, or custom types, but they must be serializable.

## Call remote service methods

Calling methods on a service by using the remoting stack is done by using a local proxy to the service through the `microsoft.serviceFabric.services.remoting.client.ServiceProxyBase` class. The `ServiceProxyBase` method creates a local proxy by using the same interface that the service implements. With that proxy, you can simply call methods on the interface remotely.

```
MyService helloWorldClient = ServiceProxyBase.create(MyService.class, new
URI("fabric:/MyApplication/MyHelloWorldService"));

CompletableFuture<String> message = helloWorldClient.helloWorldAsync();
```

The remoting framework propagates exceptions thrown at the service to the client. So exception-handling logic at the client by using `ServiceProxyBase` can directly handle exceptions that the service throws.

## Service Proxy Lifetime

ServiceProxy creation is a lightweight operation, so user can create as many as they need it. Service Proxy can be re-used as long as user need it. User can re-use the same proxy in case of Exception. Each ServiceProxy contains communication client used to send messages over the wire. While invoking API, we have internal check to see if communication client used is valid. Based on that result, we re-create the communication client. Hence user do not need to recreate serviceproxy in case of Exception.

### ServiceProxyFactory Lifetime

`FabricServiceProxyFactory` is a factory that creates proxy for different remoting interfaces. If you use API `ServiceProxyBase.create` for creating proxy, then framework creates a `FabricServiceProxyFactory`. It is useful to create one manually when you need to override `ServiceRemotingClientFactory` properties. Factory is an expensive operation. `FabricServiceProxyFactory` maintains cache of communication clients. Best practice is to cache `FabricServiceProxyFactory` for as long as possible.

## Remoting Exception Handling

All the remote exception thrown by service API, are sent back to the client either as `RuntimeException` or `FabricException`.

ServiceProxy does handle all Failover Exception for the service partition it is created for. It re-resolves the endpoints if there is Failover Exceptions(Non-Transient Exceptions) and retries the call with the correct endpoint. Number of retries for failover Exception is indefinite. In case of TransientExceptions, it only retries the call.

Default retry parameters are provied by [OperationRetrySettings]. ([https://docs.microsoft.com/en-us/java/api/microsoft.servicefabric.services.communication.client\\_operation\\_retry\\_settings](https://docs.microsoft.com/en-us/java/api/microsoft.servicefabric.services.communication.client_operation_retry_settings)) User can configure these values by passing `OperationRetrySettings` object to `ServiceProxyFactory` constructor.

## Next steps

- [Securing communication for Reliable Services](#)

# WCF-based communication stack for Reliable Services

6/27/2017 • 2 min to read • [Edit Online](#)

The Reliable Services framework allows service authors to choose the communication stack that they want to use for their service. They can plug in the communication stack of their choice via the **ICommunicationListener** returned from the [CreateServiceReplicaListeners](#) or [CreateServiceInstanceListeners](#) methods. The framework provides an implementation of the communication stack based on the Windows Communication Foundation (WCF) for service authors who want to use WCF-based communication.

## WCF Communication Listener

The WCF-specific implementation of **ICommunicationListener** is provided by the **Microsoft.ServiceFabric.Services.Communication.Wcf.Runtime.WcfCommunicationListener** class.

Let's say we have a service contract of type `ICalculator`

```
[ServiceContract]
public interface ICalculator
{
    [OperationContract]
    Task<int> Add(int value1, int value2);
}
```

We can create a WCF communication listener in the service the following manner.

```
protected override IEnumerable<ServiceReplicaListener> CreateServiceReplicaListeners()
{
    return new[] { new ServiceReplicaListener((context) =>
        new WcfCommunicationListener<ICalculator>(
            wcfServiceObject:this,
            serviceContext:context,
            //
            // The name of the endpoint configured in the ServiceManifest under the Endpoints section
            // that identifies the endpoint that the WCF ServiceHost should listen on.
            //
            endpointResourceName: "WcfServiceEndpoint",

            //
            // Populate the binding information that you want the service to use.
            //
            listenerBinding: WcfUtility.CreateTcpListenerBinding()
        )
    );
}
```

## Writing clients for the WCF communication stack

For writing clients to communicate with services by using WCF, the framework provides **WcfClientCommunicationFactory**, which is the WCF-specific implementation of [ClientCommunicationFactoryBase](#).

```

public WcfCommunicationClientFactory(
    Binding clientBinding = null,
    IEnumerable<IExceptionHandler> exceptionHandlers = null,
    IServicePartitionResolver servicePartitionResolver = null,
    string traceId = null,
    object callback = null);

```

The WCF communication channel can be accessed from the **WcfCommunicationClient** created by the **WcfCommunicationClientFactory**.

```

public class WcfCommunicationClient : ServicePartitionClient<WcfCommunicationClient<ICalculator>>
{
    public WcfCommunicationClient(ICommunicationClientFactory<WcfCommunicationClient<ICalculator>>
        communicationClientFactory, Uri serviceUri, ServicePartitionKey partitionKey = null, TargetReplicaSelector
        targetReplicaSelector = TargetReplicaSelector.Default, string listenerName = null, OperationRetrySettings
        retrySettings = null)
        : base(communicationClientFactory, serviceUri, partitionKey, targetReplicaSelector, listenerName,
        retrySettings)
    {
    }
}

```

Client code can use the **WcfCommunicationClientFactory** along with the **WcfCommunicationClient** which implements **ServicePartitionClient** to determine the service endpoint and communicate with the service.

```

// Create binding
Binding binding = WcfUtility.CreateTcpClientBinding();
// Create a partition resolver
IServicePartitionResolver partitionResolver = ServicePartitionResolver.GetDefault();
// create a WcfCommunicationClientFactory object.
var wcfClientFactory = new WcfCommunicationClientFactory<ICalculator>
    (clientBinding: binding, servicePartitionResolver: partitionResolver);

//
// Create a client for communicating with the ICalculator service that has been created with the
// Singleton partition scheme.
//
var calculatorServiceCommunicationClient = new WcfCommunicationClient(
    wcfClientFactory,
    ServiceUri,
    ServicePartitionKey.Singleton);

//
// Call the service to perform the operation.
//
var result = calculatorServiceCommunicationClient.InvokeWithRetryAsync(
    client => client.Channel.Add(2, 3)).Result;

```

#### NOTE

The default ServicePartitionResolver assumes that the client is running in same cluster as the service. If that is not the case, create a ServicePartitionResolver object and pass in the cluster connection endpoints.

## Next steps

- [Remote procedure call with Reliable Services remoting](#)
- [Web API with OWIN in Reliable Services](#)

- Securing communication for Reliable Services

# Help secure communication for services in Azure Service Fabric

6/27/2017 • 4 min to read • [Edit Online](#)

Security is one of the most important aspects of communication. The Reliable Services application framework provides a few prebuilt communication stacks and tools that you can use to improve security. This article talks about how to improve security when you're using service remoting and the Windows Communication Foundation (WCF) communication stack.

## Help secure a service when you're using service remoting

We are using an existing [example](#) that explains how to set up remoting for reliable services. To help secure a service when you're using service remoting, follow these steps:

1. Create an interface, `IHelloWorldStateful`, that defines the methods that will be available for a remote procedure call on your service. Your service will use `FabricTransportServiceRemotingListener`, which is declared in the `Microsoft.ServiceFabric.Services.Remoting.FabricTransport.Runtime` namespace. This is an `ICommunicationListener` implementation that provides remoting capabilities.

```
public interface IHelloWorldStateful : IService
{
    Task<string> GetHelloWorld();
}

internal class HelloWorldStateful : StatefulService, IHelloWorldStateful
{
    protected override IEnumerable<ServiceReplicaListener> CreateServiceReplicaListeners()
    {
        return new[]{
            new ServiceReplicaListener(
                (context) => new FabricTransportServiceRemotingListener(context, this));
        }
    }

    public Task<string> GetHelloWorld()
    {
        return Task.FromResult("Hello World!");
    }
}
```

2. Add listener settings and security credentials.

Make sure that the certificate that you want to use to help secure your service communication is installed on all the nodes in the cluster. There are two ways that you can provide listener settings and security credentials:

- a. Provide them directly in the service code:

```

protected override IEnumerable<ServiceReplicaListener> CreateServiceReplicaListeners()
{
    FabricTransportRemotingListenerSettings listenerSettings = new
FabricTransportRemotingListenerSettings
    {
        MaxMessageSize = 10000000,
        SecurityCredentials = GetSecurityCredentials()
    };
    return new[]
    {
        new ServiceReplicaListener(
            (context) => new
FabricTransportServiceRemotingListener(context, this, listenerSettings))
    };
}

private static SecurityCredentials GetSecurityCredentials()
{
    // Provide certificate details.
    var x509Credentials = new X509Credentials
    {
        FindType = X509FindType.FindByThumbprint,
        FindValue = "4FEF3950642138446CC364A396E1E881DB76B48C",
        StoreLocation = StoreLocation.LocalMachine,
        StoreName = "My",
        ProtectionLevel = ProtectionLevel.EncryptAndSign
    };
    x509Credentials.RemoteCommonNames.Add("ServiceFabric-Test-Cert");
    x509Credentials.RemoteCertThumbprints.Add("9FEF3950642138446CC364A396E1E881DB76B483");
    return x509Credentials;
}

```

b. Provide them by using a [config package](#):

Add a `TransportSettings` section in the settings.xml file.

```

<Section Name="HelloWorldStatefulTransportSettings">
    <Parameter Name="MaxMessageSize" Value="10000000" />
    <Parameter Name="SecurityCredentialsType" Value="X509" />
    <Parameter Name="CertificateFindType" Value="FindByThumbprint" />
    <Parameter Name="CertificateFindValue" Value="4FEF3950642138446CC364A396E1E881DB76B48C" />
    <Parameter Name="CertificateRemoteThumbprints"
Value="9FEF3950642138446CC364A396E1E881DB76B483" />
    <Parameter Name="CertificateStoreLocation" Value="LocalMachine" />
    <Parameter Name="CertificateStoreName" Value="My" />
    <Parameter Name="CertificateProtectionLevel" Value="EncryptAndSign" />
    <Parameter Name="CertificateRemoteCommonNames" Value="ServiceFabric-Test-Cert" />
</Section>

```

In this case, the `CreateServiceReplicaListeners` method will look like this:

```

protected override IEnumerable<ServiceReplicaListener> CreateServiceReplicaListeners()
{
    return new[]
    {
        new ServiceReplicaListener(
            (context) => new FabricTransportServiceRemotingListener(
                context, this, FabricTransportRemotingListenerSettings
                .LoadFrom("HelloWorldStatefulTransportSettings")))
    };
}

```

If you add a `TransportSettings` section in the settings.xml file,

`FabricTransportRemotingListenerSettings` will load all the settings from this section by default.

```
<!--"TransportSettings" section .-->
<Section Name="TransportSettings">
    ...
</Section>
```

In this case, the `CreateServiceReplicaListeners` method will look like this:

```
protected override IEnumerable<ServiceReplicaListener> CreateServiceReplicaListeners()
{
    return new[]
    {
        return new[]{
            new ServiceReplicaListener(
                (context) => new FabricTransportServiceRemotingListener(context, this));
        };
    }
}
```

3. When you call methods on a secured service by using the remoting stack, instead of using the

`Microsoft.ServiceFabric.Services.Remoting.Client.ServiceProxy` class to create a service proxy, use

`Microsoft.ServiceFabric.Services.Remoting.Client.ServiceProxyFactory`. Pass in

`FabricTransportRemotingSettings`, which contains `SecurityCredentials`.

```
var x509Credentials = new X509Credentials
{
    FindType = X509FindType.FindByThumbprint,
    FindValue = "9FEF3950642138446CC364A396E1E881DB76B483",
    StoreLocation = StoreLocation.LocalMachine,
    StoreName = "My",
    ProtectionLevel = ProtectionLevel.EncryptAndSign
};
x509Credentials.RemoteCommonNames.Add("ServiceFabric-Test-Cert");
x509Credentials.RemoteCertThumbprints.Add("4FEF3950642138446CC364A396E1E881DB76B48C");

FabricTransportRemotingSettings transportSettings = new FabricTransportRemotingSettings
{
    SecurityCredentials = x509Credentials,
};

ServiceProxyFactory serviceProxyFactory = new ServiceProxyFactory(
    (c) => new FabricTransportServiceRemotingClientFactory(transportSettings));

IHelloWorldStateful client = serviceProxyFactory.CreateServiceProxy<IHelloWorldStateful>(
    new Uri("fabric:/MyApplication/MyHelloWorldService"));

string message = await client.GetHelloWorld();
```

If the client code is running as part of a service, you can load `FabricTransportRemotingSettings` from the `settings.xml` file. Create a `HelloWorldClientTransportSettings` section that is similar to the service code, as shown earlier. Make the following changes to the client code:

```
ServiceProxyFactory serviceProxyFactory = new ServiceProxyFactory()
    (c) => new
FabricTransportServiceRemotingClientFactory(FabricTransportRemotingSettings.LoadFrom("HelloWorldClientT
ransportSettings")));
IHelloWorldStateful client = serviceProxyFactory.CreateServiceProxy<IHelloWorldStateful>(
    new Uri("fabric:/MyApplication/MyHelloWorldService"));

string message = await client.GetHelloWorld();
```

If the client is not running as part of a service, you can create a `client_name.settings.xml` file in the same location where the `client_name.exe` is. Then create a `TransportSettings` section in that file.

Similar to the service, if you add a `TransportSettings` section in `client settings.xml/client_name.settings.xml`, `FabricTransportRemotingSettings` loads all the settings from this section by default.

In that case, the earlier code is even further simplified:

```
IHelloWorldStateful client = ServiceProxy.Create<IHelloWorldStateful>(
    new Uri("fabric:/MyApplication/MyHelloWorldService"));

string message = await client.GetHelloWorld();
```

## Help secure a service when you're using a WCF-based communication stack

We are using an existing [example](#) that explains how to set up a WCF-based communication stack for reliable services. To help secure a service when you're using a WCF-based communication stack, follow these steps:

1. For the service, you need to help secure the WCF communication listener (`WcfCommunicationListener`) that you create. To do this, modify the `CreateServiceReplicaListeners` method.

```

protected override IEnumerable<ServiceReplicaListener> CreateServiceReplicaListeners()
{
    return new[]
    {
        new ServiceReplicaListener(
            this.CreateWcfCommunicationListener())
    };
}

private WcfCommunicationListener<ICalculator> CreateWcfCommunicationListener(StatefulServiceContext
context)
{
    var wcfCommunicationListener = new WcfCommunicationListener<ICalculator>(
        serviceContext:context,
        wcfServiceObject:this,
        // For this example, we will be using NetTcpBinding.
        listenerBinding: GetNetTcpBinding(),
        endpointResourceName:"WcfServiceEndpoint");

    // Add certificate details in the ServiceHost credentials.
    wcfCommunicationListener.ServiceHost.Credentials.ServiceCertificate.SetCertificate(
        StoreLocation.LocalMachine,
        StoreName.My,
        X509FindType.FindByThumbprint,
        "9DC906B169DC4FAFFD1697AC781E806790749D2F");
    return wcfCommunicationListener;
}

private static NetTcpBinding GetNetTcpBinding()
{
    NetTcpBinding b = new NetTcpBinding(SecurityMode.TransportWithMessageCredential);
    b.Security.Message.ClientCredentialType = MessageCredentialType.Certificate;
    return b;
}

```

2. In the client, the `WcfCommunicationClient` class that was created in the previous [example](#) remains unchanged. But you need to override the `CreateClientAsync` method of `WcfCommunicationClientFactory`:

```

public class SecureWcfCommunicationClientFactory<TServiceContract> :
    WcfCommunicationClientFactory<TServiceContract> where TServiceContract : class
{
    private readonly Binding clientBinding;
    private readonly object callbackObject;
    public SecureWcfCommunicationClientFactory(
        Binding clientBinding,
        IEnumerable<IExceptionHandler> exceptionHandlers = null,
        IServicePartitionResolver servicePartitionResolver = null,
        string traceId = null,
        object callback = null)
        : base(clientBinding, exceptionHandlers, servicePartitionResolver, traceId, callback)
    {
        this.clientBinding = clientBinding;
        this.callbackObject = callback;
    }

    protected override Task<WcfCommunicationClient<TServiceContract>> CreateClientAsync(string
endpoint, CancellationToken cancellationToken)
    {
        var endpointAddress = new EndpointAddress(new Uri(endpoint));
        ChannelFactory<TServiceContract> channelFactory;
        if (this.callbackObject != null)
        {
            channelFactory = new DuplexChannelFactory<TServiceContract>(
                this.callbackObject,
                this.clientBinding,
                endpointAddress);
        }
        else
        {
            channelFactory = new ChannelFactory<TServiceContract>(this.clientBinding, endpointAddress);
        }
        // Add certificate details to the ChannelFactory credentials.
        // These credentials will be used by the clients created by
        // SecureWcfCommunicationClientFactory.
        channelFactory.Credentials.ClientCertificate.SetCertificate(
            StoreLocation.LocalMachine,
            StoreName.My,
            X509FindType.FindByThumbprint,
            "9DC906B169DC4FAFFD1697AC781E806790749D2F");
        var channel = channelFactory.CreateChannel();
        var clientChannel = ((IClientChannel)channel);
        clientChannel.OperationTimeout = this.clientBinding.ReceiveTimeout;
        return Task.FromResult(this.CreateWcfCommunicationClient(channel));
    }
}

```

Use `SecureWcfCommunicationClientFactory` to create a WCF communication client (`WcfCommunicationClient`). Use the client to invoke service methods.

```

IServicePartitionResolver partitionResolver = ServicePartitionResolver.GetDefault();

var wcfClientFactory = new SecureWcfCommunicationClientFactory<ICalculator>(clientBinding:
    GetNetTcpBinding(), servicePartitionResolver: partitionResolver);

var calculatorServiceCommunicationClient = new WcfCommunicationClient(
    wcfClientFactory,
    ServiceUri,
    ServicePartitionKey.Singleton);

var result = calculatorServiceCommunicationClient.InvokeWithRetryAsync(
    client => client.Channel.Add(2, 3)).Result;

```

## Next steps

- [Web API with OWIN in Reliable Services](#)

# Help secure communication for services in Azure Service Fabric

6/30/2017 • 2 min to read • [Edit Online](#)

## Help secure a service when you're using service remoting

We'll be using an existing [example](#) that explains how to set up remoting for reliable services. To help secure a service when you're using service remoting, follow these steps:

1. Create an interface, `HelloWorldStateless`, that defines the methods that will be available for a remote procedure call on your service. Your service will use `FabricTransportServiceRemotingListener`, which is declared in the `microsoft.serviceFabric.services.remoting.fabricTransport.runtime` package. This is an `CommunicationListener` implementation that provides remoting capabilities.

```
public interface HelloWorldStateless extends Service {
    CompletableFuture<String> getHelloWorld();
}

class HelloWorldStatelessImpl extends StatelessService implements HelloWorldStateless {
    @Override
    protected List<ServiceInstanceListener> createServiceInstanceListeners() {
        ArrayList<ServiceInstanceListener> listeners = new ArrayList<>();
        listeners.add(new ServiceInstanceListener((context) -> {
            return new FabricTransportServiceRemotingListener(context, this);
        }));
        return listeners;
    }

    public CompletableFuture<String> getHelloWorld() {
        return CompletableFuture.completedFuture("Hello World!");
    }
}
```

2. Add listener settings and security credentials.

Make sure that the certificate that you want to use to help secure your service communication is installed on all the nodes in the cluster. There are two ways that you can provide listener settings and security credentials:

- a. Provide them by using a [config package](#):

Add a `TransportSettings` section in the `settings.xml` file.

```
<!--Section name should always end with "TransportSettings".-->
<!--Here we are using a prefix "HelloWorldStateless".-->
<Section Name="HelloWorldStatelessTransportSettings">
    <Parameter Name="MaxMessageSize" Value="10000000" />
    <Parameter Name="SecurityCredentialsType" Value="X509_2" />
    <Parameter Name="CertificatePath"
Value="/path/to/cert/BD1C71E248B8C6834C151174DECDBDC02DE1D954.crt" />
    <Parameter Name="CertificateProtectionLevel" Value="EncryptandSign" />
    <Parameter Name="CertificateRemoteThumbprints"
Value="BD1C71E248B8C6834C151174DECDBDC02DE1D954" />
</Section>
```

In this case, the `createServiceInstanceListeners` method will look like this:

```
protected List<ServiceInstanceListener> createServiceInstanceListeners() {  
    ArrayList<ServiceInstanceListener> listeners = new ArrayList<>();  
    listeners.add(new ServiceInstanceListener((context) -> {  
        return new FabricTransportServiceRemotingListener(context, this,  
FabricTransportRemotingListenerSettings.loadFrom(HelloWorldStatelessTransportSettings));  
    }));  
    return listeners;  
}
```

If you add a `TransportSettings` section in the settings.xml file without any prefix, `FabricTransportListenerSettings` will load all the settings from this section by default.

```
<!--"TransportSettings" section without any prefix.-->  
<Section Name="TransportSettings">  
    ...  
</Section>
```

In this case, the `CreateServiceInstanceListeners` method will look like this:

```
protected List<ServiceInstanceListener> createServiceInstanceListeners() {  
    ArrayList<ServiceInstanceListener> listeners = new ArrayList<>();  
    listeners.add(new ServiceInstanceListener((context) -> {  
        return new FabricTransportServiceRemotingListener(context, this);  
    }));  
    return listeners;  
}
```

3. When you call methods on a secured service by using the remoting stack, instead of using the

`microsoft.serviceFabric.services.remoting.client.ServiceProxyBase` class to create a service proxy, use `microsoft.serviceFabric.services.remoting.client.FabricServiceProxyFactory`.

If the client code is running as part of a service, you can load `FabricTransportSettings` from the settings.xml file. Create a TransportSettings section that is similar to the service code, as shown earlier. Make the following changes to the client code:

```
FabricServiceProxyFactory serviceProxyFactory = new FabricServiceProxyFactory(c -> {  
    return new  
FabricTransportServiceRemotingClientFactory(FabricTransportRemotingSettings.loadFrom("TransportPrefixTra  
nsportSettings"), null, null, null, null);  
}, null)  
  
HelloWorldStateless client = serviceProxyFactory.createServiceProxy(HelloWorldStateless.class,  
new URI("fabric:/MyApplication/MyHelloWorldService"));  
  
CompletableFuture<String> message = client.getHelloWorld();
```

# Configure stateful reliable services

10/2/2017 • 9 min to read • [Edit Online](#)

There are two sets of configuration settings for reliable services. One set is global for all reliable services in the cluster while the other set is specific to a particular reliable service.

## Global Configuration

The global reliable service configuration is specified in the cluster manifest for the cluster under the KtlLogger section. It allows configuration of the shared log location and size plus the global memory limits used by the logger. The cluster manifest is a single XML file that holds settings and configurations that apply to all nodes and services in the cluster. The file is typically called ClusterManifest.xml. You can see the cluster manifest for your cluster using the Get-ServiceFabricClusterManifest powershell command.

### Configuration names

NAME	UNIT	DEFAULT VALUE	REMARKS
WriteBufferMemoryPoolMinimumInKB	Kilobytes	8388608	Minimum number of KB to allocate in kernel mode for the logger write buffer memory pool. This memory pool is used for caching state information before writing to disk.
WriteBufferMemoryPoolMaximumInKB	Kilobytes	No Limit	Maximum size to which the logger write buffer memory pool can grow.
SharedLogId	GUID	""	Specifies a unique GUID to use for identifying the default shared log file used by all reliable services on all nodes in the cluster that do not specify the SharedLogId in their service specific configuration. If SharedLogId is specified, then SharedLogPath must also be specified.
SharedLogPath	Fully qualified path name	""	Specifies the fully qualified path where the shared log file used by all reliable services on all nodes in the cluster that do not specify the SharedLogPath in their service specific configuration. However, if SharedLogPath is specified, then SharedLogId must also be specified.

NAME	UNIT	DEFAULT VALUE	REMARKS
SharedLogSizeInMB	Megabytes	8192	Specifies the number of MB of disk space to statically allocate for the shared log. The value must be 2048 or larger.

In Azure ARM or on-premises JSON template, the example below shows how to change the the shared transaction log that gets created to back any reliable collections for stateful services.

```
"fabricSettings": [
    {
        "name": "KtlLogger",
        "parameters": [
            {
                "name": "SharedLogSizeInMB",
                "value": "4096"
            }
        ]
    }
]
```

### Sample local developer cluster manifest section

If you want to change this on your local development environment, you need to edit the local clustermanifest.xml file.

```
<Section Name="KtlLogger">
<Parameter Name="SharedLogSizeInMB" Value="4096"/>
<Parameter Name="WriteBufferMemoryPoolMinimumInKB" Value="8192" />
<Parameter Name="WriteBufferMemoryPoolMaximumInKB" Value="8192" />
<Parameter Name="SharedLogId" Value="{7668BB54-FE9C-48ed-81AC-FF89E60ED2EF}"/>
<Parameter Name="SharedLogPath" Value="f:\SharedLog.Log"/>
</Section>
```

### Remarks

The logger has a global pool of memory allocated from non paged kernel memory that is available to all reliable services on a node for caching state data before being written to the dedicated log associated with the reliable service replica. The pool size is controlled by the WriteBufferMemoryPoolMinimumInKB and WriteBufferMemoryPoolMaximumInKB settings. WriteBufferMemoryPoolMinimumInKB specifies both the initial size of this memory pool and the lowest size to which the memory pool may shrink. WriteBufferMemoryPoolMaximumInKB is the highest size to which the memory pool may grow. Each reliable service replica that is opened may increase the size of the memory pool by a system determined amount up to WriteBufferMemoryPoolMaximumInKB. If there is more demand for memory from the memory pool than is available, requests for memory will be delayed until memory is available. Therefore if the write buffer memory pool is too small for a particular configuration then performance may suffer.

The SharedLogId and SharedLogPath settings are always used together to define the GUID and location for the default shared log for all nodes in the cluster. The default shared log is used for all reliable services that do not specify the settings in the settings.xml for the specific service. For best performance, shared log files should be placed on disks that are used solely for the shared log file to reduce contention.

SharedLogSizeInMB specifies the amount of disk space to preallocate for the default shared log on all nodes. SharedLogId and SharedLogPath do not need to be specified in order for SharedLogSizeInMB to be specified.

## Service Specific Configuration

You can modify stateful Reliable Services' default configurations by using the configuration package (Config) or the service implementation (code).

- **Config** - Configuration via the config package is accomplished by changing the Settings.xml file that is generated in the Microsoft Visual Studio package root under the Config folder for each service in the application.
- **Code** - Configuration via code is accomplished by creating a ReliableStateManager using a ReliableStateManagerConfiguration object with the appropriate options set.

By default, the Azure Service Fabric runtime looks for predefined section names in the Settings.xml file and consumes the configuration values while creating the underlying runtime components.

#### **NOTE**

Do **not** delete the section names of the following configurations in the Settings.xml file that is generated in the Visual Studio solution unless you plan to configure your service via code. Renaming the config package or section names will require a code change when configuring the ReliableStateManager.

## **Replicator security configuration**

Replicator security configurations are used to secure the communication channel that is used during replication. This means that services will not be able to see each other's replication traffic, ensuring that the data that is made highly available is also secure. By default, an empty security configuration section prevents replication security.

#### **Default section name**

ReplicatorSecurityConfig

#### **NOTE**

To change this section name, override the replicatorSecuritySectionName parameter to the ReliableStateManagerConfiguration constructor when creating the ReliableStateManager for this service.

## **Replicator configuration**

Replicator configurations configure the replicator that is responsible for making the stateful Reliable Service's state highly reliable by replicating and persisting the state locally. The default configuration is generated by the Visual Studio template and should suffice. This section talks about additional configurations that are available to tune the replicator.

#### **Default section name**

ReplicatorConfig

#### **NOTE**

To change this section name, override the replicatorSettingsSectionName parameter to the ReliableStateManagerConfiguration constructor when creating the ReliableStateManager for this service.

## **Configuration names**

NAME	UNIT	DEFAULT VALUE	REMARKS

NAME	UNIT	DEFAULT VALUE	REMARKS
BatchAcknowledgementInterval	Seconds	0.015	Time period for which the replicator at the secondary waits after receiving an operation before sending back an acknowledgement to the primary. Any other acknowledgements to be sent for operations processed within this interval are sent as one response.
ReplicatorEndpoint	N/A	No default--required parameter	IP address and port that the primary/secondary replicator will use to communicate with other replicators in the replica set. This should reference a TCP resource endpoint in the service manifest. Refer to <a href="#">Service manifest resources</a> to read more about defining endpoint resources in a service manifest.
MaxPrimaryReplicationQueueSize	Number of operations	8192	Maximum number of operations in the primary queue. An operation is freed up after the primary replicator receives an acknowledgement from all the secondary replicators. This value must be greater than 64 and a power of 2.
MaxSecondaryReplicationQueueSize	Number of operations	16384	Maximum number of operations in the secondary queue. An operation is freed up after making its state highly available through persistence. This value must be greater than 64 and a power of 2.
CheckpointThresholdInMB	MB	50	Amount of log file space after which the state is checkpointed.
MaxRecordSizeInKB	KB	1024	Largest record size that the replicator may write in the log. This value must be a multiple of 4 and greater than 16.

<b>NAME</b>	<b>UNIT</b>	<b>DEFAULT VALUE</b>	<b>REMARKS</b>
MinLogSizeInMB	MB	0 (system determined)	Minimum size of the transactional log. The log will not be allowed to truncate to a size below this setting. 0 indicates that the replicator will determine the minimum log size. Increasing this value increases the possibility of doing partial copies and incremental backups since chances of relevant log records being truncated is lowered.
TruncationThresholdFactor	Factor	2	Determines at what size of the log, truncation will be triggered. Truncation threshold is determined by MinLogSizeInMB multiplied by TruncationThresholdFactor. TruncationThresholdFactor must be greater than 1. MinLogSizeInMB * TruncationThresholdFactor must be less than MaxStreamSizeInMB.
ThrottlingThresholdFactor	Factor	4	Determines at what size of the log, the replica will start being throttled. Throttling threshold (in MB) is determined by Max((MinLogSizeInMB * ThrottlingThresholdFactor), (CheckpointThresholdInMB * ThrottlingThresholdFactor)). Throttling threshold (in MB) must be greater than truncation threshold (in MB). Truncation threshold (in MB) must be less than MaxStreamSizeInMB.
MaxAccumulatedBackupLogSizeInMB	MB	800	Max accumulated size (in MB) of backup logs in a given backup log chain. An incremental backup requests will fail if the incremental backup would generate a backup log that would cause the accumulated backup logs since the relevant full backup to be larger than this size. In such cases, user is required to take a full backup.

NAME	UNIT	DEFAULT VALUE	REMARKS
SharedLogId	GUID	""	Specifies a unique GUID to use for identifying the shared log file used with this replica. Typically, services should not use this setting. However, if SharedLogId is specified, then SharedLogPath must also be specified.
SharedLogPath	Fully qualified path name	""	Specifies the fully qualified path where the shared log file for this replica will be created. Typically, services should not use this setting. However, if SharedLogPath is specified, then SharedLogId must also be specified.
SlowApiMonitoringDuration	Seconds	300	Sets the monitoring interval for managed API calls. Example: user provided backup callback function. After the interval has passed, a warning health report will be sent to the Health Manager.

### Sample configuration via code

```
class Program
{
    /// <summary>
    /// This is the entry point of the service host process.
    /// </summary>
    static void Main()
    {
        ServiceRuntime.RegisterServiceAsync("HelloWorldStatefulType",
            context => new HelloWorldStateful(context,
                new ReliableStateManager(context,
                    new ReliableStateManagerConfiguration(
                        new ReliableStateManagerReplicatorSettings()
                    {
                        RetryInterval = TimeSpan.FromSeconds(3)
                    }
                ))).GetAwaiter().GetResult();
    }
}
```

```
class MyStatefulService : StatefulService
{
    public MyStatefulService(StatefulServiceContext context, IReliableStateManagerReplica stateManager)
        : base(context, stateManager)
    { }
    ...
}
```

### Sample configuration file

```

<?xml version="1.0" encoding="utf-8"?>
<Settings xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://schemas.microsoft.com/2011/01/fabric">
  <Section Name="ReplicatorConfig">
    <Parameter Name="ReplicatorEndpoint" Value="ReplicatorEndpoint" />
    <Parameter Name="BatchAcknowledgementInterval" Value="0.05"/>
    <Parameter Name="CheckpointThresholdInMB" Value="512" />
  </Section>
  <Section Name="ReplicatorSecurityConfig">
    <Parameter Name="CredentialType" Value="X509" />
    <Parameter Name="FindType" Value="FindByThumbprint" />
    <Parameter Name="FindValue" Value="9d c9 06 b1 69 dc 4f af fd 16 97 ac 78 1e 80 67 90 74 9d 2f" />
    <Parameter Name="StoreLocation" Value="LocalMachine" />
    <Parameter Name="StoreName" Value="My" />
    <Parameter Name="ProtectionLevel" Value="EncryptAndSign" />
    <Parameter Name="AllowedCommonNames" Value="My-Test-SAN1-Alice,My-Test-SAN1-Bob" />
  </Section>
</Settings>

```

## Remarks

BatchAcknowledgementInterval controls replication latency. A value of '0' results in the lowest possible latency, at the cost of throughput (as more acknowledgement messages must be sent and processed, each containing fewer acknowledgements). The larger the value for BatchAcknowledgementInterval, the higher the overall replication throughput, at the cost of higher operation latency. This directly translates to the latency of transaction commits.

The value for CheckpointThresholdInMB controls the amount of disk space that the replicator can use to store state information in the replica's dedicated log file. Increasing this to a higher value than the default could result in faster reconfiguration times when a new replica is added to the set. This is due to the partial state transfer that takes place due to the availability of more history of operations in the log. This can potentially increase the recovery time of a replica after a crash.

The MaxRecordSizeInKB setting defines the maximum size of a record that can be written by the replicator into the log file. In most cases, the default 1024-KB record size is optimal. However, if the service is causing larger data items to be part of the state information, then this value might need to be increased. There is little benefit in making MaxRecordSizeInKB smaller than 1024, as smaller records use only the space needed for the smaller record. We expect that this value would need to be changed in only rare cases.

The SharedLogId and SharedLogPath settings are always used together to make a service use a separate shared log from the default shared log for the node. For best efficiency, as many services as possible should specify the same shared log. Shared log files should be placed on disks that are used solely for the shared log file to reduce head movement contention. We expect that this value would need to be changed in only rare cases.

## Next steps

- [Debug your Service Fabric application in Visual Studio](#)
- [Developer reference for Reliable Services](#)

# Reliable Services notifications

6/30/2017 • 6 min to read • [Edit Online](#)

Notifications allow clients to track the changes that are being made to an object that they're interested in. Two types of objects support notifications: *Reliable State Manager* and *Reliable Dictionary*.

Common reasons for using notifications are:

- Building materialized views, such as secondary indexes or aggregated filtered views of the replica's state. An example is a sorted index of all keys in Reliable Dictionary.
- Sending monitoring data, such as the number of users added in the last hour.

Notifications are fired as part of applying operations. Because of that, notifications should be handled as fast as possible, and synchronous events shouldn't include any expensive operations.

## Reliable State Manager notifications

Reliable State Manager provides notifications for the following events:

- Transaction
  - Commit
- State manager
  - Rebuild
  - Addition of a reliable state
  - Removal of a reliable state

Reliable State Manager tracks the current inflight transactions. The only change in transaction state that causes a notification to be fired is a transaction being committed.

Reliable State Manager maintains a collection of reliable states like Reliable Dictionary and Reliable Queue. Reliable State Manager fires notifications when this collection changes: a reliable state is added or removed, or the entire collection is rebuilt. The Reliable State Manager collection is rebuilt in three cases:

- Recovery: When a replica starts, it recovers its previous state from the disk. At the end of recovery, it uses **NotifyStateManagerChangedEventArgs** to fire an event that contains the set of recovered reliable states.
- Full copy: Before a replica can join the configuration set, it has to be built. Sometimes, this requires a full copy of Reliable State Manager's state from the primary replica to be applied to the idle secondary replica. Reliable State Manager on the secondary replica uses **NotifyStateManagerChangedEventArgs** to fire an event that contains the set of reliable states that it acquired from the primary replica.
- Restore: In disaster recovery scenarios, the replica's state can be restored from a backup via **RestoreAsync**. In such cases, Reliable State Manager on the primary replica uses **NotifyStateManagerChangedEventArgs** to fire an event that contains the set of reliable states that it restored from the backup.

To register for transaction notifications and/or state manager notifications, you need to register with the **TransactionChanged** or **StateManagerChanged** events on Reliable State Manager. A common place to register with these event handlers is the constructor of your stateful service. When you register on the constructor, you won't miss any notification that's caused by a change during the lifetime of **IReliableStateManager**.

```

public MyService(StatefulServiceContext context)
    : base(MyService.EndpointName, context, CreateReliableStateManager(context))
{
    this.StateManager.TransactionChanged += this.OnTransactionChangedHandler;
    this.StateManager.StateManagerChanged += this.OnStateManagerChangedHandler;
}

```

The **TransactionChanged** event handler uses **NotifyTransactionChangedEventArgs** to provide details about the event. It contains the action property (for example, **NotifyTransactionChangedAction.Commit**) that specifies the type of change. It also contains the transaction property that provides a reference to the transaction that changed.

#### **NOTE**

Today, **TransactionChanged** events are raised only if the transaction is committed. The action is then equal to **NotifyTransactionChangedAction.Commit**. But in the future, events might be raised for other types of transaction state changes. We recommend checking the action and processing the event only if it's one that you expect.

Following is an example **TransactionChanged** event handler.

```

private void OnTransactionChangedHandler(object sender, NotifyTransactionChangedEventArgs e)
{
    if (e.Action == NotifyTransactionChangedAction.Commit)
    {
        this.lastCommitLsn = e.Transaction.CommitSequenceNumber;
        this.lastTransactionId = e.Transaction.TransactionId;

        this.lastCommittedTransactionList.Add(e.Transaction.TransactionId);
    }
}

```

The **StateManagerChanged** event handler uses **NotifyStateManagerChangedEventArgs** to provide details about the event. **NotifyStateManagerChangedEventArgs** has two subclasses: **NotifyStateManagerRebuildEventArgs** and **NotifyStateManagerSingleEntityChangedEventArgs**. You use the action property in **NotifyStateManagerChangedEventArgs** to cast **NotifyStateManagerChangedEventArgs** to the correct subclass:

- **NotifyStateManagerChangedAction.Rebuild:** **NotifyStateManagerRebuildEventArgs**
- **NotifyStateManagerChangedAction.Add** and **NotifyStateManagerChangedAction.Remove:** **NotifyStateManagerSingleEntityChangedEventArgs**

Following is an example **StateManagerChanged** notification handler.

```

public void OnStateManagerChangedHandler(object sender, NotifyStateManagerChangedEventArgs e)
{
    if (e.Action == NotifyStateManagerChangedAction.Rebuild)
    {
        this.ProcessStateManagerRebuildNotification(e);

        return;
    }

    this.ProcessStateManagerSingleEntityNotification(e);
}

```

## Reliable Dictionary notifications

Reliable Dictionary provides notifications for the following events:

- Rebuild: Called when **ReliableDictionary** has recovered its state from a recovered or copied local state or backup.
- Clear: Called when the state of **ReliableDictionary** has been cleared through the **ClearAsync** method.
- Add: Called when an item has been added to **ReliableDictionary**.
- Update: Called when an item in **IReliableDictionary** has been updated.
- Remove: Called when an item in **IReliableDictionary** has been deleted.

To get Reliable Dictionary notifications, you need to register with the **DictionaryChanged** event handler on **IReliableDictionary**. A common place to register with these event handlers is in the **ReliableStateManager.StateManagerChanged** add notification. Registering when **IReliableDictionary** is added to **IReliableStateManager** ensures that you won't miss any notifications.

```
private void ProcessStateManagerSingleEntityNotification(NotifyStateManagerChangedEventArgs e)
{
    var operation = e as NotifyStateManagerSingleEntityChangedEventArgs;

    if (operation.Action == NotifyStateManagerChangedAction.Add)
    {
        if (operation.ReliableState is IReliableDictionary<TKey, TValue>)
        {
            var dictionary = (IReliableDictionary<TKey, TValue>)operation.ReliableState;
            dictionary.RebuildNotificationAsyncCallback = this.OnDictionaryRebuildNotificationHandlerAsync;
            dictionary.DictionaryChanged += this.OnDictionaryChangedHandler;
        }
    }
}
```

#### NOTE

**ProcessStateManagerSingleEntityNotification** is the sample method that the preceding **OnStateManagerChangedHandler** example calls.

The preceding code sets the **IReliableNotificationAsyncCallback** interface, along with **DictionaryChanged**. Because **NotifyDictionaryRebuildEventArgs** contains an **IEnumerable** interface--which needs to be enumerated asynchronously--rebuild notifications are fired through **RebuildNotificationAsyncCallback** instead of **OnDictionaryChangedHandler**.

```
public async Task OnDictionaryRebuildNotificationHandlerAsync(
    IReliableDictionary<TKey, TValue> origin,
    NotifyDictionaryRebuildEventArgs<TKey, TValue> rebuildNotification)
{
    this.secondaryIndex.Clear();

    var enumerator = e.State.GetAsyncEnumerator();
    while (await enumerator.MoveNextAsync(CancellationToken.None))
    {
        this.secondaryIndex.Add(enumerator.Current.Key, enumerator.Current.Value);
    }
}
```

#### NOTE

In the preceding code, as part of processing the rebuild notification, first the maintained aggregated state is cleared. Because the reliable collection is being rebuilt with a new state, all previous notifications are irrelevant.

The **DictionaryChanged** event handler uses **NotifyDictionaryChangedEventArgs** to provide details about the event. **NotifyDictionaryChangedEventArgs** has five subclasses. Use the action property in **NotifyDictionaryChangedEventArgs** to cast **NotifyDictionaryChangedEventArgs** to the correct subclass:

- **NotifyDictionaryChangedAction.Rebuild**: **NotifyDictionaryRebuildEventArgs**
- **NotifyDictionaryChangedAction.Clear**: **NotifyDictionaryClearEventArgs**
- **NotifyDictionaryChangedAction.Add** and **NotifyDictionaryChangedAction.Remove**: **NotifyDictionaryItemAddedEventArgs**
- **NotifyDictionaryChangedAction.Update**: **NotifyDictionaryItemUpdatedEventArgs**
- **NotifyDictionaryChangedAction.Remove**: **NotifyDictionaryItemRemovedEventArgs**

```
public void OnDictionaryChangedHandler(object sender, NotifyDictionaryChangedEventArgs<TKey, TValue> e)
{
    switch (e.Action)
    {
        case NotifyDictionaryChangedAction.Clear:
            var clearEvent = e as NotifyDictionaryClearEventArgs<TKey, TValue>;
            this.ProcessClearNotification(clearEvent);
            return;

        case NotifyDictionaryChangedAction.Add:
            var addEvent = e as NotifyDictionaryItemAddedEventArgs<TKey, TValue>;
            this.ProcessAddNotification(addEvent);
            return;

        case NotifyDictionaryChangedAction.Update:
            var updateEvent = e as NotifyDictionaryItemUpdatedEventArgs<TKey, TValue>;
            this.ProcessUpdateNotification(updateEvent);
            return;

        case NotifyDictionaryChangedAction.Remove:
            var deleteEvent = e as NotifyDictionaryItemRemovedEventArgs<TKey, TValue>;
            this.ProcessRemoveNotification(deleteEvent);
            return;

        default:
            break;
    }
}
```

## Recommendations

- Do complete notification events as fast as possible.
- Do *not* execute any expensive operations (for example, I/O operations) as part of synchronous events.
- Do check the action type before you process the event. New action types might be added in the future.

Here are some things to keep in mind:

- Notifications are fired as part of the execution of an operation. For example, a restore notification is fired as the last step of a restore operation. A restore will not finish until the notification event is processed.
- Because notifications are fired as part of the applying operations, clients see only notifications for locally committed operations. And because operations are guaranteed only to be locally committed (in other words,

logged), they might or might not be undone in the future.

- On the redo path, a single notification is fired for each applied operation. This means that if transaction T1 includes Create(X), Delete(X), and Create(X), you'll get one notification for the creation of X, one for the deletion, and one for the creation again, in that order.
- For transactions that contain multiple operations, operations are applied in the order in which they were received on the primary replica from the user.
- As part of processing false progress, some operations might be undone. Notifications are raised for such undo operations, rolling the state of the replica back to a stable point. One important difference of undo notifications is that events that have duplicate keys are aggregated. For example, if transaction T1 is being undone, you'll see a single notification to Delete(X).

## Next steps

- [Reliable Collections](#)
- [Reliable Services quick start](#)
- [Reliable Services backup and restore \(disaster recovery\)](#)
- [Developer reference for Reliable Collections](#)

# Back up and restore Reliable Services and Reliable Actors

8/18/2017 • 14 min to read • [Edit Online](#)

Azure Service Fabric is a high-availability platform that replicates the state across multiple nodes to maintain this high availability. Thus, even if one node in the cluster fails, the services continue to be available. While this in-built redundancy provided by the platform may be sufficient for some, in certain cases it is desirable for the service to back up data (to an external store).

## NOTE

It is critical to backup and restore your data (and test that it works as expected) so you can recover from data loss scenarios.

For example, a service may want to back up data in order to protect from the following scenarios:

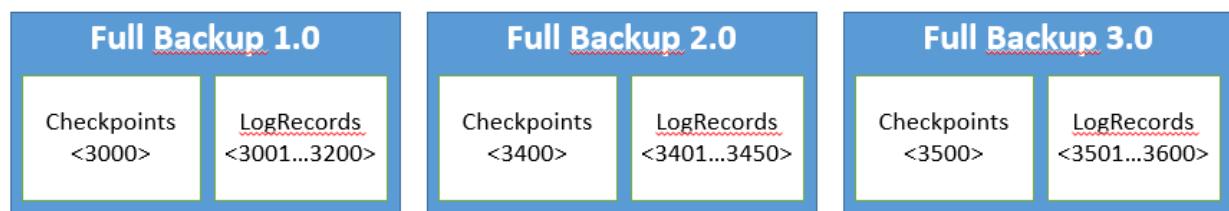
- In the event of the permanent loss of an entire Service Fabric cluster.
- Permanent loss of a majority of the replicas of a service partition
- Administrative errors whereby the state accidentally gets deleted or corrupted. For example, this may happen if an administrator with sufficient privilege erroneously deletes the service.
- Bugs in the service that cause data corruption. For example, this may happen when a service code upgrade starts writing faulty data to a Reliable Collection. In such a case, both the code and the data may have to be reverted to an earlier state.
- Offline data processing. It might be convenient to have offline processing of data for business intelligence that happens separately from the service that generates the data.

The Backup/Restore feature allows services built on the Reliable Services API to create and restore backups. The backup APIs provided by the platform allow backup(s) of a service partition's state, without blocking read or write operations. The restore APIs allow a service partition's state to be restored from a chosen backup.

## Types of Backup

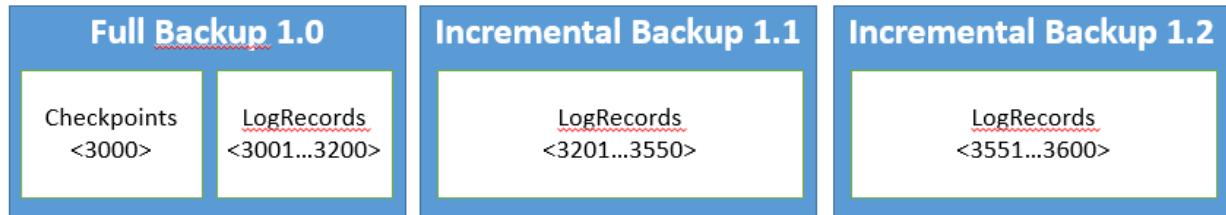
There are two backup options: Full and Incremental. A full backup is a backup that contains all the data required to recreate the state of the replica: checkpoints and all log records. Since it has the checkpoints and the log, a full backup can be restored by itself.

The problem with full backups arises when the checkpoints are large. For example, a replica that has 16 GB of state will have checkpoints that add up approximately to 16 GB. If we have a Recovery Point Objective of five minutes, the replica needs to be backed up every five minutes. Each time it backs up it needs to copy 16 GB of checkpoints in addition to 50 MB (configurable using `CheckpointThresholdInMB`) worth of logs.



The solution to this problem is incremental backups, where backup only contains the changed log records since

the last backup.



Since incremental backups are only changes since the last backup (does not include the checkpoints), they tend to be faster but they cannot be restored on their own. To restore an incremental backup, the entire backup chain is required. A backup chain is a chain of backups starting with a full backup and followed by a number of contiguous incremental backups.

## Backup Reliable Services

The service author has full control of when to make backups and where backups will be stored.

To start a backup, the service needs to invoke the inherited member function `BackupAsync`.

Backups can be made only from primary replicas, and they require write status to be granted.

As shown below, `BackupAsync` takes in a `BackupDescription` object, where one can specify a full or incremental backup, as well as a callback function, `Func<< BackupInfo, CancellationToken, Task<bool>>>` that is invoked when the backup folder has been created locally and is ready to be moved out to some external storage.

```
BackupDescription myBackupDescription = new  
BackupDescription(backupOption.Incremental,this.BackupCallbackAsync);  
  
await this.BackupAsync(myBackupDescription);
```

Request to take an incremental backup can fail with `FabricMissingFullBackupException`. This exception indicates that one of the following things is happening:

- the replica has never taken a full backup since it has become primary,
- some of the log records since the last backup has been truncated or
- replica passed the `MaxAccumulatedBackupLogSizeInMB` limit.

Users can increase the likelihood of being able to do incremental backups by configuring `MinLogSizeInMB` or `TruncationThresholdFactor`. Note that increasing these values increases the per replica disk usage. For more information, see [Reliable Services Configuration](#)

`BackupInfo` provides information regarding the backup, including the location of the folder where the runtime saved the backup (`BackupInfo.Directory`). The callback function can move the `BackupInfo.Directory` to an external store or another location. This function also returns a bool that indicates whether it was able to successfully move the backup folder to its target location.

The following code demonstrates how the `BackupCallbackAsync` method can be used to upload the backup to Azure Storage:

```

private async Task<bool> BackupCallbackAsync(BackupInfo backupInfo, CancellationToken cancellationToken)
{
    var backupId = Guid.NewGuid();

    await externalBackupStore.UploadBackupFolderAsync(backupInfo.Directory, backupId, cancellationToken);

    return true;
}

```

In the preceding example, `ExternalBackupStore` is the sample class that is used to interface with Azure Blob storage, and `UploadBackupFolderAsync` is the method that compresses the folder and places it in the Azure Blob store.

Note that:

- There can be only one backup operation in-flight per replica at any given time. More than one `BackupAsync` call at a time will throw `FabricBackupInProgressException` to limit inflight backups to one.
- If a replica fails over while a backup is in progress, the backup may not have been completed. Thus, once the failover finishes, it is the service's responsibility to restart the backup by invoking `BackupAsync` as necessary.

## Restore Reliable Services

In general, the cases when you might need to perform a restore operation fall into one of these categories:

- The service partition lost data. For example, the disk for two out of three replicas for a partition (including the primary replica) gets corrupted or wiped. The new primary may need to restore data from a backup.
- The entire service is lost. For example, an administrator removes the entire service and thus the service and the data need to be restored.
- The service replicated corrupt application data (e.g., because of an application bug). In this case, the service has to be upgraded or reverted to remove the cause of the corruption, and non-corrupt data has to be restored.

While many approaches are possible, we offer some examples on using `RestoreAsync` to recover from the above scenarios.

## Partition data loss in Reliable Services

In this case, the runtime would automatically detect the data loss and invoke the `OnDataLossAsync` API.

The service author needs to perform the following to recover:

- Override the virtual base class method `OnDataLossAsync`.
- Find the latest backup in the external location that contains the service's backups.
- Download the latest backup (and uncompress the backup into the backup folder if it was compressed).
- The `OnDataLossAsync` method provides a `RestoreContext`. Call the `RestoreAsync` API on the provided `RestoreContext`.
- Return true if the restoration was a success.

Following is an example implementation of the `OnDataLossAsync` method:

```

protected override async Task<bool> OnDataLossAsync(RestoreContext restoreCtx, CancellationToken cancellationToken)
{
    var backupFolder = await this.externalBackupStore.DownloadLastBackupAsync(cancellationToken);

    var restoreDescription = new RestoreDescription(backupFolder);

    await restoreCtx.RestoreAsync(restoreDescription);

    return true;
}

```

`RestoreDescription` passed in to the `RestoreContext.RestoreAsync` call contains a member called `BackupFolderPath`. When restoring a single full backup, this `BackupFolderPath` should be set to the local path of the folder that contains your full backup. When restoring a full backup and a number of incremental backups, `BackupFolderPath` should be set to the local path of the folder that not only contains the full backup, but also all the incremental backups. `RestoreAsync` call can throw `FabricMissingFullBackupException` if the `BackupFolderPath` provided does not contain a full backup. It can also throw `ArgumentException` if `BackupFolderPath` has a broken chain of incremental backups. For example, if it contains the full backup, the first incremental and the third incremental backup but no the second incremental backup.

#### NOTE

The `RestorePolicy` is set to `Safe` by default. This means that the `RestoreAsync` API will fail with `ArgumentException` if it detects that the backup folder contains a state that is older than or equal to the state contained in this replica. `RestorePolicy.Force` can be used to skip this safety check. This is specified as part of `RestoreDescription`.

## Deleted or lost service

If a service is removed, you must first re-create the service before the data can be restored. It is important to create the service with the same configuration, e.g., partitioning scheme, so that the data can be restored seamlessly. Once the service is up, the API to restore data (`OnDataLossAsync` above) has to be invoked on every partition of this service. One way of achieving this is by using `[FabricClient.TestManagementClient.StartPartitionDataLossAsync]` (<https://msdn.microsoft.com/library/mt693569.aspx>) on every partition.

From this point, implementation is the same as the above scenario. Each partition needs to restore the latest relevant backup from the external store. One caveat is that the partition ID may have now changed, since the runtime creates partition IDs dynamically. Thus, the service needs to store the appropriate partition information and service name to identify the correct latest backup to restore from for each partition.

#### NOTE

It is not recommended to use `FabricClient.ServiceManager.InvokeDataLossAsync` on each partition to restore the entire service, since that may corrupt your cluster state.

## Replication of corrupt application data

If the newly deployed application upgrade has a bug, that may cause corruption of data. For example, an application upgrade may start to update every phone number record in a Reliable Dictionary with an invalid area code. In this case, the invalid phone numbers will be replicated since Service Fabric is not aware of the nature of the data that is being stored.

The first thing to do after you detect such an egregious bug that causes data corruption is to freeze the service at the application level and, if possible, upgrade to the version of the application code that does not have the bug. However, even after the service code is fixed, the data may still be corrupt and thus data may need to be restored. In such cases, it may not be sufficient to restore the latest backup, since the latest backups may also be corrupt. Thus, you have to find the last backup that was made before the data got corrupted.

If you are not sure which backups are corrupt, you could deploy a new Service Fabric cluster and restore the backups of affected partitions just like the above "Deleted or lost service" scenario. For each partition, start restoring the backups from the most recent to the least. Once you find a backup that does not have the corruption, move/delete all backups of this partition that were more recent (than that backup). Repeat this process for each partition. Now, when `OnDataLossAsync` is called on the partition in the production cluster, the last backup found in the external store will be the one picked by the above process.

Now, the steps in the "Deleted or lost service" section can be used to restore the state of the service to the state before the buggy code corrupted the state.

Note that:

- When you restore, there is a chance that the backup being restored is older than the state of the partition before the data was lost. Because of this, you should restore only as a last resort to recover as much data as possible.
- The string that represents the backup folder path and the paths of files inside the backup folder can be greater than 255 characters, depending on the `FabricDataRoot` path and Application Type name's length. This can cause some .NET methods, like `Directory.Move`, to throw the `PathTooLongException` exception. One workaround is to directly call kernel32 APIs, like `CopyFile`.

## Backup and restore Reliable Actors

Reliable Actors Framework is built on top of Reliable Services. The `ActorService` which hosts the actor(s) is a stateful reliable service. Hence, all the backup and restore functionality available in Reliable Services is also available to Reliable Actors (except behaviors that are state provider specific). Since backups will be taken on a per-partition basis, states for all actors in that partition will be backed up (and restoration is similar and will happen on a per-partition basis). To perform backup/restore, the service owner should create a custom actor service class that derives from `ActorService` class and then do backup/restore similar to Reliable Services as described above in previous sections.

```
class MyCustomActorService : ActorService
{
    public MyCustomActorService(StatefulServiceContext context, ActorTypeInformation actorTypeInfo)
        : base(context, actorTypeInfo)
    {
    }

    //
    // Method overrides and other code.
    //
}
```

When you create a custom actor service class, you need to register that as well when registering the actor.

```
ActorRuntime.RegisterActorAsync<MyActor>(
    (context, typeInfo) => new MyCustomActorService(context, typeInfo)).GetAwaiter().GetResult();
```

The default state provider for Reliable Actors is `KvsActorStateProvider`. Incremental backup is not enabled by default for `KvsActorStateProvider`. You can enable incremental backup by creating `KvsActorStateProvider` with

the appropriate setting in its constructor and then passing it to ActorService constructor as shown in following code snippet:

```
class MyCustomActorService : ActorService
{
    public MyCustomActorService(StatefulServiceContext context, ActorTypeInformation actorTypeInfo)
        : base(context, actorTypeInfo, null, null, new KvsActorStateProvider(true)) // Enable
    incremental backup
    {
    }

    //
    // Method overrides and other code.
    //
}
```

After incremental backup has been enabled, taking an incremental backup can fail with FabricMissingFullBackupException for one of following reasons and you will need to take a full backup before taking incremental backup(s):

- The replica has never taken a full backup since it became primary.
- Some of the log records were truncated since last backup was taken.

When incremental backup is enabled, `KvsActorStateProvider` does not use circular buffer to manage its log records and periodically truncates it. If no backup is taken by user for a period of 45 minutes, the system automatically truncates the log records. This interval can be configured by specifying

`logTruncationIntervalInMinutes` in `KvsActorStateProvider` constructor (similar to when enabling incremental backup). The log records may also get truncated if primary replica need to build another replica by sending all its data.

When doing restore from a backup chain, similar to Reliable Services, the `BackupFolderPath` should contain subdirectories with one subdirectory containing full backup and others subdirectories containing incremental backup(s). The restore API will throw `FabricException` with appropriate error message if the backup chain validation fails.

**NOTE**

`KvsActorStateProvider` currently ignores the option `RestorePolicy.Safe`. Support for this feature is planned in an upcoming release.

## Testing Backup and Restore

It is important to ensure that critical data is being backed up, and can be restored from. This can be done by invoking the `Start-ServiceFabricPartitionDataLoss` cmdlet in PowerShell that can induce data loss in a particular partition to test whether the data backup and restore functionality for your service is working as expected. It is also possible to programmatically invoke data loss and restore from that event as well.

**NOTE**

You can find a sample implementation of backup and restore functionality in the Web Reference App on GitHub. Please look at the `Inventory.Service` service for more details.

## Under the hood: more details on backup and restore

Here's some more details on backup and restore.

## Backup

The Reliable State Manager provides the ability to create consistent backups without blocking any read or write operations. To do so, it utilizes a checkpoint and log persistence mechanism. The Reliable State Manager takes fuzzy (lightweight) checkpoints at certain points to relieve pressure from the transactional log and improve recovery times. When `BackupAsync` is called, the Reliable State Manager instructs all Reliable objects to copy their latest checkpoint files to a local backup folder. Then, the Reliable State Manager copies all log records, starting from the "start pointer" to the latest log record into the backup folder. Since all the log records up to the latest log record are included in the backup and the Reliable State Manager preserves write-ahead logging, the Reliable State Manager guarantees that all transactions that are committed (`CommitAsync` has returned successfully) are included in the backup.

Any transaction that commits after `BackupAsync` has been called may or may not be in the backup. Once the local backup folder has been populated by the platform (i.e., local backup is completed by the runtime), the service's backup callback is invoked. This callback is responsible for moving the backup folder to an external location such as Azure Storage.

## Restore

The Reliable State Manager provides the ability to restore from a backup by using the `RestoreAsync` API. The `RestoreAsync` method on `RestoreContext` can be called only inside the `OnDataLossAsync` method. The bool returned by `OnDataLossAsync` indicates whether the service restored its state from an external source. If the `OnDataLossAsync` returns true, Service Fabric will rebuild all other replicas from this primary. Service Fabric ensures that replicas that will receive `OnDataLossAsync` call first transition to the primary role but are not granted read status or write status. This implies that for StatefulService implementers, `RunAsync` will not be called until `OnDataLossAsync` finishes successfully. Then, `OnDataLossAsync` will be invoked on the new primary. Until a service completes this API successfully (by returning true or false) and finishes the relevant reconfiguration, the API will keep being called one at a time.

`RestoreAsync` first drops all existing state in the primary replica that it was called on. Then the Reliable State Manager creates all the Reliable objects that exist in the backup folder. Next, the Reliable objects are instructed to restore from their checkpoints in the backup folder. Finally, the Reliable State Manager recovers its own state from the log records in the backup folder and performs recovery. As part of the recovery process, operations starting from the "starting point" that have commit log records in the backup folder are replayed to the Reliable objects. This step ensures that the recovered state is consistent.

## Next steps

- [Reliable Collections](#)
- [Reliable Services quick start](#)
- [Reliable Services notifications](#)
- [Reliable Services configuration](#)
- [Developer reference for Reliable Collections](#)

# Getting started with Reliable Actors

6/30/2017 • 4 min to read • [Edit Online](#)

This article explains the basics of Azure Service Fabric Reliable Actors and walks you through creating, debugging, and deploying a simple Reliable Actor application in Visual Studio.

## Installation and setup

Before you start, ensure that you have the Service Fabric development environment set up on your machine. If you need to set it up, see detailed instructions on [how to set up the development environment](#).

## Basic concepts

To get started with Reliable Actors, you only need to understand a few basic concepts:

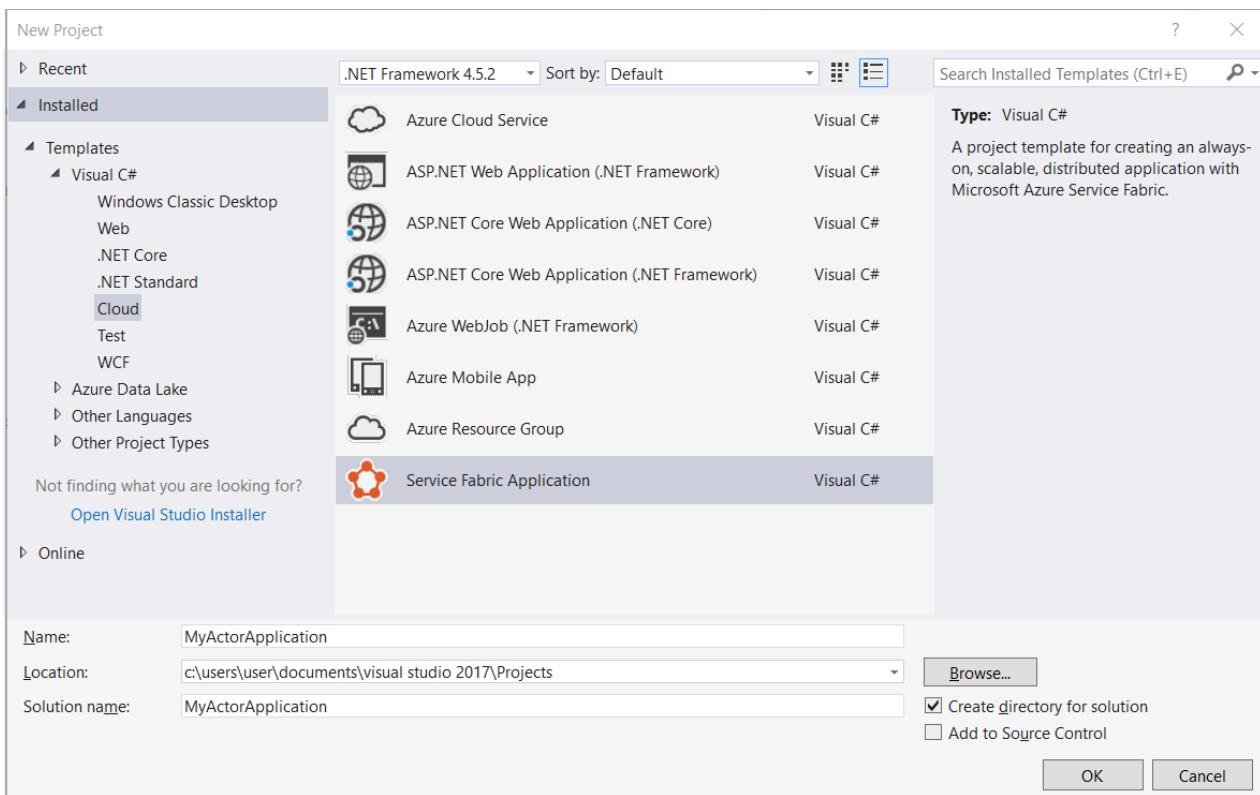
- **Actor service.** Reliable Actors are packaged in Reliable Services that can be deployed in the Service Fabric infrastructure. Actor instances are activated in a named service instance.
- **Actor registration.** As with Reliable Services, a Reliable Actor service needs to be registered with the Service Fabric runtime. In addition, the actor type needs to be registered with the Actor runtime.
- **Actor interface.** The actor interface is used to define a strongly typed public interface of an actor. In the Reliable Actor model terminology, the actor interface defines the types of messages that the actor can understand and process. The actor interface is used by other actors and client applications to "send" (asynchronously) messages to the actor. Reliable Actors can implement multiple interfaces.
- **ActorProxy class.** The ActorProxy class is used by client applications to invoke the methods exposed through the actor interface. The ActorProxy class provides two important functionalities:
  - Name resolution: It is able to locate the actor in the cluster (find the node of the cluster where it is hosted).
  - Failure handling: It can retry method invocations and re-resolve the actor location after, for example, a failure that requires the actor to be relocated to another node in the cluster.

The following rules that pertain to actor interfaces are worth mentioning:

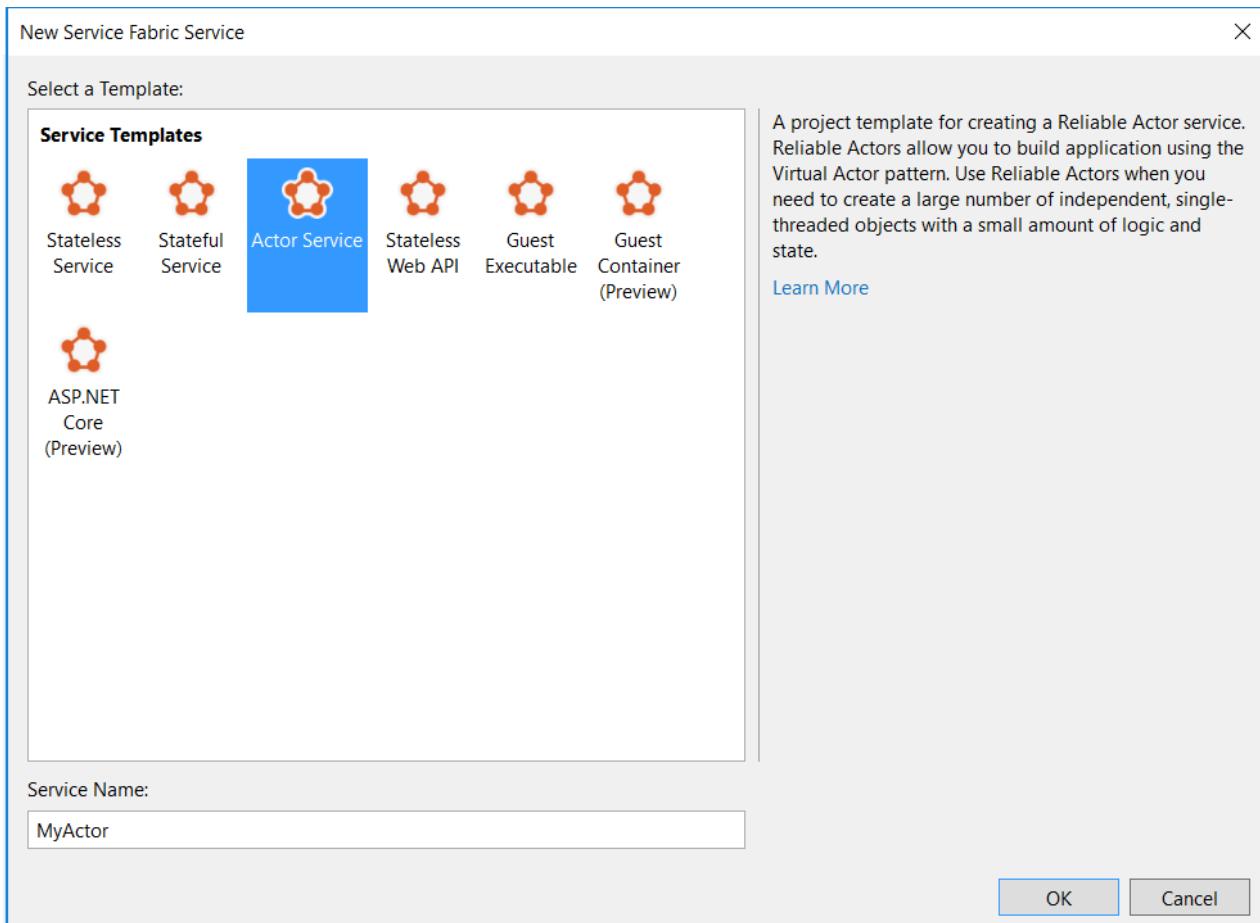
- Actor interface methods cannot be overloaded.
- Actor interface methods must not have out, ref, or optional parameters.
- Generic interfaces are not supported.

## Create a new project in Visual Studio

Launch Visual Studio 2015 or Visual Studio 2017 as an administrator, and create a new Service Fabric application project:

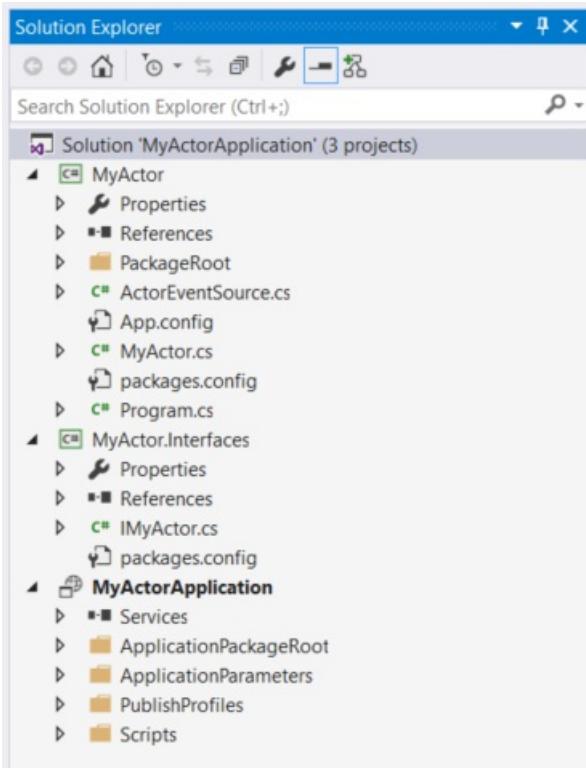


In the next dialog box, you can choose the type of project that you want to create.



For the HelloWorld project, let's use the Service Fabric Reliable Actors service.

After you have created the solution, you should see the following structure:



## Reliable Actors basic building blocks

A typical Reliable Actors solution is composed of three projects:

- **The application project (MyActorApplication).** This is the project that packages all of the services together for deployment. It contains the *ApplicationManifest.xml* and PowerShell scripts for managing the application.
- **The interface project (MyActor.Interfaces).** This is the project that contains the interface definition for the actor. In the MyActor.Interfaces project, you can define the interfaces that will be used by the actors in the solution. Your actor interfaces can be defined in any project with any name, however the interface defines the actor contract that is shared by the actor implementation and the clients calling the actor, so it typically makes sense to define it in an assembly that is separate from the actor implementation and can be shared by multiple other projects.

```
public interface IMyActor : IActor
{
    Task<string> HelloWorld();
}
```

- **The actor service project (MyActor).** This is the project used to define the Service Fabric service that is going to host the actor. It contains the implementation of the actor. An actor implementation is a class that derives from the base type `Actor` and implements the interface(s) that are defined in the MyActor.Interfaces project. An actor class must also implement a constructor that accepts an `ActorService` instance and an `ActorId` and passes them to the base `Actor` class. This allows for constructor dependency injection of platform dependencies.

```
[StatePersistence(StatePersistence.Persisted)]
class MyActor : Actor, IMyActor
{
    public MyActor(ActorService actorService, ActorId actorId)
        : base(actorService, actorId)
    {
    }

    public Task<string> HelloWorld()
    {
        return Task.FromResult("Hello world!");
    }
}
```

The actor service must be registered with a service type in the Service Fabric runtime. In order for the Actor Service to run your actor instances, your actor type must also be registered with the Actor Service. The `ActorRuntime` registration method performs this work for actors.

```
internal static class Program
{
    private static void Main()
    {
        try
        {
            ActorRuntime.RegisterActorAsync<MyActor>(
                (context, actorType) => new ActorService(context, actorType, () => new
MyActor())).GetAwaiter().GetResult();

            Thread.Sleep(Timeout.Infinite);
        }
        catch (Exception e)
        {
            ActorEventSource.Current.ActorHostInitializationFailed(e.ToString());
            throw;
        }
    }
}
```

If you start from a new project in Visual Studio and you have only one actor definition, the registration is included by default in the code that Visual Studio generates. If you define other actors in the service, you need to add the actor registration by using:

```
ActorRuntime.RegisterActorAsync<MyOtherActor>();
```

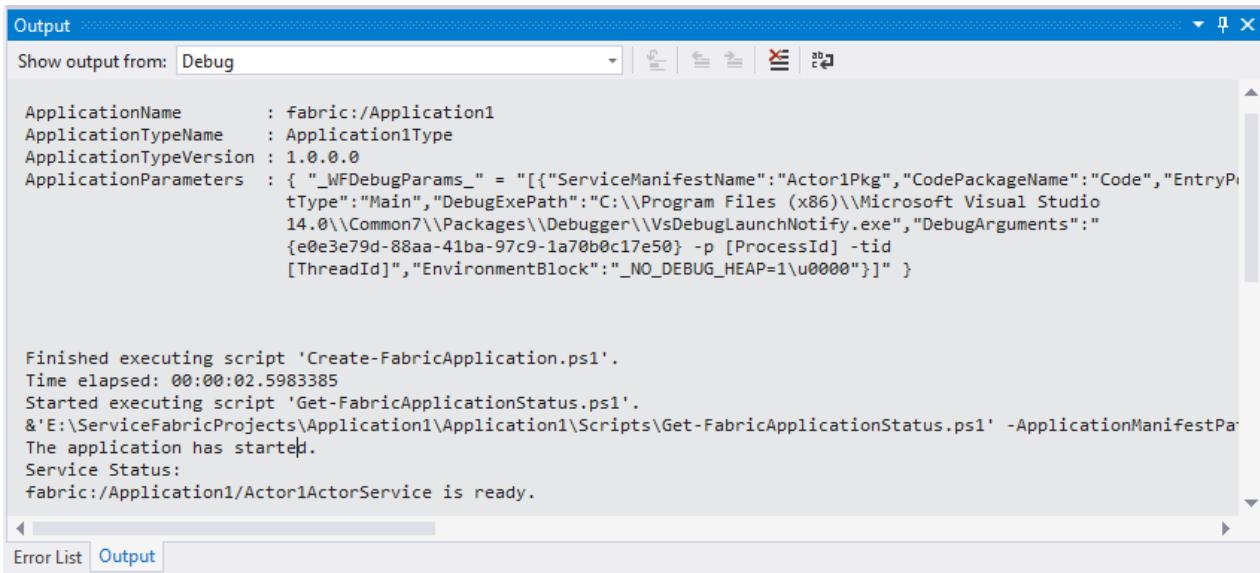
#### TIP

The Service Fabric Actors runtime emits some [events](#) and [performance counters](#) related to actor methods. They are useful in diagnostics and performance monitoring.

## Debugging

The Service Fabric tools for Visual Studio support debugging on your local machine. You can start a debugging session by hitting the F5 key. Visual Studio builds (if necessary) packages. It also deploys the application on the local Service Fabric cluster and attaches the debugger.

During the deployment process, you can see the progress in the **Output** window.



The screenshot shows the Visual Studio Output window with the title 'Output' at the top. A dropdown menu 'Show output from:' is set to 'Debug'. Below the dropdown are several icons: a magnifying glass, a refresh symbol, a copy icon, a clipboard icon, and a refresh/circular arrow icon. The main pane displays command-line output:

```
ApplicationName      : fabric:/Application1
ApplicationTypeName   : Application1Type
ApplicationTypeVersion: 1.0.0.0
ApplicationParameters: { "_WFDebugParams_" = [{"ServiceManifestName": "Actor1Pkg", "CodePackageName": "Code", "EntryPointType": "Main", "DebugExePath": "C:\Program Files (x86)\Microsoft Visual Studio 14.0\Common7\Packages\Debugger\VsDebugLaunchNotify.exe", "DebugArguments": "{e0e3e79d-88aa-41ba-97c9-1a70b0c17e50} -p [ProcessId] -tid [ThreadId]", "EnvironmentBlock": "_NO_DEBUG_HEAP=1\u0000"}] }
```

Finished executing script 'Create-FabricApplication.ps1'.
Time elapsed: 00:00:02.5983385
Started executing script 'Get-FabricApplicationStatus.ps1'.
&'E:\ServiceFabricProjects\Application1\Application1\Scripts\Get-FabricApplicationStatus.ps1' -ApplicationManifestPath
The application has started.
Service Status:
fabric:/Application1/Actor1ActorService is ready.

## Next steps

Learn more about [how Reliable Actors use the Service Fabric platform](#).

# Create your first Java Service Fabric Reliable Actors application on Linux

9/29/2017 • 9 min to read • [Edit Online](#)

This quick-start helps you create your first Azure Service Fabric Java application in a Linux development environment in just a few minutes. When you are finished, you'll have a simple Java single-service application running on the local development cluster.

## Prerequisites

Before you get started, install the Service Fabric SDK, the Service Fabric CLI, and setup a development cluster in your [Linux development environment](#). If you are using Mac OS X, you can [set up a Linux development environment in a virtual machine using Vagrant](#).

Also install the [Service Fabric CLI](#).

### Install and set up the generators for Java

Service Fabric provides scaffolding tools which will help you create a Service Fabric Java application from terminal using Yeoman template generator. Please follow the steps below to ensure you have the Service Fabric yeoman template generator for Java working on your machine.

1. Install nodejs and NPM on your machine

```
sudo apt-get install npm  
sudo apt install nodejs-legacy
```

2. Install [Yeoman](#) template generator on your machine from NPM

```
sudo npm install -g yo
```

3. Install the Service Fabric Yeoman Java application generator from NPM

```
sudo npm install -g generator-azuresfjava
```

## Basic concepts

To get started with Reliable Actors, you only need to understand a few basic concepts:

- **Actor service.** Reliable Actors are packaged in Reliable Services that can be deployed in the Service Fabric infrastructure. Actor instances are activated in a named service instance.
- **Actor registration.** As with Reliable Services, a Reliable Actor service needs to be registered with the Service Fabric runtime. In addition, the actor type needs to be registered with the Actor runtime.
- **Actor interface.** The actor interface is used to define a strongly typed public interface of an actor. In the Reliable Actor model terminology, the actor interface defines the types of messages that the actor can understand and process. The actor interface is used by other actors and client applications to "send" (asynchronously) messages to the actor. Reliable Actors can implement multiple interfaces.
- **ActorProxy class.** The ActorProxy class is used by client applications to invoke the methods exposed through the actor interface. The ActorProxy class provides two important functionalities:

- Name resolution: It is able to locate the actor in the cluster (find the node of the cluster where it is hosted).
- Failure handling: It can retry method invocations and re-resolve the actor location after, for example, a failure that requires the actor to be relocated to another node in the cluster.

The following rules that pertain to actor interfaces are worth mentioning:

- Actor interface methods cannot be overloaded.
- Actor interface methods must not have out, ref, or optional parameters.
- Generic interfaces are not supported.

## Create the application

A Service Fabric application contains one or more services, each with a specific role in delivering the application's functionality. The generator you installed in the last section, makes it easy to create your first service and to add more later. You can also create, build, and deploy Service Fabric Java applications using a plugin for Eclipse. See [Create and deploy your first Java application using Eclipse](#). For this quick start, use Yeoman to create an application with a single service that stores and gets a counter value.

1. In a terminal, type `yo azuresfjava`.
2. Name your application.
3. Choose the type of your first service and name it. For this tutorial, choose a Reliable Actor Service. For more information about the other types of services, see [Service Fabric programming model overview](#).

```
vagrant@vagrant:~/src$ clear
vagrant@vagrant:~/src$ yo azuresfjava

          Welcome to Service
          Fabric java app
          generator

? Name your application myapp
? Choose a framework for your service Reliable Actor Service
? Enter the name of actor service : myactorsvc
  create myapp/myactorsvc/src/reliableactor/myactorsvcImpl.java
  create myapp/myactorsvc/src/reliableactor/myactorsvcActorHost.java
  create myapp/myactorsvc/build.gradle
  create myapp/myapp/myactorsvcPkg/ServiceManifest.xml
  create myapp/myapp/ApplicationManifest.xml
  create myapp/myapp/myactorsvcPkg/Code/entryPoint.sh
  create myapp/myapp/myactorsvcPkg/Config/Settings.xml
  create myapp/myactorsvc/settings.gradle
  create myapp/myactorsvcInterface/src/reliableactor/myactorsvc.java
  create myapp/myactorsvcInterface/build.gradle
  create myapp/myactorsvcTestClient/src/reliableactor/test/myactorsvcTestClient.java
  create myapp/myactorsvcTestClient/build.gradle
  create myapp/myactorsvcTestClient/testclient.sh
  create myapp/myactorsvcTestClient/settings.gradle
  create myapp/build.gradle
  create myapp/settings.gradle
  create myapp/install.sh
  create myapp/uninstall.sh
  create myapp/myapp/myactorsvcPkg/Code/_readme.txt
  create myapp/myapp/myactorsvcPkg/Config/_readme.txt
  create myapp/myapp/myactorsvcPkg/Data/_readme.txt
vagrant@vagrant:~/src$
```

If you name the application "HelloWorldActorApplication" and the actor "HelloWorldActor", the following scaffolding is created:

```
HelloWorldActorApplication/
├── build.gradle
├── HelloWorldActor
│   ├── build.gradle
│   ├── settings.gradle
│   └── src
│       └── reliableactor
│           ├── HelloWorldActorHost.java
│           └── HelloWorldActorImpl.java
└── HelloWorldActorApplication
    ├── ApplicationManifest.xml
    └── HelloWorldActorPkg
        ├── Code
        │   ├── entryPoint.sh
        │   └── _readme.txt
        ├── Config
        │   ├── _readme.txt
        │   └── Settings.xml
        ├── Data
        │   └── _readme.txt
        └── ServiceManifest.xml
├── HelloWorldActorInterface
│   ├── build.gradle
│   └── src
│       └── reliableactor
│           └── HelloWorldActor.java
└── HelloWorldActorTestClient
    ├── build.gradle
    ├── settings.gradle
    ├── src
    │   └── reliableactor
    │       └── test
    │           └── HelloWorldActorTestClient.java
    └── testclient.sh
├── install.sh
├── settings.gradle
└── uninstall.sh
```

## Reliable Actors basic building blocks

The basic concepts described earlier translate into the basic building blocks of a Reliable Actor service.

### Actor interface

This contains the interface definition for the actor. This interface defines the actor contract that is shared by the actor implementation and the clients calling the actor, so it typically makes sense to define it in a place that is separate from the actor implementation and can be shared by multiple other services or client applications.

```
HelloWorldActorInterface/src/reliableactor/HelloWorldActor.java :
```

```
public interface HelloWorldActor extends Actor {
    @Readonly
    CompletableFuture<Integer> getCountAsync();

    CompletableFuture<?> setCountAsync(int count);
}
```

### Actor service

This contains your actor implementation and actor registration code. The actor class implements the actor

interface. This is where your actor does its work.

```
HelloWorldActor/src/reliableactor/HelloWorldActorImpl :
```

```
@ActorServiceAttribute(name = "HelloWorldActor.HelloWorldActorService")
@StatePersistenceAttribute(statePersistence = StatePersistence.Persisted)
public class HelloWorldActorImpl extends ReliableActor implements HelloWorldActor {
    Logger logger = Logger.getLogger(this.getClass().getName());

    protected CompletableFuture<?> onActivateAsync() {
        logger.log(Level.INFO, "onActivateAsync");
        return this.stateManager().tryAddStateAsync("count", 0);
    }

    @Override
    public CompletableFuture<Integer> getCountAsync() {
        logger.log(Level.INFO, "Getting current count value");
        return this.stateManager().getStateAsync("count");
    }

    @Override
    public CompletableFuture<?> setCountAsync(int count) {
        logger.log(Level.INFO, "Setting current count value {0}", count);
        return this.stateManager().addOrUpdateStateAsync("count", count, (key, value) -> count > value ? count
: value);
    }
}
```

## Actor registration

The actor service must be registered with a service type in the Service Fabric runtime. In order for the Actor Service to run your actor instances, your actor type must also be registered with the Actor Service. The `ActorRuntime` registration method performs this work for actors.

```
HelloWorldActor/src/reliableactor/HelloWorldActorHost :
```

```
public class HelloWorldActorHost {

    public static void main(String[] args) throws Exception {
        try {
            ActorRuntime.registerActorAsync(HelloWorldActorImpl.class, (context, actorType) -> new
ActorServiceImpl(context, actorType, ()-> new HelloWorldActorImpl()), Duration.ofSeconds(10));

            Thread.sleep(Long.MAX_VALUE);
        } catch (Exception e) {
            e.printStackTrace();
            throw e;
        }
    }
}
```

## Build the application

The Service Fabric Yeoman templates include a build script for [Gradle](#), which you can use to build the application from the terminal. Service Fabric Java dependencies get fetched from Maven. To build and work on the Service Fabric Java applications, you need to ensure that you have JDK and Gradle installed. If not yet installed, you can run the following to install JDK(openjdk-8-jdk) and Gradle -

```
sudo apt-get install openjdk-8-jdk-headless  
sudo apt-get install gradle
```

To build and package the application, run the following:

```
cd myapp  
gradle
```

## Deploy the application

Once the application is built, you can deploy it to the local cluster.

1. Connect to the local Service Fabric cluster.

```
sfctl cluster select --endpoint http://localhost:19080
```

2. Run the install script provided in the template to copy the application package to the cluster's image store, register the application type, and create an instance of the application.

```
./install.sh
```

Deploying the built application is the same as any other Service Fabric application. See the documentation on [managing a Service Fabric application with the Service Fabric CLI](#) for detailed instructions.

Parameters to these commands can be found in the generated manifests inside the application package.

Once the application has been deployed, open a browser and navigate to [Service Fabric Explorer](#) at <http://localhost:19080/Explorer>. Then, expand the **Applications** node and note that there is now an entry for your application type and another for the first instance of that type.

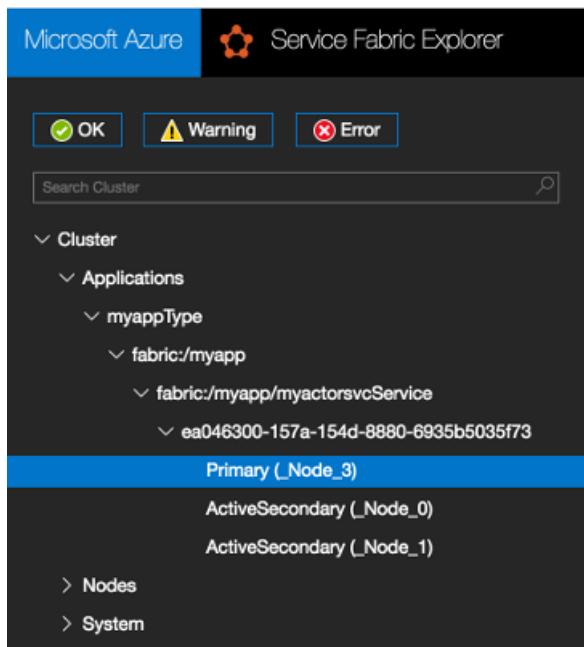
## Start the test client and perform a failover

Actors do not do anything on their own, they require another service or client to send them messages. The actor template includes a simple test script that you can use to interact with the actor service.

1. Run the script using the watch utility to see the output of the actor service. The test script calls the `setCountAsync()` method on the actor to increment a counter, calls the `getCountAsync()` method on the actor to get the new counter value, and displays that value to the console.

```
cd myactorsvcTestClient  
watch -n 1 ./testclient.sh
```

2. In Service Fabric Explorer, locate the node hosting the primary replica for the actor service. In the screenshot below, it is node 3. The primary service replica handles read and write operations. Changes in service state are then replicated out to the secondary replicas, running on nodes 0 and 1 in the screen shot below.



3. In **Nodes**, click the node you found in the previous step, then select **Deactivate (restart)** from the Actions menu. This action restarts the node running the primary service replica and forces a failover to one of the secondary replicas running on another node. That secondary replica is promoted to primary, another secondary replica is created on a different node, and the primary replica begins to take read/write operations. As the node restarts, watch the output from the test client and note that the counter continues to increment despite the failover.

## Remove the application

Use the uninstall script provided in the template to delete the application instance, unregister the application package, and remove the application package from the cluster's image store.

```
./uninstall.sh
```

In Service Fabric explorer you see that the application and application type no longer appear in the **Applications** node.

## Service Fabric Java libraries on Maven

Service Fabric Java libraries have been hosted in Maven. You can add the dependencies in the `pom.xml` or `build.gradle` of your projects to use Service Fabric Java libraries from **mavenCentral**.

### Actors

Service Fabric Reliable Actor support for your application.

```
<dependency>
  <groupId>com.microsoft.servicefabric</groupId>
  <artifactId>sf-actors-preview</artifactId>
  <version>0.12.0</version>
</dependency>
```

```
repositories {
    mavenCentral()
}
dependencies {
    compile 'com.microsoft.servicefabric:sf-actors-preview:0.12.0'
}
```

## Services

Service Fabric Reliable Services support for your application.

```
<dependency>
<groupId>com.microsoft.servicefabric</groupId>
<artifactId>sf-services-preview</artifactId>
<version>0.12.0</version>
</dependency>
```

```
repositories {
    mavenCentral()
}
dependencies {
    compile 'com.microsoft.servicefabric:sf-services-preview:0.12.0'
}
```

## Others

### Transport

Transport layer support for Service Fabric Java application. You do not need to explicitly add this dependency to your Reliable Actor or Service applications, unless you program at the transport layer.

```
<dependency>
<groupId>com.microsoft.servicefabric</groupId>
<artifactId>sf-transport-preview</artifactId>
<version>0.12.0</version>
</dependency>
```

```
repositories {
    mavenCentral()
}
dependencies {
    compile 'com.microsoft.servicefabric:sf-transport-preview:0.12.0'
}
```

### Fabric support

System level support for Service Fabric, which talks to native Service Fabric runtime. You do not need to explicitly add this dependency to your Reliable Actor or Service applications. This gets fetched automatically from Maven, when you include the other dependencies above.

```
<dependency>
<groupId>com.microsoft.servicefabric</groupId>
<artifactId>sf-preview</artifactId>
<version>0.12.0</version>
</dependency>
```

```
repositories {  
    mavenCentral()  
}  
dependencies {  
    compile 'com.microsoft.servicefabric:sf-preview:0.12.0'  
}
```

## Migrating old Service Fabric Java applications to be used with Maven

We have recently moved Service Fabric Java libraries from Service Fabric Java SDK to Maven repository. While the new applications you generate using Yeoman or Eclipse, will generate latest updated projects (which will be able to work with Maven), you can update your existing Service Fabric stateless or actor Java applications, which were using the Service Fabric Java SDK earlier, to use the Service Fabric Java dependencies from Maven. Please follow the steps mentioned [here](#) to ensure your older application works with Maven.

## Next steps

- [Create your first Service Fabric Java application on Linux using Eclipse](#)
- [Learn more about Reliable Actors](#)
- [Interact with Service Fabric clusters using the Service Fabric CLI](#)
- Learn about [Service Fabric support options](#)
- [Getting started with Service Fabric CLI](#)

# Actor events

10/12/2017 • 1 min to read • [Edit Online](#)

Actor events provide a way to send best-effort notifications from the actor to the clients. Actor events are designed for actor-to-client communication and shouldn't be used for actor-to-actor communication.

The following code snippets show how to use actor events in your application.

Define an interface that describes the events published by the actor. This interface must be derived from the `IActorEvents` interface. The arguments of the methods must be [data contract serializable](#). The methods must return void, as event notifications are one way and best effort.

```
public interface IGameEvents : IActorEvents
{
    void GameScoreUpdated(Guid gameId, string currentScore);
}
```

```
public interface GameEvents implements ActorEvents
{
    void gameScoreUpdated(UUID gameId, String currentScore);
}
```

Declare the events published by the actor in the actor interface.

```
public interface IGameActor : IActor, IActorEventPublisher<IGameEvents>
{
    Task UpdateGameStatus(GameStatus status);

    Task<string> GetGameScore();
}
```

```
public interface GameActor extends Actor, ActorEventPublisherE<GameEvents>
{
    CompletableFuture<?> updateGameStatus(GameStatus status);

    CompletableFuture<String> getGameScore();
}
```

On the client side, implement the event handler.

```
class GameEventsHandler : IGameEvents
{
    public void GameScoreUpdated(Guid gameId, string currentScore)
    {
        Console.WriteLine(@"Updates: Game: {0}, Score: {1}", gameId, currentScore);
    }
}
```

```

class GameEventsHandler implements GameEvents {
    public void gameScoreUpdated(UUID gameId, String currentScore)
    {
        System.out.println("Updates: Game: "+gameId+" ,Score: "+currentScore);
    }
}

```

On the client, create a proxy to the actor that publishes the event and subscribe to its events.

```

var proxy = ActorProxy.Create<IGameActor>(
    new ActorId(Guid.Parse(arg)), ApplicationName);

await proxy.SubscribeAsync<IGameEvents>(new GameEventsHandler());

```

```

GameActor actorProxy = ActorProxyBase.create<GameActor>(GameActor.class, new ActorId(UUID.fromString(args)));

return ActorProxyEventUtility.subscribeAsync(actorProxy, new GameEventsHandler());

```

In the event of failovers, the actor may fail over to a different process or node. The actor proxy manages the active subscriptions and automatically re-subscribes them. You can control the re-subscription interval through the `ActorProxyEventExtensions.SubscribeAsync<TEvent>` API. To unsubscribe, use the `ActorProxyEventExtensions.UnsubscribeAsync<TEvent>` API.

On the actor, publish the events as they happen. If there are subscribers to the event, the Actors runtime sends them the notification.

```

var ev = GetEvent<IGameEvents>();
ev.GameScoreUpdated(Id.GetGuidId(), score);

```

```

GameEvents event = getEvent<GameEvents>(GameEvents.class);
event.gameScoreUpdated(Id.getUUIDId(), score);

```

## Next steps

- [Actor reentrancy](#)
- [Actor diagnostics and performance monitoring](#)
- [Actor API reference documentation](#)
- [C# Sample code](#)
- [C# .NET Core Sample code](#)
- [Java Sample code](#)

# Actor timers and reminders

10/27/2017 • 5 min to read • [Edit Online](#)

Actors can schedule periodic work on themselves by registering either timers or reminders. This article shows how to use timers and reminders and explains the differences between them.

## Actor timers

Actor timers provide a simple wrapper around a .NET or Java timer to ensure that the callback methods respect the turn-based concurrency guarantees that the Actors runtime provides.

Actors can use the `RegisterTimer` (C#) or `registerTimer` (Java) and `UnregisterTimer` (C#) or `unregisterTimer` (Java) methods on their base class to register and unregister their timers. The example below shows the use of timer APIs. The APIs are very similar to the .NET timer or Java timer. In this example, when the timer is due, the Actors runtime will call the `MoveObject` (C#) or `moveObject` (Java) method. The method is guaranteed to respect the turn-based concurrency. This means that no other actor methods or timer/reminder callbacks will be in progress until this callback completes execution.

```
class VisualObjectActor : Actor, IVisualObject
{
    private IActorTimer _updateTimer;

    public VisualObjectActor(ActorService actorService, ActorId actorId)
        : base(actorService, actorId)
    {
    }

    protected override Task OnActivateAsync()
    {
        ...

        _updateTimer = RegisterTimer(
            MoveObject,                      // Callback method
            null,                            // Parameter to pass to the callback method
            TimeSpan.FromMilliseconds(15),   // Amount of time to delay before the callback is invoked
            TimeSpan.FromMilliseconds(15));  // Time interval between invocations of the callback method

        return base.OnActivateAsync();
    }

    protected override Task OnDeactivateAsync()
    {
        if (_updateTimer != null)
        {
            UnregisterTimer(_updateTimer);
        }

        return base.OnDeactivateAsync();
    }

    private Task MoveObject(object state)
    {
        ...
        return Task.FromResult(true);
    }
}
```

```

public class VisualObjectActorImpl extends FabricActor implements VisualObjectActor
{
    private ActorTimer updateTimer;

    public VisualObjectActorImpl(FabricActorService actorService, ActorId actorId)
    {
        super(actorService, actorId);
    }

    @Override
    protected CompletableFuture onActivateAsync()
    {
        ...

        return this.stateManager()
            .getOrAddStateAsync(
                stateName,
                VisualObject.createRandom(
                    this.getId().toString(),
                    new Random(this.getId().toString().hashCode())))
            .thenApply((r) -> {
                this.registerTimer(
                    (o) -> this.moveObject(o),           // Callback method
                    "moveObject",
                    null,                                // Parameter to pass to the
callback method
                    Duration.ofMillis(10),               // Amount of time to delay
before the callback is invoked
                    Duration.ofMillis(timerIntervalInMilliSeconds)); // Time interval between
invocations of the callback method
                return null;
            });
    }

    @Override
    protected CompletableFuture onDeactivateAsync()
    {
        if (updateTimer != null)
        {
            unregisterTimer(updateTimer);
        }

        return super.onDeactivateAsync();
    }

    private CompletableFuture moveObject(Object state)
    {
        ...
        return this.stateManager().getStateAsync(this.stateName).thenCompose(v -> {
            VisualObject v1 = (VisualObject)v;
            v1.move();
            return (CompletableFuture<?>)this.stateManager().setStateAsync(stateName, v1).
                thenApply(r -> {
                    ...
                    return null;});
        });
    }
}

```

The next period of the timer starts after the callback completes execution. This implies that the timer is stopped while the callback is executing and is started when the callback finishes.

The Actors runtime saves changes made to the actor's State Manager when the callback finishes. If an error occurs in saving the state, that actor object will be deactivated and a new instance will be activated.

All timers are stopped when the actor is deactivated as part of garbage collection. No timer callbacks are invoked

after that. Also, the Actors runtime does not retain any information about the timers that were running before deactivation. It is up to the actor to register any timers that it needs when it is reactivated in the future. For more information, see the section on [actor garbage collection](#).

## Actor reminders

Reminders are a mechanism to trigger persistent callbacks on an actor at specified times. Their functionality is similar to timers. But unlike timers, reminders are triggered under all circumstances until the actor explicitly unregisters them or the actor is explicitly deleted. Specifically, reminders are triggered across actor deactivations and failovers because the Actors runtime persists information about the actor's reminders using actor state provider. Please note that the reliability of reminders is tied to the state reliability guarantees provided by the actor state provider. This means that for actors whose state persistence is set to None, the reminders will not fire after a failover.

To register a reminder, an actor calls the `RegisterReminderAsync` method provided on the base class, as shown in the following example:

```
protected override async Task OnActivateAsync()
{
    string reminderName = "Pay cell phone bill";
    int amountInDollars = 100;

    IActorReminder reminderRegistration = await this.RegisterReminderAsync(
        reminderName,
        BitConverter.GetBytes(amountInDollars),
        TimeSpan.FromDays(3),
        TimeSpan.FromDays(1));
}
```

```
@Override
protected CompletableFuture onActivateAsync()
{
    String reminderName = "Pay cell phone bill";
    int amountInDollars = 100;

    ActorReminder reminderRegistration = this.registerReminderAsync(
        reminderName,
        state,
        dueTime,      //The amount of time to delay before firing the reminder
        period);     //The time interval between firing of reminders
}
```

In this example, `"Pay cell phone bill"` is the reminder name. This is a string that the actor uses to uniquely identify a reminder. `BitConverter.GetBytes(amountInDollars)` (C#) is the context that is associated with the reminder. It will be passed back to the actor as an argument to the reminder callback, i.e.

`IRemindable.ReceiveReminderAsync` (C#) or `Remindable.receiveReminderAsync` (Java).

Actors that use reminders must implement the `IRemindable` interface, as shown in the example below.

```

public class ToDoListActor : Actor, IToDoListActor, IRemindable
{
    public ToDoListActor(ActorService actorService, ActorId actorId)
        : base(actorService, actorId)
    {
    }

    public Task ReceiveReminderAsync(string reminderName, byte[] context, TimeSpan dueTime, TimeSpan period)
    {
        if (reminderName.Equals("Pay cell phone bill"))
        {
            int amountToPay = BitConverter.ToInt32(context, 0);
            System.Console.WriteLine("Please pay your cell phone bill of ${0}!", amountToPay);
        }
        return Task.FromResult(true);
    }
}

```

```

public class ToDoListActorImpl extends FabricActor implements ToDoListActor, Remindable
{
    public ToDoListActor(FabricActorService actorService, ActorId actorId)
    {
        super(actorService, actorId);
    }

    public CompletableFuture receiveReminderAsync(String reminderName, byte[] context, Duration dueTime,
Duration period)
    {
        if (reminderName.equals("Pay cell phone bill"))
        {
            int amountToPay = ByteBuffer.wrap(context).getInt();
            System.out.println("Please pay your cell phone bill of " + amountToPay);
        }
        return CompletableFuture.completedFuture(true);
    }
}

```

When a reminder is triggered, the Reliable Actors runtime will invoke the `ReceiveReminderAsync` (C#) or `receiveReminderAsync` (Java) method on the Actor. An actor can register multiple reminders, and the `ReceiveReminderAsync` (C#) or `receiveReminderAsync` (Java) method is invoked when any of those reminders is triggered. The actor can use the reminder name that is passed in to the `ReceiveReminderAsync` (C#) or `receiveReminderAsync` (Java) method to figure out which reminder was triggered.

The Actors runtime saves the actor's state when the `ReceiveReminderAsync` (C#) or `receiveReminderAsync` (Java) call finishes. If an error occurs in saving the state, that actor object will be deactivated and a new instance will be activated.

To unregister a reminder, an actor calls the `UnregisterReminderAsync` (C#) or `unregisterReminderAsync` (Java) method, as shown in the examples below.

```

IActorReminder reminder = GetReminder("Pay cell phone bill");
Task reminderUnregistration = UnregisterReminderAsync(reminder);

```

```

ActorReminder reminder = getReminder("Pay cell phone bill");
CompletableFuture reminderUnregistration = unregisterReminderAsync(reminder);

```

As shown above, the `UnregisterReminderAsync` (C#) or `unregisterReminderAsync` (Java) method accepts an `IActorReminder` (C#) or `ActorReminder` (Java) interface. The actor base class supports a `GetReminder` (C#) or `getReminder` (Java) method that can be used to retrieve the `IActorReminder` (C#) or `ActorReminder` (Java) interface

by passing in the reminder name. This is convenient because the actor does not need to persist the `IActorReminder` (C#) or `ActorReminder` (Java) interface that was returned from the `RegisterReminder` (C#) or `registerReminder` (Java) method call.

## Next Steps

Learn about Reliable Actor events and reentrancy:

- [Actor events](#)
- [Actor reentrancy](#)

# Configuring Reliable Actors--KVSActorStateProvider

10/2/2017 • 3 min to read • [Edit Online](#)

You can modify the default configuration of KVSActorStateProvider by changing the settings.xml file that is generated in the Microsoft Visual Studio package root under the Config folder for the specified actor.

The Azure Service Fabric runtime looks for predefined section names in the settings.xml file and consumes the configuration values while creating the underlying runtime components.

## NOTE

Do **not** delete or modify the section names of the following configurations in the settings.xml file that is generated in the Visual Studio solution.

## Replicator security configuration

Replicator security configurations are used to secure the communication channel that is used during replication. This means that services cannot see each other's replication traffic, ensuring that the data that is made highly available is also secure. By default, an empty security configuration section prevents replication security.

### Section name

<ActorName>ServiceReplicatorSecurityConfig

## Replicator configuration

Replicator configurations configure the replicator that is responsible for making the Actor State Provider state highly reliable. The default configuration is generated by the Visual Studio template and should suffice. This section talks about additional configurations that are available to tune the replicator.

### Section name

<ActorName>ServiceReplicatorConfig

### Configuration names

NAME	UNIT	DEFAULT VALUE	REMARKS
BatchAcknowledgementInterval	Seconds	0.015	Time period for which the replicator at the secondary waits after receiving an operation before sending back an acknowledgement to the primary. Any other acknowledgements to be sent for operations processed within this interval are sent as one response.

NAME	UNIT	DEFAULT VALUE	REMARKS
ReplicatorEndpoint	N/A	No default--required parameter	IP address and port that the primary/secondary replicator will use to communicate with other replicators in the replica set. This should reference a TCP resource endpoint in the service manifest. Refer to <a href="#">Service manifest resources</a> to read more about defining endpoint resources in the service manifest.
RetryInterval	Seconds	5	Time period after which the replicator re-transmits a message if it does not receive an acknowledgement for an operation.
MaxReplicationMessageSize	Bytes	50 MB	Maximum size of replication data that can be transmitted in a single message.
MaxPrimaryReplicationQueueSize	Number of operations	1024	Maximum number of operations in the primary queue. An operation is freed up after the primary replicator receives an acknowledgement from all the secondary replicators. This value must be greater than 64 and a power of 2.
MaxSecondaryReplicationQueueSize	Number of operations	2048	Maximum number of operations in the secondary queue. An operation is freed up after making its state highly available through persistence. This value must be greater than 64 and a power of 2.

## Store configuration

Store configurations are used to configure the local store that is used to persist the state that is being replicated. The default configuration is generated by the Visual Studio template and should suffice. This section talks about additional configurations that are available to tune the local store.

### Section name

<ActorName>ServiceLocalStoreConfig

### Configuration names

NAME	UNIT	DEFAULT VALUE	REMARKS

NAME	UNIT	DEFAULT VALUE	REMARKS
MaxAsyncCommitDelayInMilliseconds	Milliseconds	200	Sets the maximum batching interval for durable local store commits.
MaxVerPages	Number of pages	16384	The maximum number of version pages in the local store database. It determines the maximum number of outstanding transactions.

## Sample configuration file

```

<?xml version="1.0" encoding="utf-8"?>
<Settings xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://schemas.microsoft.com/2011/01/fabric">
  <Section Name="MyActorServiceReplicatorConfig">
    <Parameter Name="ReplicatorEndpoint" Value="MyActorServiceReplicatorEndpoint" />
    <Parameter Name="BatchAcknowledgementInterval" Value="0.05"/>
  </Section>
  <Section Name="MyActorServiceLocalStoreConfig">
    <Parameter Name="MaxVerPages" Value="8192" />
  </Section>
  <Section Name="MyActorServiceReplicatorSecurityConfig">
    <Parameter Name="CredentialType" Value="X509" />
    <Parameter Name="FindType" Value="FindByThumbprint" />
    <Parameter Name="FindValue" Value="9d c9 06 b1 69 dc 4f af fd 16 97 ac 78 1e 80 67 90 74 9d 2f" />
    <Parameter Name="StoreLocation" Value="LocalMachine" />
    <Parameter Name="StoreName" Value="My" />
    <Parameter Name="ProtectionLevel" Value="EncryptAndSign" />
    <Parameter Name="AllowedCommonNames" Value="My-Test-SAN1-Alice,My-Test-SAN1-Bob" />
  </Section>
</Settings>

```

## Remarks

The BatchAcknowledgementInterval parameter controls replication latency. A value of '0' results in the lowest possible latency, at the cost of throughput (as more acknowledgement messages must be sent and processed, each containing fewer acknowledgements). The larger the value for BatchAcknowledgementInterval, the higher the overall replication throughput, at the cost of higher operation latency. This directly translates to the latency of transaction commits.

# Configure FabricTransport settings for Reliable Actors

6/27/2017 • 2 min to read • [Edit Online](#)

Here are the settings that you can configure:

- C#: [FabricTransportRemotingSettings](#)
- Java: [FabricTransportRemotingSettings](#)

You can modify the default configuration of FabricTransport in following ways.

## Assembly attribute

The [FabricTransportActorRemotingProvider](#) attribute needs to be applied on the actor client and actor service assemblies.

The following example shows how to change the default value of FabricTransport OperationTimeout settings:

```
using Microsoft.ServiceFabric.Actors.Remoting.FabricTransport;
[assembly:FabricTransportActorRemotingProvider(OperationTimeoutInSeconds = 600)]
```

Second example changes default Values of FabricTransport MaxMessageSize and OperationTimeoutInSeconds.

```
using Microsoft.ServiceFabric.Actors.Remoting.FabricTransport;
[assembly:FabricTransportActorRemotingProvider(OperationTimeoutInSeconds = 600,MaxMessageSize = 134217728)]
```

## Config package

You can use a [config package](#) to modify the default configuration.

### Configure FabricTransport settings for the actor service

Add a TransportSettings section in the settings.xml file.

By default, actor code looks for SectionName as "<ActorName>TransportSettings". If that's not found, it checks for SectionName as "TransportSettings".

```
<Section Name="MyActorServiceTransportSettings">
  <Parameter Name="MaxMessageSize" Value="10000000" />
  <Parameter Name="OperationTimeoutInSeconds" Value="300" />
  <Parameter Name="SecurityCredentialsType" Value="X509" />
  <Parameter Name="CertificateFindType" Value="FindByThumbprint" />
  <Parameter Name="CertificateFindValue" Value="4FED3950642138446CC364A396E1E881DB76B48C" />
  <Parameter Name="CertificateRemoteThumbnails" Value="b3449b018d0f6839a2c5d62b5b6c6ac822b6f662" />
  <Parameter Name="CertificateStoreLocation" Value="LocalMachine" />
  <Parameter Name="CertificateStoreName" Value="My" />
  <Parameter Name="CertificateProtectionLevel" Value="EncryptAndSign" />
  <Parameter Name="CertificateRemoteCommonNames" Value="ServiceFabric-Test-Cert" />
</Section>
```

### Configure FabricTransport settings for the actor client assembly

If the client is not running as part of a service, you can create a "<Client Exe Name>.settings.xml" file in the same location as the client .exe file. Then add a TransportSettings section in that file. SectionName should be "TransportSettings".

```

<?xml version="1.0" encoding="utf-8"?>
<Settings xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://schemas.microsoft.com/2011/01/fabric">
  <Section Name="TransportSettings">
    <Parameter Name="SecurityCredentialsType" Value="X509" />
    <Parameter Name="OperationTimeoutInSeconds" Value="300" />
    <Parameter Name="CertificateFindType" Value="FindByThumbprint" />
    <Parameter Name="CertificateFindValue" Value="b3449b018d0f6839a2c5d62b5b6c6ac822b6f662" />
    <Parameter Name="CertificateRemoteThumbprints" Value="4FEF3950642138446CC364A396E1E881DB76B48C" />
    <Parameter Name="OperationTimeoutInSeconds" Value="300" />
    <Parameter Name="CertificateStoreLocation" Value="LocalMachine" />
    <Parameter Name="CertificateStoreName" Value="My" />
    <Parameter Name="CertificateProtectionLevel" Value="EncryptAndSign" />
    <Parameter Name="CertificateRemoteCommonNames" Value="WinFabric-Test-SAN1-Alice" />
  </Section>
</Settings>

```

- Configuring FabricTransport Settings for Secure Actor Service/Client With Secondary Certificate. Secondary certificate information can be added by adding parameter CertificateFindValuebySecondary. Below is the example for the Listener TransportSettings.

```

<Section Name="TransportSettings">
  <Parameter Name="SecurityCredentialsType" Value="X509" />
  <Parameter Name="CertificateFindType" Value="FindByThumbprint" />
  <Parameter Name="CertificateFindValue" Value="b3449b018d0f6839a2c5d62b5b6c6ac822b6f662" />
  <Parameter Name="CertificateFindValuebySecondary" Value="h9449b018d0f6839a2c5d62b5b6c6ac822b6f690" />
  <Parameter Name="CertificateRemoteThumbprints"
    Value="4FEF3950642138446CC364A396E1E881DB76B48C,a9449b018d0f6839a2c5d62b5b6c6ac822b6f667" />
  <Parameter Name="CertificateStoreLocation" Value="LocalMachine" />
  <Parameter Name="CertificateStoreName" Value="My" />
  <Parameter Name="CertificateProtectionLevel" Value="EncryptAndSign" />
</Section>

```

Below is the example for the Client TransportSettings.

```

<Section Name="TransportSettings">
  <Parameter Name="SecurityCredentialsType" Value="X509" />
  <Parameter Name="CertificateFindType" Value="FindByThumbprint" />
  <Parameter Name="CertificateFindValue" Value="4FEF3950642138446CC364A396E1E881DB76B48C" />
  <Parameter Name="CertificateFindValuebySecondary" Value="a9449b018d0f6839a2c5d62b5b6c6ac822b6f667" />
  <Parameter Name="CertificateRemoteThumbprints"
    Value="b3449b018d0f6839a2c5d62b5b6c6ac822b6f662,h9449b018d0f6839a2c5d62b5b6c6ac822b6f690" />
  <Parameter Name="CertificateStoreLocation" Value="LocalMachine" />
  <Parameter Name="CertificateStoreName" Value="My" />
  <Parameter Name="CertificateProtectionLevel" Value="EncryptAndSign" />
</Section>

```

- Configuring FabricTransport Settings for Securing Actor Service/Client Using Subject Name. User needs to provide findType as FindBySubjectName,add CertificateIssuerThumbprints and CertificateRemoteCommonNames values. Below is the example for the Listener TransportSettings.

```
<Section Name="TransportSettings">
<Parameter Name="SecurityCredentialsType" Value="X509" />
<Parameter Name="CertificateFindType" Value="FindBySubjectName" />
<Parameter Name="CertificateFindValue" Value="CN = WinFabric-Test-SAN1-Alice" />
<Parameter Name="CertificateIssuerThumbnails" Value="b3449b018d0f6839a2c5d62b5b6c6ac822b6f662" />
<Parameter Name="CertificateRemoteCommonNames" Value="WinFabric-Test-SAN1-Bob" />
<Parameter Name="CertificateStoreLocation" Value="LocalMachine" />
<Parameter Name="CertificateStoreName" Value="My" />
<Parameter Name="CertificateProtectionLevel" Value="EncryptAndSign" />
</Section>
```

Below is the example for the Client TransportSettings.

```
<Section Name="TransportSettings">
<Parameter Name="SecurityCredentialsType" Value="X509" />
<Parameter Name="CertificateFindType" Value="FindBySubjectName" />
<Parameter Name="CertificateFindValue" Value="CN = WinFabric-Test-SAN1-Bob" />
<Parameter Name="CertificateStoreLocation" Value="LocalMachine" />
<Parameter Name="CertificateStoreName" Value="My" />
<Parameter Name="CertificateRemoteCommonNames" Value="WinFabric-Test-SAN1-Alice" />
<Parameter Name="CertificateProtectionLevel" Value="EncryptAndSign" />
</Section>
```

# Configuring Reliable Actors-- ReliableDictionaryActorStateProvider

10/2/2017 • 7 min to read • [Edit Online](#)

You can modify the default configuration of ReliableDictionaryActorStateProvider by changing the settings.xml file generated in the Visual Studio package root under the Config folder for the specified actor.

The Azure Service Fabric runtime looks for predefined section names in the settings.xml file and consumes the configuration values while creating the underlying runtime components.

## NOTE

Do **not** delete or modify the section names of the following configurations in the settings.xml file that is generated in the Visual Studio solution.

There are also global settings that affect the configuration of ReliableDictionaryActorStateProvider.

## Global Configuration

The global configuration is specified in the cluster manifest for the cluster under the KtlLogger section. It allows configuration of the shared log location and size plus the global memory limits used by the logger. Note that changes in the cluster manifest affect all services that use ReliableDictionaryActorStateProvider and reliable stateful services.

The cluster manifest is a single XML file that holds settings and configurations that apply to all nodes and services in the cluster. The file is typically called ClusterManifest.xml. You can see the cluster manifest for your cluster using the Get-ServiceFabricClusterManifest powershell command.

### Configuration names

NAME	UNIT	DEFAULT VALUE	REMARKS
WriteBufferMemoryPoolMinumumInKB	Kilobytes	8388608	Minimum number of KB to allocate in kernel mode for the logger write buffer memory pool. This memory pool is used for caching state information before writing to disk.
WriteBufferMemoryPoolMaximumInKB	Kilobytes	No Limit	Maximum size to which the logger write buffer memory pool can grow.

Name	Unit	Default Value	Remarks
SharedLogId	GUID	""	Specifies a unique GUID to use for identifying the default shared log file used by all reliable services on all nodes in the cluster that do not specify the SharedLogId in their service specific configuration. If SharedLogId is specified, then SharedLogPath must also be specified.
SharedLogPath	Fully qualified path name	""	Specifies the fully qualified path where the shared log file used by all reliable services on all nodes in the cluster that do not specify the SharedLogPath in their service specific configuration. However, if SharedLogPath is specified, then SharedLogId must also be specified.
SharedLogSizeInMB	Megabytes	8192	Specifies the number of MB of disk space to statically allocate for the shared log. The value must be 2048 or larger.

## Sample cluster manifest section

```
<Section Name="KtlLogger">
  <Parameter Name="WriteBufferMemoryPoolMinimumInKB" Value="8192" />
  <Parameter Name="WriteBufferMemoryPoolMaximumInKB" Value="8192" />
  <Parameter Name="SharedLogId" Value="{7668BB54-FE9C-48ed-81AC-FF89E60ED2EF}" />
  <Parameter Name="SharedLogPath" Value="f:\SharedLog\Log" />
  <Parameter Name="SharedLogSizeInMB" Value="16383" />
</Section>
```

## Remarks

The logger has a global pool of memory allocated from non paged kernel memory that is available to all reliable services on a node for caching state data before being written to the dedicated log associated with the reliable service replica. The pool size is controlled by the WriteBufferMemoryPoolMinimumInKB and WriteBufferMemoryPoolMaximumInKB settings. WriteBufferMemoryPoolMinimumInKB specifies both the initial size of this memory pool and the lowest size to which the memory pool may shrink.

WriteBufferMemoryPoolMaximumInKB is the highest size to which the memory pool may grow. Each reliable service replica that is opened may increase the size of the memory pool by a system determined amount up to WriteBufferMemoryPoolMaximumInKB. If there is more demand for memory from the memory pool than is available, requests for memory will be delayed until memory is available. Therefore if the write buffer memory pool is too small for a particular configuration then performance may suffer.

The SharedLogId and SharedLogPath settings are always used together to define the GUID and location for the default shared log for all nodes in the cluster. The default shared log is used for all reliable services that do not specify the settings in the settings.xml for the specific service. For best performance, shared log files should be placed on disks that are used solely for the shared log file to reduce contention.

SharedLogSizeInMB specifies the amount of disk space to preallocate for the default shared log on all nodes. SharedLogId and SharedLogPath do not need to be specified in order for SharedLogSizeInMB to be specified.

## Replicator security configuration

Replicator security configurations are used to secure the communication channel that is used during replication. This means that services cannot see each other's replication traffic, ensuring the data that is made highly available is also secure. By default, an empty security configuration section prevents replication security.

### Section name

<ActorName>ServiceReplicatorSecurityConfig

## Replicator configuration

Replicator configurations are used to configure the replicator that is responsible for making the Actor State Provider state highly reliable by replicating and persisting the state locally. The default configuration is generated by the Visual Studio template and should suffice. This section talks about additional configurations that are available to tune the replicator.

### Section name

<ActorName>ServiceReplicatorConfig

### Configuration names

NAME	UNIT	DEFAULT VALUE	REMARKS
BatchAcknowledgementInterval	Seconds	0.015	Time period for which the replicator at the secondary waits after receiving an operation before sending back an acknowledgement to the primary. Any other acknowledgements to be sent for operations processed within this interval are sent as one response.
ReplicatorEndpoint	N/A	No default--required parameter	IP address and port that the primary/secondary replicator will use to communicate with other replicators in the replica set. This should reference a TCP resource endpoint in the service manifest. Refer to <a href="#">Service manifest resources</a> to read more about defining endpoint resources in service manifest.
MaxReplicationMessageSize	Bytes	50 MB	Maximum size of replication data that can be transmitted in a single message.

<b>NAME</b>	<b>UNIT</b>	<b>DEFAULT VALUE</b>	<b>REMARKS</b>
MaxPrimaryReplicationQueueSize	Number of operations	8192	Maximum number of operations in the primary queue. An operation is freed up after the primary replicator receives an acknowledgement from all the secondary replicators. This value must be greater than 64 and a power of 2.
MaxSecondaryReplicationQueueSize	Number of operations	16384	Maximum number of operations in the secondary queue. An operation is freed up after making its state highly available through persistence. This value must be greater than 64 and a power of 2.
CheckpointThresholdInMB	MB	200	Amount of log file space after which the state is checkpointed.
MaxRecordSizeInKB	KB	1024	Largest record size that the replicator may write in the log. This value must be a multiple of 4 and greater than 16.
OptimizeLogForLowerDiskUsage	Boolean	true	When true, the log is configured so that the replica's dedicated log file is created by using an NTFS sparse file. This lowers the actual disk space usage for the file. When false, the file is created with fixed allocations, which provide the best write performance.
SharedLogId	guid	""	Specifies a unique guid to use for identifying the shared log file used with this replica. Typically, services should not use this setting. However, if SharedLogId is specified, then SharedLogPath must also be specified.
SharedLogPath	Fully qualified path name	""	Specifies the fully qualified path where the shared log file for this replica will be created. Typically, services should not use this setting. However, if SharedLogPath is specified, then SharedLogId must also be specified.

# Sample configuration file

```
<?xml version="1.0" encoding="utf-8"?>
<Settings xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://schemas.microsoft.com/2011/01/fabric">
  <Section Name="MyActorServiceReplicatorConfig">
    <Parameter Name="ReplicatorEndpoint" Value="MyActorServiceReplicatorEndpoint" />
    <Parameter Name="BatchAcknowledgementInterval" Value="0.05"/>
    <Parameter Name="CheckpointThresholdInMB" Value="180" />
  </Section>
  <Section Name="MyActorServiceReplicatorSecurityConfig">
    <Parameter Name="CredentialType" Value="X509" />
    <Parameter Name="FindType" Value="FindByThumbprint" />
    <Parameter Name="FindValue" Value="9d c9 06 b1 69 dc 4f af fd 16 97 ac 78 1e 80 67 90 74 9d 2f" />
    <Parameter Name="StoreLocation" Value="LocalMachine" />
    <Parameter Name="StoreName" Value="My" />
    <Parameter Name="ProtectionLevel" Value="EncryptAndSign" />
    <Parameter Name="AllowedCommonNames" Value="My-Test-SAN1-Alice,My-Test-SAN1-Bob" />
  </Section>
</Settings>
```

## Remarks

The BatchAcknowledgementInterval parameter controls replication latency. A value of '0' results in the lowest possible latency, at the cost of throughput (as more acknowledgement messages must be sent and processed, each containing fewer acknowledgements). The larger the value for BatchAcknowledgementInterval, the higher the overall replication throughput, at the cost of higher operation latency. This directly translates to the latency of transaction commits.

The CheckpointThresholdInMB parameter controls the amount of disk space that the replicator can use to store state information in the replica's dedicated log file. Increasing this to a higher value than the default could result in faster reconfiguration times when a new replica is added to the set. This is due to the partial state transfer that takes place due to the availability of more history of operations in the log. This can potentially increase the recovery time of a replica after a crash.

If you set OptimizeForLowerDiskUsage to true, log file space will be over-provisioned so that active replicas can store more state information in their log files, while inactive replicas will use less disk space. This makes it possible to host more replicas on a node. If you set OptimizeForLowerDiskUsage to false, the state information is written to the log files more quickly.

The MaxRecordSizeInKB setting defines the maximum size of a record that can be written by the replicator into the log file. In most cases, the default 1024-KB record size is optimal. However, if the service is causing larger data items to be part of the state information, then this value might need to be increased. There is little benefit in making MaxRecordSizeInKB smaller than 1024, as smaller records use only the space needed for the smaller record. We expect that this value would need to be changed only in rare cases.

The SharedLogId and SharedLogPath settings are always used together to make a service use a separate shared log from the default shared log for the node. For best efficiency, as many services as possible should specify the same shared log. Shared log files should be placed on disks that are used solely for the shared log file, to reduce head movement contention. We expect that these values would need to be changed only in rare cases.

# Update your previous Java Service Fabric application to fetch Java libraries from Maven

8/23/2017 • 6 min to read • [Edit Online](#)

We have recently moved Service Fabric Java binaries from the Service Fabric Java SDK to Maven hosting. Now you can use **mavencentral** to fetch the latest Service Fabric Java dependencies. This quick-start helps you update your existing Java applications, which you earlier created to be used with Service Fabric Java SDK, using either Yeoman template or Eclipse, to be compatible with the Maven based build.

## Prerequisites

1. First you need to uninstall the existing Java SDK.

```
sudo dpkg -r servicefabricsdkjava
```

2. Install the latest Service Fabric CLI following the steps mentioned [here](#).
3. To build and work on the Service Fabric Java applications, you need to ensure that you have JDK 1.8 and Gradle installed. If not yet installed, you can run the following to install JDK 1.8 (openjdk-8-jdk) and Gradle -

```
sudo apt-get install openjdk-8-jdk-headless  
sudo apt-get install gradle
```

4. Update the install/uninstall scripts of your application to use the new Service Fabric CLI following the steps mentioned [here](#). You can refer to our getting-started [examples](#) for reference.

### TIP

After uninstalling the Service Fabric Java SDK, Yeoman will not work. Follow the Prerequisites mentioned [here](#) to have Service Fabric Yeoman Java template generator up and working.

## Service Fabric Java libraries on Maven

Service Fabric Java libraries have been hosted in Maven. You can add the dependencies in the `pom.xml` or `build.gradle` of your projects to use Service Fabric Java libraries from **mavenCentral**.

### Actors

Service Fabric Reliable Actor support for your application.

```
<dependency>  
  <groupId>com.microsoft.servicefabric</groupId>  
  <artifactId>sf-actors-preview</artifactId>  
  <version>0.10.0</version>  
</dependency>
```

```
repositories {
    mavenCentral()
}
dependencies {
    compile 'com.microsoft.servicefabric:sf-actors-preview:0.10.0'
}
```

## Services

Service Fabric Stateless Service support for your application.

```
<dependency>
    <groupId>com.microsoft.servicefabric</groupId>
    <artifactId>sf-services-preview</artifactId>
    <version>0.10.0</version>
</dependency>
```

```
repositories {
    mavenCentral()
}
dependencies {
    compile 'com.microsoft.servicefabric:sf-services-preview:0.10.0'
}
```

## Others

### Transport

Transport layer support for Service Fabric Java application. You do not need to explicitly add this dependency to your Reliable Actor or Service applications, unless you program at the transport layer.

```
<dependency>
    <groupId>com.microsoft.servicefabric</groupId>
    <artifactId>sf-transport-preview</artifactId>
    <version>0.10.0</version>
</dependency>
```

```
repositories {
    mavenCentral()
}
dependencies {
    compile 'com.microsoft.servicefabric:sf-transport-preview:0.10.0'
}
```

### Fabric support

System level support for Service Fabric, which talks to native Service Fabric runtime. You do not need to explicitly add this dependency to your Reliable Actor or Service applications. This gets fetched automatically from Maven, when you include the other dependencies above.

```
<dependency>
    <groupId>com.microsoft.servicefabric</groupId>
    <artifactId>sf-preview</artifactId>
    <version>0.10.0</version>
</dependency>
```

```

repositories {
    mavenCentral()
}
dependencies {
    compile 'com.microsoft.servicefabric:sf-preview:0.10.0'
}

```

## Migrating Service Fabric Stateless Service

To be able to build your existing Service Fabric stateless Java service using Service Fabric dependencies fetched from Maven, you need to update the `build.gradle` file inside the Service. Previously it used to be like as follows -

```

dependencies {
    compile fileTree(dir: '/opt/microsoft/sdk/servicefabric/java/packages/lib', include: ['*.jar'])
    compile project(':Interface')
}
.
.
.
jar {
    manifest {
        attributes(
            'Main-Class': 'statelessservice.MyStatelessServiceHost',
            "Class-Path": configurations.compile.collect { 'lib/' + it.getName() }.join(' '))
        baseName "MyStateless"
        destinationDir = file('../MyStatelessApplication/MyStatelessPkg/Code')
    }
.
.
.
task copyDeps <<{
    copy {
        from("/opt/microsoft/sdk/servicefabric/java/packages/lib")
        into("../MyStatelessApplication/MyStatelessPkg/Code/lib")
        include('*.jar')
    }
    copy {
        from("/opt/microsoft/sdk/servicefabric/java/packages/lib")
        into("../MyStatelessApplication/MyStatelessPkg/Code/lib")
        include('libj*.so')
    }
}

```

Now, to fetch the dependencies from Maven, the **updated** `build.gradle` would have the corresponding parts as follows -

```

repositories {
    mavenCentral()
}

configurations {
    azuresf
}

dependencies {
    compile project(':Interface')
    azuresf ('com.microsoft.servicefabric:sf-services-preview:0.10.0')
    compile fileTree(dir: 'lib', include: '*.jar')
}

task explodeDeps(type: Copy, dependsOn:configurations.azuresf) { task ->
    configurations.azuresf.filter { it.toString().contains("native-preview") }.each{
        from zipTree(it)
    }
    configurations.azuresf.filter { !it.toString().contains("native-preview") }.each {
        from it
    }
    into "lib"
    include "libj*.so", "*.jar"
}

compileJava.dependsOn(explodeDeps)
.

.

.

jar {
    manifest {
        def mpath = configurations.compile.collect {'lib/'+it.getName()}.join (' ')
        mpath = mpath + ' ' + configurations.azuresf.collect {'lib/'+it.getName()}.join (' ')
        attributes(
            'Main-Class': 'statelessservice.MyStatelessServiceHost',
            "Class-Path": mpath)
        baseName "MyStateless"
        destinationDir = file('../MyStatelessApplication/MyStatelessPkg/Code')
    }
}
.

.

.

task copyDeps <<{
    copy {
        copy {
            from("lib/")
            into("../MyStatelessApplication/MyStatelessPkg/Code/lib")
            include('*')
        }
    }
}

```

In general, to get an overall idea about how the build script would look like for a Service Fabric stateless Java service, you can refer to any sample from our getting-started examples. Here is the [build.gradle](#) for the EchoServer sample.

## Migrating Service Fabric Actor Service

To be able to build your existing Service Fabric Actor Java application using Service Fabric dependencies fetched from Maven, you need to update the `build.gradle` file inside the interface package and in the Service package. If you have a TestClient package, you need to update that as well. So, for your actor `Myactor`, the following would be the places where you need to update -

```
./Myactor/build.gradle  
./MyactorInterface/build.gradle  
./MyactorTestClient/build.gradle
```

#### Updating build script for the interface project

Previously it used to be like as follows -

```
dependencies {  
    compile fileTree(dir: '/opt/microsoft/sdk/servicefabric/java/packages/lib', include: ['*.jar'])  
}  
. . .
```

Now, to fetch the dependencies from Maven, the **updated** `build.gradle` would have the corresponding parts as follows -

```
repositories {  
    mavenCentral()  
}  
  
configurations {  
    azuresf  
}  
  
dependencies {  
    azuresf ('com.microsoft.servicefabric:sf-actors-preview:0.10.0')  
    compile fileTree(dir: 'lib', include: '*.jar')  
}  
  
task explodeDeps(type: Copy, dependsOn:configurations.azuresf) { task ->  
    configurations.azuresf.filter { it.toString().contains("native-preview") }.each{  
        from zipTree(it)  
    }  
    configurations.azuresf.filter { !it.toString().contains("native-preview") }.each {  
        from it  
    }  
    into "lib"  
    include "libj*.so", "*.jar"  
}  
  
compileJava.dependsOn(explodeDeps)  
. . .
```

#### Updating build script for the actor project

Previously it used to be like as follows -

```

dependencies {
    compile fileTree(dir: '/opt/microsoft/sdk/servicefabric/java/packages/lib', include: ['*.jar'])
    compile project(':MyactorInterface')
}

.
.

.

jar {
    manifest {
        attributes(
            'Main-Class': 'reliableactor.MyactorHost',
            'Class-Path': configurations.compile.collect { 'lib/' + it.getName() }.join(' '))
        baseName "myactor"
        destinationDir = file('../myjavaapp/MyactorPkg/Code')
    }
}
.

.

.

task copyDeps<< {
    copy {
        from("/opt/microsoft/sdk/servicefabric/java/packages/lib")
        into("../myjavaapp/MyactorPkg/Code/lib")
        include('*.jar')
    }
    copy {
        from("/opt/microsoft/sdk/servicefabric/java/packages/lib")
        into("../myjavaapp/MyactorPkg/Code/lib")
        include('libj*.so')
    }
    copy {
        from("../MyactorInterface/out/lib")
        into("../myjavaapp/MyactorPkg/Code/lib")
        include('*.jar')
    }
}
}

```

Now, to fetch the dependencies from Maven, the **updated** `build.gradle` would have the corresponding parts as follows -

```

repositories {
    mavenCentral()
}

configurations {
    azuresf
}

dependencies {
    compile project(':MyactorInterface')
    azuresf ('com.microsoft.servicefabric:sf-actors-preview:0.10.0')
    compile fileTree(dir: 'lib', include: '*.jar')
}

task explodeDeps(type: Copy, dependsOn:configurations.azuresf) { task ->
    configurations.azuresf.filter { it.toString().contains("native-preview") }.each{
        from zipTree(it)
    }
    configurations.azuresf.filter { !it.toString().contains("native-preview") }.each {
        from it
    }
    into "lib"
    include "libj*.so", "*.jar"
}

compileJava.dependsOn(explodeDeps)
.

.

.

jar {
    manifest {
        def mpath = configurations.compile.collect {'lib/'+it.getName()}.join (' ')
        mpath = mpath + ' ' + configurations.azuresf.collect {'lib/'+it.getName()}.join (' ')
        attributes(
            'Main-Class': 'reliableactor.MyactorHost',
            "Class-Path": mpath)
        baseName "myactor"
        destinationDir = file('../myjavaapp/MyactorPkg/Code')}
}

.

.

.

task copyDeps<< {
    copy {
        from("lib/")
        into("../myjavaapp/MyactorPkg/Code/lib")
        include('*')
    }
    copy {
        from("../MyactorInterface/out/lib")
        into("../myjavaapp/MyactorPkg/Code/lib")
        include('*.jar')
    }
}

```

#### **Updating build script for the test client project**

Changes here are similar to the changes discussed in previous section, that is, the actor project. Previously the Gradle script used to be like as follows -

```
dependencies {
    compile fileTree(dir: '/opt/microsoft/sdk/servicefabric/java/packages/lib', include: ['*.jar'])
    compile project(':MyactorInterface')
}

.
.
.
.

jar
{
    manifest {
        attributes(
            'Main-Class': 'reliableactor.test.MyactorTestClient',
            "Class-Path": configurations.compile.collect { 'lib/' + it.getName() }.join(' '))
    }
    baseName "myactor-test"
    destinationDir = file('out/lib')
}

.
.

task copyDeps<< {
    copy {
        from("/opt/microsoft/sdk/servicefabric/java/packages/lib")
        into("./out/lib/lib")
        include('*.jar')
    }
    copy {
        from("/opt/microsoft/sdk/servicefabric/java/packages/lib")
        into("./out/lib/lib")
        include('libj*.so')
    }
    copy {
        from("../MyactorInterface/out/lib")
        into("./out/lib/lib")
        include('*.jar')
    }
}
```

Now, to fetch the dependencies from Maven, the **updated** `build.gradle` would have the corresponding parts as follows -

```

repositories {
    mavenCentral()
}

configurations {
    azuresf
}

dependencies {
    compile project(':MyactorInterface')
    azuresf ('com.microsoft.servicefabric:sf-actors-preview:0.10.0')
    compile fileTree(dir: 'lib', include: '*.jar')
}

task explodeDeps(type: Copy, dependsOn:configurations.azuresf) { task ->
    configurations.azuresf.filter { it.toString().contains("native-preview") }.each{
        from zipTree(it)
    }
    configurations.azuresf.filter { !it.toString().contains("native-preview") }.each {
        from it
    }
    into "lib"
    include "libj*.so", "*.jar"
}

compileJava.dependsOn(explodeDeps)
.

.

.

jar {
    manifest {
        def mpath = configurations.compile.collect {'lib/'+it.getName()}.join (' ')
        mpath = mpath + ' ' + configurations.azuresf.collect {'lib/'+it.getName()}.join (' ')
        attributes(
            'Main-Class': 'reliableactor.test.MyactorTestClient',
            "Class-Path": mpath)
        baseName "myactor-test"
        destinationDir = file('./out/lib')
    }
}
.

.

.

task copyDeps<< {
    copy {
        from("lib/")
        into("./out/lib/lib")
        include('*')
    }
    copy {
        from("../MyactorInterface/out/lib")
        into("./out/lib/lib")
        include('*.*')
    }
}

```

## Next steps

- [Create and deploy your first Service Fabric Java application on Linux by using Yeoman](#)
- [Create and deploy your first Service Fabric Java application on Linux by using Service Fabric Plugin for Eclipse](#)
- [Interact with Service Fabric clusters using the Service Fabric CLI](#)

# Connect to a secure service with the reverse proxy

8/11/2017 • 5 min to read • [Edit Online](#)

This article explains how to establish secure connection between the reverse proxy and services, thus enabling an end to end secure channel.

Connecting to secure services is supported only when reverse proxy is configured to listen on HTTPS. Rest of the document assumes this is the case. Refer to [Reverse proxy in Azure Service Fabric](#) to configure the reverse proxy in Service Fabric.

## Secure connection establishment between the reverse proxy and services

### Reverse proxy authenticating to services:

The reverse proxy identifies itself to services using its certificate, specified with **reverseProxyCertificate** property in the [Cluster Resource type section](#). Services can implement the logic to verify the certificate presented by the reverse proxy. The services can specify the accepted client certificate details as configuration settings in the configuration package. This can be read at runtime and used to validate the certificate presented by the reverse proxy. Refer to [Manage application parameters](#) to add the configuration settings.

### Reverse proxy verifying the service's identity via the certificate presented by the service:

To perform server certificate validation of the certificates presented by the services, reverse proxy supports one of the following options: None, ServiceCommonNameAndIssuer, and ServiceCertificateThumbprints. To select one of these three options, specify the **ApplicationCertificateValidationPolicy** in the parameters section of ApplicationGateway/Http element under [fabricSettings](#).

```
{
  "fabricSettings": [
    ...
    {
      "name": "ApplicationGateway/Http",
      "parameters": [
        {
          "name": "ApplicationCertificateValidationPolicy",
          "value": "None"
        }
      ]
    },
    ...
  ]
}
```

Refer to the next section for details about additional configuration for each of these options.

### Service certificate validation options

- **None**: Reverse proxy skips verification of the proxied service certificate and establishes the secure connection. This is the default behavior. Specify the **ApplicationCertificateValidationPolicy** with value **None** in the parameters section of ApplicationGateway/Http element.
- **ServiceCommonNameAndIssuer**: Reverse proxy verifies the certificate presented by the service based on certificate's common name and immediate issuer's thumbprint: Specify the **ApplicationCertificateValidationPolicy** with value **ServiceCommonNameAndIssuer** in the parameters

section of ApplicationGateway/Http element.

```
{  
  "fabricSettings": [  
    ...  
    {  
      "name": "ApplicationGateway/Http",  
      "parameters": [  
        {  
          "name": "ApplicationCertificateValidationPolicy",  
          "value": "ServiceCommonNameAndIssuer"  
        }  
      ]  
    },  
    ...  
  ]
```

To specify the list of service common name and issuer thumbprints, add a **ApplicationGateway/Http/ServiceCommonNameAndIssuer** element under fabricSettings, as shown below. Multiple certificate common name and issuer thumbprint pairs can be added in the parameters array element.

If the endpoint reverse proxy is connecting to presents a certificate who's common name and issuer thumbprint matches any of the values specified here, SSL channel is established. Upon failure to match the certificate details, reverse proxy fails the client's request with a 502 (Bad Gateway) status code. The HTTP status line will also contain the phrase "Invalid SSL Certificate."

```
{  
  "fabricSettings": [  
    ...  
    {  
      "name": "ApplicationGateway/Http/ServiceCommonNameAndIssuer",  
      "parameters": [  
        {  
          "name": "WinFabric-Test-Certificate-CN1",  
          "value": "b3 44 9b 01 8d 0f 68 39 a2 c5 d6 2b 5b 6c 6a c8 22 b4 22 11"  
        },  
        {  
          "name": "WinFabric-Test-Certificate-CN2",  
          "value": "b3 44 9b 01 8d 0f 68 39 a2 c5 d6 2b 5b 6c 6a c8 22 11 33 44"  
        }  
      ]  
    },  
    ...  
  ]
```

- **ServiceCertificateThumbprints:** Reverse proxy will verify the proxied service certificate based on its thumbprint. You can choose to go this route when the services are configured with self signed certificates: Specify the **ApplicationCertificateValidationPolicy** with value **ServiceCertificateThumbprints** in the parameters section of ApplicationGateway/Http element.

```
{
  "fabricSettings": [
    ...
    {
      "name": "ApplicationGateway/Http",
      "parameters": [
        {
          "name": "ApplicationCertificateValidationPolicy",
          "value": "ServiceCertificateThumbprints"
        }
      ]
    },
    ...
  ]
}
```

Also specify the thumbprints with a **ServiceCertificateThumbprints** entry under parameters section of ApplicationGateway/Http element. Multiple thumbprints can be specified as a comma-separated list in the value field, as shown below:

```
{
  "fabricSettings": [
    ...
    {
      "name": "ApplicationGateway/Http",
      "parameters": [
        ...
        {
          "name": "ServiceCertificateThumbprints",
          "value": "78 12 20 5a 39 d2 23 76 da a0 37 f0 5a ed e3 60 1a 7e 64 bf,78 12 20 5a 39 d2 23 76
da a0 37 f0 5a ed e3 60 1a 7e 64 b9"
        }
      ]
    },
    ...
  ]
}
```

If the thumbprint of the server certificate is listed in this config entry, reverse proxy succeeds the SSL connection. Otherwise, it terminates the connection and fails the client's request with a 502 (Bad Gateway). The HTTP status line will also contain the phrase "Invalid SSL Certificate."

## Endpoint selection logic when services expose secure as well as unsecured endpoints

Service fabric supports configuring multiple endpoints for a service. See [Specify resources in a service manifest](#).

Reverse proxy selects one of the endpoints to forward the request based on the **ListenerName** query parameter. If this is not specified, it can pick any endpoint from the endpoints list. Now this can be an HTTP or HTTPS endpoint. There might be scenarios/requirements where you want the reverse proxy to operate in a "secure only mode", i.e. you don't want the secure reverse proxy to forward requests to unsecured endpoints. This can be achieved by specifying the **SecureOnlyMode** configuration entry with value **true** in the parameters section of ApplicationGateway/Http element.

```
{
  "fabricSettings": [
    ...
    {
      "name": "ApplicationGateway/Http",
      "parameters": [
        ...
        {
          "name": "SecureOnlyMode",
          "value": true
        }
      ]
    },
    ...
  ]
}
```

When operating in **SecureOnlyMode**, if client has specified a **ListenerName** corresponding to an HTTP(unsecured) endpoint, reverse proxy fails the request with a 404 (Not Found) HTTP status code.

## Setting up client certificate authentication through the reverse proxy

SSL termination happens at the reverse proxy and all the client certificate data is lost. For the services to perform client certificate authentication, set the **ForwardClientCertificate** setting in the parameters section of ApplicationGateway/Http element.

1. When **ForwardClientCertificate** is set to **false**, reverse proxy will not request for the client certificate during its SSL handshake with the client. This is the default behavior.
2. When **ForwardClientCertificate** is set to **true**, reverse proxy requests for the client's certificate during its SSL handshake with the client. It will then forward the client certificate data in a custom HTTP header named **X-Client-Certificate**. The header value is the base64 encoded PEM format string of the client's certificate. The service can succeed/fail the request with appropriate status code after inspecting the certificate data. If the client does not present a certificate, reverse proxy forwards an empty header and let the service handle the case.

Reverse proxy is a mere forwarder. It will not perform any validation of the client's certificate.

## Next steps

- Refer to [Configure reverse proxy to connect to secure services](#) for Azure Resource Manager template samples to configure secure reverse proxy with the different service certificate validation options.
- See an example of HTTP communication between services in a [sample project on GitHub](#).
- [Remote procedure calls with Reliable Services remoting](#)
- [Web API that uses OWIN in Reliable Services](#)
- [Manage cluster certificates](#)

# Build a web service front end for your application using ASP.NET Core

6/27/2017 • 9 min to read • [Edit Online](#)

By default, Azure Service Fabric services do not provide a public interface to the web. To expose your application's functionality to HTTP clients, you have to create a web project to act as an entry point and then communicate from there to your individual services.

In this tutorial, we pick up where we left off in the [Creating your first application in Visual Studio](#) tutorial and add an ASP.NET Core web service in front of the stateful counter service. If you have not already done so, you should go back and step through that tutorial first.

## Set up your environment for ASP.NET Core

You need Visual Studio 2017 to follow along with this tutorial. Any edition will do.

- [Install Visual Studio 2017](#)

To develop ASP.NET Core Service Fabric applications, you should have the following workloads installed:

- **Azure development** (under *Web & Cloud*)
- **ASP.NET and web development** (under *Web & Cloud*)
- **.NET Core cross-platform development** (under *Other Toolsets*)

### NOTE

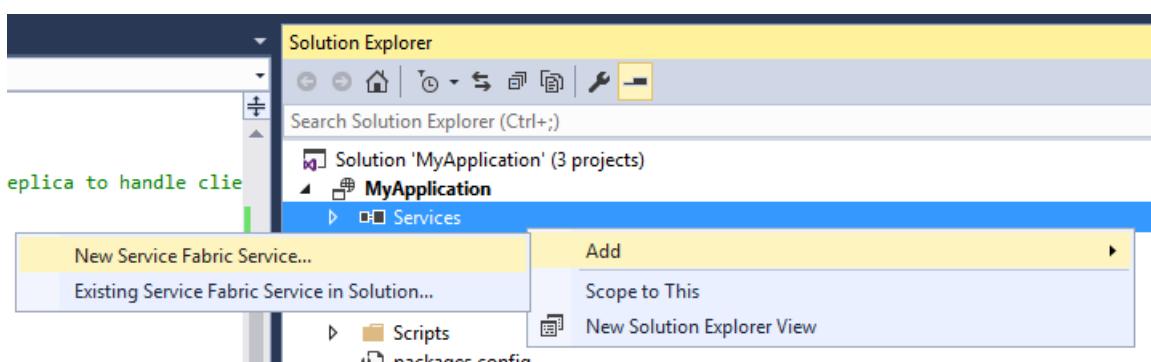
The .NET Core tools for Visual Studio 2015 are no longer being updated, however if you are still using Visual Studio 2015, you need to have [.NET Core VS 2015 Tooling Preview 2](#) installed.

## Add an ASP.NET Core service to your application

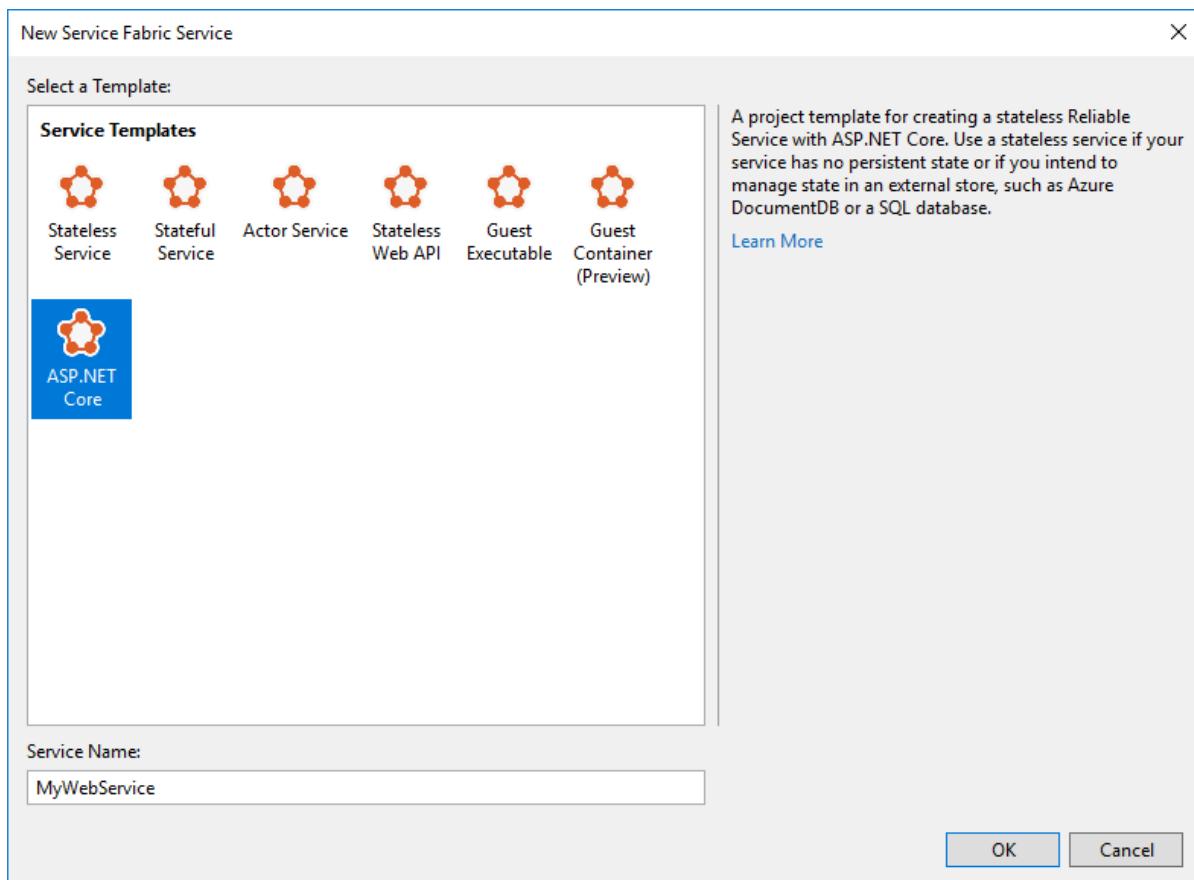
ASP.NET Core is a lightweight, cross-platform web development framework that you can use to create modern web UI and web APIs.

Let's add an ASP.NET Web API project to our existing application.

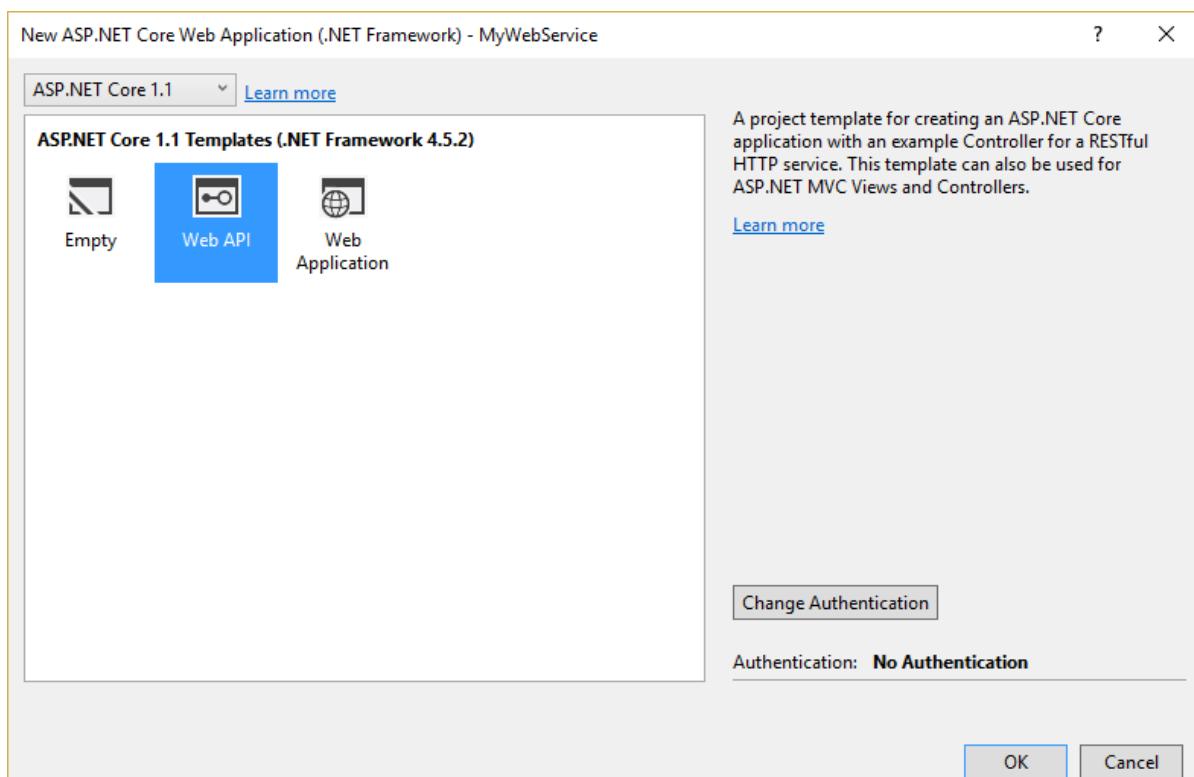
1. In Solution Explorer, right-click **Services** within the application project and choose **Add > New Service Fabric Service**.



2. On the **Create a Service** page, choose **ASP.NET Core** and give it a name.



3. The next page provides a set of ASP.NET Core project templates. Note that these are the same choices that you would see if you created an ASP.NET Core project outside of a Service Fabric application, with a small amount of additional code to register the service with the Service Fabric runtime. For this tutorial, choose **Web API**. However, you can apply the same concepts to building a full web application.



Once your Web API project is created, you should have two services in your application. As you continue to build your application, you can add more services in exactly the same way. Each can be independently versioned and upgraded.

## Run the application

To get a sense of what we've done, let's deploy the new application and take a look at the default behavior that the ASP.NET Core Web API template provides.

1. Press F5 in Visual Studio to debug the app.
2. When deployment is complete, Visual Studio launches a browser to the root of the ASP.NET Web API service. The ASP.NET Core Web API template doesn't provide default behavior for the root, so you should see a 404 error in the browser.
3. Add `/api/values` to the location in the browser. This invokes the `Get` method on the `ValuesController` in the Web API template. It returns the default response that is provided by the template--a JSON array that contains two strings:



By the end of the tutorial, this page will show the most recent counter value from our stateful service instead of the default strings.

## Connect the services

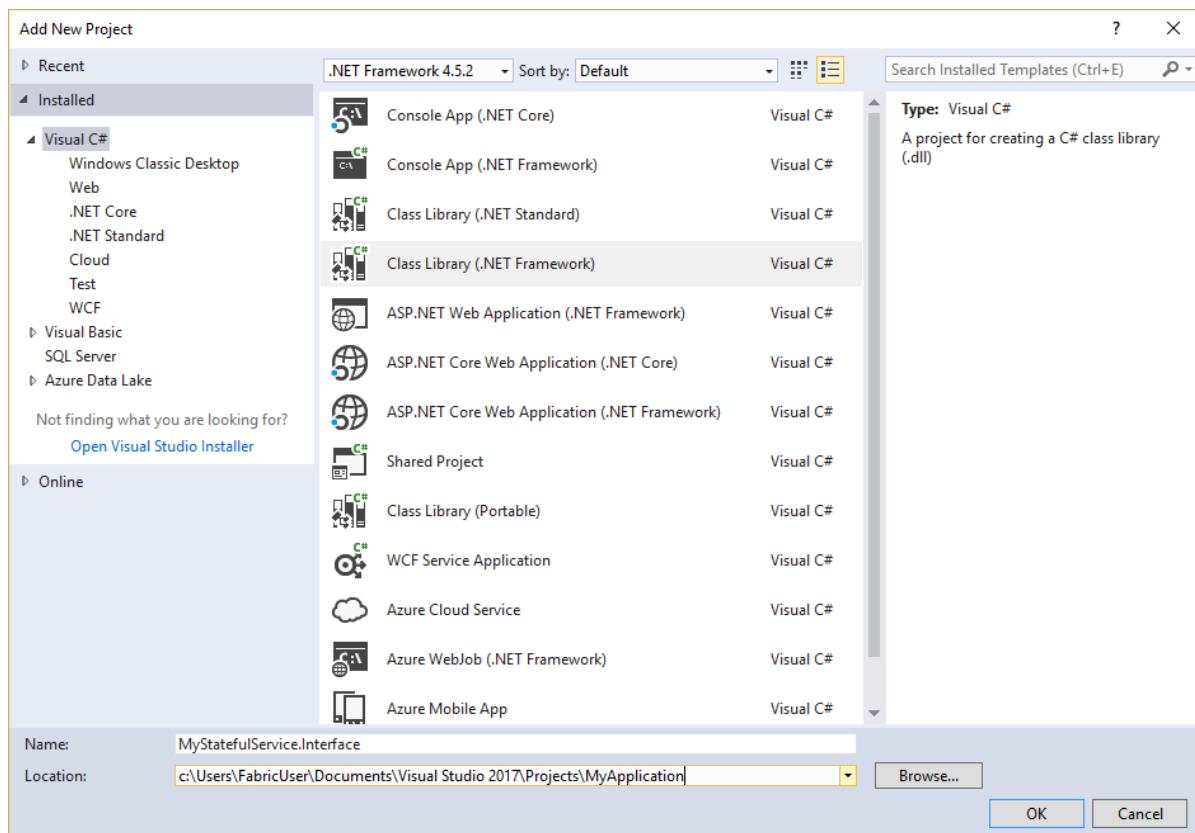
Service Fabric provides complete flexibility in how you communicate with reliable services. Within a single application, you might have services that are accessible via TCP, other services that are accessible via an HTTP REST API, and still other services that are accessible via web sockets. For background on the options available and the tradeoffs involved, see [Communicating with services](#). In this tutorial, we use [Service Fabric Service Remoting](#), provided in the SDK.

In the Service Remoting approach (modeled on remote procedure calls or RPCs), you define an interface to act as the public contract for the service. Then, you use that interface to generate a proxy class for interacting with the service.

### Create the remoting interface

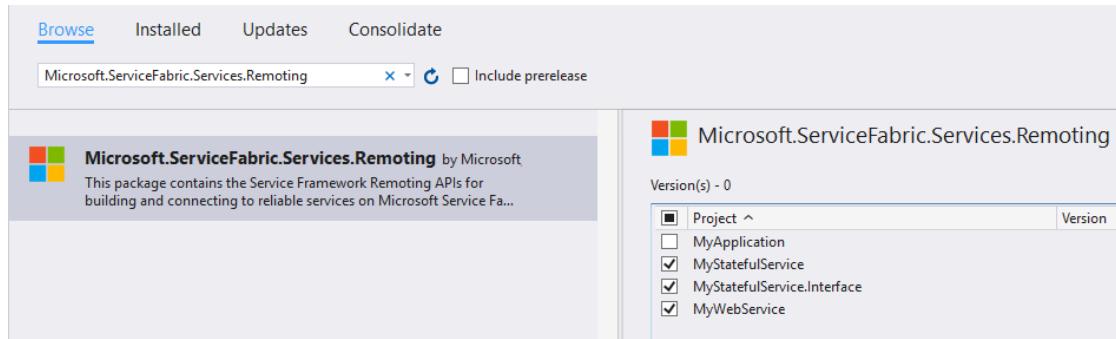
Let's start by creating the interface to act as the contract between the stateful service and other services, in this case the ASP.NET Core web project. This interface must be shared by all services that use it to make RPC calls, so we'll create it in its own Class Library project.

1. In Solution Explorer, right-click your solution and choose **Add > New Project**.
2. Choose the **Visual C#** entry in the left navigation pane and then select the **Class Library** template. Ensure that the .NET Framework version is set to **4.5.2**.



3. Install the **Microsoft.ServiceFabric.Services.Remoting** NuGet package. Search for **Microsoft.ServiceFabric.Services.Remoting** in the NuGet package manager and install it for all projects in the solution that use Service Remoting, including:

- The Class Library project that contains the service interface
- The Stateful Service project
- The ASP.NET Core web service project



4. In the class library, create an interface with a single method, `GetCountAsync`, and extend the interface from `Microsoft.ServiceFabric.Services.Remoting.IService`. The remoting interface must derive from this interface to indicate that it is a Service Remoting interface.

```

using Microsoft.ServiceFabric.Services.Remoting;
using System.Threading.Tasks;

...

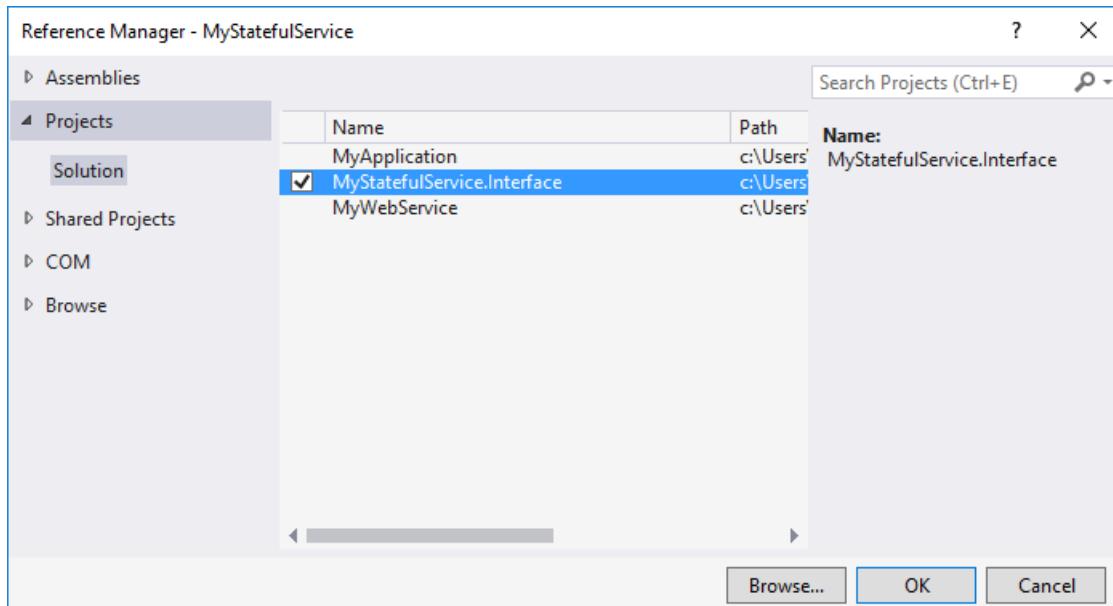
namespace MyStatefulService.Interface
{
    public interface ICounter : IService
    {
        Task<long> GetCountAsync();
    }
}

```

## Implement the interface in your stateful service

Now that we have defined the interface, we need to implement it in the stateful service.

1. In your stateful service, add a reference to the class library project that contains the interface.



2. Locate the class that inherits from `StatefulService`, such as `MyStatefulService`, and extend it to implement the `ICounter` interface.

```

using MyStatefulService.Interface;

...

public class MyStatefulService : StatefulService, ICounter
{
    ...
}

```

3. Now implement the single method that is defined in the `ICounter` interface, `GetCountAsync`.

```

public async Task<long> GetCountAsync()
{
    var myDictionary =
        await this.StateManager.GetOrAddAsync<IReliableDictionary<string, long>>("myDictionary");

    using (var tx = this.StateManager.CreateTransaction())
    {
        var result = await myDictionary.TryGetValueAsync(tx, "Counter");
        return result.HasValue ? result.Value : 0;
    }
}

```

## Expose the stateful service using a service remoting listener

With the `ICounter` interface implemented, the final step is to open the Service Remoting communication channel. For stateful services, Service Fabric provides an overridable method called `CreateServiceReplicaListeners`. With this method, you can specify one or more communication listeners, based on the type of communication that you want to enable for your service.

### NOTE

The equivalent method for opening a communication channel to stateless services is called `CreateServiceInstanceListeners`.

In this case, we replace the existing `CreateServiceReplicaListeners` method and provide an instance of `ServiceRemotingListener`, which creates an RPC endpoint that is callable from clients through `ServiceProxy`.

The `CreateServiceRemotingListener` extension method on the `IService` interface allows you to easily create a `ServiceRemotingListener` with all default settings. To use this extension method, ensure you have the `Microsoft.ServiceFabric.Services.Remoting.Runtime` namespace imported.

```

using Microsoft.ServiceFabric.Services.Remoting.Runtime;

...

protected override IEnumerable<ServiceReplicaListener> CreateServiceReplicaListeners()
{
    return new List<ServiceReplicaListener>()
    {
        new ServiceReplicaListener(
            context) =>
                this.CreateServiceRemotingListener(context))
    };
}

```

## Use the `ServiceProxy` class to interact with the service

Our stateful service is now ready to receive traffic from other services over RPC. So all that remains is adding the code to communicate with it from the ASP.NET web service.

1. In your ASP.NET project, add a reference to the class library that contains the `ICounter` interface.
2. Earlier you added the **Microsoft.ServiceFabric.Services.Remoting** NuGet package to the ASP.NET project. This package provides the `ServiceProxy` class which you use to make RPC calls to the stateful service. Ensure this NuGet package is installed in the ASP.NET Core web service project.
3. In the **Controllers** folder, open the `ValuesController` class. Note that the `Get` method currently just returns a hard-coded string array of "value1" and "value2"--which matches what we saw earlier in the browser. Replace this implementation with the following code:

```

using MyStatefulService.Interface;
using Microsoft.ServiceFabric.Services.Client;
using Microsoft.ServiceFabric.Services.Remoting.Client;

...

[HttpGet]
public async Task<IEnumerable<string>> Get()
{
    ICounter counter =
        ServiceProxy.Create<ICounter>(new Uri("fabric:/MyApplication/MyStatefulService"), new
    ServicePartitionKey(0));

    long count = await counter.GetCountAsync();

    return new string[] { count.ToString() };
}

```

The first line of code is the key one. To create the `ICounter` proxy to the stateful service, you need to provide two pieces of information: a partition ID and the name of the service.

You can use partitioning to scale stateful services by breaking up their state into different buckets, based on a key that you define, such as a customer ID or postal code. In our trivial application, the stateful service only has one partition, so the key doesn't matter. Any key that you provide will lead to the same partition. To learn more about partitioning services, see [How to partition Service Fabric Reliable Services](#).

The service name is a URI of the form `fabric:/<application_name>/<service_name>`.

With these two pieces of information, Service Fabric can uniquely identify the machine that requests should be sent to. The `ServiceProxy` class also seamlessly handles the case where the machine that hosts the stateful service partition fails and another machine must be promoted to take its place. This abstraction makes writing the client code to deal with other services significantly simpler.

Once we have the proxy, we simply invoke the `GetCountAsync` method and return its result.

4. Press F5 again to run the modified application. As before, Visual Studio automatically launches the browser to the root of the web project. Add the "api/values" path, and you should see the current counter value returned.



Refresh the browser periodically to see the counter value update.

## Kestrel and WebListener

The default ASP.NET Core web server, known as Kestrel, is [not currently supported for handling direct internet traffic](#). As a result, the ASP.NET Core stateless service template for Service Fabric uses [WebListener](#) by default.

To learn more about Kestrel and WebListener in Service Fabric services, please refer to [ASP.NET Core in Service Fabric Reliable Services](#).

## Connecting to a Reliable Actor service

This tutorial focused on adding a web front end that communicated with a stateful service. However, you can follow a very similar model to talk to actors. When you create a Reliable Actor project, Visual Studio automatically generates an interface project for you. You can use that interface to generate an actor proxy in the web project to communicate with the actor. The communication channel is provided automatically. So you do not need to do anything that is equivalent to establishing a `ServiceRemotingListener` like you did for the stateful service in this tutorial.

## How web services work on your local cluster

In general, you can deploy exactly the same Service Fabric application to a multi-machine cluster that you deployed on your local cluster and be highly confident that it works as you expect. This is because your local cluster is simply a five-node configuration that is collapsed to a single machine.

When it comes to web services, however, there is one key nuance. When your cluster sits behind a load balancer, as it does in Azure, you must ensure that your web services are deployed on every machine since the load balancer simply round-robs traffic across the machines. You can do this by setting the `InstanceCount` for the service to the special value of "-1".

By contrast, when you run a web service locally, you need to ensure that only one instance of the service is running. Otherwise, you run into conflicts from multiple processes that are listening on the same path and port. As a result, the web service instance count should be set to "1" for local deployments.

To learn how to configure different values for different environment, see [Managing application parameters for multiple environments](#).

## Next steps

Now that you have a web front end set up for your application with ASP.NET Core, learn more about [ASP.NET Core in Service Fabric Reliable Services](#) for an in-depth understanding of how ASP.NET Core works with Service Fabric.

Next, [learn more about communicating with services](#) in general to get a complete picture of how service communication works in Service Fabric.

Once you have a good understanding of how service communication works, [create a cluster in Azure and deploy your application to the cloud](#).

# Managing secrets in Service Fabric applications

6/30/2017 • 5 min to read • [Edit Online](#)

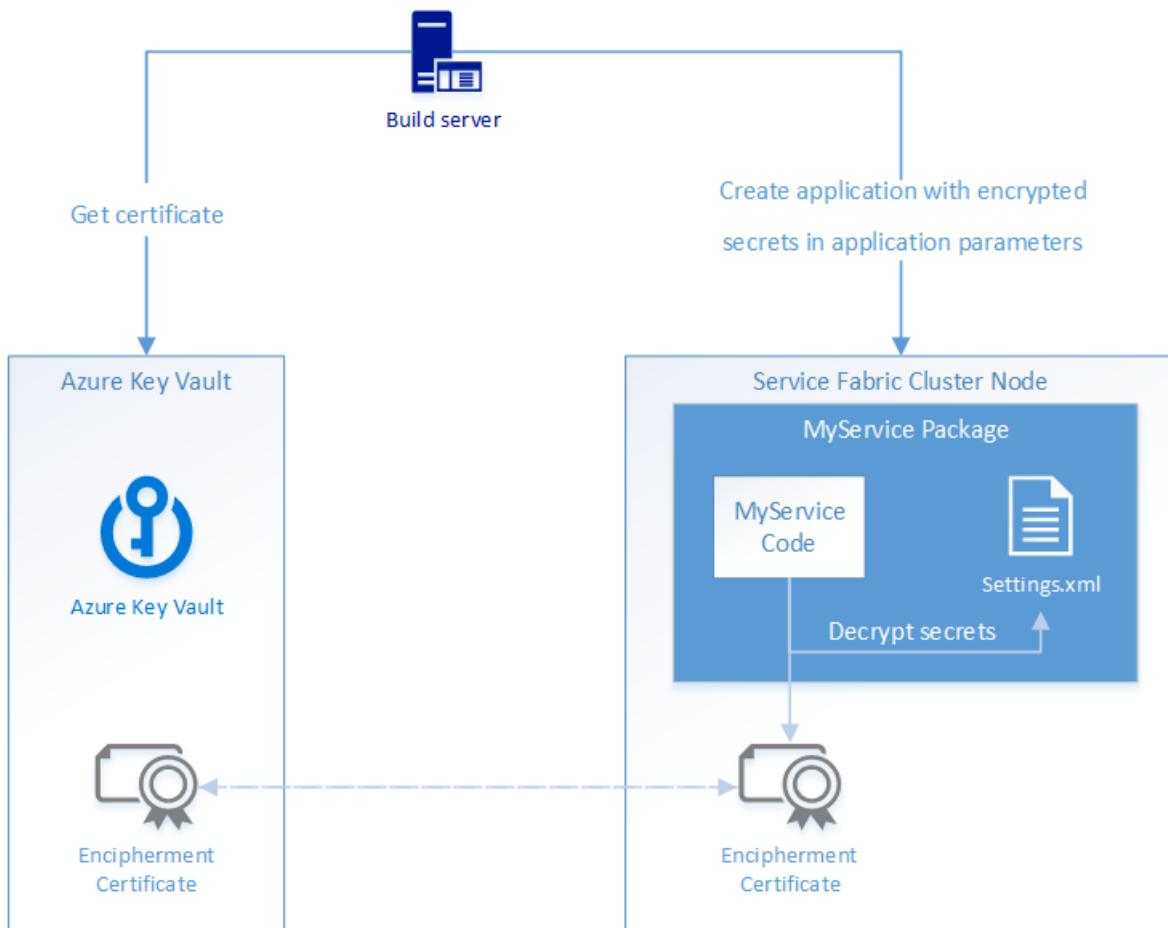
This guide walks you through the steps of managing secrets in a Service Fabric application. Secrets can be any sensitive information, such as storage connection strings, passwords, or other values that should not be handled in plain text.

This guide uses Azure Key Vault to manage keys and secrets. However, *using* secrets in an application is cloud platform-agnostic to allow applications to be deployed to a cluster hosted anywhere.

## Overview

The recommended way to manage service configuration settings is through [service configuration packages](#). Configuration packages are versioned and updatable through managed rolling upgrades with health-validation and auto rollback. This is preferred to global configuration as it reduces the chances of a global service outage. Encrypted secrets are no exception. Service Fabric has built-in features for encrypting and decrypting values in a configuration package Settings.xml file using certificate encryption.

The following diagram illustrates the basic flow for secret management in a Service Fabric application:



There are four main steps in this flow:

1. Obtain a data encipherment certificate.
2. Install the certificate in your cluster.
3. Encrypt secret values when deploying an application with the certificate and inject them into a service's Settings.xml configuration file.

4. Read encrypted values out of Settings.xml by decrypting with the same encipherment certificate.

Azure Key Vault is used here as a safe storage location for certificates and as a way to get certificates installed on Service Fabric clusters in Azure. If you are not deploying to Azure, you do not need to use Key Vault to manage secrets in Service Fabric applications.

## Data encipherment certificate

A data encipherment certificate is used strictly for encryption and decryption of configuration values in a service's Settings.xml and is not used for authentication or signing of cipher text. The certificate must meet the following requirements:

- The certificate must contain a private key.
- The certificate must be created for key exchange, exportable to a Personal Information Exchange (.pfx) file.
- The certificate key usage must include Data Encipherment (10), and should not include Server Authentication or Client Authentication.

For example, when creating a self-signed certificate using PowerShell, the `KeyUsage` flag must be set to `DataEncipherment`:

```
New-SelfSignedCertificate -Type DocumentEncryptionCert -KeyUsage DataEncipherment -Subject mydataenciphermentcert -Provider 'Microsoft Enhanced Cryptographic Provider v1.0'
```

## Install the certificate in your cluster

This certificate must be installed on each node in the cluster. It will be used at runtime to decrypt values stored in a service's Settings.xml. See [how to create a cluster using Azure Resource Manager](#) for setup instructions.

## Encrypt application secrets

The Service Fabric SDK has built-in secret encryption and decryption functions. Secret values can be encrypted at build-time and then decrypted and read programmatically in service code.

The following PowerShell command is used to encrypt a secret. This command only encrypts the value; it does **not** sign the cipher text. You must use the same encipherment certificate that is installed in your cluster to produce ciphertext for secret values:

```
Invoke-ServiceFabricEncryptText -CertStore -CertThumbprint "<thumbprint>" -Text "mysecret" -StoreLocation CurrentUser -StoreName My
```

The resulting base-64 string contains both the secret ciphertext as well as information about the certificate that was used to encrypt it. The base-64 encoded string can be inserted into a parameter in your service's Settings.xml configuration file with the `IsEncrypted` attribute set to `true`:

```
<?xml version="1.0" encoding="utf-8" ?>
<Settings xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://schemas.microsoft.com/2011/01/fabric">
  <Section Name="MySettings">
    <Parameter Name="MySecret" IsEncrypted="true" Value="I6jCCAeYCAxgFhBXABFxzAt ... gNBRyeWFXl2VydmjZNwJIM=" />
  </Section>
</Settings>
```

## Inject application secrets into application instances

Ideally, deployment to different environments should be as automated as possible. This can be accomplished by performing secret encryption in a build environment and providing the encrypted secrets as parameters when creating application instances.

#### Use overridable parameters in Settings.xml

The Settings.xml configuration file allows overridable parameters that can be provided at application creation time. Use the `MustOverride` attribute instead of providing a value for a parameter:

```
<?xml version="1.0" encoding="utf-8" ?>
<Settings xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://schemas.microsoft.com/2011/01/fabric">
  <Section Name="MySettings">
    <Parameter Name="MySecret" IsEncrypted="true" Value="" MustOverride="true" />
  </Section>
</Settings>
```

To override values in Settings.xml, declare an override parameter for the service in ApplicationManifest.xml:

```
<ApplicationManifest ... >
  <Parameters>
    <Parameter Name="MySecret" DefaultValue="" />
  </Parameters>
  <ServiceManifestImport>
    <ServiceManifestRef ServiceManifestName="Stateful1Pkg" ServiceManifestVersion="1.0.0" />
    <ConfigOverrides>
      <ConfigOverride Name="Config">
        <Settings>
          <Section Name="MySettings">
            <Parameter Name="MySecret" Value="[MySecret]" IsEncrypted="true" />
          </Section>
        </Settings>
      </ConfigOverride>
    </ConfigOverrides>
  </ServiceManifestImport>
```

Now the value can be specified as an *application parameter* when creating an instance of the application. Creating an application instance can be scripted using PowerShell, or written in C#, for easy integration in a build process.

Using PowerShell, the parameter is supplied to the `New-ServiceFabricApplication` command as a [hash table](#):

```
PS C:\Users\vturecek> New-ServiceFabricApplication -ApplicationName fabric:/MyApp -ApplicationTypeName
MyAppType -ApplicationTypeVersion 1.0.0 -ApplicationParameter @{"MySecret" = "I6jCCAeYCAxgFhBXABFxzAt ...
gNBRYeWFX12VydjmjZNwJIM="}
```

Using C#, application parameters are specified in an `ApplicationDescription` as a `NameValuePairCollection`:

```
FabricClient fabricClient = new FabricClient();

NameValuePairCollection applicationParameters = new NameValueCollection();
applicationParameters["MySecret"] = "I6jCCAeYCAxgFhBXABFxzAt ... gNBRYeWFX12VydjmjZNwJIM=";

ApplicationDescription applicationDescription = new ApplicationDescription(
    applicationName: new Uri("fabric:/MyApp"),
    applicationTypeName: "MyAppType",
    applicationTypeVersion: "1.0.0",
    applicationParameters: applicationParameters
);

await fabricClient.ApplicationManager.CreateApplicationAsync(applicationDescription);
```

## Decrypt secrets from service code

Services in Service Fabric run under NETWORK SERVICE by default on Windows and don't have access to certificates installed on the node without some extra setup.

When using a data encipherment certificate, you need to make sure NETWORK SERVICE or whatever user account the service is running under has access to the certificate's private key. Service Fabric will handle granting access for your service automatically if you configure it to do so. This configuration can be done in ApplicationManifest.xml by defining users and security policies for certificates. In the following example, the NETWORK SERVICE account is given read access to a certificate defined by its thumbprint:

```
<ApplicationManifest ... >
  <Principals>
    <Users>
      <User Name="Service1" AccountType="NetworkService" />
    </Users>
  </Principals>
  <Policies>
    <SecurityAccessPolicies>
      <SecurityAccessPolicy GrantRights="Read" PrincipalRef="Service1" ResourceRef="MyCert"
ResourceType="Certificate"/>
    </SecurityAccessPolicies>
  </Policies>
  <Certificates>
    <SecretsCertificate Name="MyCert" X509FindType="FindByThumbprint" X509FindValue="[YourCertThumbprint]"/>
  </Certificates>
</ApplicationManifest>
```

### NOTE

When copying a certificate thumbprint from the certificate store snap-in on Windows, an invisible character is placed at the beginning of the thumbprint string. This invisible character can cause an error when trying to locate a certificate by thumbprint, so be sure to delete this extra character.

## Use application secrets in service code

The API for accessing configuration values from Settings.xml in a configuration package allows for easy decrypting of values that have the `IsEncrypted` attribute set to `true`. Since the encrypted text contains information about the certificate used for encryption, you do not need to manually find the certificate. The certificate just needs to be installed on the node that the service is running on. Simply call the `DecryptValue()` method to retrieve the original secret value:

```
ConfigurationPackage configPackage =
this.Context.CodePackageActivationContext.GetConfigurationPackageObject("Config");
SecureString mySecretValue =
configPackage.Settings.Sections["MySettings"].Parameters["MySecret"].DecryptValue()
```

## Next Steps

Learn more about [running applications with different security permissions](#)

# Configure security policies for your application

10/2/2017 • 14 min to read • [Edit Online](#)

By using Azure Service Fabric, you can secure applications that are running in the cluster under different user accounts. Service Fabric also helps secure the resources that are used by applications at the time of deployment under the user accounts--for example, files, directories, and certificates. This makes running applications, even in a shared hosted environment, more secure from one another.

By default, Service Fabric applications run under the account that the Fabric.exe process runs under. Service Fabric also provides the capability to run applications under a local user account or local system account, which is specified within the application manifest. Supported local system account types are **LocalUser**, **NetworkService**, **LocalService**, and **LocalSystem**.

When you're running Service Fabric on Windows Server in your datacenter by using the standalone installer, you can use Active Directory domain accounts, including group managed service accounts.

You can define and create user groups so that one or more users can be added to each group to be managed together. This is useful when there are multiple users for different service entry points and they need to have certain common privileges that are available at the group level.

## Configure the policy for a service setup entry point

As described in the [application model](#), the setup entry point, **SetupEntryPoint**, is a privileged entry point that runs with the same credentials as Service Fabric (typically the *NetworkService* account) before any other entry point. The executable that is specified by **EntryPoint** is typically the long-running service host. So having a separate setup entry point avoids having to run the service host executable with high privileges for extended periods of time. The executable that **EntryPoint** specifies is run after **SetupEntryPoint** exits successfully. The resulting process is monitored and restarted, and begins again with **SetupEntryPoint** if it ever terminates or crashes.

The following is a simple service manifest example that shows the **SetupEntryPoint** and the main **EntryPoint** for the service.

```

<?xml version="1.0" encoding="utf-8" ?>
<ServiceManifest Name="MyServiceManifest" Version="SvcManifestVersion1"
  xmlns="http://schemas.microsoft.com/2011/01/fabric" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Description>An example service manifest</Description>
  <ServiceTypes>
    <StatelessServiceType ServiceTypeName="MyServiceType" />
  </ServiceTypes>
  <CodePackage Name="Code" Version="1.0.0">
    <SetupEntryPoint>
      <ExeHost>
        <Program>MySetup.bat</Program>
        <WorkingFolder>CodePackage</WorkingFolder>
      </ExeHost>
    </SetupEntryPoint>
    <EntryPoint>
      <ExeHost>
        <Program>MyServiceHost.exe</Program>
      </ExeHost>
    </EntryPoint>
  </CodePackage>
  <ConfigPackage Name="Config" Version="1.0.0" />
</ServiceManifest>

```

## Configure the policy by using a local account

After you configure the service to have a setup entry point, you can change the security permissions that it runs under in the application manifest. The following example shows how to configure the service to run under user administrator account privileges.

```

<?xml version="1.0" encoding="utf-8"?>
<ApplicationManifest xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ApplicationTypeName="MyApplicationType" ApplicationTypeVersion="1.0.0"
  xmlns="http://schemas.microsoft.com/2011/01/fabric">
  <ServiceManifestImport>
    <ServiceManifestRef ServiceManifestName="MyServiceTypePkg" ServiceManifestVersion="1.0.0" />
    <ConfigOverrides />
    <Policies>
      <RunAsPolicy CodePackageRef="Code" UserRef="SetupAdminUser" EntryPointType="Setup" />
    </Policies>
  </ServiceManifestImport>
  <Principals>
    <Users>
      <User Name="SetupAdminUser">
        <MemberOf>
          <SystemGroup Name="Administrators" />
        </MemberOf>
      </User>
    </Users>
  </Principals>
</ApplicationManifest>

```

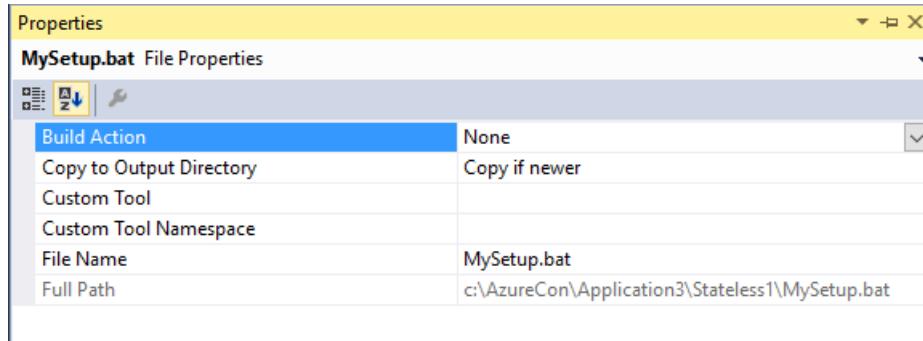
First, create a **Principals** section with a user name, such as SetupAdminUser. This indicates that the user is a member of the Administrators system group.

Next, under the **ServiceManifestImport** section, configure a policy to apply this principal to **SetupEntryPoint**. This tells Service Fabric that when the **MySetup.bat** file is run, it should be **RunAs** with administrator privileges. Given that you have *not* applied a policy to the main entry point, the code in **MyServiceHost.exe** runs under the system **NetworkService** account. This is the default account that all service entry points are run as.

Let's now add the file MySetup.bat to the Visual Studio project to test the administrator privileges. In Visual Studio, right-click the service project and add a new file called MySetup.bat.

Next, ensure that the MySetup.bat file is included in the service package. By default, it is not. Select the file, right-

click to get the context menu, and choose **Properties**. In the Properties dialog box, ensure that **Copy to Output Directory** is set to **Copy if newer**. See the following screenshot.



Now open the MySetup.bat file and add the following commands:

```
REM Set a system environment variable. This requires administrator privilege
setx -m TestVariable "MyValue"
echo System TestVariable set to > out.txt
echo %TestVariable% >> out.txt

REM To delete this system variable us
REM REG delete "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Environment" /v
TestVariable /f
```

Next, build and deploy the solution to a local development cluster. After the service has started, as shown in Service Fabric Explorer, you can see that the MySetup.bat file was successful in a two ways. Open a PowerShell command prompt and type:

```
PS C:\ [Environment]::GetEnvironmentVariable("TestVariable","Machine")
MyValue
```

Then, note the name of the node where the service was deployed and started in Service Fabric Explorer--for example, Node 2. Next, navigate to the application instance work folder to find the out.txt file that shows the value of **TestVariable**. For example, if this service was deployed to Node 2, then you can go to this path for the **MyApplicationType**:

```
C:\SfDevCluster\Data\_App\Node.2\MyApplicationType_App\work\out.txt
```

### Configure the policy by using local system accounts

Often, it's preferable to run the startup script by using a local system account rather than an administrator account. Running the RunAs policy as a member of the Administrators group typically doesn't work well because machines have User Access Control (UAC) enabled by default. In such cases, **the recommendation is to run the SetupEntryPoint as LocalSystem, instead of as a local user added to Administrators group**. The following example shows setting the SetupEntryPoint to run as LocalSystem:

```

<?xml version="1.0" encoding="utf-8"?>
<ApplicationManifest xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ApplicationTypeName="MyApplicationType" ApplicationTypeVersion="1.0.0"
xmlns="http://schemas.microsoft.com/2011/01/fabric">
  <ServiceManifestImport>
    <ServiceManifestRef ServiceManifestName="MyServiceTypePkg" ServiceManifestVersion="1.0.0" />
    <ConfigOverrides />
    <Policies>
      <RunAsPolicy CodePackageRef="Code" UserRef="SetupLocalSystem" EntryPointType="Setup" />
    </Policies>
  </ServiceManifestImport>
  <Principals>
    <Users>
      <User Name="SetupLocalSystem" AccountType="LocalSystem" />
    </Users>
  </Principals>
</ApplicationManifest>

```

For Linux clusters, to run a service or the setup entry point as **root**, you can specify the **AccountType** as **LocalSystem**.

## Start PowerShell commands from a setup entry point

To run PowerShell from the **SetupEntryPoint** point, you can run **PowerShell.exe** in a batch file that points to a PowerShell file. First, add a PowerShell file to the service project--for example, **MySetup.ps1**. Remember to set the *Copy if newer* property so that the file is also included in the service package. The following example shows a sample batch file that starts a PowerShell file called MySetup.ps1, which sets a system environment variable called **TestVariable**.

MySetup.bat to start a PowerShell file:

```
powershell.exe -ExecutionPolicy Bypass -Command ".\MySetup.ps1"
```

In the PowerShell file, add the following to set a system environment variable:

```
[Environment]::SetEnvironmentVariable("TestVariable", "MyValue", "Machine")
[Environment]::GetEnvironmentVariable("TestVariable","Machine") > out.txt
```

### NOTE

By default, when the batch file runs, it looks at the application folder called **work** for files. In this case, when MySetup.bat runs, we want this to find the MySetup.ps1 file in the same folder, which is the application **code package** folder. To change this folder, set the working folder:

```

<SetupEntryPoint>
  <ExeHost>
    <Program>MySetup.bat</Program>
    <WorkingFolder>CodePackage</WorkingFolder>
  </ExeHost>
</SetupEntryPoint>

```

## Use console redirection for local debugging

Occasionally, it's useful to see the console output from running a script for debugging purposes. To do this, you can set a console redirection policy, which writes the output to a file. The file output is written to the application

folder called **log** on the node where the application is deployed and run. (See where to find this in the preceding example.)

#### WARNING

Never use the console redirection policy in an application that is deployed in production because this can affect the application failover. *Only* use this for local development and debugging purposes.

The following example shows setting the console redirection with a FileRetentionCount value:

```
<SetupEntryPoint>
  <ExeHost>
    <Program>MySetup.bat</Program>
    <WorkingFolder>CodePackage</WorkingFolder>
    <ConsoleRedirection FileRetentionCount="10"/>
  </ExeHost>
</SetupEntryPoint>
```

If you now change the MySetup.ps1 file to write an **Echo** command, this will write to the output file for debugging purposes:

```
Echo "Test console redirection which writes to the application log folder on the node that the application is
deployed to"
```

**After you debug your script, immediately remove this console redirection policy.**

## Configure a policy for service code packages

In the preceding steps, you saw how to apply a RunAs policy to SetupEntryPoint. Let's look a little deeper into how to create different principals that can be applied as service policies.

### Create local user groups

You can define and create user groups that allow one or more users to be added to a group. This is useful if there are multiple users for different service entry points and they need to have certain common privileges that are available at the group level. The following example shows a local group called **LocalAdminGroup** that has administrator privileges. Two users, Customer1 and Customer2, are made members of this local group.

```

<Principals>
  <Groups>
    <Group Name="LocalAdminGroup">
      <Membership>
        <SystemGroup Name="Administrators"/>
      </Membership>
    </Group>
  </Groups>
  <Users>
    <User Name="Customer1">
      <MemberOf>
        <Group NameRef="LocalAdminGroup" />
      </MemberOf>
    </User>
    <User Name="Customer2">
      <MemberOf>
        <Group NameRef="LocalAdminGroup" />
      </MemberOf>
    </User>
  </Users>
</Principals>

```

## Create local users

You can create a local user that can be used to help secure a service within the application. When a **LocalUser** account type is specified in the principals section of the application manifest, Service Fabric creates local user accounts on machines where the application is deployed. By default, these accounts do not have the same names as those specified in the application manifest (for example, Customer3 in the following sample). Instead, they are dynamically generated and have random passwords.

```

<Principals>
  <Users>
    <User Name="Customer3" AccountType="LocalUser" />
  </Users>
</Principals>

```

If an application requires that the user account and password be same on all machines (for example, to enable NTLM authentication), the cluster manifest must set NTLMAuthenticationEnabled to true. The cluster manifest must also specify an NTLMAuthenticationPasswordSecret that is used to generate the same password across all machines.

```

<Section Name="Hosting">
  <Parameter Name="EndpointProviderEnabled" Value="true"/>
  <Parameter Name="NTLMAuthenticationEnabled" Value="true"/>
  <Parameter Name="NTLMAuthenticationPassworkSecret" Value="*****" IsEncrypted="true"/>
</Section>

```

## Assign policies to the service code packages

The **RunAsPolicy** section for a **ServiceManifestImport** specifies the account from the principals section that should be used to run a code package. It also associates code packages from the service manifest with user accounts in the principals section. You can specify this for the setup or main entry points, or you can specify **All** to apply it to both. The following example shows different policies being applied:

```

<Policies>
  <RunAsPolicy CodePackageRef="Code" UserRef="LocalAdmin" EntryPointType="Setup"/>
  <RunAsPolicy CodePackageRef="Code" UserRef="Customer3" EntryPointType="Main"/>
</Policies>

```

If **EntryPointType** is not specified, the default is set to EntryPointType="Main". Specifying **SetupEntryPoint** is especially useful when you want to run certain high-privilege setup operations under a system account. The actual service code can run under a lower-privilege account.

### Apply a default policy to all service code packages

You use the **DefaultRunAsPolicy** section to specify a default user account for all code packages that don't have a specific **RunAsPolicy** defined. If most of the code packages that are specified in the service manifest used by an application need to run under the same user, the application can just define a default RunAs policy with that user account. The following example specifies that if a code package does not have a **RunAsPolicy** specified, the code package should run under the **MyDefaultAccount** specified in the principals section.

```
<Policies>
  <DefaultRunAsPolicy UserRef="MyDefaultAccount"/>
</Policies>
```

### Use an Active Directory domain group or user

For an instance of Service Fabric that was installed on Windows Server by using the standalone installer, you can run the service under the credentials for an Active Directory user or group account. This is Active Directory on-premises within your domain and is not with Azure Active Directory (Azure AD). By using a domain user or group, you can then access other resources in the domain (for example, file shares) that have been granted permissions.

The following example shows an Active Directory user called *TestUser* with their domain password encrypted by using a certificate called *MyCert*. You can use the `Invoke-ServiceFabricEncryptText` PowerShell command to create the secret cipher text. See [Managing secrets in Service Fabric applications](#) for details.

You must deploy the private key of the certificate to decrypt the password to the local machine by using an out-of-band method (in Azure, this is via Azure Resource Manager). Then, when Service Fabric deploys the service package to the machine, it is able to decrypt the secret and (along with the user name) authenticate with Active Directory to run under those credentials.

```
<Principals>
  <Users>
    <User Name="TestUser" AccountType="DomainUser" AccountName="Domain\User" Password="[Put encrypted password here using MyCert certificate]" PasswordEncrypted="true" />
  </Users>
</Principals>
<Policies>
  <DefaultRunAsPolicy UserRef="TestUser" />
  <SecurityAccessPolicies>
    <SecurityAccessPolicy ResourceRef="MyCert" PrincipalRef="TestUser" GrantRights="Full" ResourceType="Certificate" />
  </SecurityAccessPolicies>
</Policies>
<Certificates>
```

### Use a Group Managed Service Account.

For an instance of Service Fabric that was installed on Windows Server by using the standalone installer, you can run the service as a group Managed Service Account (gMSA). Note that this is Active Directory on-premises within your domain and is not with Azure Active Directory (Azure AD). By using a gMSA there is no password or encrypted password stored in the `Application Manifest`.

The following example shows how to create a gMSA account called *svc-Test\$*; how to deploy that managed service account to the cluster nodes; and how to configure the user principal.

#### Prerequisites.

- The domain needs a KDS root key.

- The domain needs to be at a Windows Server 2012 or later functional level.

#### Example

- Have an active directory domain administrator create a group managed service account using the `New-ADServiceAccount` commandlet and ensure that the `PrincipalsAllowedToRetrieveManagedPassword` includes all of the service fabric cluster nodes. Note that `AccountName`, `DnsHostName`, and `ServicePrincipalName` must be unique.

```
New-ADServiceAccount -name svc-Test$ -DnsHostName svc-test.contoso.com -ServicePrincipalNames http/svc-test.contoso.com -PrincipalsAllowedToRetrieveManagedPassword SfNode0$,SfNode1$,SfNode2$,SfNode3$,SfNode4$
```

- On each of the service fabric cluster nodes (for example, `SfNode0$,SfNode1$,SfNode2$,SfNode3$,SfNode4$`), install and test the gMSA.

```
Add-WindowsFeature RSAT-AD-PowerShell Install-AdServiceAccount svc-Test$ Test-AdServiceAccount svc-Test$
```

- Configure the User principal, and configure the RunAsPolicy to reference the user.

```
xml <?xml version="1.0" encoding="utf-8"?> <ApplicationManifest xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ApplicationTypeName="MyApplicationType" ApplicationTypeVersion="1.0.0" xmlns="http://schemas.microsoft.com/2011/01/fabric"> <ServiceManifestImport> <ServiceManifestRef ServiceManifestName="MyServiceTypePkg" ServiceManifestVersion="1.0.0" /> <ConfigOverrides /> <Policies> <RunAsPolicy CodePackageRef="Code" UserRef="DomaingMSA"/> </Policies> </ServiceManifestImport> <Principals> <Users> <User Name="DomaingMSA" AccountType="ManagedServiceAccount" AccountName="domain\svc-Test$"/> </Users> </Principals> </ApplicationManifest>
```

## Assign a security access policy for HTTP and HTTPS endpoints

If you apply a RunAs policy to a service and the service manifest declares endpoint resources with the HTTP protocol, you must specify a **SecurityAccessPolicy** to ensure that ports allocated to these endpoints are correctly access-control listed for the RunAs user account that the service runs under. Otherwise, **http.sys** does not have access to the service, and you get failures with calls from the client. The following example applies the Customer1 account to an endpoint called **EndpointName**, which gives it full access rights.

```
<Policies>
  <RunAsPolicy CodePackageRef="Code" UserRef="Customer1" />
  <!--SecurityAccessPolicy is needed if RunAsPolicy is defined and the Endpoint is http -->
  <SecurityAccessPolicy ResourceRef="EndpointName" PrincipalRef="Customer1" />
</Policies>
```

For the HTTPS endpoint, you also have to indicate the name of the certificate to return to the client. You can do this by using **EndpointBindingPolicy**, with the certificate defined in a certificates section in the application manifest.

```
<Policies>
  <RunAsPolicy CodePackageRef="Code" UserRef="Customer1" />
  <!--SecurityAccessPolicy is needed if RunAsPolicy is defined and the Endpoint is http -->
  <SecurityAccessPolicy ResourceRef="EndpointName" PrincipalRef="Customer1" />
  <!--EndpointBindingPolicy is needed if the EndpointName is secured with https -->
  <EndpointBindingPolicy EndpointRef="EndpointName" CertificateRef="Cert1" />
</Policies>
```

## Upgrading multiple applications with https endpoints

You need to be careful not to use the **same port** for different instances of the same application when using https. The reason is that Service Fabric won't be able to upgrade the cert for one of the application instances. For example, if application 1 or application 2 both want to upgrade their cert 1 to cert 2. When the upgrade happens, Service Fabric might have cleaned up the cert 1 registration with http.sys even though the other application is still using it. To prevent this, Service Fabric detects that there is already another application instance registered on the port with the certificate (due to http.sys) and fails the operation.

Hence Service Fabric does not support upgrading two different services using **the same port** in different

application instances. In other words, you cannot use the same certificate on different services on the same port. If you need to have a shared certificate on the same port, you need to ensure that the services are placed on different machines with placement constraints. Or consider using Service Fabric dynamic ports if possible for each service in each application instance.

If you see an upgrade fail with https, an error warning saying "The Windows HTTP Server API does not support multiple certificates for applications that share a port."

## A complete application manifest example

The following application manifest shows many of the different settings:

```
<?xml version="1.0" encoding="utf-8"?>
<ApplicationManifest xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ApplicationTypeName="Application3Type" ApplicationTypeVersion="1.0.0"
xmlns="http://schemas.microsoft.com/2011/01/fabric">
    <Parameters>
        <Parameter Name="Stateless1_InstanceCount" DefaultValue="-1" />
    </Parameters>
    <ServiceManifestImport>
        <ServiceManifestRef ServiceManifestName="Stateless1Pkg" ServiceManifestVersion="1.0.0" />
        <ConfigOverrides />
    <Policies>
        <RunAsPolicy CodePackageRef="Code" UserRef="Customer1" />
        <RunAsPolicy CodePackageRef="Code" UserRef="LocalAdmin" EntryPointType="Setup" />
        <!--SecurityAccessPolicy is needed if RunAsPolicy is defined and the Endpoint is http -->
        <SecurityAccessPolicy ResourceRef="EndpointName" PrincipalRef="Customer1" />
        <!--EndpointBindingPolicy is needed the EndpointName is secured with https -->
        <EndpointBindingPolicy EndpointRef="EndpointName" CertificateRef="Cert1" />
    </Policies>
    </ServiceManifestImport>
    <DefaultServices>
        <Service Name="Stateless1">
            <StatelessService ServiceTypeName="Stateless1Type" InstanceCount="[Stateless1_InstanceCount]">
                <SingletonPartition />
            </StatelessService>
        </Service>
    </DefaultServices>
    <Principals>
        <Groups>
            <Group Name="LocalAdminGroup">
                <Membership>
                    <SystemGroup Name="Administrators" />
                </Membership>
            </Group>
        </Groups>
        <Users>
            <User Name="LocalAdmin">
                <MemberOf>
                    <Group NameRef="LocalAdminGroup" />
                </MemberOf>
            </User>
            <!--Customer1 below create a local account that this service runs under -->
            <User Name="Customer1" />
            <User Name="MyDefaultAccount" AccountType="NetworkService" />
        </Users>
    </Principals>
    <Policies>
        <DefaultRunAsPolicy UserRef="LocalAdmin" />
    </Policies>
    <Certificates>
        <EndpointCertificate Name="Cert1" X509FindValue="FF EE E0 TT JJ DD JJ EE EE XX 23 4T 66 "/">
    </Certificates>
</ApplicationManifest>
```

## Next steps

- [Understand the application model](#)
- [Specify resources in a service manifest](#)
- [Deploy an application](#)

# Use Visual Studio to simplify writing and managing your Service Fabric applications

6/27/2017 • 3 min to read • [Edit Online](#)

You can manage your Azure Service Fabric applications and services through Visual Studio. Once you've [set up your development environment](#), you can use Visual Studio to create Service Fabric applications, add services, or package, register, and deploy applications in your local development cluster.

## Deploy your Service Fabric application

By default, deploying an application combines the following steps into one simple operation:

1. Creating the application package
2. Uploading the application package to the image store
3. Registering the application type
4. Removing any running application instances
5. Creating an application instance

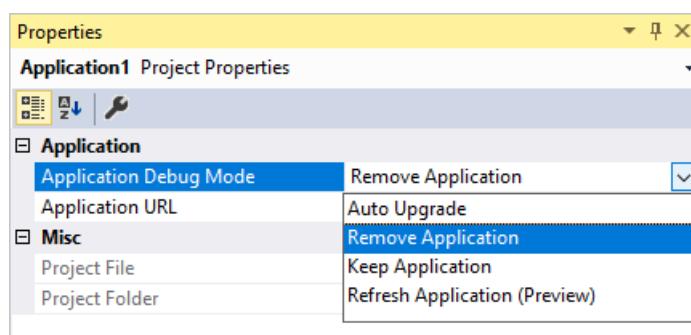
In Visual Studio, pressing **F5** deploys your application and attach the debugger to all application instances. You can use **Ctrl+F5** to deploy an application without debugging, or you can publish to a local or remote cluster by using the publish profile. For more information, see [Publish an application to a remote cluster by using Visual Studio](#).

### Application Debug Mode

Visual Studio provide a property called **Application Debug Mode**, which controls how you want Visual Studios to handle Application deployment as part of debugging.

#### To set the Application Debug Mode property

1. On the Service Fabric application project's (.sfproj) shortcut menu, choose \*\*Properties\*\* (or press the **F4** key).
2. In the **Properties** window, set the **Application Debug Mode** property.



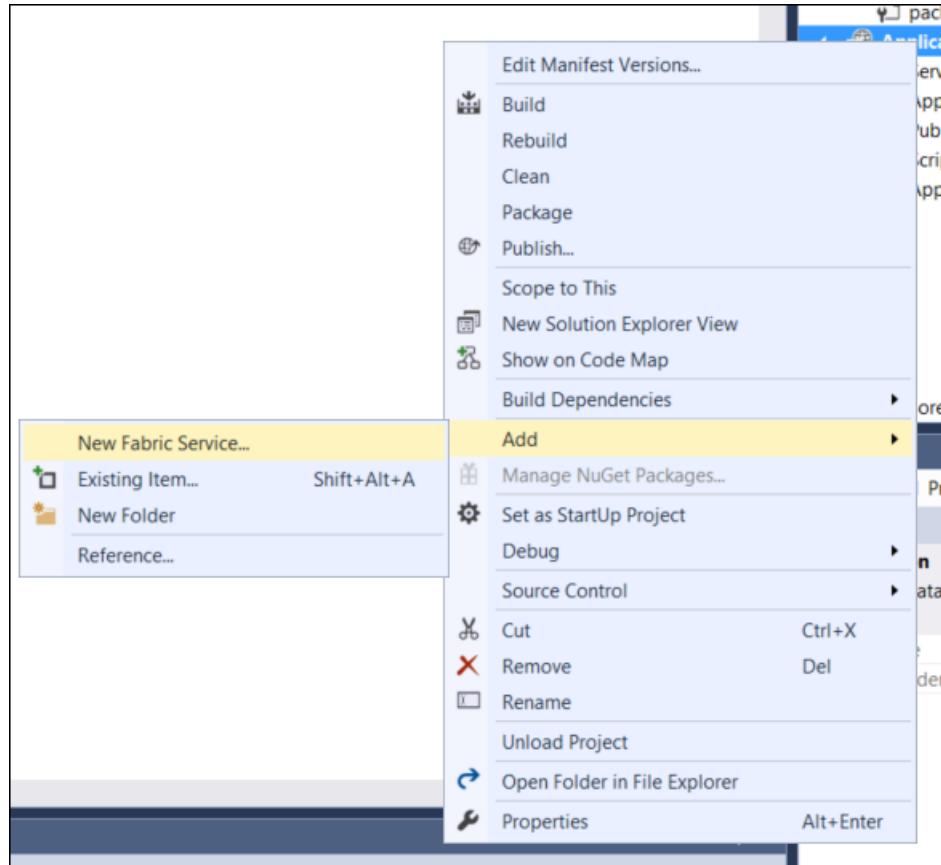
#### Application Debug Modes

1. **Refresh Application** This mode enables you to quickly change and debug your code and supports editing static web files while debugging. This mode only works if your local development cluster is in [1-Node mode](#).
2. **Remove Application** causes the application to be removed when the debug session ends.
3. **Auto Upgrade** The application continues to run when the debug session ends. The next debug session will treat the deployment as an upgrade. The upgrade process preserves any data that you entered in a previous debug session.
4. **Keep Application** The application keeps running in the cluster when the debug session ends. At the beginning of the next debug session, the application will be removed.

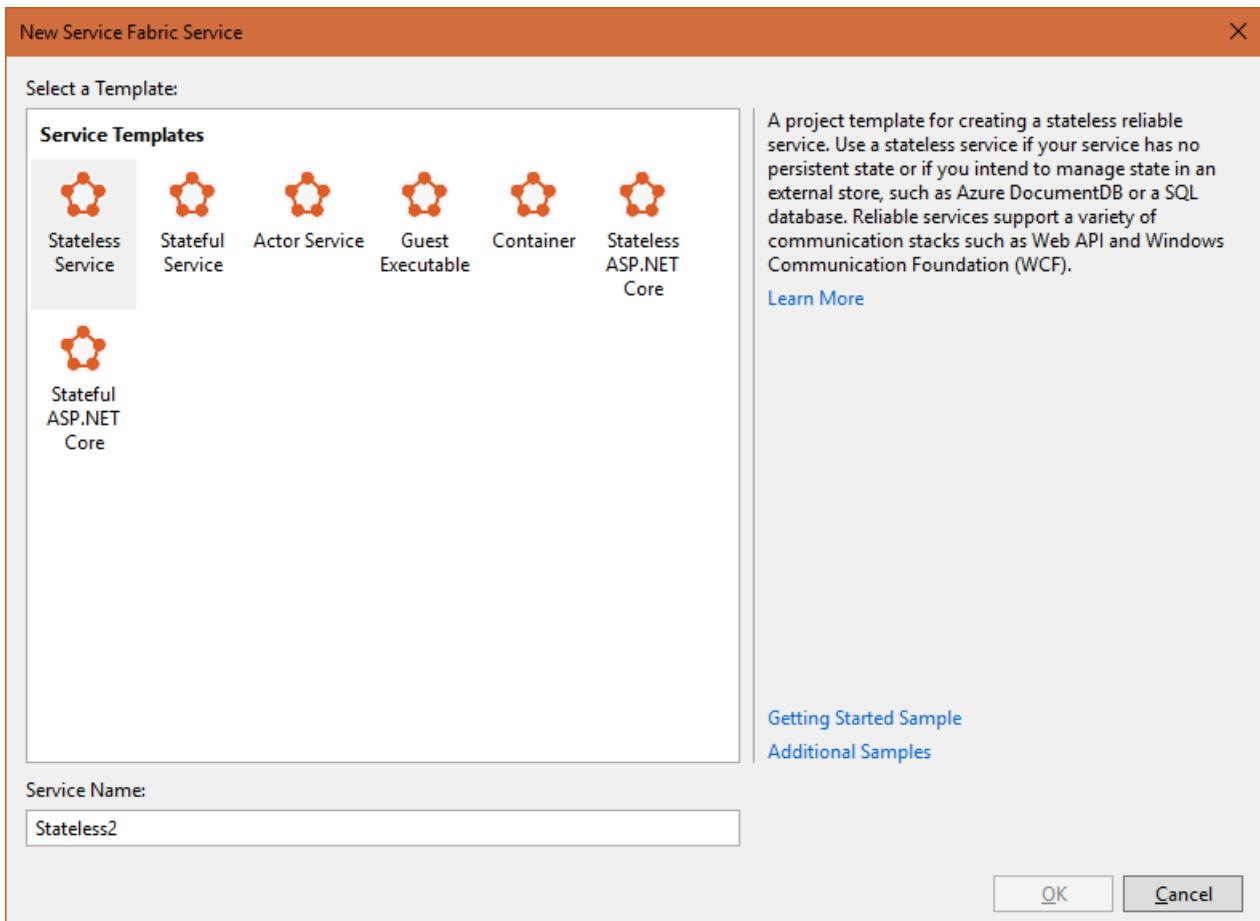
For **Auto Upgrade** data is preserved by applying the application upgrade capabilities of Service Fabric. For more information about upgrading applications and how you might perform an upgrade in a real environment, see [Service Fabric application upgrade](#).

## Add a service to your Service Fabric application

You can add new services to your application to extend its functionality. To ensure that the service is included in your application package, add the service through the **New Fabric Service...** menu item.



Select a Service Fabric project type to add to your application, and specify a name for the service. See [Choosing a framework for your service](#) to help you decide which service type to use.



The new service is added to your solution and existing application package. The service references and a default service instance will be added to the application manifest, causing the service to be created and started the next time you deploy the application.

```

ApplicationManifest.xml* # X
<?xml version="1.0" encoding="utf-8"?>
<ApplicationManifest ApplicationTypeName="Application56Type"
    ApplicationTypeVersion="1.0.0.0"
    xmlns="http://schemas.microsoft.com/2011/01/fabric"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Parameters>
        <Parameter Name="MyFrontEnd_InstanceCount" DefaultValue="1" />
        <Parameter Name="Stateless1_InstanceCount" DefaultValue="1" />
    </Parameters>
    <!-- Import the ServiceManifest from the ServicePackage. The ServiceManifestName and ServiceManifestVersion
        should match the Name and Version attributes of the ServiceManifest element defined in the
        ServiceManifest.xml file. -->
    <ServiceManifestImport>
        <ServiceManifestRef ServiceManifestName="MyFrontEndPkg" ServiceManifestVersion="1.0.0.0" />
        <ConfigOverrides />
    </ServiceManifestImport>
    <ServiceManifestImport>
        <ServiceManifestRef ServiceManifestName="Stateless1Pkg" ServiceManifestVersion="1.0.0.0" />
        <ConfigOverrides />
    </ServiceManifestImport>
    <DefaultServices>
        <!-- The section below creates instances of service types, when an instance of this
            application type is created. You can also create one or more instances of service type using the
            ServiceFabric PowerShell module.

            The attribute ServiceTypeName below must match the name defined in the imported ServiceManifest.xml file. -->
        <Service Name="MyFrontEnd">
            <StatelessService ServiceTypeName="MyFrontEndType" InstanceCount="[MyFrontEnd_InstanceCount]">
                <SingletonPartition />
            </StatelessService>
        </Service>
        <Service Name="Stateless1">
            <StatelessService ServiceTypeName="Stateless1Type" InstanceCount="[Stateless1_InstanceCount]">
                <SingletonPartition />
            </StatelessService>
        </Service>
    </DefaultServices>
</ApplicationManifest>

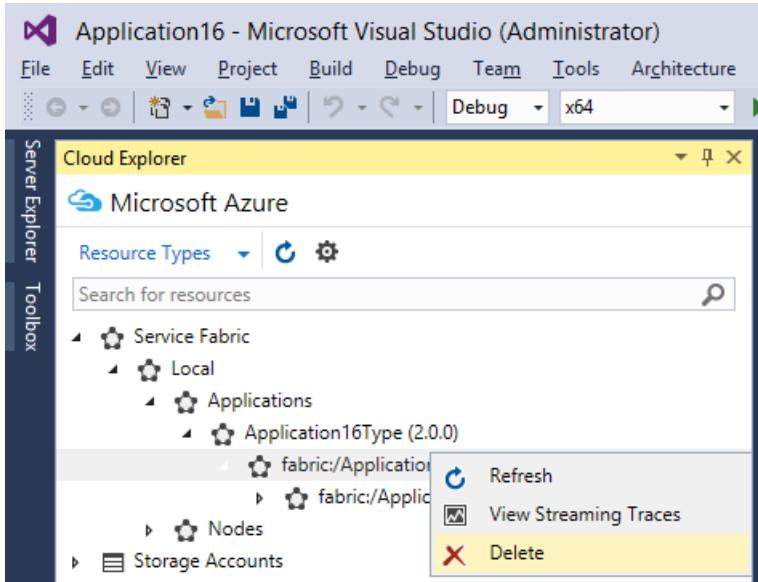
```

## Package your Service Fabric application

To deploy the application and its services to a cluster, you need to create an application package. The package organizes the application manifest, service manifests, and other necessary files in a specific layout. Visual Studio sets up and manages the package in the application project's folder, in the 'pkg' directory. Clicking **Package** from the **Application** context menu creates or updates the application package.

## Remove applications and application types using Cloud Explorer

You can perform basic cluster management operations from within Visual Studio using Cloud Explorer, which you can launch from the **View** menu. For instance, you can delete applications and unprovision application types on local or remote clusters.



### TIP

For richer cluster management functionality, see [Visualizing your cluster with Service Fabric Explorer](#).

## Next steps

- [Service Fabric application model](#)
- [Service Fabric application deployment](#)
- [Managing application parameters for multiple environments](#)
- [Debugging your Service Fabric application](#)
- [Visualizing your cluster by using Service Fabric Explorer](#)

# Configure secure connections to a Service Fabric cluster from Visual Studio

8/7/2017 • 2 min to read • [Edit Online](#)

Learn how to use Visual Studio to securely access an Azure Service Fabric cluster with access control policies configured.

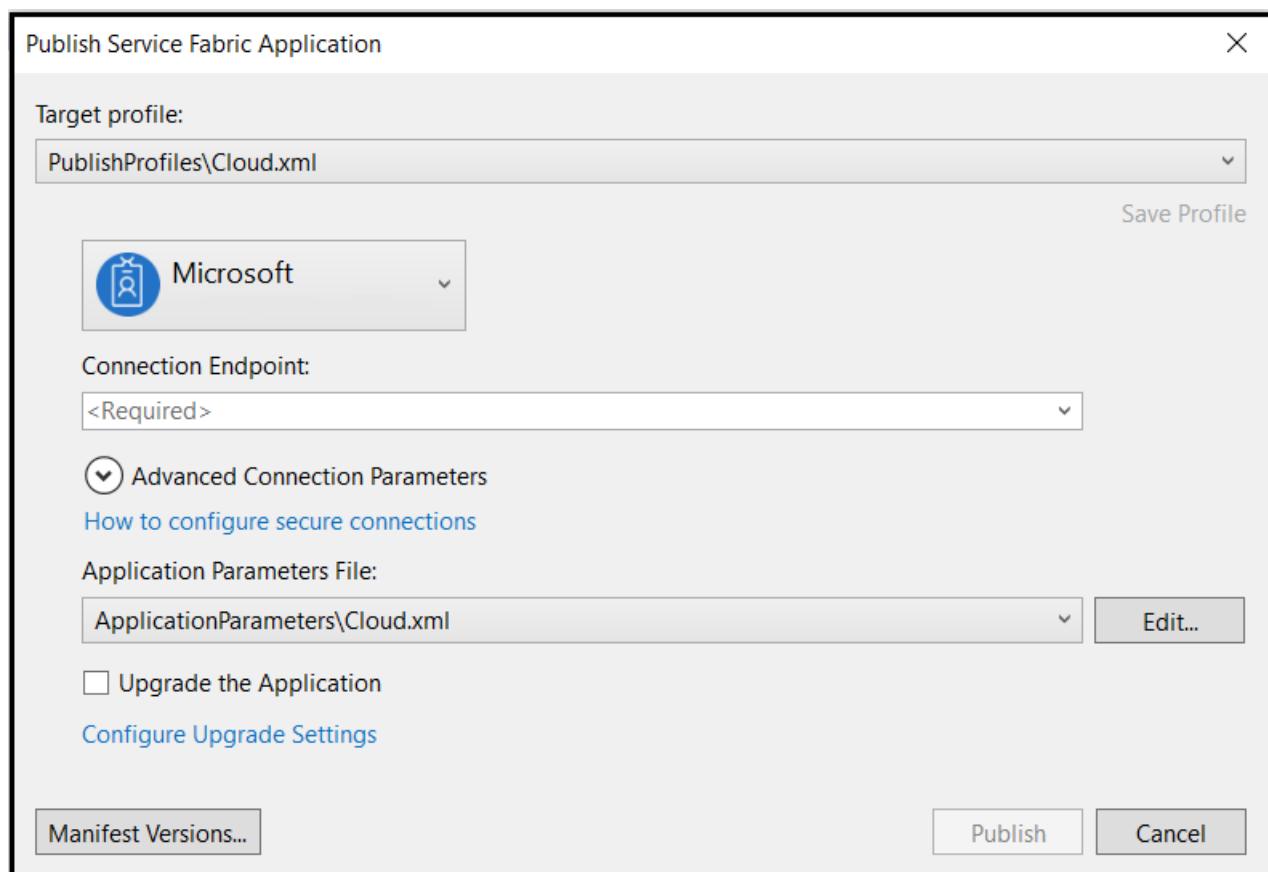
## Cluster connection types

Two types of connections are supported by the Azure Service Fabric cluster: **non-secure** connections and **x509 certificate-based** secure connections. (For Service Fabric clusters hosted on-premises, **Windows** and **dSTS** authentications are also supported.) You have to configure the cluster connection type when the cluster is being created. Once it's created, the connection type can't be changed.

The Visual Studio Service Fabric tools support all authentication types for connecting to a cluster for publishing. See [Setting up a Service Fabric cluster from the Azure portal](#) for instructions on how to set up a secure Service Fabric cluster.

## Configure cluster connections in publish profiles

If you publish a Service Fabric project from Visual Studio, use the **Publish Service Fabric Application** dialog box to choose an Azure Service Fabric cluster. Under **Connection endpoint**, select an existing cluster under your subscription.



The **Publish Service Fabric Application** dialog box automatically validates the cluster connection. If prompted, sign in to your Azure account. If validation passes, it means that your system has the correct certificates installed to

connect to the cluster securely, or your cluster is non-secure. Validation failures can be caused by network issues or by not having your system correctly configured to connect to a secure cluster.



Connection Endpoint:  
:19000 

### To connect to a secure cluster

1. Make sure you can access one of the client certificates that the destination cluster trusts. The certificate is usually shared as a Personal Information Exchange (.pfx) file. See [Setting up a Service Fabric cluster from the Azure portal](#) for how to configure the server to grant access to a client.
2. Install the trusted certificate. To do this, double-click the .pfx file, or use the PowerShell script Import-PfxCertificate to import the certificates. Install the certificate to **Cert:\LocalMachine\My**. It's OK to accept all default settings while importing the certificate.
3. Choose the **Publish...** command on the shortcut menu of the project to open the **Publish Azure Application** dialog box and then select the target cluster. The tool automatically resolves the connection and saves the secure connection parameters in the publish profile.
4. Optional: You can edit the publish profile to specify a secure cluster connection.

Since you're manually editing the Publish Profile XML file to specify the certificate information, be sure to note the certificate store name, store location, and certificate thumbprint. You'll need to provide these values for the certificate's store name and store location. See [How to: Retrieve the Thumbprint of a Certificate](#) for more information.

You can use the *ClusterConnectionParameters* parameters to specify the PowerShell parameters to use when connecting to the Service Fabric cluster. Valid parameters are any that are accepted by the Connect-ServiceFabricCluster cmdlet. See [Connect-ServiceFabricCluster](#) for a list of available parameters.

If you're publishing to a remote cluster, you need to specify the appropriate parameters for that specific cluster. The following is an example of connecting to a non-secure cluster:

```
<ClusterConnectionParameters ConnectionEndpoint="mycluster.westus.cloudapp.azure.com:19000" />
```

Here's an example for connecting to an x509 certificate-based secure cluster:

```
<ClusterConnectionParameters  
ConnectionEndpoint="mycluster.westus.cloudapp.azure.com:19000"  
X509Credential="true"  
ServerCertThumbprint="0123456789012345678901234567890123456789"  
FindType="FindByThumbprint"  
FindValue="9876543210987654321098765432109876543210"  
StoreLocation="CurrentUser"  
StoreName="My" />
```

5. Edit any other necessary settings, such as upgrade parameters and Application Parameter file location, and then publish your application from the **Publish Service Fabric Application** dialog box in Visual Studio.

## Next steps

For more information about accessing Service Fabric clusters, see [Visualizing your cluster by using Service Fabric Explorer](#).

# Manage application parameters for multiple environments

8/18/2017 • 5 min to read • [Edit Online](#)

You can create Azure Service Fabric clusters by using anywhere from one to many thousands of machines. While application binaries can run without modification across this wide spectrum of environments, you often want to configure the application differently, depending on the number of machines you're deploying to.

As a simple example, consider `InstanceCount` for a stateless service. When you are running applications in Azure, you generally want to set this parameter to the special value of "-1". This configuration ensures that your service is running on every node in the cluster (or every node in the node type if you have set a placement constraint). However, this configuration is not suitable for a single-machine cluster since you cannot have multiple processes listening on the same endpoint on a single machine. Instead, you typically set `InstanceCount` to "1".

## Specifying environment-specific parameters

The solution to this configuration issue is a set of parameterized default services and application parameter files that fill in those parameter values for a given environment. Default services and application parameters are configured in the application and service manifests. The schema definition for the `ServiceManifest.xml` and `ApplicationManifest.xml` files is installed with the Service Fabric SDK and tools to `C:\Program Files\Microsoft SDKs\Service Fabric\schemas\ServiceFabricServiceModel.xsd`.

### Default services

Service Fabric applications are made up of a collection of service instances. While it is possible for you to create an empty application and then create all service instances dynamically, most applications have a set of core services that should always be created when the application is instantiated. These are referred to as "default services". They are specified in the application manifest, with placeholders for per-environment configuration included in square brackets:

```
<DefaultServices>
  <Service Name="Stateful1">
    <StatefulService
      ServiceTypeName="Stateful1Type"
      TargetReplicaSetSize="[Stateful1_TargetReplicaSetSize]"
      MinReplicaSetSize="[Stateful1_MinReplicaSetSize]"

      <UniformInt64Partition
        PartitionCount="[Stateful1_PartitionCount]"
        LowKey="-9223372036854775808"
        HighKey="9223372036854775807"
      />
    </StatefulService>
  </Service>
</DefaultServices>
```

Each of the named parameters must be defined within the `Parameters` element of the application manifest:

```

<Parameters>
    <Parameter Name="Stateful1_MinReplicaSetSize" DefaultValue="3" />
    <Parameter Name="Stateful1_PartitionCount" DefaultValue="1" />
    <Parameter Name="Stateful1_TargetReplicaSetSize" DefaultValue="3" />
</Parameters>

```

The `DefaultValue` attribute specifies the value to be used in the absence of a more-specific parameter for a given environment.

#### **NOTE**

Not all service instance parameters are suitable for per-environment configuration. In the example above, the `LowKey` and `HighKey` values for the service's partitioning scheme are explicitly defined for all instances of the service since the partition range is a function of the data domain, not the environment.

### **Per-environment service configuration settings**

The [Service Fabric application model](#) enables services to include configuration packages that contain custom key-value pairs that are readable at run time. The values of these settings can also be differentiated by environment by specifying a `ConfigOverride` in the application manifest.

Suppose that you have the following setting in the `Config\Settings.xml` file for the `Stateful1` service:

```

<Section Name="MyConfigSection">
    <Parameter Name="MaxQueueSize" Value="25" />
</Section>

```

To override this value for a specific application/environment pair, create a `configOverride` when you import the service manifest in the application manifest.

```

<ConfigOverrides>
    <ConfigOverride Name="Config">
        <Settings>
            <Section Name="MyConfigSection">
                <Parameter Name="MaxQueueSize" Value="[Stateful1_MaxQueueSize]" />
            </Section>
        </Settings>
    </ConfigOverride>
</ConfigOverrides>

```

This parameter can then be configured by environment as shown above. You can do this by declaring it in the parameters section of the application manifest and specifying environment-specific values in the application parameter files.

#### **NOTE**

In the case of service configuration settings, there are three places where the value of a key can be set: the service configuration package, the application manifest, and the application parameter file. Service Fabric will always choose from the application parameter file first (if specified), then the application manifest, and finally the configuration package.

### **Setting and using environment variables**

You can specify and set environment variables in the `ServiceManifest.xml` file and then override these in the `ApplicationManifest.xml` file on a per instance basis. The example below shows two environment variables, one with a value set and the other is overridden. You can use application parameters to set environment variables values in the same way that these were used for config overrides.

```

<?xml version="1.0" encoding="utf-8" ?>
<ServiceManifest Name="MyServiceManifest" Version="SvcManifestVersion1"
xmlns="http://schemas.microsoft.com/2011/01/fabric" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Description>An example service manifest</Description>
  <ServiceTypes>
    <StatelessServiceType ServiceTypeName="MyServiceType" />
  </ServiceTypes>
  <CodePackage Name="MyCode" Version="CodeVersion1">
    <SetupEntryPoint>
      <ExeHost>
        <Program>MySetup.bat</Program>
      </ExeHost>
    </SetupEntryPoint>
    <EntryPoint>
      <ExeHost>
        <Program>MyServiceHost.exe</Program>
      </ExeHost>
    </EntryPoint>
    <EnvironmentVariables>
      <EnvironmentVariable Name="MyEnvVariable" Value="" />
      <EnvironmentVariable Name="HttpGatewayPort" Value="19080" />
    </EnvironmentVariables>
  </CodePackage>
  <ConfigPackage Name="MyConfig" Version="ConfigVersion1" />
  <DataPackage Name="MyData" Version="DataVersion1" />
</ServiceManifest>

```

To override the environment variables in the ApplicationManifest.xml, reference the code package in the ServiceManifest with the `EnvironmentOverrides` element.

```

<ServiceManifestImport>
  <ServiceManifestRef ServiceManifestName="FrontEndServicePkg" ServiceManifestVersion="1.0.0" />
  <EnvironmentOverrides CodePackageRef="MyCode">
    <EnvironmentVariable Name="MyEnvVariable" Value="mydata" />
  </EnvironmentOverrides>
</ServiceManifestImport>

```

Once the named service instance is created you can access the environment variables from code. e.g. In C# you can do the following

```

string EnvVariable = Environment.GetEnvironmentVariable("MyEnvVariable");

```

## Service Fabric environment variables

Service Fabric has built in environment variables set for each service instance. The full list of environment variables is below, where the ones in bold are the ones that you will use in your service, the other being used by Service Fabric runtime.

- Fabric\_ApplicationHostId
- Fabric\_ApplicationHostType
- Fabric\_ApplicationId
- **Fabric\_ApplicationName**
- Fabric\_CodePackageInstanceId
- **Fabric\_CodePackageName**
- **Fabric\_Endpoint\_[YourServiceName]TypeEndpoint**
- **Fabric\_Folder\_App\_Log**
- **Fabric\_Folder\_App\_Temp**
- **Fabric\_Folder\_App\_Work**

- **Fabric\_Folder\_Application**
- Fabric\_NodeId
- **Fabric\_NodeIPorFQDN**
- **Fabric\_NodeName**
- Fabric\_RuntimeConnectionAddress
- Fabric\_ServicePackageInstanceId
- Fabric\_ServicePackageName
- Fabric\_ServicePackageVersionInstance
- FabricPackageFileName

The code below shows how to list the Service Fabric environment variables

```
foreach (DictionaryEntry de in Environment.GetEnvironmentVariables())
{
    if (de.Key.ToString().StartsWith("Fabric"))
    {
        Console.WriteLine(" Environment variable {0} = {1}", de.Key, de.Value);
    }
}
```

The following are examples of environment variables for an application type called `GuestExe.Application` with a service type called `FrontEndService` when run on your local dev machine.

- **Fabric\_ApplicationName = fabric:/GuestExe.Application**
- **Fabric\_CodePackageName = Code**
- **Fabric\_Endpoint\_FrontEndServiceTypeEndpoint = 80**
- **Fabric\_NodeIPorFQDN = localhost**
- **Fabric\_NodeName = \_Node\_2**

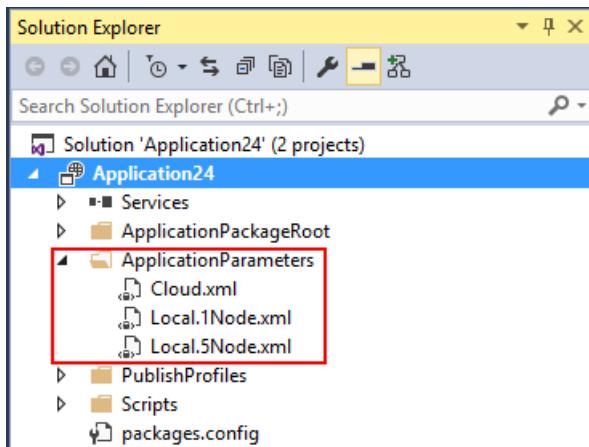
### Application parameter files

The Service Fabric application project can include one or more application parameter files. Each of them defines the specific values for the parameters that are defined in the application manifest:

```
<!-- ApplicationParameters\Local.xml -->

<Application Name="fabric:/Application1" xmlns="http://schemas.microsoft.com/2011/01/fabric">
    <Parameters>
        <Parameter Name ="Stateful1_MinReplicaSetSize" Value="3" />
        <Parameter Name="Stateful1_PartitionCount" Value="1" />
        <Parameter Name="Stateful1_TargetReplicaSetSize" Value="3" />
    </Parameters>
</Application>
```

By default, a new application includes three application parameter files, named Local.1Node.xml, Local.5Node.xml, and Cloud.xml:



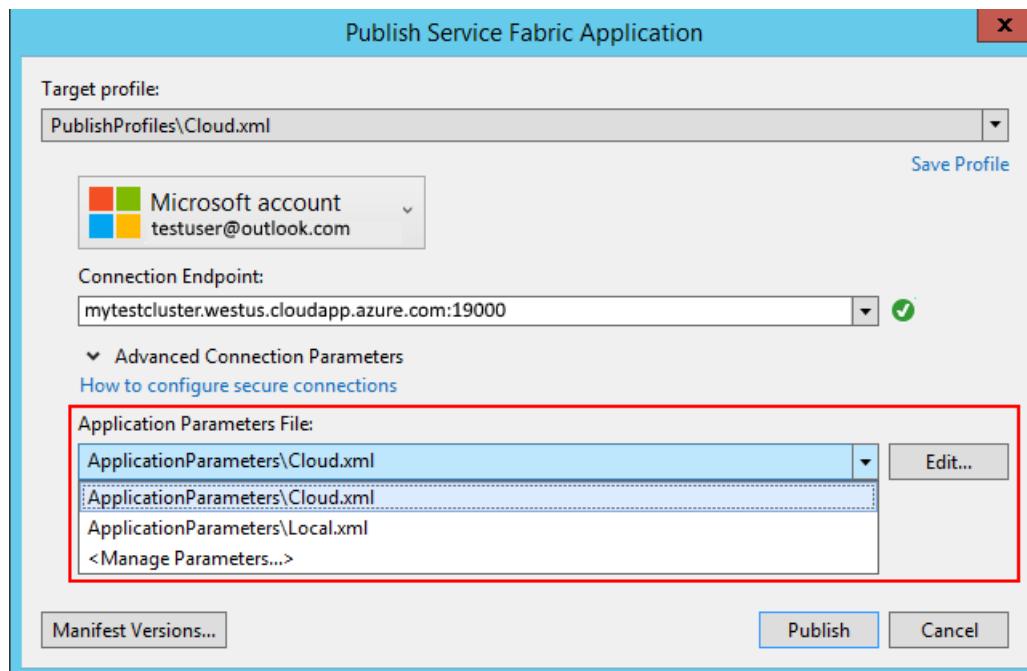
To create a parameter file, simply copy and paste an existing one and give it a new name.

## Identifying environment-specific parameters during deployment

At deployment time, you need to choose the appropriate parameter file to apply with your application. You can do this through the Publish dialog in Visual Studio or through PowerShell.

### Deploy from Visual Studio

You can choose from the list of available parameter files when you publish your application in Visual Studio.



### Deploy from PowerShell

The `Deploy-FabricApplication.ps1` PowerShell script included in the application project template accepts a publish profile as a parameter and the PublishProfile contains a reference to the application parameters file.

```
./Deploy-FabricApplication -ApplicationPackagePath <app_package_path> -PublishProfileFile  
<publishprofile_path>
```

## Next steps

To learn more about some of the core concepts that are discussed in this topic, see the [Service Fabric technical overview](#). For information about other app management capabilities that are available in Visual Studio, see [Manage your Service Fabric applications in Visual Studio](#).

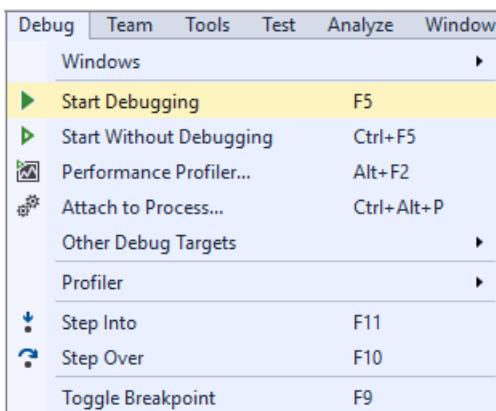
# Debug your Service Fabric application by using Visual Studio

6/30/2017 • 4 min to read • [Edit Online](#)

## Debug a local Service Fabric application

You can save time and money by deploying and debugging your Azure Service Fabric application in a local computer development cluster. Visual Studio 2017 or Visual Studio 2015 can deploy the application to the local cluster and automatically connect the debugger to all instances of your application.

1. Start a local development cluster by following the steps in [Setting up your Service Fabric development environment](#).
2. Press **F5** or click **Debug > Start Debugging**.



3. Set breakpoints in your code and step through the application by clicking commands in the **Debug** menu.

### NOTE

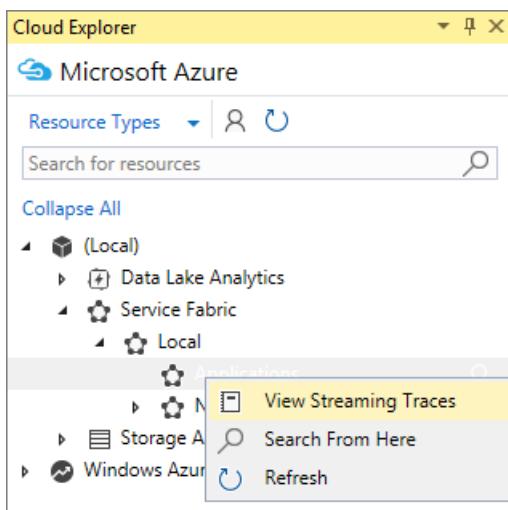
Visual Studio attaches to all instances of your application. While you're stepping through code, breakpoints may get hit by multiple processes resulting in concurrent sessions. Try disabling the breakpoints after they're hit, by making each breakpoint conditional on the thread ID or by using diagnostic events.

4. The **Diagnostic Events** window will automatically open so you can view diagnostic events in real time.

Diagnostic Events (Local) > MyStatefulService.cs			
	Timestamp	Event Name	Message
▶	10:20:23.192	ServiceMessage	Current Counter Value: 473
▶	10:20:23.192	ServiceMessage	Working-492
▶	10:20:22.188	ServiceMessage	Working-491
▶	10:20:22.173	ServiceMessage	Current Counter Value: 472
▶	10:20:21.187	ServiceMessage	Working-490
▶	10:20:21.156	ServiceMessage	Current Counter Value: 471
▶	10:20:20.184	ServiceMessage	Working-489
▶	10:20:20.137	ServiceMessage	Current Counter Value: 470
▶	10:20:19.168	ServiceMessage	Working-488
▶	10:20:19.106	ServiceMessage	Current Counter Value: 469
▶	10:20:18.152	ServiceMessage	Working-487
▶	10:20:18.074	ServiceMessage	Current Counter Value: 468
▶	10:20:17.136	ServiceMessage	Working-486
▶	10:20:17.057	ServiceMessage	Current Counter Value: 467
▶	10:20:16.120	ServiceMessage	Working-485
▶	10:20:16.026	ServiceMessage	Current Counter Value: 466
▶	10:20:15.108	ServiceMessage	Working-484
▶	10:20:14.998	ServiceMessage	Current Counter Value: 465
▶	10:20:14.092	ServiceMessage	Working-483
▶	10:20:13.967	ServiceMessage	Current Counter Value: 464

Paused (119 of 119 events shown) [Clear Filter](#)

5. You can also open the **Diagnostic Events** window in Cloud Explorer. Under **Service Fabric**, right-click any node and choose **View Streaming Traces**.



If you want to filter your traces to a specific service or application, simply enable streaming traces on that specific service or application.

6. The diagnostic events can be seen in the automatically generated **ServiceEventSource.cs** file and are called from application code.

```
ServiceEventSource.Current.ServiceMessage(this, "My ServiceMessage with a parameter {0}", result.Value.ToString());
```

7. The **Diagnostic Events** window supports filtering, pausing, and inspecting events in real time. The filter is a simple string search of the event message, including its contents.

The screenshot shows the Visual Studio Diagnostic Events window. The title bar says "Diagnostic Events (Local) > X ServiceEventSource.cs MyStatefulService.cs\*". The main area displays a table with columns: Timestamp, Event Name, and Message. A single event is selected, highlighted in blue, showing detailed properties like Timestamp, ProviderName, Id, Message, ProcessId, Level, Keywords, EventName, and Payload. The payload contains service metadata such as fabric URI, service type, replica ID, partition ID, application name, application type name, node name, and message content. Below the table, there are several other events listed with similar details. At the bottom, there are buttons for "Paused" and "Clear Filter".

Timestamp	Event Name	Message
0:20:23.192	ServiceMessage	Current Counter Value: 473
10:20:22.173	ServiceMessage	Current Counter Value: 472
10:20:21.156	ServiceMessage	Current Counter Value: 471
10:20:20.137	ServiceMessage	Current Counter Value: 470

Paused (59 of 119 events shown) Clear Filter

- Debugging services is like debugging any other application. You will normally set Breakpoints through Visual Studio for easy debugging. Even though Reliable Collections replicate across multiple nodes, they still implement `IEnumerable`. This means that you can use the Results View in Visual Studio while debugging to see what you've stored inside. Simply set a breakpoint anywhere in your code.

The screenshot shows the Visual Studio code editor with a C# file open. Line 25 contains a breakpoint. The code implements a `RunAsync` method that uses a `ReliableDictionary` to store a counter value. A tooltip over the `myDictionary.AddOrUpdateAsync` call shows the current state of the dictionary. The `Results View` window is open, showing a list of items with keys and values. One item is expanded to show its key ("Counter-1") and value (0).

```

25 protected override async Task RunAsync(CancellationToken cancellationToken)
26 {
27     // TODO: Replace the following with your own logic.
28     IReliableDictionary<string, long> myDictionary = await this.StateManager.GetOrAddAsync<IReliableDictionary<string, long>>("Counter");
29
30     while (!cancellationToken.IsCancellationRequested)
31     {
32         using (var tx = this.StateManager.CreateTransaction())
33         {
34             var result = await myDictionary.TryGetValueAsync(tx, "Counter-1");
35             ServiceEventSource.Current.ServiceMessage(
36                 this,
37                 "Current Counter Value: {0}",
38                 result.HasValue ? result.Value.ToString() : "Value does not exist.");
39
40             await myDictionary.AddOrUpdateAsync(tx, "Counter-1", 0, (k, v) => ++v); // myDictionary[Microsoft.ServiceFabric.Data.Collections.ReliableDictionaryWrapper<string, long>] ≤ 1,254ms elapsed
41         }
42     }
43 }
44
45
46
47
48
49
50

```

myDictionary[Microsoft.ServiceFabric.Data.Collections.ReliableDictionaryWrapper<string, long>]

- urn:myDictionary
- (byte[265])
- urn:myDictionary
- (System.Fabric.Collections.Distributed.DistributedDictionary<string, long>)
- Results View

Key	Value
Counter-1	0

## Debug a remote Service Fabric application

If your Service Fabric applications are running on a Service Fabric cluster in Azure, you are able to remotely debug these, directly from Visual Studio.

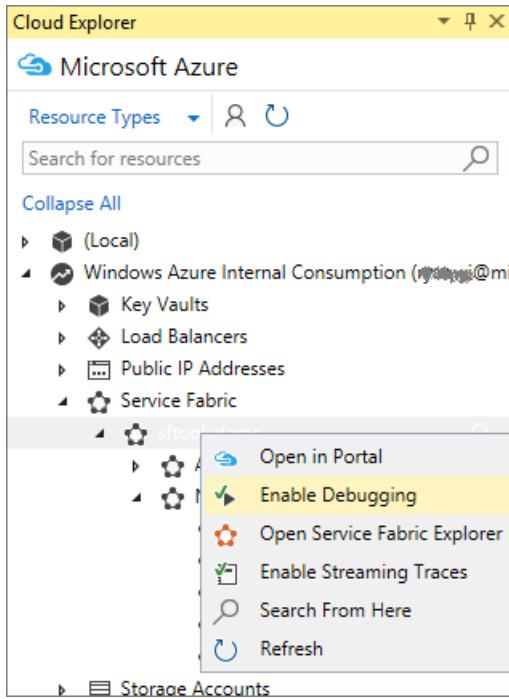
### NOTE

The feature requires [Service Fabric SDK 2.0](#) and [Azure SDK for .NET 2.9](#).

## WARNING

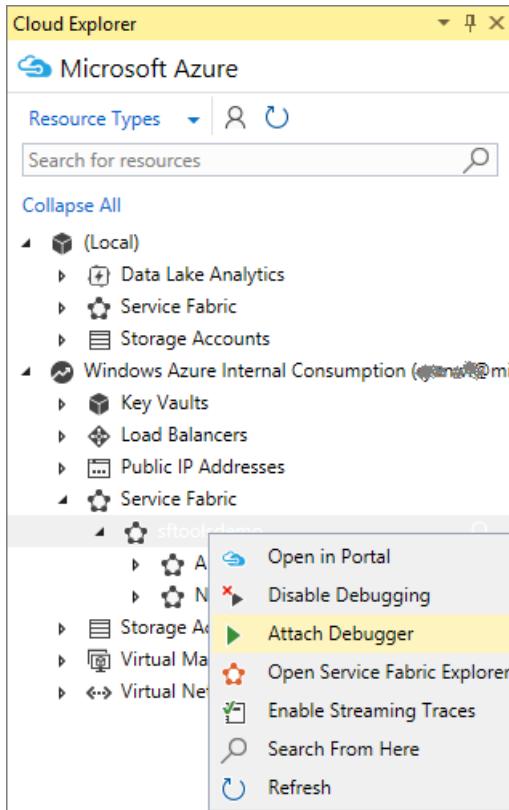
Remote debugging is meant for dev/test scenarios and not to be used in production environments, because of the impact on the running applications.

1. Navigate to your cluster in **Cloud Explorer**, right-click and choose **Enable Debugging**

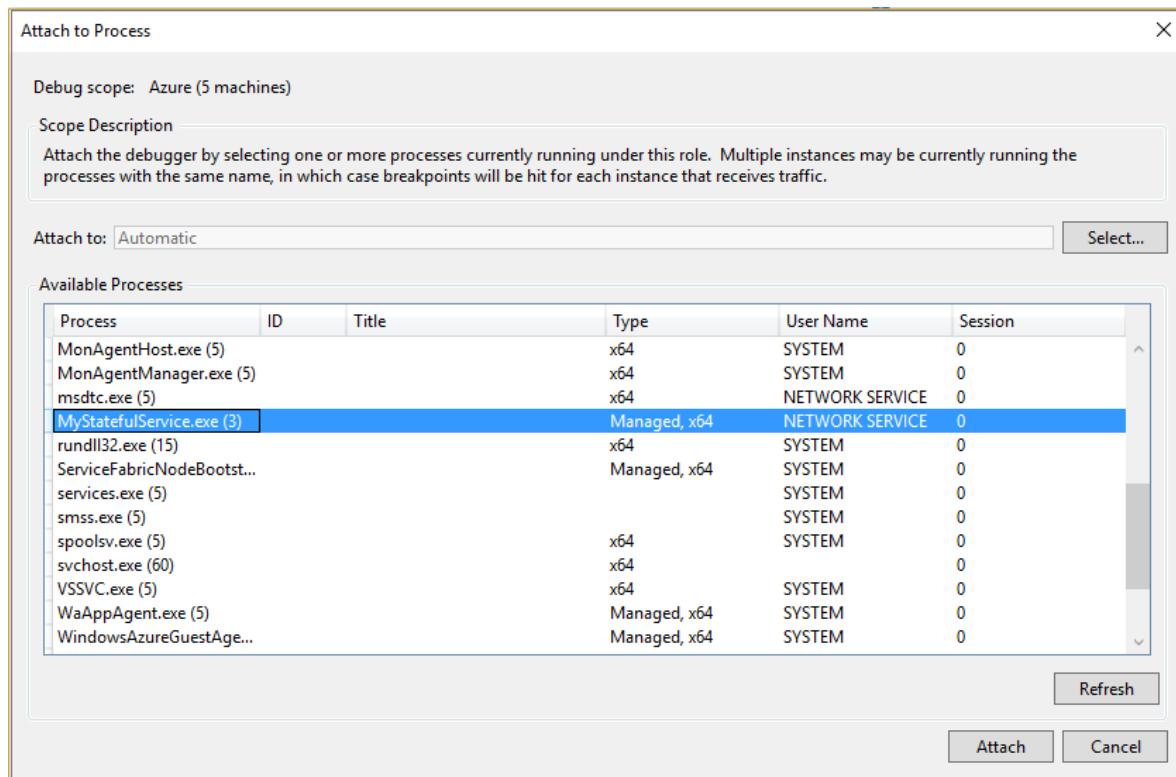


This will kick off the process of enabling the remote debugging extension on your cluster nodes, as well as required network configurations.

2. Right-click the cluster node in **Cloud Explorer**, and choose **Attach Debugger**



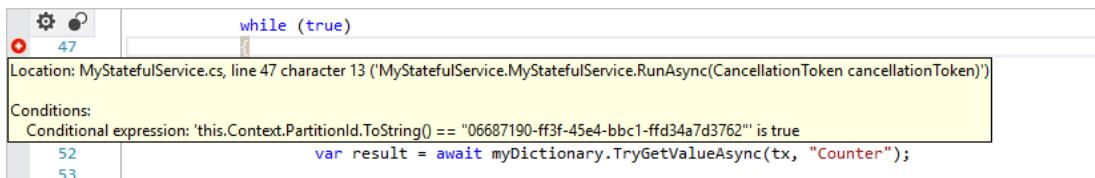
3. In the **Attach to process** dialog, choose the process you want to debug, and click **Attach**



The name of the process you want to attach to, equals the name of your service project assembly name.

The debugger will attach to all nodes running the process.

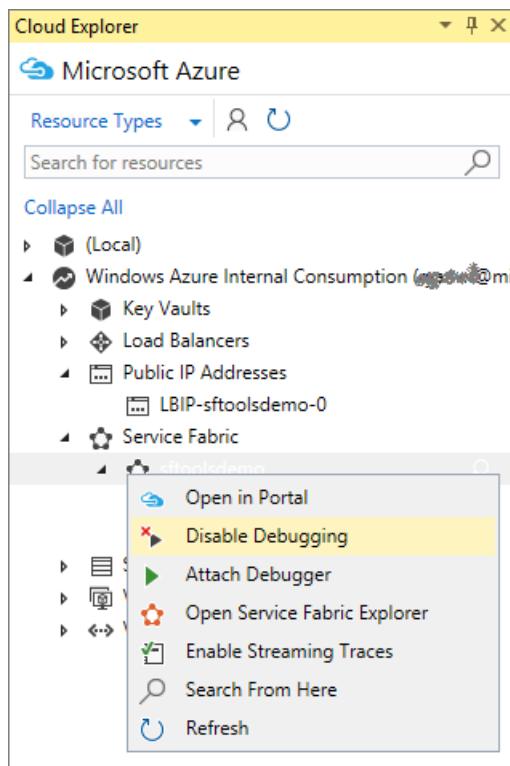
- In the case where you are debugging a stateless service, all instances of the service on all nodes are part of the debug session.
- If you are debugging a stateful service, only the primary replica of any partition will be active and therefore caught by the debugger. If the primary replica moves during the debug session, the processing of that replica will still be part of the debug session.
- In order to only catch relevant partitions or instances of a given service, you can use conditional breakpoints to only break a specific partition or instance.



#### NOTE

Currently we do not support debugging a Service Fabric cluster with multiple instances of the same service executable name.

- Once you finish debugging your application, you can disable the remote debugging extension by right-clicking the cluster in **Cloud Explorer** and choose **Disable Debugging**



## Streaming traces from a remote cluster node

You are also able to stream traces directly from a remote cluster node to Visual Studio. This feature allows you to stream ETW trace events, produced on a Service Fabric cluster node.

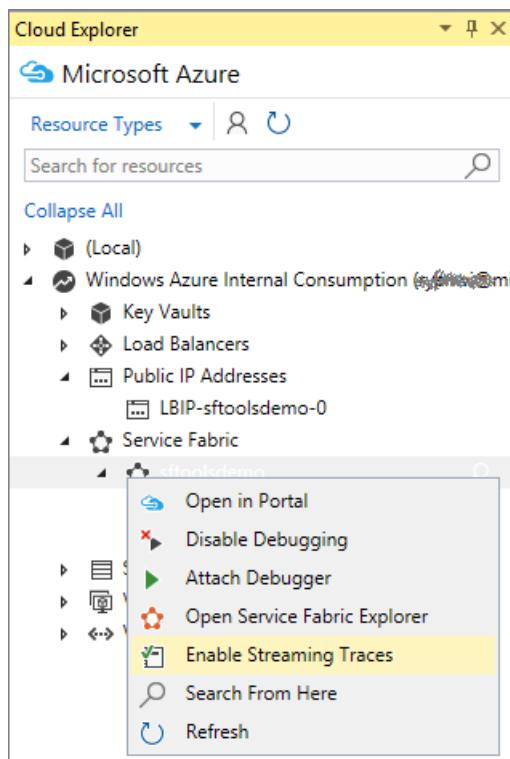
### NOTE

This feature requires [Service Fabric SDK 2.0](#) and [Azure SDK for .NET 2.9](#). This feature only supports clusters running in Azure.

### WARNING

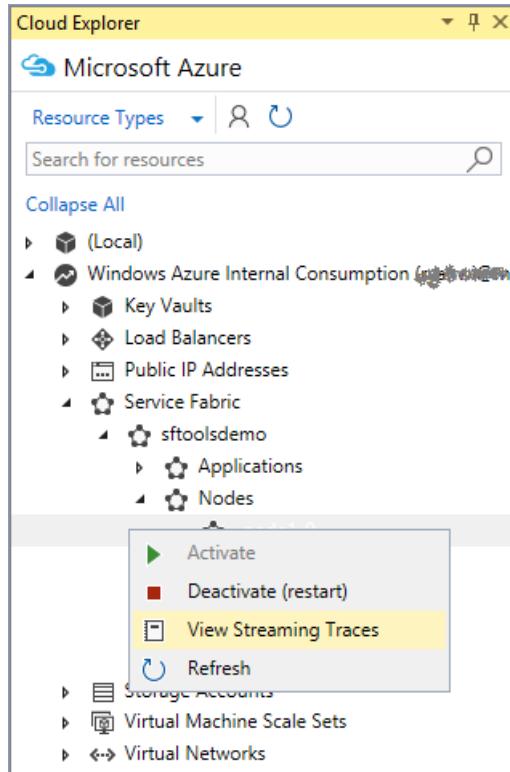
Streaming traces is meant for dev/test scenarios and not to be used in production environments, because of the impact on the running applications. In a production scenario, you should rely on forwarding events using Azure Diagnostics.

1. Navigate to your cluster in **Cloud Explorer**, right-click and choose **Enable Streaming Traces**



This will kick off the process of enabling the streaming traces extension on your cluster nodes, as well as required network configurations.

2. Expand the **Nodes** element in **Cloud Explorer**, right-click the node you want to stream traces from and choose **View Streaming Traces**



Repeat step 2 for as many nodes as you want to see traces from. Each nodes stream will show up in a dedicated window.

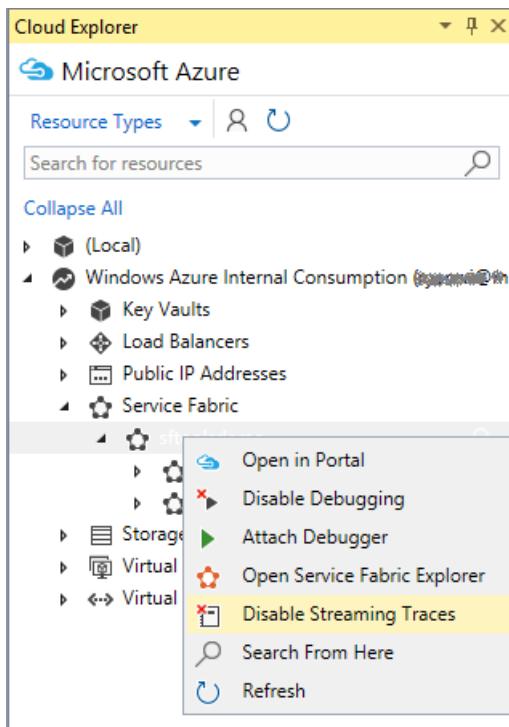
You are now able to see the traces emitted by Service Fabric, and your services. If you want to filter the events to only show a specific application, simply type in the name of the application in the filter.

Diagnostic Events (sftools - \_sft\_2) ▾ X MyStatefulService.cs

SFAApplication-MyStatefulService

Timestamp	Event Name	Message
21:31:48.502	ServiceMessage	Current Counter Value: 4857
		{ "Timestamp": "2016-03-29T21:31:48.5029804+00:00", "ProviderName": "MyCompany-MyCompany.SFAApplication-MyStatefulService", "Id": 2, "Message": "Current Counter Value: 4857", "ProcessId": 8124, "Level": "Informational", "Keywords": "0x0000F00000000000", "EventName": "ServiceMessage", "Payload": { "serviceName": "fabric:/MyCompany.SFAApplication/MyStatefulService", "serviceTypeName": "MyStatefulServiceType", "replicaOrInstanceId": 131037554715799500, "partitionId": "b8d6cac4-dd9a-477e-9446-75daa93e4a90", "applicationName": "fabric:/MyCompany.SFAApplication", "applicationTypeName": "MyCompany.SFAApplicationType", "nodeName": "_sft_2", "message": "Current Counter Value: 4857" } }
21:31:47.483	ServiceMessage	Current Counter Value: 4861
21:31:47.483	ServiceMessage	Current Counter Value: 4856
21:31:46.460	ServiceMessage	Current Counter Value: 4855
21:31:46.460	ServiceMessage	Current Counter Value: 4860
21:31:45.432	ServiceMessage	Current Counter Value: 4859
21:31:45.432	ServiceMessage	Current Counter Value: 4854

3. Once you are done streaming traces from your cluster, you can disable remote streaming traces, by right-clicking the cluster in **Cloud Explorer** and choose **Disable Streaming Traces**



## Next steps

- [Test a Service Fabric service.](#)
- [Manage your Service Fabric applications in Visual Studio.](#)

# Common exceptions and errors when working with the FabricClient APIs

8/31/2017 • 1 min to read • [Edit Online](#)

The [FabricClient](#) APIs enable cluster and application administrators to perform administrative tasks on a Service Fabric application, service, or cluster. For example, application deployment, upgrade, and removal, checking the health a cluster, or testing a service. Application developers and cluster administrators can use the FabricClient APIs to develop tools for managing the Service Fabric cluster and applications.

There are many different types of operations which can be performed using FabricClient. Each method can throw exceptions for errors due to incorrect input, runtime errors, or transient infrastructure issues. See the API reference documentation to find which exceptions are thrown by a specific method. There are some exceptions, however, which can be thrown by many different [FabricClient](#) APIs. The following table lists the exceptions that are common across the FabricClient APIs.

EXCEPTION	THROWN WHEN
<a href="#">System.Fabric.FabricObjectClosedException</a>	The <a href="#">FabricClient</a> object is in a closed state. Dispose of the <a href="#">FabricClient</a> object you are using and instantiate a new <a href="#">FabricClient</a> object.
<a href="#">System.TimeoutException</a>	The operation timed out. <a href="#">OperationTimedOut</a> is returned when the operation takes more than MaxOperationTimeout to complete.
<a href="#">System.UnauthorizedAccessException</a>	The access check for the operation failed. E_ACCESSDENIED is returned.
<a href="#">System.Fabric.FabricException</a>	A runtime error occurred while performing the operation. Any of the FabricClient methods can potentially throw <a href="#">FabricException</a> , the <a href="#">ErrorCode</a> property indicates the exact cause of the exception. Error codes are defined in the <a href="#">FabricErrorCode</a> enumeration.
<a href="#">System.Fabric.FabricTransientException</a>	The operation failed due to a transient error condition of some kind. For example, an operation may fail because a quorum of replicas is temporarily not reachable. Transient exceptions correspond to failed operations that can be retried.

Some common [FabricErrorCode](#) errors that can be returned in a [FabricException](#):

ERROR	CONDITION
CommunicationError	A communication error caused the operation to fail, retry the operation.
InvalidCredentialType	The credential type is invalid.
InvalidX509FindType	The X509FindType is invalid.
InvalidX509StoreLocation	The X509 store location is invalid.

ERROR	CONDITION
InvalidX509StoreName	The X509 store name is invalid.
InvalidX509Thumbprint	The X509 certificate thumbprint string is invalid.
InvalidProtectionLevel	The protection level is invalid.
InvalidX509Store	The X509 certificate store cannot be opened.
InvalidSubjectName	The subject name is invalid.
InvalidAllowedCommonNameList	The format of common name list string is invalid. It should be a comma-separated list.

# Monitor and diagnose services in a local machine development setup

10/16/2017 • 3 min to read • [Edit Online](#)

Monitoring, detecting, diagnosing, and troubleshooting allow for services to continue with minimal disruption to the user experience. While monitoring and diagnostics are critical in an actual deployed production environment, the efficiency will depend on adopting a similar model during development of services to ensure they work when you move to a real-world setup. Service Fabric makes it easy for service developers to implement diagnostics that can seamlessly work across both single-machine local development setups and real-world production cluster setups.

## Event Tracing for Windows

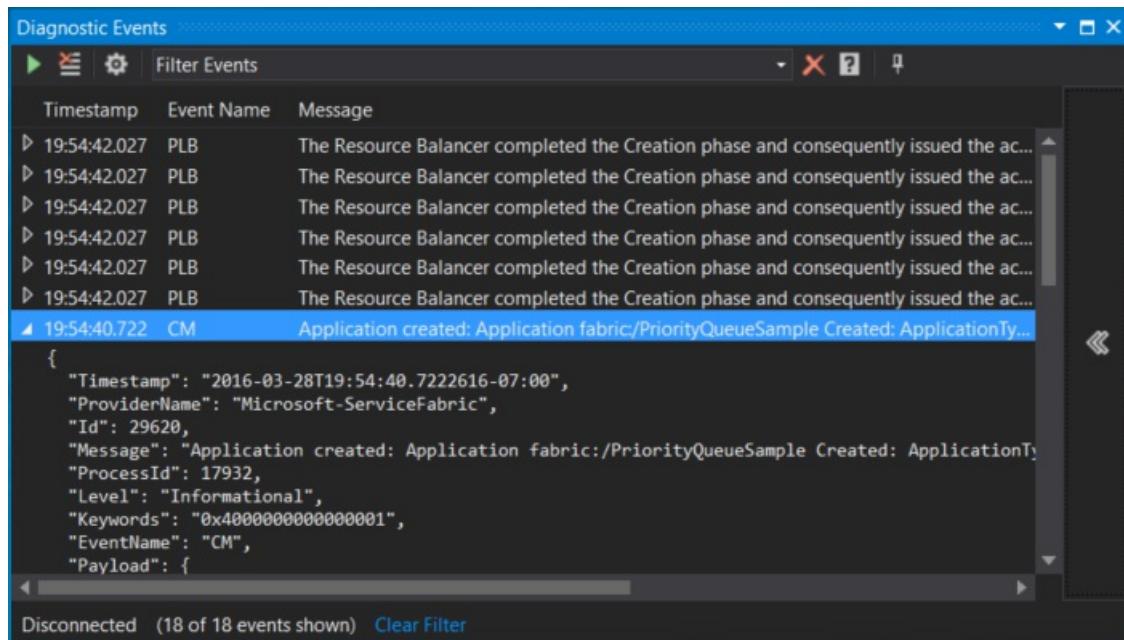
[Event Tracing for Windows](#) (ETW) is the recommended technology for tracing messages in Service Fabric. Some benefits of using ETW are:

- **ETW is fast.** It was built as a tracing technology that has minimal impact on code execution times.
- **ETW tracing works seamlessly across local development environments and also real-world cluster setups.** This means you don't have to rewrite your tracing code when you are ready to deploy your code to a real cluster.
- **Service Fabric system code also uses ETW for internal tracing.** This allows you to view your application traces interleaved with Service Fabric system traces. It also helps you to more easily understand the sequences and interrelationships between your application code and events in the underlying system.
- **There is built-in support in Service Fabric Visual Studio tools to view ETW events.** ETW events appear in the Diagnostic Events view of Visual Studio once Visual Studio is correctly configured with Service Fabric.

## View Service Fabric system events in Visual Studio

Service Fabric emits ETW events to help application developers understand what's happening in the platform. If you haven't already done so, go ahead and follow the steps in [Creating your first application in Visual Studio](#). This information will help you get an application up and running with the Diagnostics Events Viewer showing the trace messages.

1. If the diagnostics events window does not automatically show, Go to the **View** tab in Visual Studio, choose **Other Windows** and then **Diagnostic Events Viewer**.
2. Each event has standard metadata information that tells you the node, application and service the event is coming from. You can also filter the list of events by using the **Filter events** box at the top of the events window. For example, you can filter on **Node Name** or **Service Name**. And when you're looking at event details, you can also pause by using the **Pause** button at the top of the events window and resume later without any loss of events.



## Add your own custom traces to the application code

The Service Fabric Visual Studio project templates contain sample code. The code shows how to add custom application code ETW traces that show up in the Visual Studio ETW viewer alongside system traces from Service Fabric. The advantage of this method is that metadata is automatically added to traces, and the Visual Studio Diagnostic Events Viewer is already configured to display them.

For projects created from the **service templates** (stateless or stateful) just search for the `RunAsync` implementation:

1. The call to `ServiceEventSource.Current.ServiceMessage` in the `RunAsync` method shows an example of a custom ETW trace from the application code.
2. In the **ServiceEventSource.cs** file, you will find an overload for the `ServiceEventSource.ServiceMessage` method that should be used for high-frequency events due to performance reasons.

For projects created from the **actor templates** (stateless or stateful):

1. Open the "**ProjectName**.cs" file where *ProjectName* is the name you chose for your Visual Studio project.
2. Find the code `ActorEventSource.Current.ActorMessage(this, "Doing Work");` in the `DoWorkAsync` method. This is an example of a custom ETW trace written from application code.
3. In file **ActorEventSource.cs**, you will find an overload for the `ActorEventSource.ActorMessage` method that should be used for high-frequency events due to performance reasons.

After adding custom ETW tracing to your service code, you can build, deploy, and run the application again to see your event(s) in the Diagnostic Events Viewer. If you debug the application with **F5**, the Diagnostic Events Viewer will open automatically.

## Next steps

The same tracing code that you added to your application above for local diagnostics will work with tools that you can use to view these events when running your application on an Azure cluster. Check out these articles that discuss the different options for the tools and describe how you can set them up.

- [How to collect logs with Azure Diagnostics](#)
- [Event aggregation and collection using EventFlow](#)

# Service Fabric plug-in for Eclipse Java application development

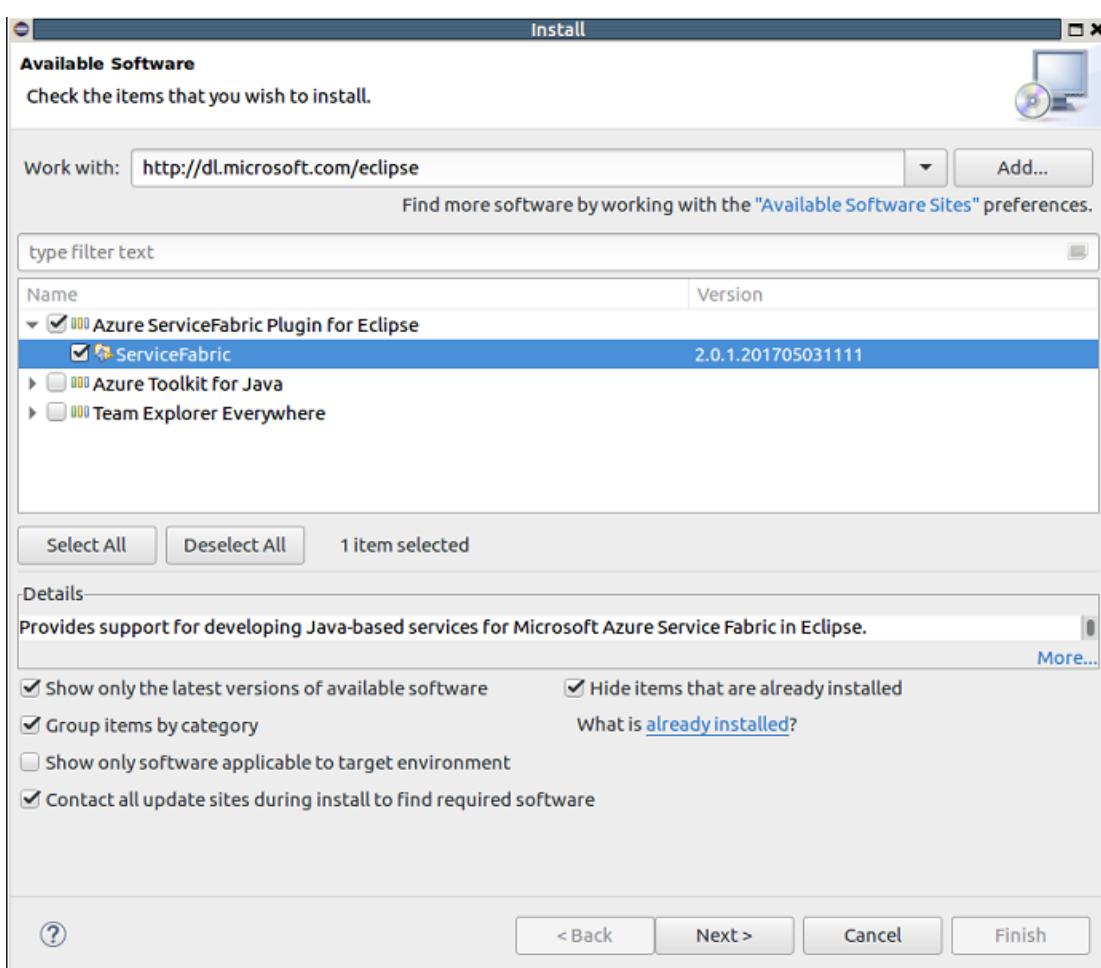
9/20/2017 • 6 min to read • [Edit Online](#)

Eclipse is one of the most widely used integrated development environments (IDEs) for Java developers. In this article, we describe how to set up your Eclipse development environment to work with Azure Service Fabric. Learn how to install the Service Fabric plug-in, create a Service Fabric application, and deploy your Service Fabric application to a local or remote Service Fabric cluster in Eclipse Neon.

## Install or update the Service Fabric plug-in in Eclipse Neon

You can install a Service Fabric plug-in in Eclipse. The plug-in can help simplify the process of building and deploying Java services.

1. Ensure that you have the latest version of Eclipse Neon and the latest version of Buildship (1.0.17 or a later version) installed:
  - To check the versions of installed components, in Eclipse Neon, go to **Help > Installation Details**.
  - To update Buildship, see [Eclipse Buildship: Eclipse Plug-ins for Gradle](#).
  - To check for and install updates for Eclipse Neon, go to **Help > Check for Updates**.
2. To install the Service Fabric plug-in, in Eclipse Neon, go to **Help > Install New Software**.
  - a. In the **Work with** box, enter <http://dl.microsoft.com/eclipse>.
  - b. Click **Add**.



- c. Select the Service Fabric plug-in, and then click **Next**.
- d. Complete the installation steps, and then accept the Microsoft Software License Terms.

If you already have the Service Fabric plug-in installed, make sure that you have the latest version. To check for available updates, go to **Help > Installation Details**. In the list of installed plug-ins, select Service Fabric, and then click **Update**. Available updates will be installed.

#### NOTE

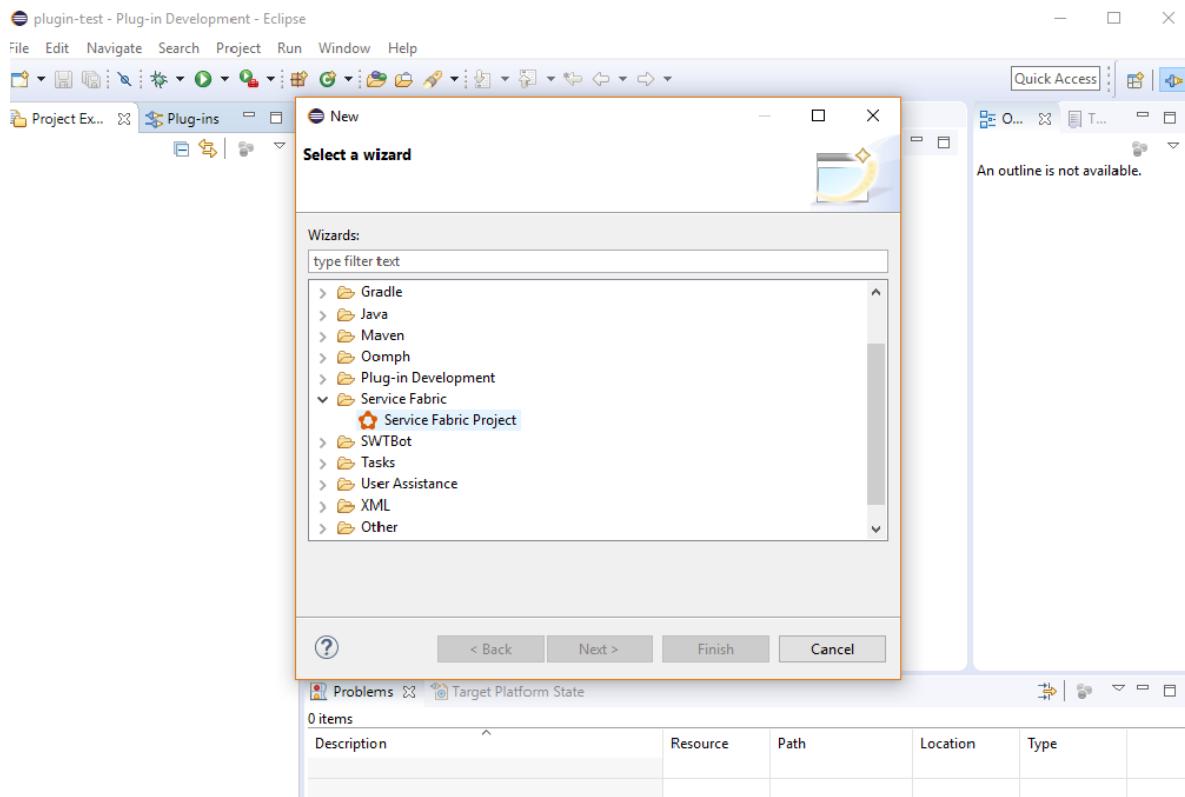
If installing or updating the Service Fabric plug-in is slow, it might be due to an Eclipse setting. Eclipse collects metadata on all changes to update sites that are registered with your Eclipse instance. To speed up the process of checking for and installing a Service Fabric plug-in update, go to **Available Software Sites**. Clear the check boxes for all sites except for the one that points to the Service Fabric plug-in location (<http://dl.microsoft.com/eclipse/azure/servicefabric>).

#### NOTE

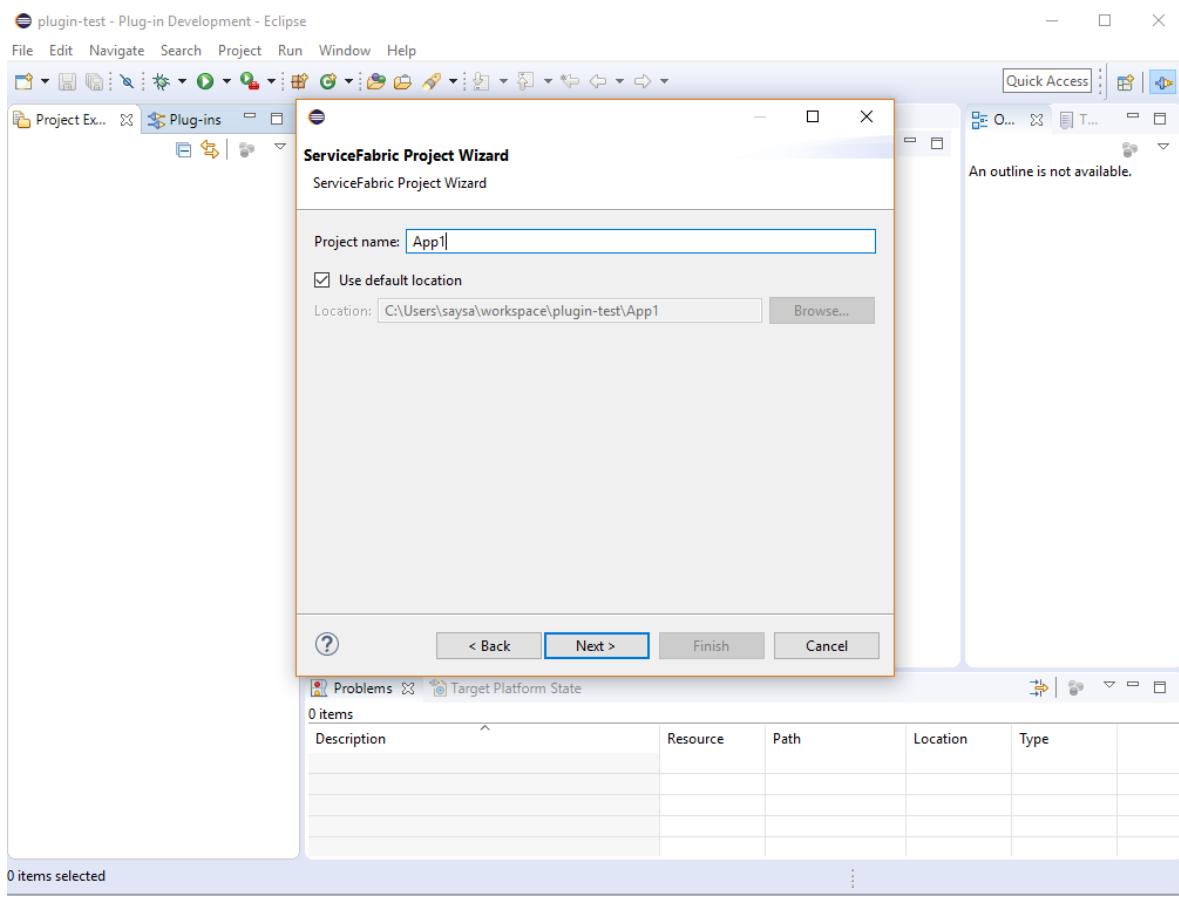
If Eclipse isn't working as expected on your Mac, or needs you run as super user), go to the **ECLIPSE\_INSTALLATION\_PATH** folder and navigate to the subfolder **Eclipse.app/Contents/MacOS**. Start Eclipse by running `./eclipse`.

## Create a Service Fabric application in Eclipse

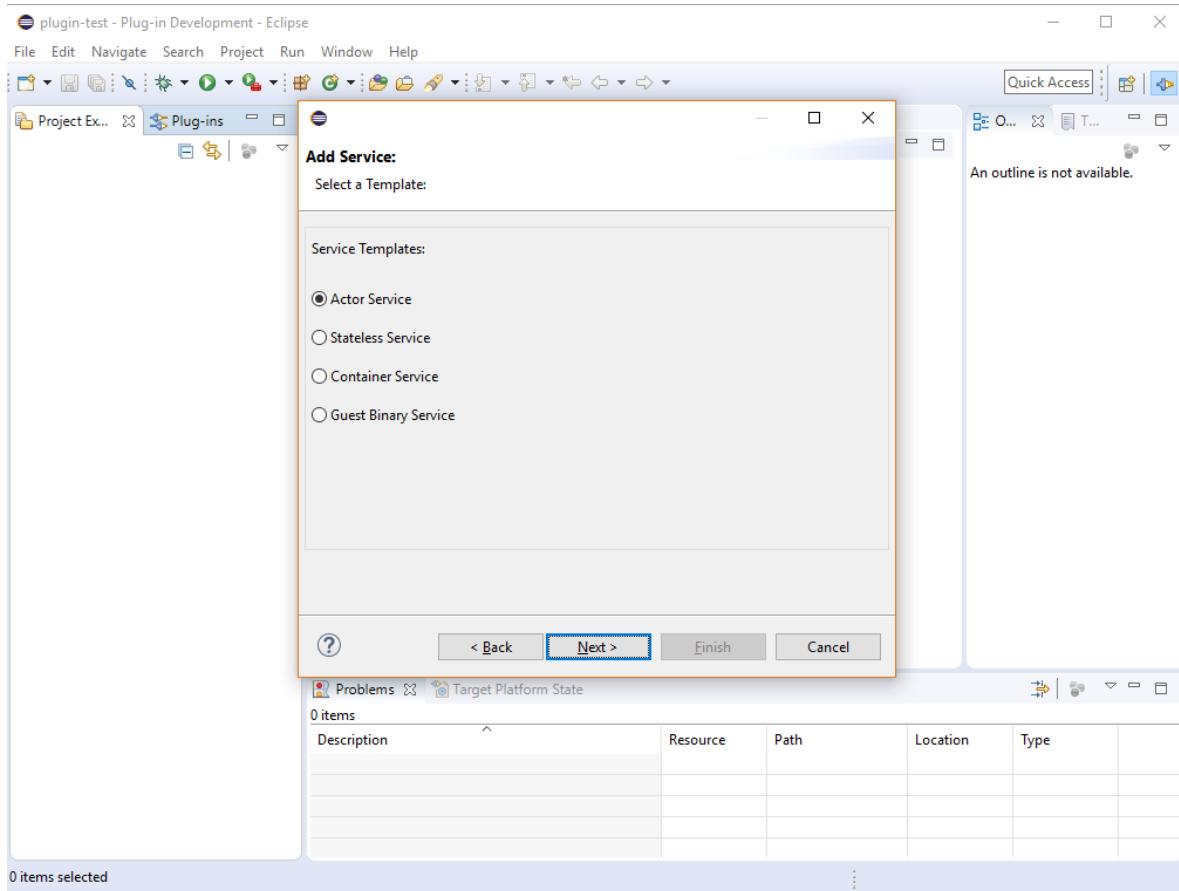
1. In Eclipse Neon, go to **File > New > Other**. Select **Service Fabric Project**, and then click **Next**.



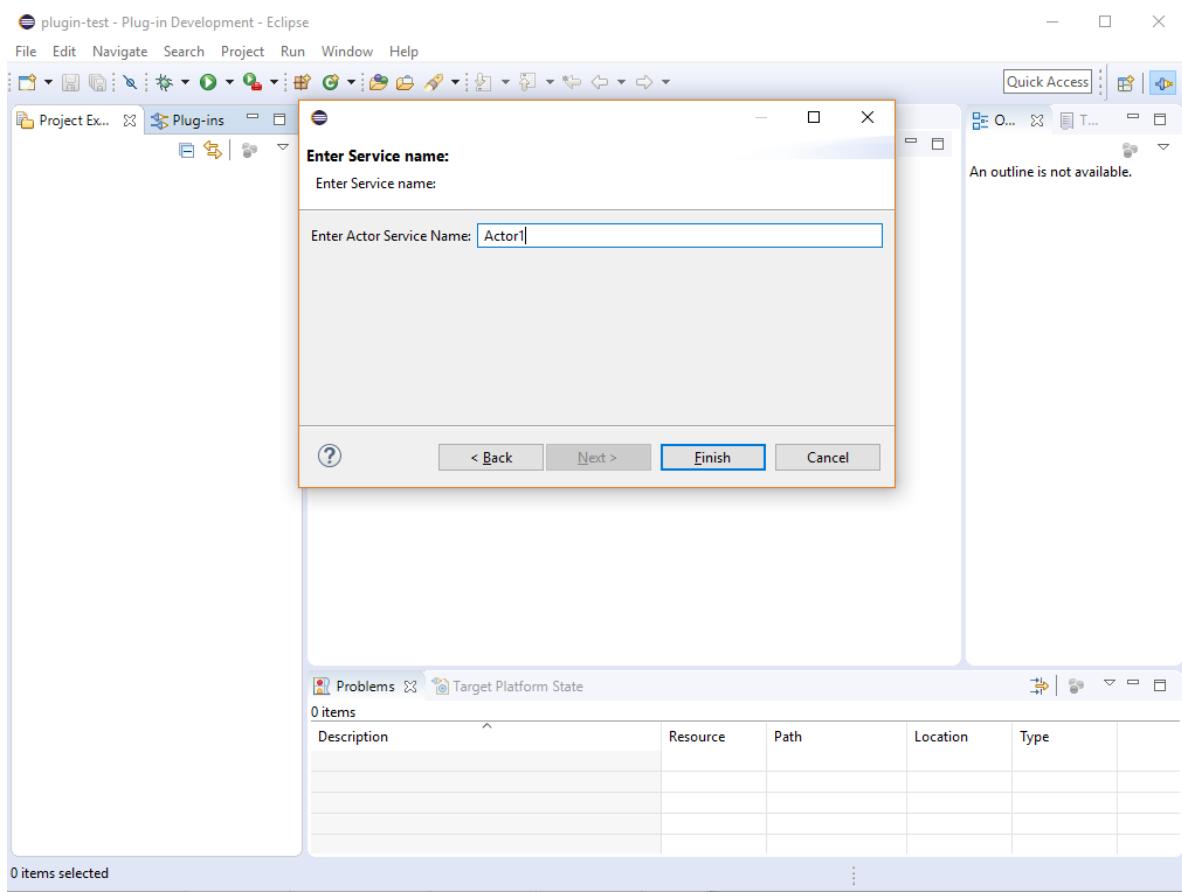
2. Enter a name for your project, and then click **Next**.



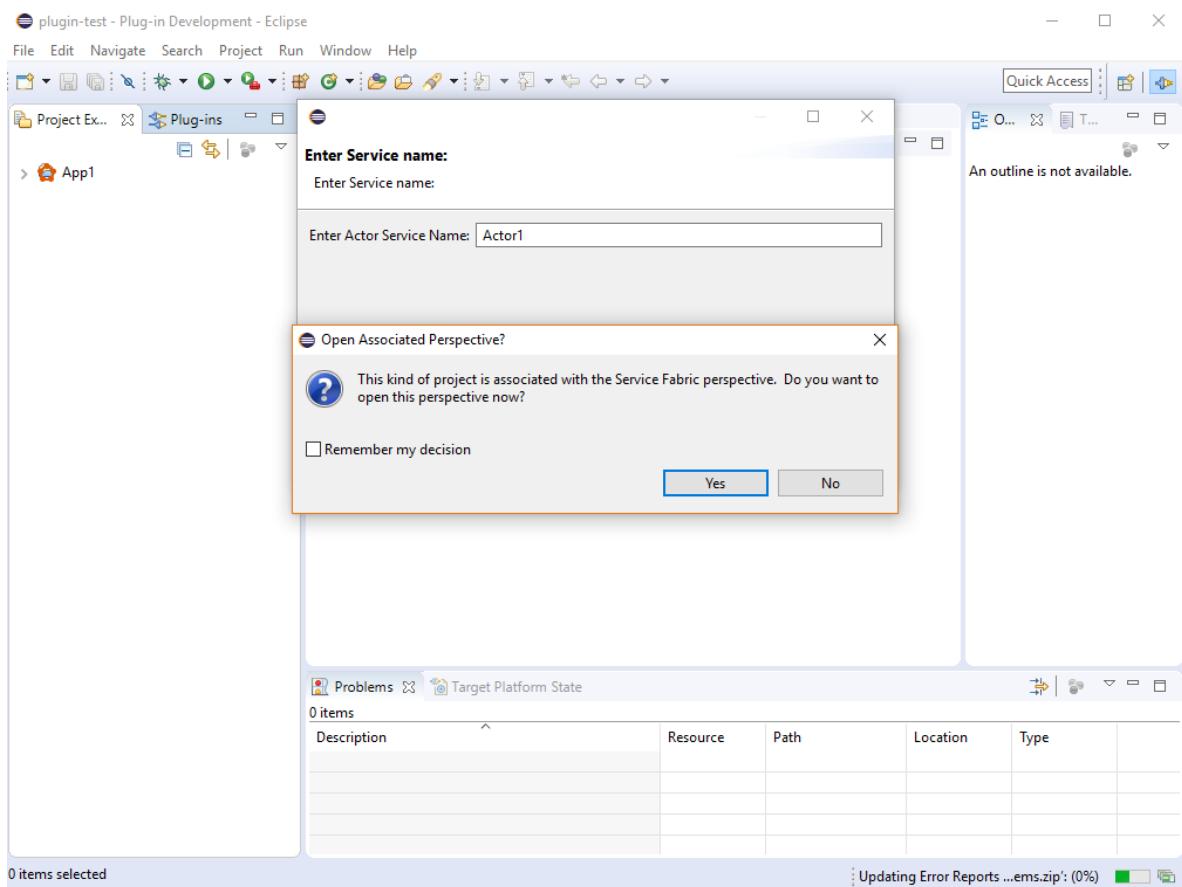
3. In the list of templates, select **Service Template**. Select your service template type (Actor, Stateless, Container, or Guest Binary), and then click **Next**.



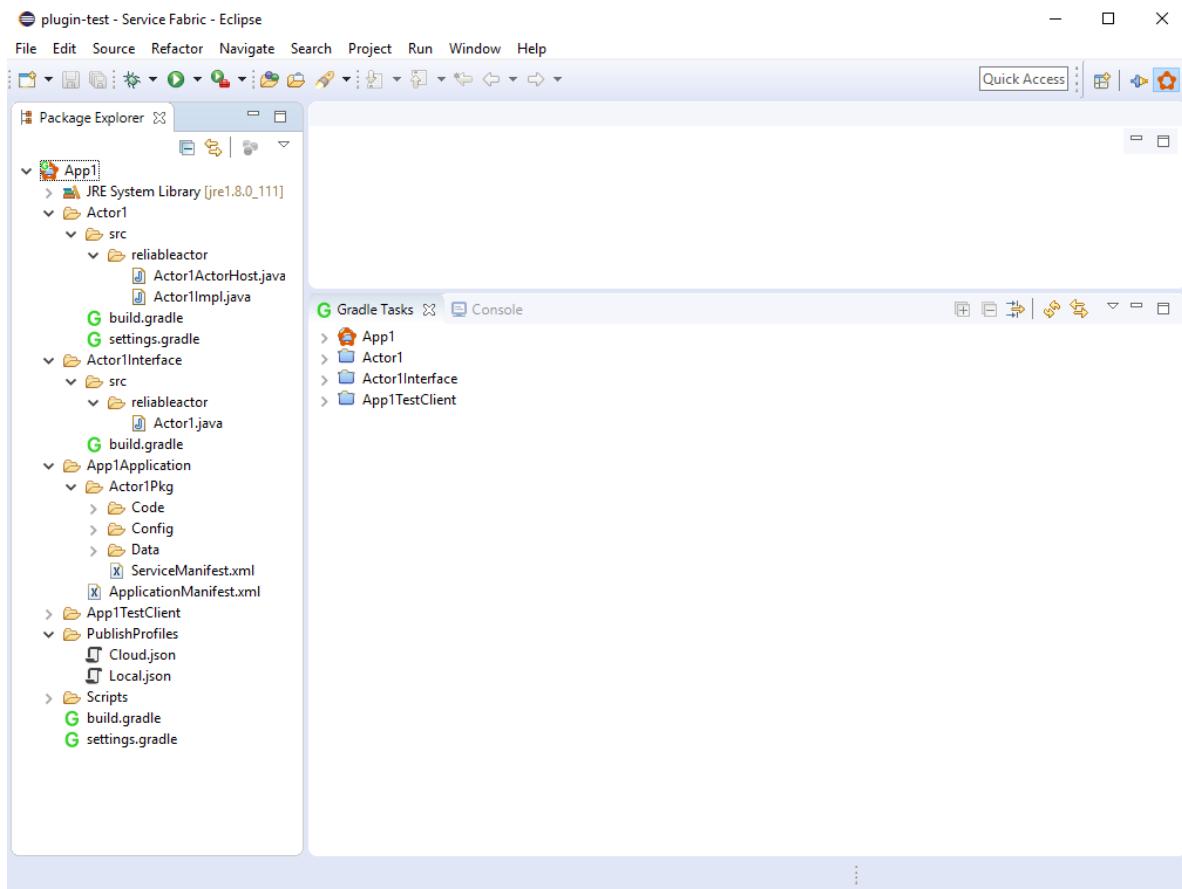
4. Enter the service name and service details, and then click **Finish**.



5. When you create your first Service Fabric project, in the **Open Associated Perspective** dialog box, click **Yes**.

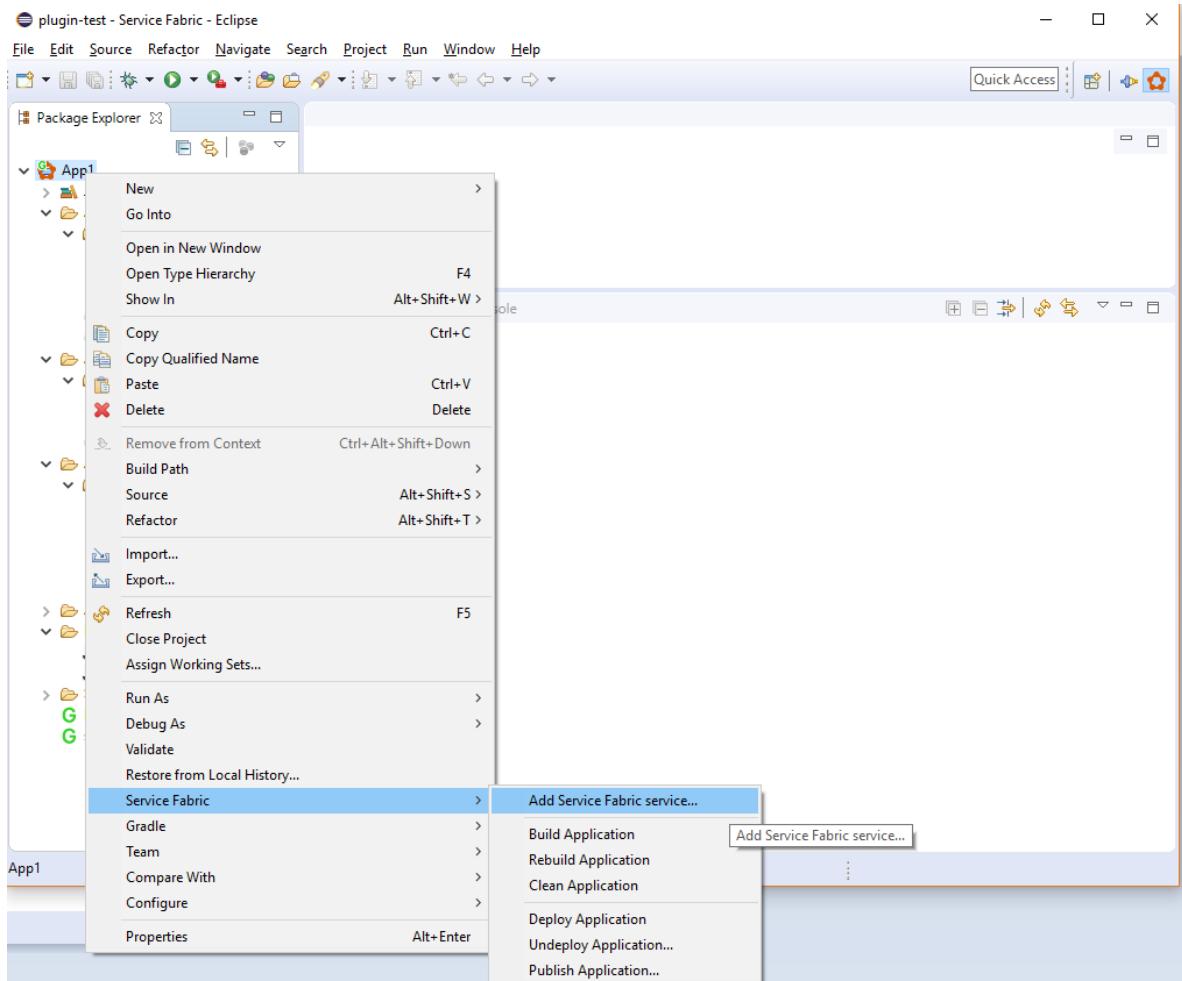


6. Your new project looks like this:



## Build and deploy a Service Fabric application in Eclipse

1. Right-click your new Service Fabric application, and then select **Service Fabric**.



2. In the submenu, select the option you want:

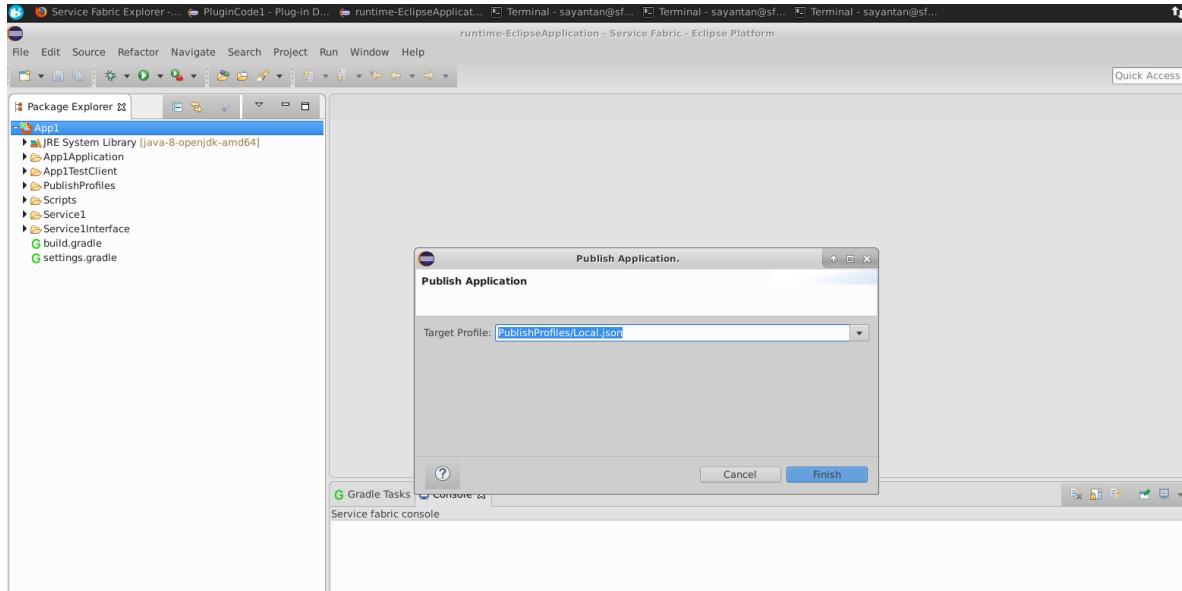
- To build the application without cleaning, click **Build Application**.
- To do a clean build of the application, click **Rebuild Application**.
- To clean the application of built artifacts, click **Clean Application**.

3. From this menu, you also can deploy, undeploy, and publish your application:

- To deploy to your local cluster, click **Deploy Application**.
- In the **Publish Application** dialog box, select a publish profile:

- **Local.json**
- **Cloud.json**

These JavaScript Object Notation (JSON) files store information (such as connection endpoints and security information) that is required to connect to your local or cloud (Azure) cluster.



An alternate way to deploy your Service Fabric application is by using Eclipse run configurations.

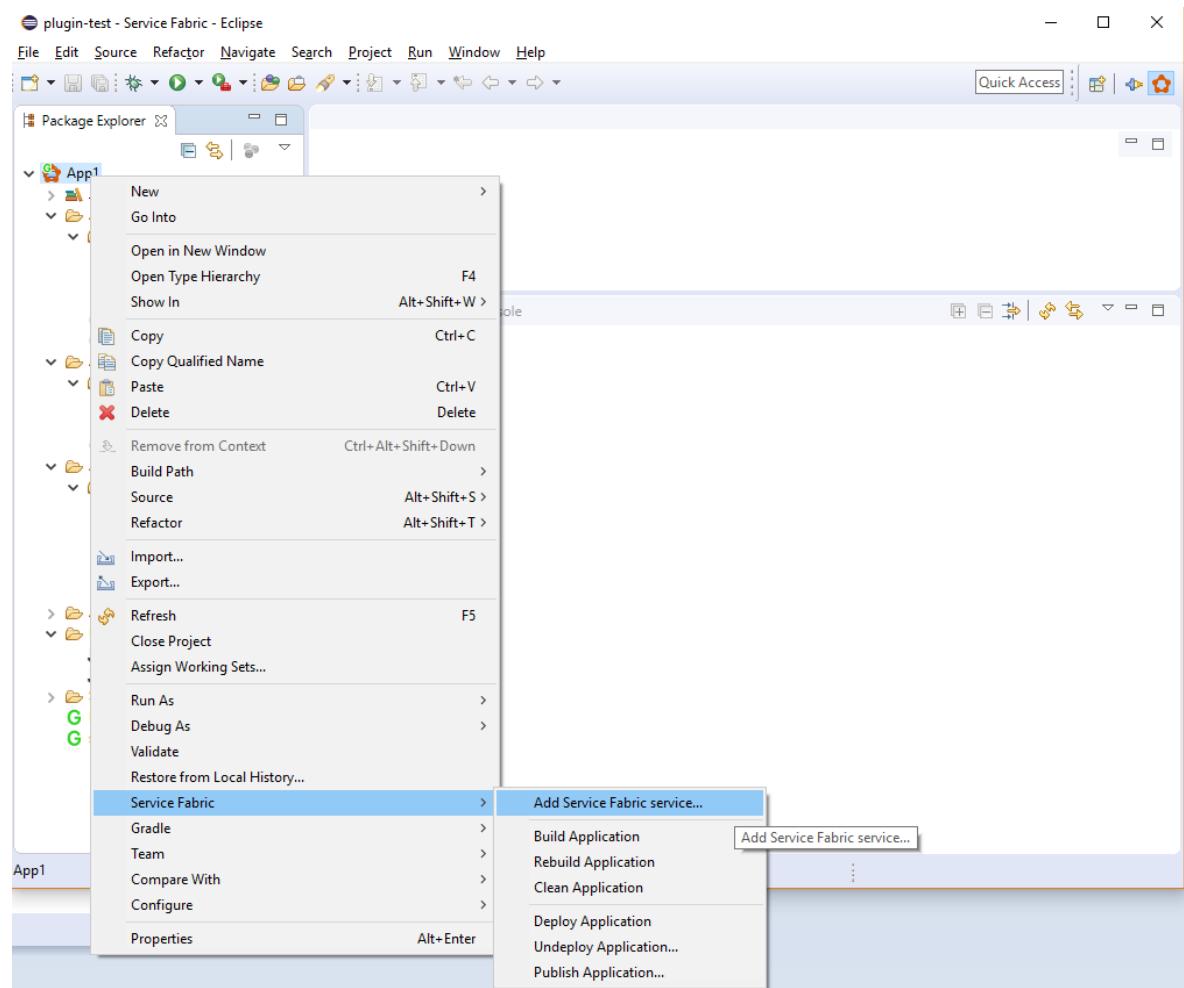
1. Go to **Run > Run Configurations**.
2. Under **Gradle Project**, select the **ServiceFabricDeployer** run configuration.
3. In the right pane, on the **Arguments** tab, for **publishProfile**, select **local** or **cloud**. The default is **local**. To deploy to a remote or cloud cluster, select **cloud**.
4. To ensure that the proper information is populated in the publish profiles, edit **Local.json** or **Cloud.json** as needed. You can add or update endpoint details and security credentials.
5. Ensure that **Working Directory** points to the application you want to deploy. To change the application, click the **Workspace** button, and then select the application you want.
6. Click **Apply**, and then click **Run**.

Your application builds and deploys within a few moments. You can monitor the deployment status in Service Fabric Explorer.

## Add a Service Fabric service to your Service Fabric application

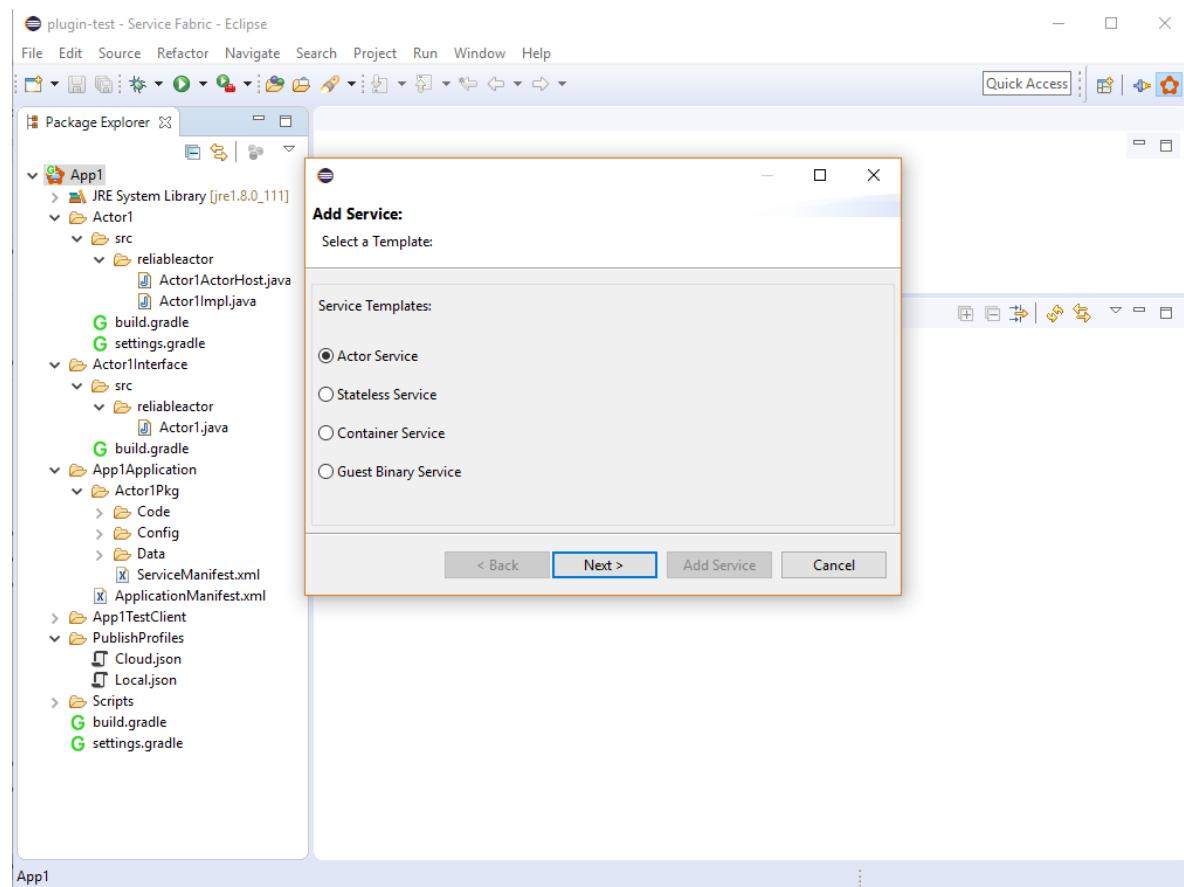
To add a Service Fabric service to an existing Service Fabric application, do the following steps:

1. Right-click the project you want to add a service to, and then click **Service Fabric**.

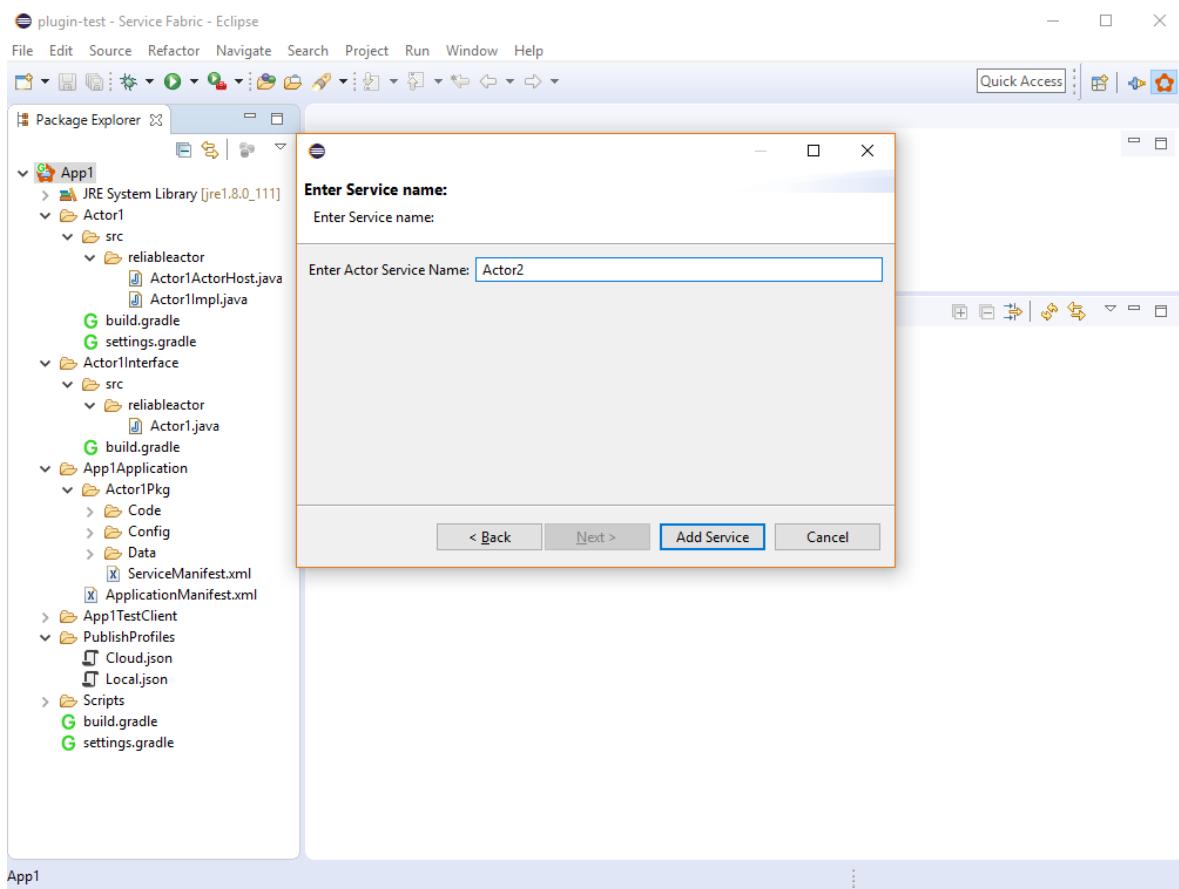


2. Click **Add Service Fabric Service**, and complete the set of steps to add a service to the project.

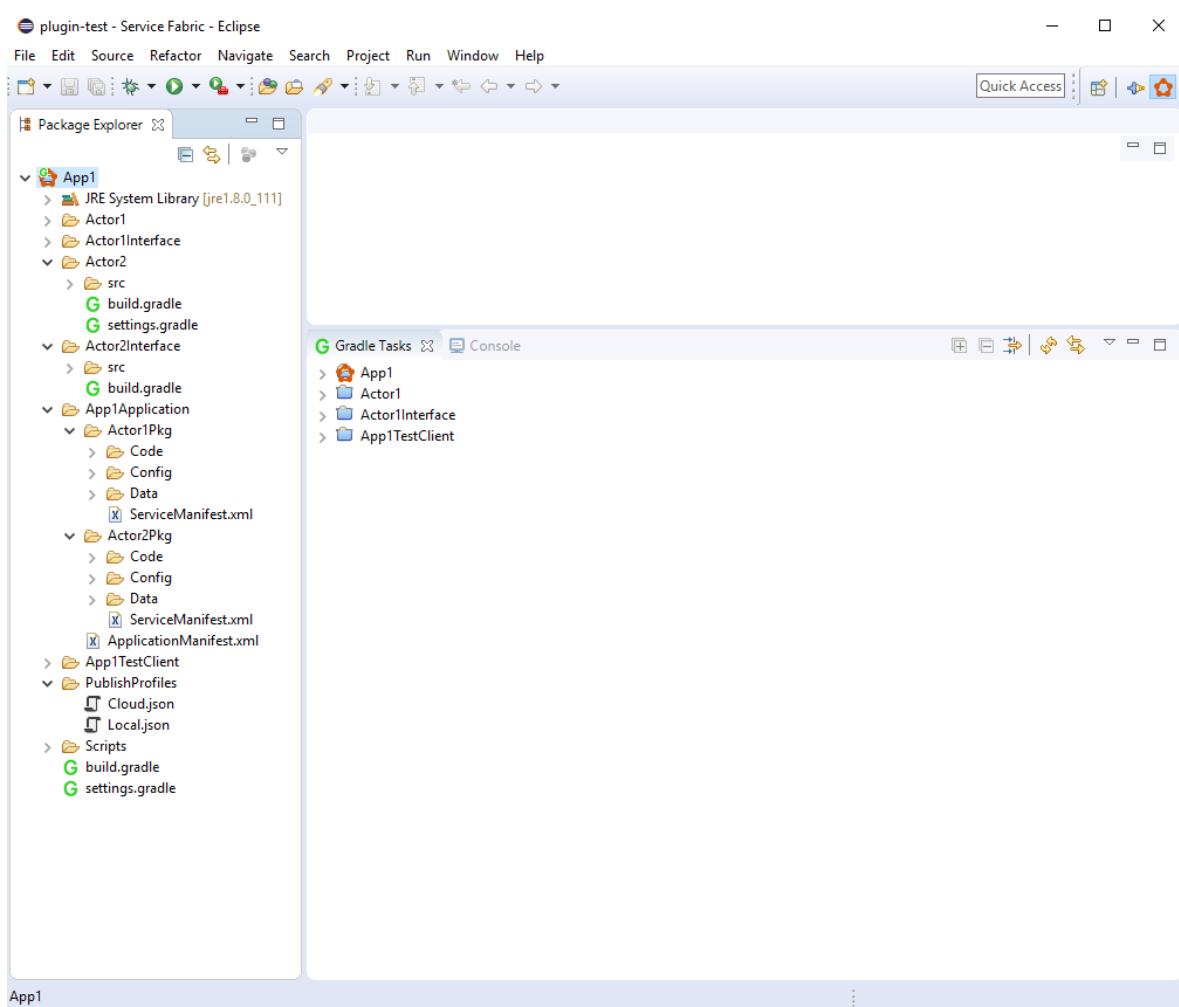
3. Select the service template you want to add to your project, and then click **Next**.



4. Enter the service name (and other details, as needed), and then click the **Add Service** button.



5. After the service is added, your overall project structure looks similar to the following project:



## Edit Manifest versions of your Service Fabric Java application

To edit manifest versions, right click on the project, go to **Service Fabric** and select **Edit Manifest Versions...** from the menu dropdown. In the wizard, you can update the manifest versions for application manifest, service manifest and the versions for **Code**, **Config** and **Data** packages.

If you check the option **Automatically update application and service versions** and then update a version, then the manifest versions will be automatically updated. To give an example, you first select the check-box, then update the version of **Code** version from 0.0.0 to 0.0.1 and click on **Finish**, then service manifest version and application manifest version will be automatically updated to 0.0.1.

## Upgrade your Service Fabric Java application

For an upgrade scenario, say you created the **App1** project by using the Service Fabric plug-in in Eclipse. You deployed it by using the plug-in to create an application named **fabric:/App1Application**. The application type is **App1ApplicationType**, and the application version is 1.0. Now, you want to upgrade your application without interrupting availability.

First, make any changes to your application, and then rebuild the modified service. Update the modified service's manifest file (ServiceManifest.xml) with the updated versions for the service (and Code, Config, or Data, as relevant). Also, modify the application's manifest (ApplicationManifest.xml) with the updated version number for the application and the modified service.

To upgrade your application by using Eclipse Neon, you can create a duplicate run configuration profile. Then, use it to upgrade your application as needed.

1. Go to **Run > Run Configurations**. In the left pane, click the small arrow to the left of **Gradle Project**.
2. Right-click **ServiceFabricDeployer**, and then select **Duplicate**. Enter a new name for this configuration, for example, **ServiceFabricUpgrader**.
3. In the right panel, on the **Arguments** tab, change **-Pconfig='deploy'** to **-Pconfig='upgrade'**, and then click **Apply**.

This process creates and saves a run configuration profile you can use at any time to upgrade your application. It also gets the latest updated application type version from the application manifest file.

The application upgrade takes a few minutes. You can monitor the application upgrade in Service Fabric Explorer.

## Migrating old Service Fabric Java applications to be used with Maven

We have recently moved Service Fabric Java libraries from Service Fabric Java SDK to Maven repository. While the new applications you generate using Eclipse, will generate latest updated projects (which will be able to work with Maven), you can update your existing Service Fabric stateless or actor Java applications, which were using the Service Fabric Java SDK earlier, to use the Service Fabric Java dependencies from Maven. Please follow the steps mentioned [here](#) to ensure your older application works with Maven.

# Debug your Java Service Fabric application using Eclipse

6/27/2017 • 1 min to read • [Edit Online](#)

1. Start a local development cluster by following the steps in [Setting up your Service Fabric development environment](#).
2. Update entryPoint.sh of the service you wish to debug, so that it starts the java process with remote debug parameters. This file can be found at the following location:

`ApplicationName\ServiceNamePkg\Code\entrypoint.sh`. Port 8001 is set for debugging in this example.

```
java -Xdebug -Xrunjdwp:transport=dt_socket,address=8001,server=y,suspend=y -  
Djava.library.path=$LD_LIBRARY_PATH -jar myapp.jar
```

3. Update the Application Manifest by setting the instance count or the replica count for the service that is being debugged to 1. This setting avoids conflicts for the port that is used for debugging. For example, for stateless services, set `InstanceCount="1"` and for stateful services set the target and min replica set sizes to 1 as follows: `TargetReplicaSetSize="1" MinReplicaSetSize="1"`.
4. Deploy the application.

5. In the Eclipse IDE, select **Run -> Debug Configurations -> Remote Java Application and input connection properties** and set the properties as follows:

Host: ipaddress  
Port: 8001

6. Set breakpoints at desired points and debug the application.

If the application is crashing, you may also want to enable core dumps. Execute `ulimit -c` in a shell and if it returns 0, then core dumps are not enabled. To enable unlimited core dumps, execute the following command:

`ulimit -c unlimited`. You can also verify the status using the command `ulimit -a`. If you wanted to update the core dump generation path, execute `echo '/tmp/core_%e.%p' | sudo tee /proc/sys/kernel/core_pattern`.

## Next steps

- [Collect logs using Linux Azure Diagnostics](#).
- [Monitor and diagnose services locally](#).

# Monitor and diagnose services in a local machine development setup

8/15/2017 • 3 min to read • [Edit Online](#)

Monitoring, detecting, diagnosing, and troubleshooting allow for services to continue with minimal disruption to the user experience. Monitoring and diagnostics are critical in an actual deployed production environment. Adopting a similar model during development of services ensures that the diagnostic pipeline works when you move to a production environment. Service Fabric makes it easy for service developers to implement diagnostics that can seamlessly work across both single-machine local development setups and real-world production cluster setups.

## Debugging Service Fabric Java applications

For Java applications, [multiple logging frameworks](#) are available. Since `java.util.logging` is the default option with the JRE, it is also used for the [code examples in github](#). The following discussion explains how to configure the `java.util.logging` framework.

Using `java.util.logging` you can redirect your application logs to memory, output streams, console files, or sockets. For each of these options, there are default handlers already provided in the framework. You can create a `app.properties` file to configure the file handler for your application to redirect all logs to a local file.

The following code snippet contains an example configuration:

```
handlers = java.util.logging.FileHandler

java.util.logging.FileHandler.level = ALL
java.util.logging.FileHandler.formatter = java.util.logging.SimpleFormatter
java.util.logging.FileHandler.limit = 1024000
java.util.logging.FileHandler.count = 10
java.util.logging.FileHandler.pattern = /tmp/servicefabric/logs/mysapp%u.%g.log
```

The folder pointed to by the `app.properties` file must exist. After the `app.properties` file is created, you need to also modify your entry point script, `entrypoint.sh` in the `<applicationfolder>/<servicePkg>/Code/` folder to set the property `java.util.logging.config.file` to `app.properties` file. The entry should look like the following snippet:

```
java -Djava.library.path=$LD_LIBRARY_PATH -Djava.util.logging.config.file=<path to app.properties> -jar <service name>.jar
```

This configuration results in logs being collected in a rotating fashion at `/tmp/servicefabric/logs/`. The log file in this case is named mysapp%u.%g.log where:

- **%u** is a unique number to resolve conflicts between simultaneous Java processes.
- **%g** is the generation number to distinguish between rotating logs.

By default if no handler is explicitly configured, the console handler is registered. One can view the logs in syslog under `/var/log/syslog`.

For more information, see the [code examples in github](#).

## Debugging Service Fabric C# applications

Multiple frameworks are available for tracing CoreCLR applications on Linux. For more information, see [GitHub: logging](#). Since EventSource is familiar to C# developers, this article uses EventSource for tracing in CoreCLR samples on Linux.

The first step is to include System.Diagnostics.Tracing so that you can write your logs to memory, output streams, or console files. For logging using EventSource, add the following project to your project.json:

```
"System.Diagnostics.StackTrace": "4.0.1"
```

You can use a custom EventListener to listen for the service event and then appropriately redirect them to trace files. The following code snippet shows a sample implementation of logging using EventSource and a custom EventListener:

```
public class ServiceEventSource : EventSource
{
    public static ServiceEventSource Current = new ServiceEventSource();

    [NonEvent]
    public void Message(string message, params object[] args)
    {
        if (this.IsEnabled())
        {
            var finalMessage = string.Format(message, args);
            this.Message(finalMessage);
        }
    }

    // TBD: Need to add method for sample event.
}
```

```
internal class ServiceEventListener : EventListener
{

    protected override void OnEventSourceCreated(EventSource eventSource)
    {
        EnableEvents(eventSource, EventLevel.LogAlways, EventKeywords.All);
    }
    protected override void OnEventWritten(EventWrittenEventArgs eventData)
    {
        using (StreamWriter Out = new StreamWriter( new FileStream("/tmp/MyServiceLog.txt",
        FileMode.Append)))
        {
            // report all event information
            Out.WriteLine("{0} {1} {2} {3} {4}", eventData.Task.ToString(), eventData.EventName,
            eventData.EventId.ToString(), eventData.Level, "");
            if (eventData.Message != null)
                Out.WriteLine(eventData.Message, eventData.Payload.ToArray());
            else
            {
                string[] sargs = eventData.Payload != null ? eventData.Payload.Select(o =>
o.ToString()).ToArray() : null;
                Out.WriteLine("{0}{1}{2}", "{0}.", sargs != null ? string.Join(", ", sargs) : "");
            }
        }
    }
}
```

The preceding snippet outputs the logs to a file in `/tmp/MyServiceLog.txt`. This file name needs to be appropriately updated. In case you want to redirect the logs to console, use the following snippet in your customized

EventListener class:

```
public static TextWriter Out = Console.Out;
```

The samples at [C# Samples](#) use EventSource and a custom EventListener to log events to a file.

## Next steps

The same tracing code added to your application also works with the diagnostics of your application on an Azure cluster. Check out these articles that discuss the different options for the tools and describe how to set them up.

- [How to collect logs with Azure Diagnostics](#)

# Service Fabric with Azure API Management Quick Start

10/4/2017 • 12 min to read • [Edit Online](#)

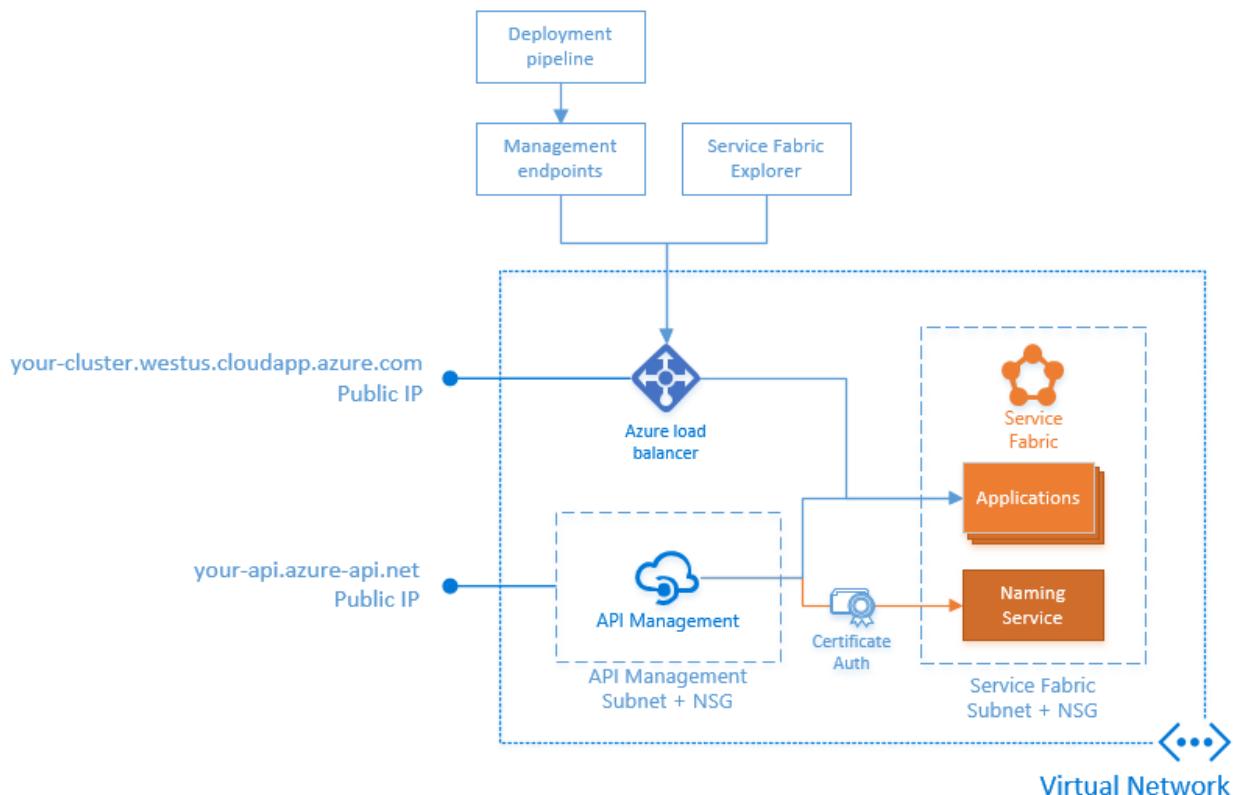
This guide shows you how to set up Azure API Management with Service Fabric and configure your first API operation to send traffic to back-end services in Service Fabric. To learn more about Azure API Management scenarios with Service Fabric, see the [overview](#) article.

## Deploy API Management and Service Fabric to Azure

The first step is to deploy API Management and a Service Fabric cluster to Azure in a shared Virtual Network. This allows API Management to communicate directly with Service Fabric so it can perform service discovery, service partition resolution, and forward traffic directly to any backend service in Service Fabric.

### Topology

This guide deploys the following topology to Azure in which API Management and Service Fabric are in subnets of the same Virtual Network:



To get started quickly, Resource Manager templates are provided for each deployment step:

- Network topology:
  - [network.json](#)
  - [network.parameters.json](#)
- Service Fabric cluster:
  - [cluster.json](#)
  - [cluster.parameters.json](#)
- API Management:

- o [apim.json](#)
- o [apim.parameters.json](#)

## Sign in to Azure and select your subscription

This guide uses [Azure PowerShell](#). When you start a new PowerShell session, sign in to your Azure account and select your subscription before you execute Azure commands.

Sign in to your Azure account:

```
Login-AzureRmAccount
```

Select your subscription:

```
Get-AzureRmSubscription  
Set-AzureRmContext -SubscriptionId <guid>
```

## Create a resource group

Create a new resource group for your deployment. Give it a name and a location.

```
New-AzureRmResourceGroup -Name <my-resource-group> -Location westus
```

## Deploy the network topology

The first step is to set up the network topology to which API Management and the Service Fabric cluster will be deployed. The [network.json](#) Resource Manager template is configured to create a Virtual Network (VNET) with two subnets and two Network Security Groups (NSG).

The [network.parameters.json](#) parameters file contains the names of the subnets and NSGs that API Management and Service Fabric will be deployed to. For this guide, the parameter values do not need to be changed. The API Management and Service Fabric Resource Manager templates use these values, so if they are modified here, you must modify them in the other Resource Manager templates accordingly.

1. Download the following Resource Manager template and parameters file:

- [network.json](#)
- [network.parameters.json](#)

2. Use the following PowerShell command to deploy the Resource Manager template and parameter files for the network setup:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName <my-resource-group> -TemplateFile .\network.json -  
TemplateParameterFile .\network.parameters.json -Verbose
```

## Deploy the Service Fabric cluster

Once the network resources have finished deploying, the next step is to deploy a Service Fabric cluster to the VNET in the subnet and NSG designated for the Service Fabric cluster. For this tutorial, the Service Fabric Resource Manager template is pre-configured to use the names of the VNET, subnet, and NSG that you set up in the previous step.

The Service Fabric cluster Resource Manager template is configured to create a secure cluster with certificate security. The certificate is used to secure node-to-node communication for your cluster and to manage user access to your Service Fabric cluster. API Management uses this certificate to access the Service Fabric Naming Service for service discovery.

This step requires having a certificate in Key Vault for cluster security. For more information on setting up a secure cluster with Key Vault, see [this guide on creating a cluster in Azure using Resource Manager](#)

#### NOTE

You may add Azure Active Directory authentication in addition to the certificate used for cluster access. Azure Active Directory is the recommended way to manage user access to your Service Fabric cluster, but is not necessary to complete this tutorial. A certificate is required either way for cluster node-to-node security and for Azure API Management authentication, which currently does not support authenticating with Azure Active Directory for a Service Fabric backend.

1. Download the following Resource Manager template and parameters file:
  - [cluster.json](#)
  - [cluster.parameters.json](#)
2. Fill in the empty parameters in the `cluster.parameters.json` file for your deployment, including the [Key Vault information](#) for your cluster certificate.
3. Use the following PowerShell command to deploy the Resource Manager template and parameter files to create the Service Fabric cluster:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName <my-resource-group> -TemplateFile .\cluster.json -TemplateParameterFile .\cluster.parameters.json -Verbose
```

## Deploy API Management

Finally, deploy API Management to the VNET in the subnet and NSG designated for API Management. You do not need to wait for the Service Fabric cluster deployment to finish before deploying API Management.

For this tutorial, the API Management Resource Manager template is pre-configured to use the names of the VNET, subnet, and NSG that you set up in the previous step.

1. Download the following Resource Manager template and parameters file:
  - [apim.json](#)
  - [apim.parameters.json](#)
2. Fill in the empty parameters in the `apim.parameters.json` for your deployment.
3. Use the following PowerShell command to deploy the Resource Manager template and parameter files for API Management:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName <my-resource-group> -TemplateFile .\apim.json -TemplateParameterFile .\apim.parameters.json -Verbose
```

## Configure API Management

Once your API Management and Service Fabric cluster are deployed, you can configure a Service Fabric backend in API Management. This allows you to create a backend service policy that sends traffic to your Service Fabric cluster.

### API Management Security

To configure the Service Fabric backend, you first need to configure API Management security settings. To configure security settings, go to your API Management service in the Azure portal.

#### Enable the API Management REST API

The API Management REST API is currently the only way to configure a backend service. The first step is to enable the API Management REST API and secure it.

1. In the API Management service, select **Management API - PREVIEW** under **Security**.
2. Check the **Enable API Management REST API** checkbox.
3. Note the Management API URL - this is the URL we'll use later to set up the Service Fabric backend
4. Generate an **access Token** by selecting an expiry date and a key, then click the **Generate** button toward the bottom of the page.
5. Copy the **access token** and save it - we'll use this in the following steps. Note this is different from the primary key and secondary key.

#### Upload a Service Fabric client certificate

API Management must authenticate with your Service Fabric cluster for service discovery using a client certificate that has access to your cluster. For simplicity, this tutorial uses the same certificate specified when creating the Service Fabric cluster, which by default can be used to access your cluster.

1. In the API Management service, select **Client certificates - PREVIEW** under **Security**.
2. Click the **+ Add** button
3. Select the private key file (.pfx) of the cluster certificate that you specified when creating your Service Fabric cluster, give it a name, and provide the private key password.

#### NOTE

This tutorial uses the same certificate for client authentication and cluster node-to-node security. You may use a separate client certificate if you have one configured to access your Service Fabric cluster.

### Configure the backend

Now that API Management security is configured, you can configure the Service Fabric backend. For Service Fabric backends, the Service Fabric cluster is the backend, rather than a specific Service Fabric service. This allows a single policy to route to more than one service in the cluster.

This step requires the access token you generated earlier and the thumbprint for your cluster certificate you uploaded to API Management in the previous step.

#### NOTE

If you used a separate client certificate in the previous step for API Management, you need the thumbprint for the client certificate in addition to the cluster certificate thumbprint in this step.

Send the following HTTP PUT request to the API Management API URL you noted earlier when enabling the API Management REST API to configure the Service Fabric backend service. You should see an [HTTP 201 Created](#) response when the command succeeds. For more information on each field, see the API Management [backend API reference documentation](#).

HTTP command and URL:

```
PUT https://your-apim.management.azure-api.net/backends/servicefabric?api-version=2016-10-10
```

Request headers:

```
Authorization: SharedAccessSignature <your access token>
Content-Type: application/json
```

Request body:

```
{
    "description": "<description>",
    "url": "<fallback service name>",
    "protocol": "http",
    "resourceId": "<cluster HTTP management endpoint>",
    "properties": {
        "serviceFabricCluster": {
            "managementEndpoints": [ "<cluster HTTP management endpoint>" ],
            "clientCertificateThumbprint": "<client cert thumbprint>",
            "serverCertificateThumbprints": [ "<cluster cert thumbprint>" ],
            "maxPartitionResolutionRetries" : 5
        }
    }
}
```

The **url** parameter here is a fully-qualified service name of a service in your cluster that all requests are routed to by default if no service name is specified in a backend policy. You may use a fake service name, such as "fabric:/fake/service" if you do not intend to have a fallback service. Be aware, the **url** must be in the format "fabric:/app/service" even if it is a fake fallback service.

Refer to the API Management [backend API reference documentation](#) for more details on each field.

#### Example

```
PUT https://your-apim.management.azure-api.net/backends/servicefabric?api-version=2016-10-10
Authorization: SharedAccessSignature
230948023984&Ld93cRGcNU6KZ4uVz7JlfTec4eX43Q9Nu8ndat0gBzs6+f559Pkf3iHX2cSge+r42pn35qGY3TitjrIl13hwCQ==
Content-Type: application/json

{
    "description": "My Service Fabric backend",
    "url": "fabric:/myapp/myservice",
    "protocol": "http",
    "resourceId": "https://your-cluster.westus.cloudapp.azure.com:19080",
    "properties": {
        "serviceFabricCluster": {
            "managementEndpoints": [ "https://your-cluster.westus.cloudapp.azure.com:19080" ],
            "clientCertificateThumbprint": "57bc463aba3aea3a12a18f36f44154f819f0fe32",
            "serverCertificateThumbprints": [ "57bc463aba3aea3a12a18f36f44154f819f0fe32" ],
            "maxPartitionResolutionRetries" : 5
        }
    }
}
```

## Deploy a Service Fabric back-end service

Now that you have the Service Fabric configured as a backend to API Management, you can author backend policies for your APIs that send traffic to your Service Fabric services. But first you need a service running in Service Fabric to accept requests.

#### Create a Service Fabric service with an HTTP endpoint

For this tutorial, we'll create a basic stateless ASP.NET Core Reliable Service using the default Web API project template. This creates an HTTP endpoint for your service, which you'll expose through Azure API Management:

```
/api/values
```

Start by [setting up your development environment for ASP.NET Core development](#).

Once your development environment is set up, start Visual Studio as Administrator and create an ASP.NET Core

service:

1. In Visual Studio, select File -> New Project.
2. Select the Service Fabric Application template under Cloud and name it "**ApiApplication**".
3. Select the ASP.NET Core service template and name the project "**Web ApiService**".
4. Select the Web API ASP.NET Core 1.1 project template.
5. Once the project is created, open `PackageRoot\ServiceManifest.xml` and remove the `Port` attribute from the endpoint resource configuration:

```
<Resources>
  <Endpoints>
    <Endpoint Protocol="http" Name="ServiceEndpoint" Type="Input" />
  </Endpoints>
</Resources>
```

This allows Service Fabric to specify a port dynamically from the application port range, which we opened through the Network Security Group in the cluster Resource Manager template, allowing traffic to flow to it from API Management.

6. Press F5 in Visual Studio to verify the web API is available locally.

Open Service Fabric Explorer and drill down to a specific instance of the ASP.NET Core service to see the base address the service is listening on. Add `/api/values` to the base address and open it in a browser. This invokes the Get method on the ValuesController in the Web API template. It returns the default response that is provided by the template, a JSON array that contains two strings:

```
["value1", "value2"]`
```

This is the endpoint that you'll expose through API Management in Azure.

7. Finally, deploy the application to your cluster in Azure. [Using Visual Studio](#), right-click the Application project and select **Publish**. Provide your cluster endpoint (for example, `mycluster.westus.cloudapp.azure.com:19000`) to deploy the application to your Service Fabric cluster in Azure.

An ASP.NET Core stateless service named `fabric:/ApiApplication/Web ApiService` should now be running in your Service Fabric cluster in Azure.

## Create an API operation

Now we're ready to create an operation in API Management that external clients use to communicate with the ASP.NET Core stateless service running in the Service Fabric cluster.

1. Log in to the Azure portal and navigate to your API Management service deployment.
2. In the API Management service blade, select **APIs - Preview**
3. Add a new API by clicking the **Blank API** box and filling out the dialog box:
  - **Web service URL:** For Service Fabric backends, this URL value is not used. You can put any value here. For this tutorial, use: `http://servicefabric`.
  - **Name:** Provide any name for your API. For this tutorial, use `Service Fabric App`.
  - **URL scheme:** Select either HTTP, HTTPS, or both. For this tutorial, use `both`.
  - **API URL Suffix:** Provide a suffix for our API. For this tutorial, use `myapp`.
4. Once the API is created, click **+ Add operation** to add a front-end API operation. Fill out the values:
  - **URL:** Select `GET` and provide a URL path for the API. For this tutorial, use `/api/values`.

By default, the URL path specified here is the URL path sent to the backend Service Fabric service. If you use the same URL path here that your service uses, in this case `/api/values`, then the operation works without further modification. You may also specify a URL path here that is different from the URL path used by your backend Service Fabric service, in which case you will also need to specify a path rewrite in your operation policy later.

- **Display name:** Provide any name for the API. For this tutorial, use `Values`.

## Configure a backend policy

The backend policy ties everything together. This is where you configure the backend Service Fabric service to which requests are routed. You can apply this policy to any API operation. The [backend configuration for Service Fabric](#) provides the following request routing controls:

- Service instance selection by specifying a Service Fabric service instance name, either hardcoded (for example, `"fabric:/myapp/myservice"`) or generated from the HTTP request (for example, `"fabric:/myapp/users/" + context.Request.MatchedParameters["name"]`).
- Partition resolution by generating a partition key using any Service Fabric partitioning scheme.
- Replica selection for stateful services.
- Resolution retry conditions that allow you to specify the conditions for re-resolving a service location and resending a request.

For this tutorial, create a backend policy that routes requests directly to the ASP.NET Core stateless service deployed earlier:

1. Select and edit the **inbound policies** for the `Values` operation by clicking the edit icon, and then selecting **Code View**.
2. In the policy code editor, add a `set-backend-service` policy under inbound policies as shown here and click the **Save** button:

```
<policies>
  <inbound>
    <base/>
    <set-backend-service
      backend-id="servicefabric"
      sf-service-instance-name="fabric:/ApiApplication/Web ApiService"
      sf-resolve-condition="@((int)context.Response.StatusCode != 200)" />
  </inbound>
  <backend>
    <base/>
  </backend>
  <outbound>
    <base/>
  </outbound>
</policies>
```

For a full set of Service Fabric back-end policy attributes, refer to the [API Management back-end documentation](#)

### Add the API to a product.

Before you can call the API, it must be added to a product where you can grant access to users.

1. In the API Management service, select **Products - PREVIEW**.
2. By default, API Management providers two products: Starter and Unlimited. Select the Unlimited product.
3. Select APIs.
4. Click the **+ Add** button.
5. Select the `Service Fabric App` API you created in the previous steps and click the **Select** button.

## Test it

You can now try sending a request to your back-end service in Service Fabric through API Management directly from the Azure portal.

1. In the API Management service, select **API - PREVIEW**.
2. In the `Service Fabric App` API you created in the previous steps, select the **Test** tab.
3. Click the **Send** button to send a test request to the backend service.

## Next steps

At this point, you should have a basic setup with Service Fabric and API Management.

This tutorial uses basic certificate-based user authentication for your Service Fabric cluster to get you set up quickly. More advanced user authentication for your Service Fabric cluster is preferable with [Azure Active Directory authentication](#).

Next, [create and configure advanced product settings in Azure API Management](#) to prepare your application for real world traffic.

# Learn about the differences between Cloud Services and Service Fabric before migrating applications.

6/30/2017 • 5 min to read • [Edit Online](#)

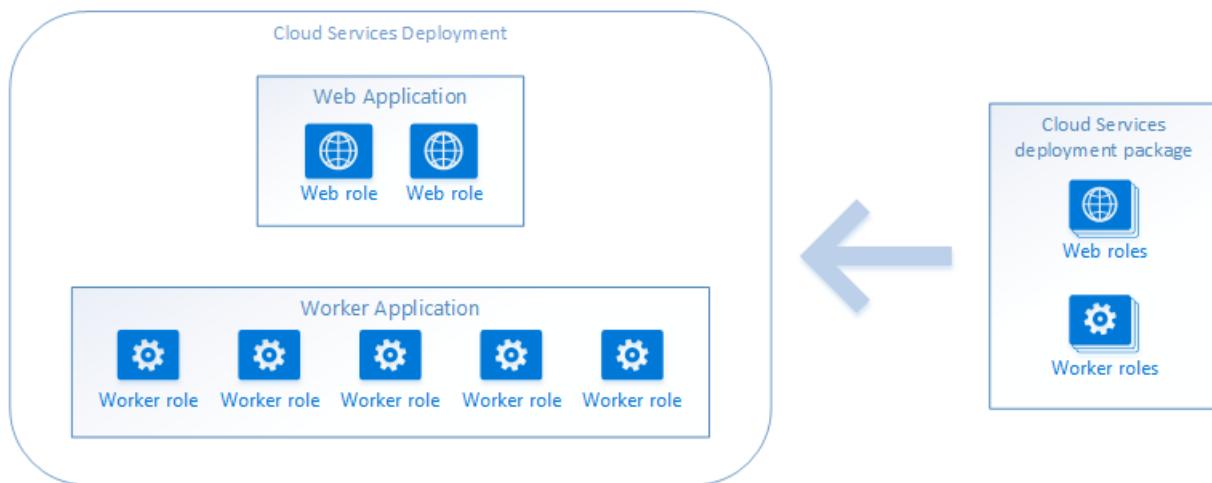
Microsoft Azure Service Fabric is the next-generation cloud application platform for highly scalable, highly reliable distributed applications. It introduces many new features for packaging, deploying, upgrading, and managing distributed cloud applications.

This is an introductory guide to migrating applications from Cloud Services to Service Fabric. It focuses primarily on architectural and design differences between Cloud Services and Service Fabric.

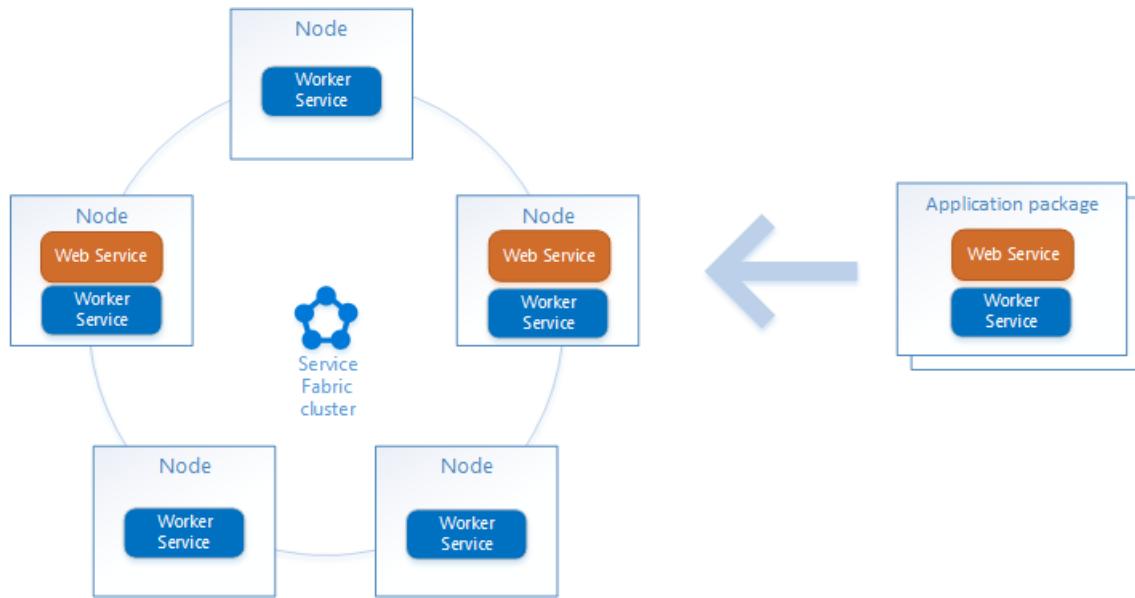
## Applications and infrastructure

A fundamental difference between Cloud Services and Service Fabric is the relationship between VMs, workloads, and applications. A workload here is defined as the code you write to perform a specific task or provide a service.

- **Cloud Services is about deploying applications as VMs.** The code you write is tightly coupled to a VM instance, such as a Web or Worker Role. To deploy a workload in Cloud Services is to deploy one or more VM instances that run the workload. There is no separation of applications and VMs, and so there is no formal definition of an application. An application can be thought of as a set of Web or Worker Role instances within a Cloud Services deployment or as an entire Cloud Services deployment. In this example, an application is shown as a set of role instances.



- **Service Fabric is about deploying applications to existing VMs or machines running Service Fabric on Windows or Linux.** The services you write are completely decoupled from the underlying infrastructure, which is abstracted away by the Service Fabric application platform, so an application can be deployed to multiple environments. A workload in Service Fabric is called a "service," and one or more services are grouped in a formally-defined application that runs on the Service Fabric application platform. Multiple applications can be deployed to a single Service Fabric cluster.

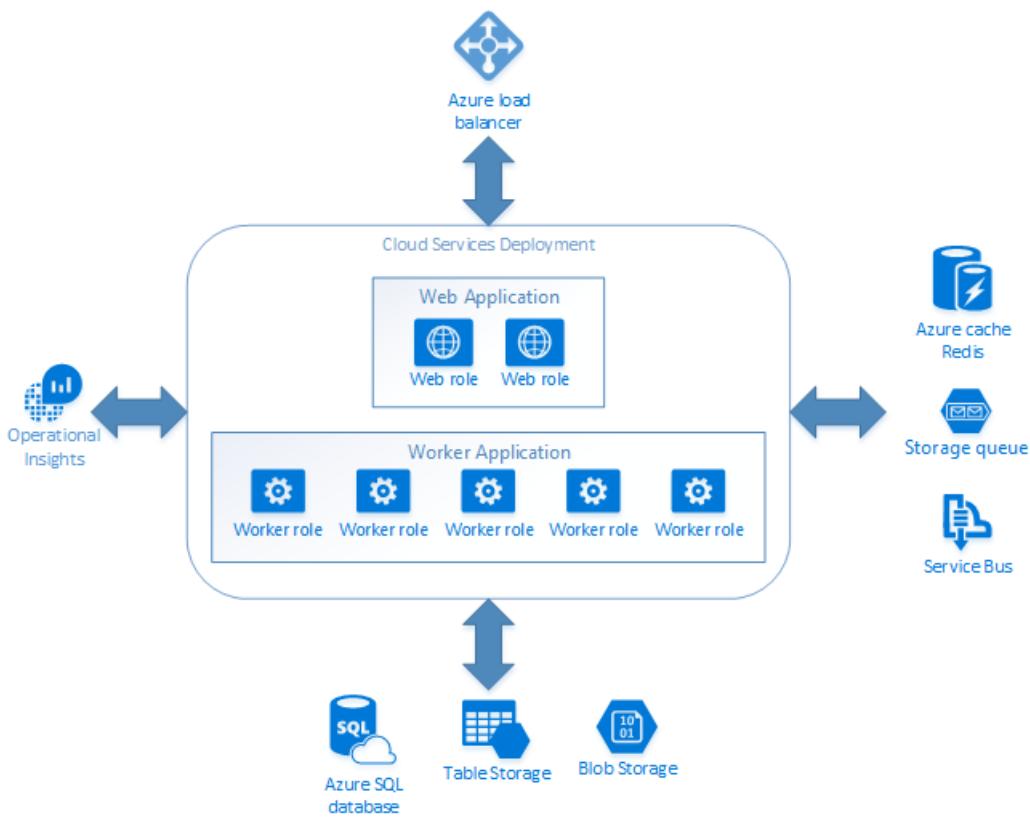


Service Fabric itself is an application platform layer that runs on Windows or Linux, whereas Cloud Services is a system for deploying Azure-managed VMs with workloads attached. The Service Fabric application model has a number of advantages:

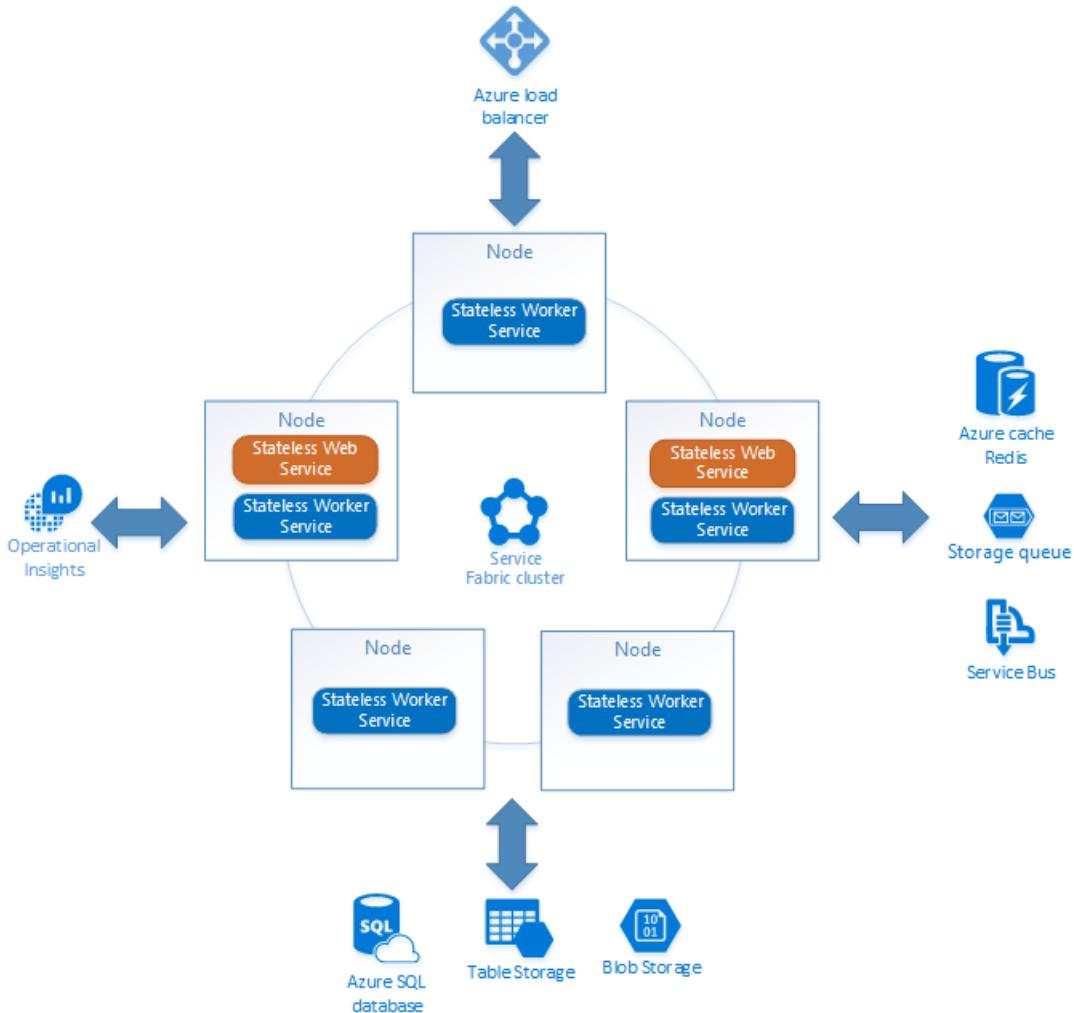
- Fast deployment times. Creating VM instances can be time consuming. In Service Fabric, VMs are only deployed once to form a cluster that hosts the Service Fabric application platform. From that point on, application packages can be deployed to the cluster very quickly.
- High-density hosting. In Cloud Services, a Worker Role VM hosts one workload. In Service Fabric, applications are separate from the VMs that run them, meaning you can deploy a large number of applications to a small number of VMs, which can lower overall cost for larger deployments.
- The Service Fabric platform can run anywhere that has Windows Server or Linux machines, whether it's Azure or on-premises. The platform provides an abstraction layer over the underlying infrastructure so your application can run on different environments.
- Distributed application management. Service Fabric is a platform that not only hosts distributed applications, but also helps manage their lifecycle independently of the hosting VM or machine lifecycle.

## Application architecture

The architecture of a Cloud Services application usually includes numerous external service dependencies, such as Service Bus, Azure Table and Blob Storage, SQL, Redis, and others to manage the state and data of an application and communication between Web and Worker Roles in a Cloud Services deployment. An example of a complete Cloud Services application might look like this:



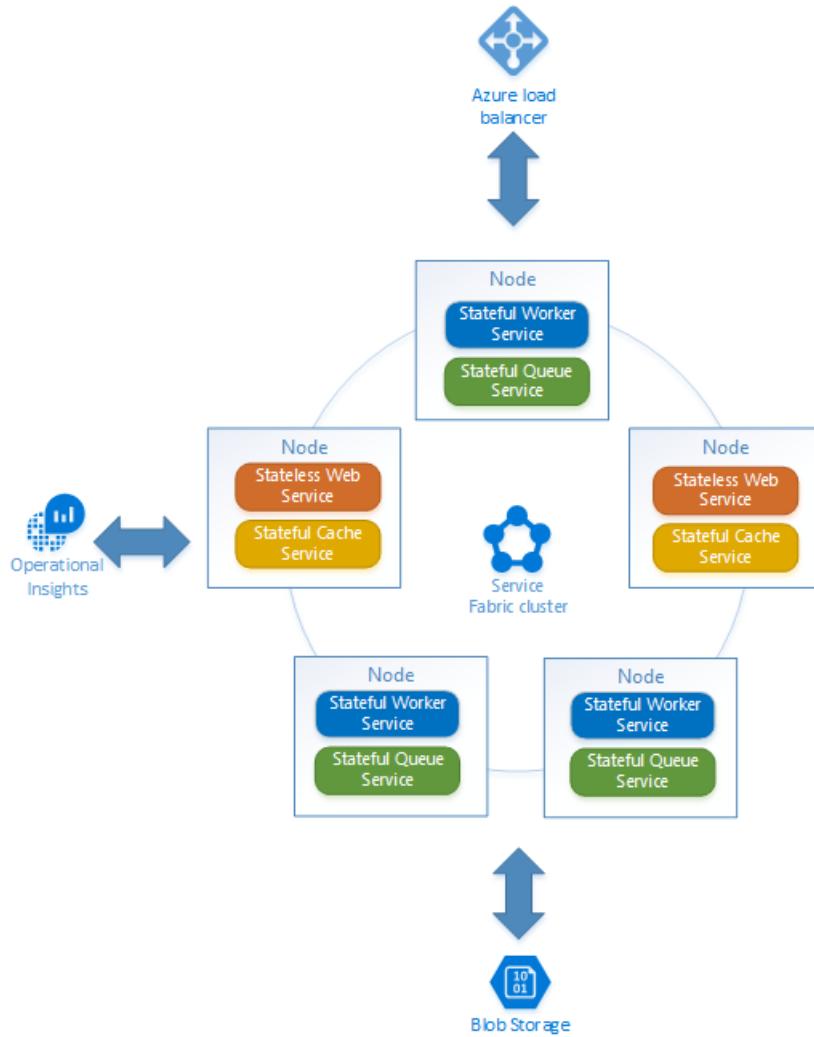
Service Fabric applications can also choose to use the same external services in a complete application. Using this example Cloud Services architecture, the simplest migration path from Cloud Services to Service Fabric is to replace only the Cloud Services deployment with a Service Fabric application, keeping the overall architecture the same. The Web and Worker Roles can be ported to Service Fabric stateless services with minimal code changes.



At this stage, the system should continue to work the same as before. Taking advantage of Service Fabric's stateful features, external state stores can be internalized as stateful services where applicable. This is more involved than a simple migration of Web and Worker Roles to Service Fabric stateless services, as it requires writing custom services that provide equivalent functionality to your application as the external services did before. The benefits of doing so include:

- Removing external dependencies
- Unifying the deployment, management, and upgrade models.

An example resulting architecture of internalizing these services could look like this:

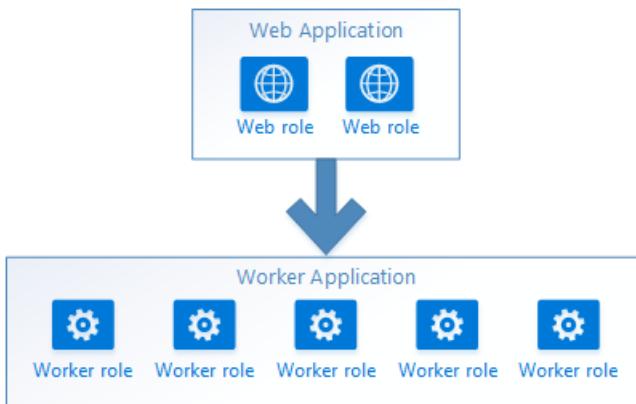


## Communication and workflow

Most Cloud Service applications consist of more than one tier. Similarly, a Service Fabric application consists of more than one service (typically many services). Two common communication models are direct communication and via an external durable storage.

### Direct communication

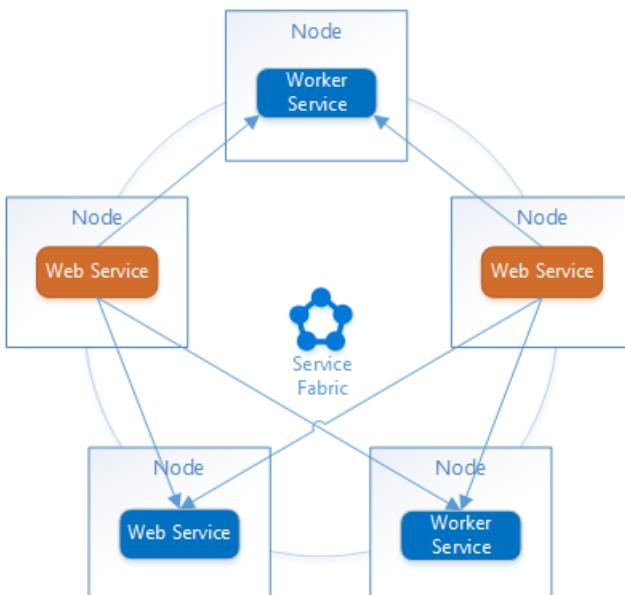
With direct communication, tiers can communicate directly through endpoint exposed by each tier. In stateless environments such as Cloud Services, this means selecting an instance of a VM role, either randomly or round-robin to balance load, and connecting to its endpoint directly.



Direct communication is a common communication model in Service Fabric. The key difference between Service Fabric and Cloud Services is that in Cloud Services you connect to a VM, whereas in Service Fabric you connect to a service. This is an important distinction for a couple reasons:

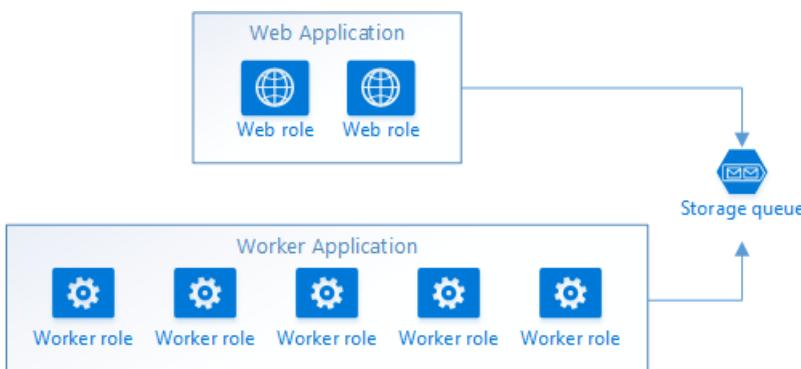
- Services in Service Fabric are not bound to the VMs that host them; services may move around in the cluster, and in fact, are expected to move around for various reasons: Resource balancing, failover, application and infrastructure upgrades, and placement or load constraints. This means a service instance's address can change at any time.
- A VM in Service Fabric can host multiple services, each with unique endpoints.

Service Fabric provides a service discovery mechanism, called the Naming Service, which can be used to resolve endpoint addresses of services.

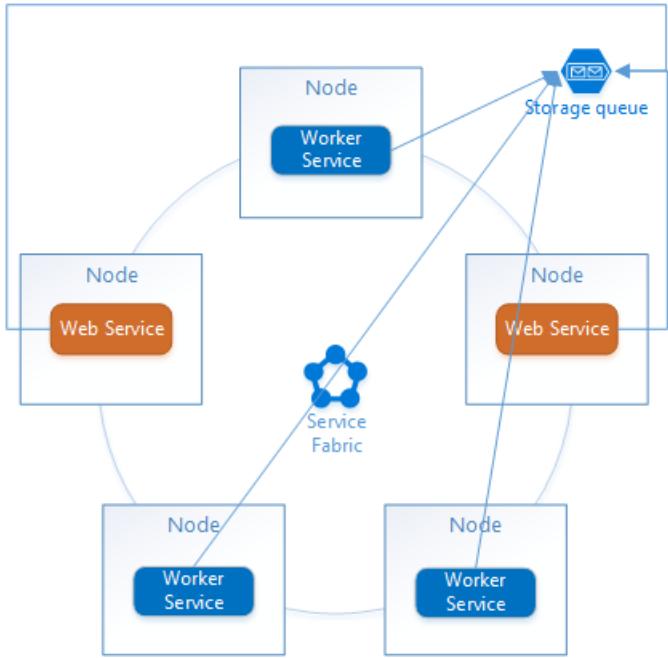


## Queues

A common communication mechanism between tiers in stateless environments such as Cloud Services is to use an external storage queue to durably store work tasks from one tier to another. A common scenario is a web tier that sends jobs to an Azure Queue or Service Bus where Worker Role instances can dequeue and process the jobs.



The same communication model can be used in Service Fabric. This can be useful when migrating an existing Cloud Services application to Service Fabric.



## Next Steps

The simplest migration path from Cloud Services to Service Fabric is to replace only the Cloud Services deployment with a Service Fabric application, keeping the overall architecture of your application roughly the same. The following article provides a guide to help convert a Web or Worker Role to a Service Fabric stateless service.

- [Simple migration: convert a Web or Worker Role to a Service Fabric stateless service](#)

# Guide to converting Web and Worker Roles to Service Fabric stateless services

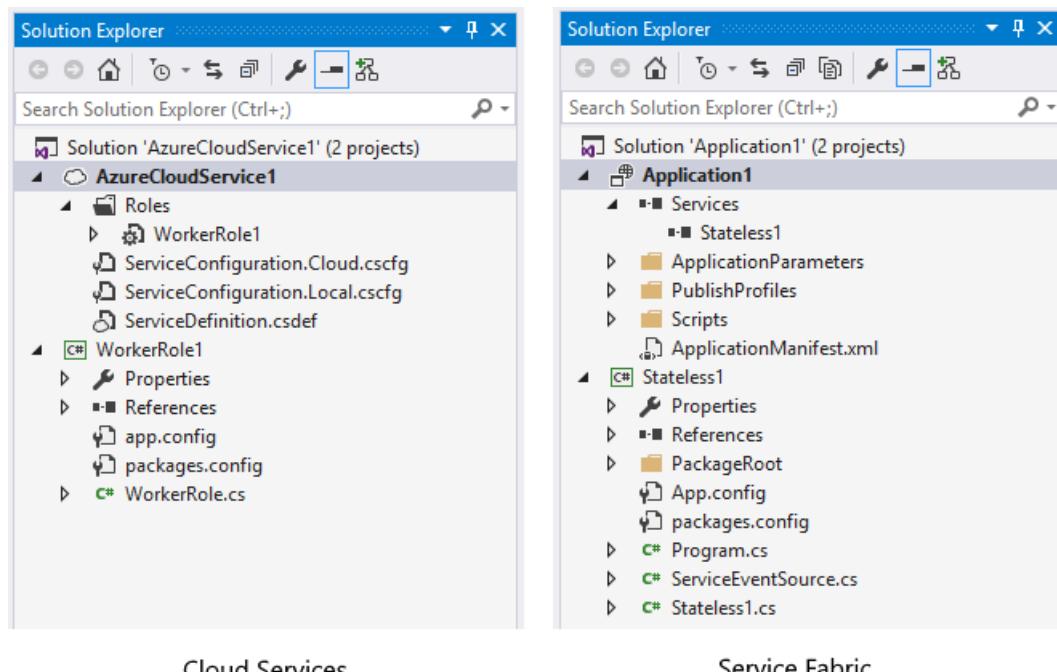
6/30/2017 • 8 min to read • [Edit Online](#)

This article describes how to migrate your Cloud Services Web and Worker Roles to Service Fabric stateless services. This is the simplest migration path from Cloud Services to Service Fabric for applications whose overall architecture is going to stay roughly the same.

## Cloud Service project to Service Fabric application project

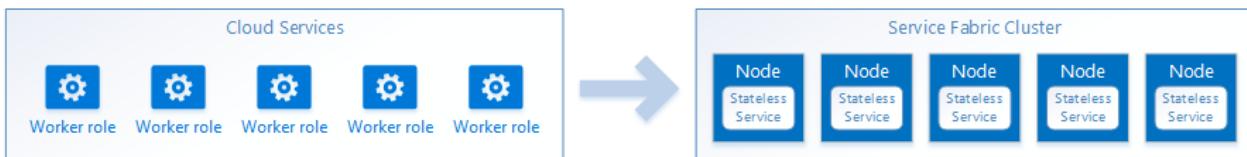
A Cloud Service project and a Service Fabric Application project have a similar structure and both represent the deployment unit for your application - that is, they each define the complete package that is deployed to run your application. A Cloud Service project contains one or more Web or Worker Roles. Similarly, a Service Fabric Application project contains one or more services.

The difference is that the Cloud Service project couples the application deployment with a VM deployment and thus contains VM configuration settings in it, whereas the Service Fabric Application project only defines an application that will be deployed to a set of existing VMs in a Service Fabric cluster. The Service Fabric cluster itself is only deployed once, either through an Resource Manager template or through the Azure portal, and multiple Service Fabric applications can be deployed to it.



## Worker Role to stateless service

Conceptually, a Worker Role represents a stateless workload, meaning every instance of the workload is identical and requests can be routed to any instance at any time. Each instance is not expected to remember the previous request. State that the workload operates on is managed by an external state store, such as Azure Table Storage or Azure Document DB. In Service Fabric, this type of workload is represented by a Stateless Service. The simplest approach to migrating a Worker Role to Service Fabric can be done by converting Worker Role code to a Stateless Service.



## Web Role to stateless service

Similar to Worker Role, a Web Role also represents a stateless workload, and so conceptually it too can be mapped to a Service Fabric stateless service. However, unlike Web Roles, Service Fabric does not support IIS. To migrate a web application from a Web Role to a stateless service requires first moving to a web framework that can be self-hosted and does not depend on IIS or System.Web, such as ASP.NET Core 1.

APPLICATION	SUPPORTED	MIGRATION PATH
ASP.NET Web Forms	No	Convert to ASP.NET Core 1 MVC
ASP.NET MVC	With Migration	Upgrade to ASP.NET Core 1 MVC
ASP.NET Web API	With Migration	Use self-hosted server or ASP.NET Core 1
ASP.NET Core 1	Yes	N/A

## Entry point API and lifecycle

Worker Role and Service Fabric service APIs offer similar entry points:

ENTRY POINT	WORKER ROLE	SERVICE FABRIC SERVICE
Processing	<code>Run()</code>	<code>RunAsync()</code>
VM start	<code>OnStart()</code>	N/A
VM stop	<code>OnStop()</code>	N/A
Open listener for client requests	N/A	<ul style="list-style-type: none"> <li>• <code>CreateServiceInstanceListener()</code> for stateless</li> <li>• <code>CreateServiceReplicaListener()</code> for stateful</li> </ul>

### Worker Role

```

using Microsoft.WindowsAzure.ServiceRuntime;

namespace WorkerRole1
{
    public class WorkerRole : RoleEntryPoint
    {
        public override void Run()
        {

        }

        public override bool OnStart()
        {
        }

        public override void OnStop()
        {
        }
    }
}

```

## Service Fabric Stateless Service

```

using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.ServiceFabric.Services.Communication.Runtime;
using Microsoft.ServiceFabric.Services.Runtime;

namespace Stateless1
{
    public class Stateless1 : StatelessService
    {
        protected override IEnumerable<ServiceInstanceListener> CreateServiceInstanceListeners()
        {
        }

        protected override Task RunAsync(CancellationToken cancellationToken)
        {
        }
    }
}

```

Both have a primary "Run" override in which to begin processing. Service Fabric services combine `Run`, `Start`, and `Stop` into a single entry point, `RunAsync`. Your service should begin working when `RunAsync` starts, and should stop working when the `RunAsync` method's `CancellationToken` is signaled.

There are several key differences between the lifecycle and lifetime of Worker Roles and Service Fabric services:

- **Lifecycle:** The biggest difference is that a Worker Role is a VM and so its lifecycle is tied to the VM, which includes events for when the VM starts and stops. A Service Fabric service has a lifecycle that is separate from the VM lifecycle, so it does not include events for when the host VM or machine starts and stop, as they are not related.
- **Lifetime:** A Worker Role instance will recycle if the `Run` method exits. The `RunAsync` method in a Service Fabric service however can run to completion and the service instance will stay up.

Service Fabric provides an optional communication setup entry point for services that listen for client requests. Both the `RunAsync` and communication entry point are optional overrides in Service Fabric services - your service may choose to only listen to client requests, or only run a processing loop, or both - which is why the `RunAsync` method is allowed to exit without restarting the service instance, because it may continue to listen for client

requests.

## Application API and environment

The Cloud Services environment API provides information and functionality for the current VM instance as well as information about other VM role instances. Service Fabric provides information related to its runtime and some information about the node a service is currently running on.

ENVIRONMENT TASK	CLOUD SERVICES	SERVICE FABRIC
Configuration Settings and change notification	<code>RoleEnvironment</code>	<code>CodePackageActivationContext</code>
Local Storage	<code>RoleEnvironment</code>	<code>CodePackageActivationContext</code>
Endpoint Information	<code>RoleInstance</code> <ul style="list-style-type: none"><li>• Current instance: <code>RoleEnvironment.CurrentRoleInstance</code></li><li>• Other roles and instance: <code>RoleEnvironment.Roles</code></li></ul>	<ul style="list-style-type: none"><li>• <code>NodeContext</code> for current Node address</li><li>• <code>FabricClient</code> and <code>ServicePartitionResolver</code> for service endpoint discovery</li></ul>
Environment Emulation	<code>RoleEnvironment.Emulated</code>	N/A
Simultaneous change event	<code>RoleEnvironment</code>	N/A

## Configuration settings

Configuration settings in Cloud Services are set for a VM role and apply to all instances of that VM role. These settings are key-value pairs set in `ServiceConfiguration.*.cscfg` files and can be accessed directly through `RoleEnvironment`. In Service Fabric, settings apply individually to each service and to each application, rather than to a VM, because a VM can host multiple services and applications. A service is composed of three packages:

- **Code:** contains the service executables, binaries, DLLs, and any other files a service needs to run.
- **Config:** all configuration files and settings for a service.
- **Data:** static data files associated with the service.

Each of these packages can be independently versioned and upgraded. Similar to Cloud Services, a config package can be accessed programmatically through an API and events are available to notify the service of a config package change. A `Settings.xml` file can be used for key-value configuration and programmatic access similar to the app settings section of an `App.config` file. However, unlike Cloud Services, a Service Fabric config package can contain any configuration files in any format, whether it's XML, JSON, YAML, or a custom binary format.

### Accessing configuration

#### Cloud Services

Configuration settings from `ServiceConfiguration.*.cscfg` can be accessed through `RoleEnvironment`. These settings are globally available to all role instances in the same Cloud Service deployment.

```
string value = RoleEnvironment.GetConfigurationSettingValue("Key");
```

#### Service Fabric

Each service has its own individual configuration package. There is no built-in mechanism for global configuration settings accessible by all applications in a cluster. When using Service Fabric's special `Settings.xml` configuration

file within a configuration package, values in Settings.xml can be overwritten at the application level, making application-level configuration settings possible.

Configuration settings are accessed within each service instance through the service's

```
CodePackageActivationContext .
```

```
ConfigurationPackage configPackage =
this.Context.CodePackageActivationContext.GetConfigurationPackageObject("Config");

// Access Settings.xml
KeyedCollection<string, ConfigurationProperty> parameters =
configPackage.Settings.Sections["MyConfigSection"].Parameters;

string value = parameters["Key"]?.Value;

// Access custom configuration file:
using (StreamReader reader = new StreamReader(Path.Combine(configPackage.Path, "CustomConfig.json")))
{
    MySettings settings = JsonConvert.DeserializeObject<MySettings>(reader.ReadToEnd());
}
```

## Configuration update events

### Cloud Services

The `RoleEnvironment.Changed` event is used to notify all role instances when a change occurs in the environment, such as a configuration change. This is used to consume configuration updates without recycling role instances or restarting a worker process.

```
RoleEnvironment.Changed += RoleEnvironmentChanged;

private void RoleEnvironmentChanged(object sender, RoleEnvironmentChangedEventArgs e)
{
    // Get the list of configuration changes
    var settingChanges = e.Changes.OfType<RoleEnvironmentConfigurationSettingChange>();
    foreach (var settingChange in settingChanges)
    {
        Trace.WriteLine("Setting: " + settingChange.ConfigurationSettingName, "Information");
    }
}
```

### Service Fabric

Each of the three package types in a service - Code, Config, and Data - have events that notify a service instance when a package is updated, added, or removed. A service can contain multiple packages of each type. For example, a service may have multiple config packages, each individually versioned and upgradeable.

These events are available to consume changes in service packages without restarting the service instance.

```
this.Context.CodePackageActivationContext.ConfigurationPackageModifiedEvent +=
    this.CodePackageActivationContext_ConfigurationPackageModifiedEvent;

private void CodePackageActivationContext_ConfigurationPackageModifiedEvent(object sender,
    PackageModifiedEventArgs<ConfigurationPackage> e)
{
    this.UpdateCustomConfig(e.NewPackage.Path);
    this.UpdateSettings(e.NewPackage.Settings);
}
```

## Startup tasks

Startup tasks are actions that are taken before an application starts. A startup task is typically used to run setup scripts using elevated privileges. Both Cloud Services and Service Fabric support start-up tasks. The main difference is that in Cloud Services, a startup task is tied to a VM because it is part of a role instance, whereas in Service Fabric a startup task is tied to a service, which is not tied to any particular VM.

CLOUD SERVICES	SERVICE FABRIC
Configuration location	ServiceDefinition.csdef
Privileges	"limited" or "elevated"
Sequencing	"simple", "background", "foreground"

### Cloud Services

In Cloud Services a startup entry point is configured per role in ServiceDefinition.csdef.

```
<ServiceDefinition>
  <Startup>
    <Task commandLine="Startup.cmd" executionContext="limited" taskType="simple" >
      <Environment>
        <Variable name="MyVersionNumber" value="1.0.0.0" />
      </Environment>
    </Task>
  </Startup>
  ...
</ServiceDefinition>
```

### Service Fabric

In Service Fabric a startup entry point is configured per service in ServiceManifest.xml:

```
<ServiceManifest>
  <CodePackage Name="Code" Version="1.0.0">
    <SetupEntryPoint>
      <ExeHost>
        <Program>Startup.bat</Program>
      </ExeHost>
    </SetupEntryPoint>
    ...
  </ServiceManifest>
```

## A note about development environment

Both Cloud Services and Service Fabric are integrated with Visual Studio with project templates and support for debugging, configuring, and deploying both locally and to Azure. Both Cloud Services and Service Fabric also provide a local development runtime environment. The difference is that while the Cloud Service development runtime emulates the Azure environment on which it runs, Service Fabric does not use an emulator - it uses the complete Service Fabric runtime. The Service Fabric environment you run on your local development machine is the same environment that runs in production.

## Next steps

Read more about Service Fabric Reliable Services and the fundamental differences between Cloud Services and Service Fabric application architecture to understand how to take advantage of the full set of Service Fabric features.

- [Getting started with Service Fabric Reliable Services](#)
- [Conceptual guide to the differences between Cloud Services and Service Fabric](#)

# Package an application

8/15/2017 • 5 min to read • [Edit Online](#)

This article describes how to package a Service Fabric application and make it ready for deployment.

## Package layout

The application manifest, one or more service manifests, and other necessary package files must be organized in a specific layout for deployment into a Service Fabric cluster. The example manifests in this article would need to be organized in the following directory structure:

```
PS D:\temp> tree /f .\MyApplicationType

D:\TEMP\MYAPPLICATIONTYPE
|   ApplicationManifest.xml
|
└── MyServiceManifest
    |   ServiceManifest.xml
    |
    ├── MyCode
    |   MyServiceHost.exe
    |
    ├── MyConfig
    |   Settings.xml
    |
    └── MyData
        init.dat
```

The folders are named to match the **Name** attributes of each corresponding element. For example, if the service manifest contained two code packages with the names **MyCodeA** and **MyCodeB**, then two folders with the same names would contain the necessary binaries for each code package.

## Use SetupEntryPoint

Typical scenarios for using **SetupEntryPoint** are when you need to run an executable before the service starts or you need to perform an operation with elevated privileges. For example:

- Setting up and initializing environment variables that the service executable needs. It is not limited to only executables written via the Service Fabric programming models. For example, npm.exe needs some environment variables configured for deploying a node.js application.
- Setting up access control by installing security certificates.

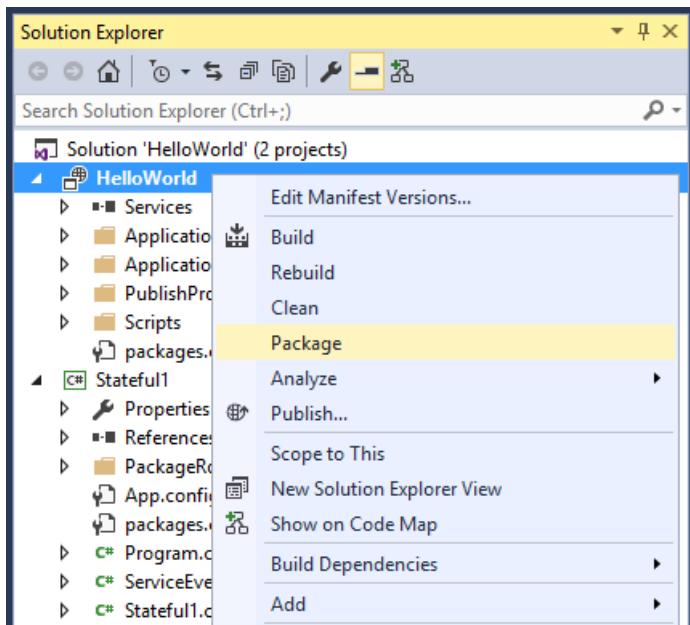
For more information on how to configure the **SetupEntryPoint**, see [Configure the policy for a service setup entry point](#)

## Configure

### Build a package by using Visual Studio

If you use Visual Studio 2015 to create your application, you can use the Package command to automatically create a package that matches the layout described above.

To create a package, right-click the application project in Solution Explorer and choose the Package command, as shown below:



When packaging is complete, you can find the location of the package in the **Output** window. The packaging step occurs automatically when you deploy or debug your application in Visual Studio.

### Build a package by command line

It is also possible to programmatically package up your application using `msbuild.exe`. Under the hood, Visual Studio is running it so the output is same.

```
D:\Temp> msbuild HelloWorld.sproj /t:Package
```

## Test the package

You can verify the package structure locally through PowerShell by using the [Test-ServiceFabricApplicationPackage](#) command. This command checks for manifest parsing issues and verify all references. This command only verifies the structural correctness of the directories and files in the package. It doesn't verify any of the code or data package contents beyond checking that all necessary files are present.

```
PS D:\temp> Test-ServiceFabricApplicationPackage .\MyApplicationType
False
Test-ServiceFabricApplicationPackage : The EntryPoint MySetup.bat is not found.
FileName:
C:\Users\servicefabric\AppData\Local\Temp\TestApplicationPackage_7195781181\nrri205a.e2h\MyApplicationType\MyServiceManifest\ServiceManifest.xml
```

This error shows that the *MySetup.bat* file referenced in the service manifest **SetupEntryPoint** is missing from the code package. After the missing file is added, the application verification passes:

```
PS D:\temp> tree /f .\MyApplicationType
D:\TEMP\MYAPPLICATIONTYPE
|   ApplicationManifest.xml
|
└── MyServiceManifest
    |   ServiceManifest.xml
    |
    └── MyCode
        |   MyServiceHost.exe
        |   MySetup.bat
        |
        └── MyConfig
            |   Settings.xml
        |
        └── MyData
            |   init.dat

PS D:\temp> Test-ServiceFabricApplicationPackage .\MyApplicationType
True
PS D:\temp>
```

If your application has [application parameters](#) defined, you can pass them in [Test-ServiceFabricApplicationPackage](#) for proper validation.

If you know the cluster where the application will be deployed, it is recommended you pass in the `ImageStoreConnectionString` parameter. In this case, the package is also validated against previous versions of the application that are already running in the cluster. For example, the validation can detect whether a package with the same version but different content was already deployed.

Once the application is packaged correctly and passes validation, evaluate based on the size and the number of files if compression is needed.

## Compress a package

When a package is large or has many files, you can compress it for faster deployment. Compression reduces the number of files and the package size. For a compressed application package, [Uploading the application package](#) may take longer compared to uploading the uncompressed package (specially if compression time is factored in), but [registering](#) and [un-registering the application type](#) are faster for a compressed application package.

The deployment mechanism is same for compressed and uncompressed packages. If the package is compressed, it is stored as such in the cluster image store and it's uncompressed on the node before the application is run. The compression replaces the valid Service Fabric package with the compressed version. The folder must allow write permissions. Running compression on an already compressed package yields no changes.

You can compress a package by running the Powershell command [Copy-ServiceFabricApplicationPackage](#) with `CompressPackage` switch. You can uncompress the package with the same command, using `UncompressPackage` switch.

The following command compresses the package without copying it to the image store. You can copy a compressed package to one or more Service Fabric clusters, as needed, using [Copy-ServiceFabricApplicationPackage](#) without the `skipCopy` flag. The package now includes zipped files for the `code`, `config`, and `data` packages. The application manifest and the service manifests are not zipped, because they are needed for many internal operations (like package sharing, application type name and version extraction for certain validations). Zipping the manifests would make these operations inefficient.

```

PS D:\temp> tree /f .\MyApplicationType

D:\TEMP\MYAPPLICATIONTYPE
|   ApplicationManifest.xml
|
└── MyServiceManifest
    |   ServiceManifest.xml
    |
    └── MyCode
        |   MyServiceHost.exe
        |   MySetup.bat
    |
    └── MyConfig
        |   Settings.xml
    |
    └── MyData
        |   init.dat

PS D:\temp> Copy-ServiceFabricApplicationPackage -ApplicationPackagePath .\MyApplicationType -CompressPackage -SkipCopy

PS D:\temp> tree /f .\MyApplicationType

D:\TEMP\MYAPPLICATIONTYPE
|   ApplicationManifest.xml
|
└── MyServiceManifest
    |   ServiceManifest.xml
    |
    └── MyCode.zip
    └── MyConfig.zip
    └── MyData.zip

```

Alternatively, you can compress and copy the package with [Copy-ServiceFabricApplicationPackage](#) in one step. If the package is large, provide a high enough timeout to allow time for both the package compression and the upload to the cluster.

```

PS D:\temp> Copy-ServiceFabricApplicationPackage -ApplicationPackagePath .\MyApplicationType -
ApplicationPackagePathInImageStore MyApplicationType -ImageStoreConnectionString fabric:ImageStore -
CompressPackage -TimeoutSec 5400

```

Internally, Service Fabric computes checksums for the application packages for validation. When using compression, the checksums are computed on the zipped versions of each package. If you copied an uncompressed version of your application package, and you want to use compression for the same package, you must change the versions of the `code`, `config`, and `data` packages to avoid checksum mismatch. If the packages are unchanged, instead of changing the version, you can use [diff provisioning](#). With this option, do not include the unchanged package instead reference it from the service manifest.

Similarly, if you uploaded a compressed version of the package and you want to use an uncompressed package, you must update the versions to avoid the checksum mismatch.

The package is now packaged correctly, validated, and compressed (if needed), so it is ready for [deployment](#) to one or more Service Fabric clusters.

### **Compress packages when deploying using Visual Studio**

You can instruct Visual Studio to compress packages on deployment, by adding the `CopyPackageParameters` element to your publish profile, and set the `CompressPackage` attribute to `true`.

```
<PublishProfile xmlns="http://schemas.microsoft.com/2015/05/fabrictools">
  <ClusterConnectionParameters ConnectionEndpoint="mycluster.westus.cloudapp.azure.com" />
  <ApplicationParameterFile Path="..\ApplicationParameters\Cloud.xml" />
  <CopyPackageParameters CompressPackage="true"/>
</PublishProfile>
```

## Next steps

[Deploy and remove applications](#) describes how to use PowerShell to manage application instances

[Managing application parameters for multiple environments](#) describes how to configure parameters and environment variables for different application instances.

[Configure security policies for your application](#) describes how to run services under security policies to restrict access.

# Get started with deploying and upgrading applications on your local cluster

7/14/2017 • 7 min to read • [Edit Online](#)

The Azure Service Fabric SDK includes a full local development environment that you can use to quickly get started with deploying and managing applications on a local cluster. In this article, you create a local cluster, deploy an existing application to it, and then upgrade that application to a new version, all from Windows PowerShell.

## NOTE

This article assumes that you already [set up your development environment](#).

## Create a local cluster

A Service Fabric cluster represents a set of hardware resources that you can deploy applications to. Typically, a cluster is made up of anywhere from five to many thousands of machines. However, the Service Fabric SDK includes a cluster configuration that can run on a single machine.

It is important to understand that the Service Fabric local cluster is not an emulator or simulator. It runs the same platform code that is found on multi-machine clusters. The only difference is that it runs the platform processes that are normally spread across five machines on one machine.

The SDK provides two ways to set up a local cluster: a Windows PowerShell script and the Local Cluster Manager system tray app. In this tutorial, we use the PowerShell script.

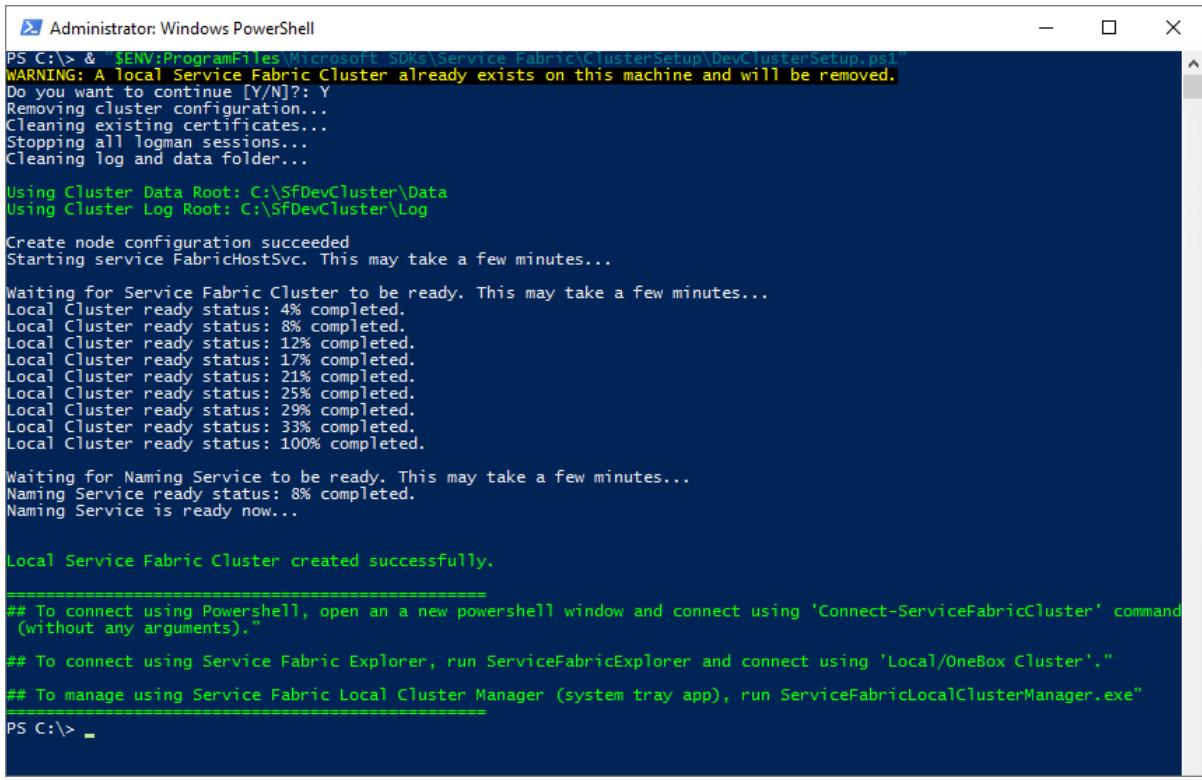
## NOTE

If you have already created a local cluster by deploying an application from Visual Studio, you can skip this section.

1. Launch a new PowerShell window as an administrator.
2. Run the cluster setup script from the SDK folder:

```
& "$ENV:ProgramFiles\Microsoft SDKs\Service Fabric\ClusterSetup\DevClusterSetup.ps1"
```

Cluster setup takes a few moments. After setup is finished, you should see output similar to:



```
Administrator: Windows PowerShell
PS C:\> & "$ENV:ProgramFiles\Microsoft SDKs\Service Fabric\ClusterSetup\DevClusterSetup.ps1"
WARNING: A Local Service Fabric Cluster already exists on this machine and will be removed.
Do you want to continue [Y/N]?: Y
Removing cluster configuration...
Cleaning existing certificates...
Stopping all logman sessions...
Cleaning log and data folder...

Using Cluster Data Root: C:\SfDevCluster\Data
Using Cluster Log Root: C:\SfDevCluster\Log

Create node configuration succeeded
Starting service FabricHostSvc. This may take a few minutes...

Waiting for Service Fabric Cluster to be ready. This may take a few minutes...
Local Cluster ready status: 4% completed.
Local Cluster ready status: 8% completed.
Local Cluster ready status: 12% completed.
Local Cluster ready status: 17% completed.
Local Cluster ready status: 21% completed.
Local Cluster ready status: 25% completed.
Local Cluster ready status: 29% completed.
Local Cluster ready status: 33% completed.
Local Cluster ready status: 100% completed.

Waiting for Naming Service to be ready. This may take a few minutes...
Naming Service ready status: 8% completed.
Naming Service is ready now...

Local Service Fabric Cluster created successfully.

=====
## To connect using Powershell, open an a new powershell window and connect using 'Connect-ServiceFabricCluster' command
## (without any arguments)."
## To connect using Service Fabric Explorer, run ServiceFabricExplorer and connect using 'Local/OneBox Cluster'.
## To manage using Service Fabric Local Cluster Manager (system tray app), run ServiceFabricLocalClusterManager.exe"
=====

PS C:\> -
```

You are now ready to try deploying an application to your cluster.

## Deploy an application

The Service Fabric SDK includes a rich set of frameworks and developer tooling for creating applications. If you are interested in learning how to create applications in Visual Studio, see [Create your first Service Fabric application in Visual Studio](#).

In this tutorial, you use an existing sample application (called WordCount) so that you can focus on the management aspects of the platform: deployment, monitoring, and upgrade.

1. Launch a new PowerShell window as an administrator.
2. Import the Service Fabric SDK PowerShell module.

```
Import-Module "$ENV:ProgramFiles\Microsoft SDKs\Service
Fabric\Tools\PSModule\ServiceFabricSDK\ServiceFabricSDK.psm1"
```

3. Create a directory to store the application that you download and deploy, such as C:\ServiceFabric.

```
mkdir c:\ServiceFabric\
cd c:\ServiceFabric\
```

4. [Download the WordCount application](#) to the location you created. Note: the Microsoft Edge browser saves the file with a .zip extension. Change the file extension to .sfpkg.
5. Connect to the local cluster:

```
Connect-ServiceFabricCluster localhost:19000
```

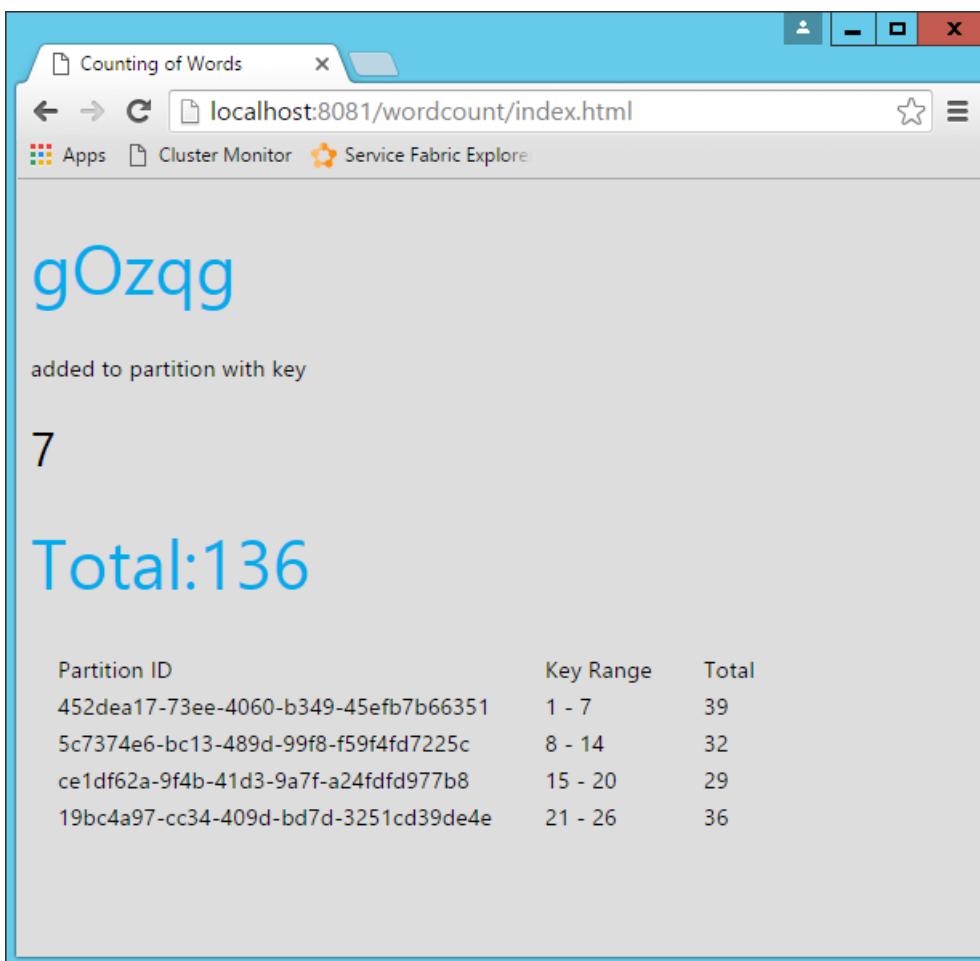
6. Create a new application using the SDK's deployment command with a name and a path to the application package.

```
Publish-NewServiceFabricApplication -ApplicationPackagePath c:\ServiceFabric\WordCountV1.sfpkg -  
ApplicationName "fabric:/WordCount"
```

If all goes well, you should see the following output:

```
Copying application to image store...  
Copy application package succeeded  
Registering application type...  
Register application type succeeded  
Removing application package from image store...  
Remove application package succeeded  
Creating application...  
Create application succeeded.
```

7. To see the application in action, launch the browser and navigate to <http://localhost:8081/wordcount/index.html>. You should see:



The WordCount application is simple. It includes client-side JavaScript code to generate random five-character "words", which are then relayed to the application via ASP.NET Web API. A stateful service tracks the number of words counted. They are partitioned based on the first character of the word. You can find the source code for the WordCount app in the [classic getting started samples](#).

The application that we deployed contains four partitions. So words beginning with A through G are stored in the first partition, words beginning with H through N are stored in the second partition, and so on.

## View application details and status

Now that we have deployed the application, let's look at some of the app details in PowerShell.

1. Query all deployed applications on the cluster:

```
Get-ServiceFabricApplication
```

Assuming that you have only deployed the WordCount app, you see something similar to:

```
PS C:\Service Fabric> Get-ServiceFabricApplication

ApplicationName      : fabric:/WordCount
ApplicationTypeName  : WordCount
ApplicationTypeVersion: 1.0.0
ApplicationStatus    : Ready
HealthState          : Ok
ApplicationParameters: { "WordCountWebService_InstanceCount" = "1" }
```

2. Go to the next level by querying the set of services that are included in the WordCount application.

```
Get-ServiceFabricService -ApplicationName 'fabric:/WordCount'
```

```
PS C:\Service Fabric> Get-ServiceFabricService -ApplicationName 'fabric:/WordCount'

ServiceName          : fabric:/WordCount/WordCountService
ServiceKind          : Stateful
ServiceTypeName      : WordCountServiceType
IsServiceGroup       : False
ServiceManifestVersion: 1.0.0
HasPersistedState   : True
ServiceStatus        : Active
HealthState          : Ok

ServiceKind          : Stateless
ServiceName          : fabric:/WordCount/WordCountWebService
ServiceTypeName      : WordCountWebServiceType
ServiceManifestVersion: 1.0.0
HealthState          : Ok
ServiceStatus        : Active
IsServiceGroup       : False
```

The application is made up of two services, the web front end, and the stateful service that manages the words.

3. Finally, look at the list of partitions for WordCountService:

```
Get-ServiceFabricPartition 'fabric:/WordCount/WordCountService'
```

```
PS C:\Service Fabric> Get-ServiceFabricPartition 'fabric:/WordCount/WordCountService'

PartitionId          : 59393164-71f9-4e45-9f30-9bdce46c4349
PartitionKind        : Int64Range
PartitionLowKey      : 1
PartitionHighKey    : 7
PartitionStatus      : Ready
LastQuorumLossDuration : 00:00:00
MinReplicaSetSize   : 2
TargetReplicaSetSize : 3
HealthState          : Ok
DataLossNumber       : 130888309725707433
ConfigurationNumber  : 8589934592

PartitionId          : d645a771-67bd-4f4f-9236-c408cd107f8
PartitionKind        : Int64Range
PartitionLowKey      : 8
PartitionHighKey    : 14
PartitionStatus      : Ready
LastQuorumLossDuration : 00:00:00
MinReplicaSetSize   : 2
TargetReplicaSetSize : 3
HealthState          : Ok
DataLossNumber       : 130888309725707433
ConfigurationNumber  : 8589934592

PartitionId          : 08ab1875-b3cd-4def-82f4-7c92807bfb50
PartitionKind        : Int64Range
PartitionLowKey      : 21
PartitionHighKey    : 26
PartitionStatus      : Ready
LastQuorumLossDuration : 00:00:00
MinReplicaSetSize   : 2
TargetReplicaSetSize : 3
HealthState          : Ok
DataLossNumber       : 130888309725707433
ConfigurationNumber  : 8589934592

PartitionId          : 474326d9-3e36-4be3-aa73-bc1e3f6fcfa41
PartitionKind        : Int64Range
PartitionLowKey      : 15
PartitionHighKey    : 20
PartitionStatus      : Ready
LastQuorumLossDuration : 00:00:00
MinReplicaSetSize   : 2
TargetReplicaSetSize : 3
HealthState          : Ok
DataLossNumber       : 130888309725707433
ConfigurationNumber  : 8589934592
```

The set of commands that you used, like all Service Fabric PowerShell commands, are available for any cluster that you might connect to, local or remote.

For a more visual way to interact with the cluster, you can use the web-based Service Fabric Explorer tool by navigating to <http://localhost:19080/Explorer> in the browser.

The screenshot shows the Microsoft Azure Service Fabric Explorer interface. The top navigation bar includes links for 'Service Fabric Explorer', 'localhost:19080/Explorer/index.html#/apptype/WordCount/app/WordCount', 'Cluster Monitor', and 'Service Fabric Explorer'. The main content area has tabs for 'Microsoft Azure' and 'Service Fabric Explorer'. On the left, there's a navigation tree with 'Cluster' expanded, showing 'Applications' (with 'WordCount' selected), 'System', and 'Nodes'. The 'WordCount' node is expanded, showing its services: 'WordCountService' (version 2.0.0, Stateful, OK, Active) and 'WordCountWebService' (version 1.0.0, Stateless, OK, Active). The 'ESSENTIALS' tab is selected, displaying the application's name ('fabric:/WordCount'), health state ('OK'), version ('2.0.0'), and status ('Ready'). The 'DETAILS' tab shows 'UNHEALTHY EVALUATIONS' with no items listed. The 'MANIFEST' tab is also present. The bottom section lists 'SERVICES' with columns for Name, Service Type, Version, Service Kind, Health State, and Status.

Name	Service Type	Version	Service Kind	Health State	Status
fabric:/WordCount/WordCountService	WordCountServiceType	2.0.0	Stateful	OK	Active
fabric:/WordCount/WordCountWebService	WordCountWebServiceType	1.0.0	Stateless	OK	Active

#### NOTE

To learn more about Service Fabric Explorer, see [Visualizing your cluster with Service Fabric Explorer](#).

## Upgrade an application

Service Fabric provides no-downtime upgrades by monitoring the health of the application as it rolls out across the cluster. Perform an upgrade of the WordCount application.

The new version of the application now counts only words that begin with a vowel. As the upgrade rolls out, we see two changes in the application's behavior. First, the rate at which the count grows should slow, since fewer words are being counted. Second, since the first partition has two vowels (A and E) and all other partitions contain only one each, its count should eventually start to outpace the others.

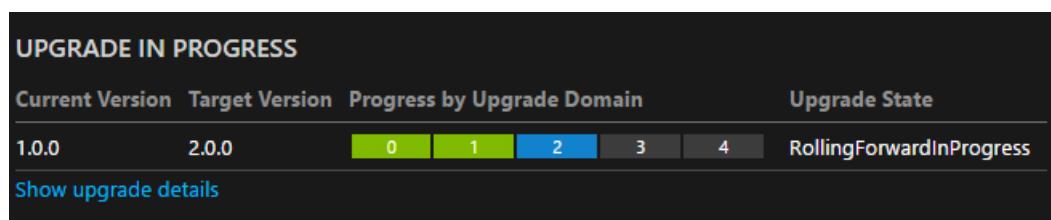
1. [Download the WordCount version 2 package](#) to the same location where you downloaded the version 1 package.
2. Return to your PowerShell window and use the SDK's upgrade command to register the new version in the cluster. Then begin upgrading the fabric:/WordCount application.

```
Publish-UpgradeServiceFabricApplication -ApplicationPackagePath C:\ServiceFabric\WordCountV2.sfpkg -  
ApplicationName "fabric:/WordCount" -UpgradeParameters @{"FailureAction"="Rollback";  
"UpgradeReplicaSetCheckTimeout"=1; "Monitored"=$true; "Force"=$true}
```

You should see the following output in PowerShell as the upgrade begins.

```
Copying application package...  
Copy application package succeeded  
Registering application type...  
Register application type succeeded  
Start upgrading application...  
  
ApplicationName : fabric:/WordCount  
ApplicationParameters : { "WordCountWebService_InstanceCount" = "1" }  
TargetApplicationTypeVersion : 2.0.0  
UpgradeKind : Rolling  
ForceRestart : False  
UpgradeMode : UnmonitoredAuto  
UpgradeReplicaSetCheckTimeout : 00:00:01  
  
Waiting for upgrade...  
Waiting for upgrade...  
Waiting for upgrade...  
Upgrade completed
```

3. While the upgrade is proceeding, you may find it easier to monitor its status from Service Fabric Explorer. Launch a browser window and navigate to <http://localhost:19080/Explorer>. Expand **Applications** in the tree on the left, then choose **WordCount**, and finally **fabric:/WordCount**. In the essentials tab, you see the status of the upgrade as it proceeds through the cluster's upgrade domains.



As the upgrade proceeds through each domain, health checks are performed to ensure that the application is behaving properly.

4. If you rerun the earlier query for the set of services in the fabric:/WordCount application, notice that the WordCountService version changed but the WordCountWebService version did not:

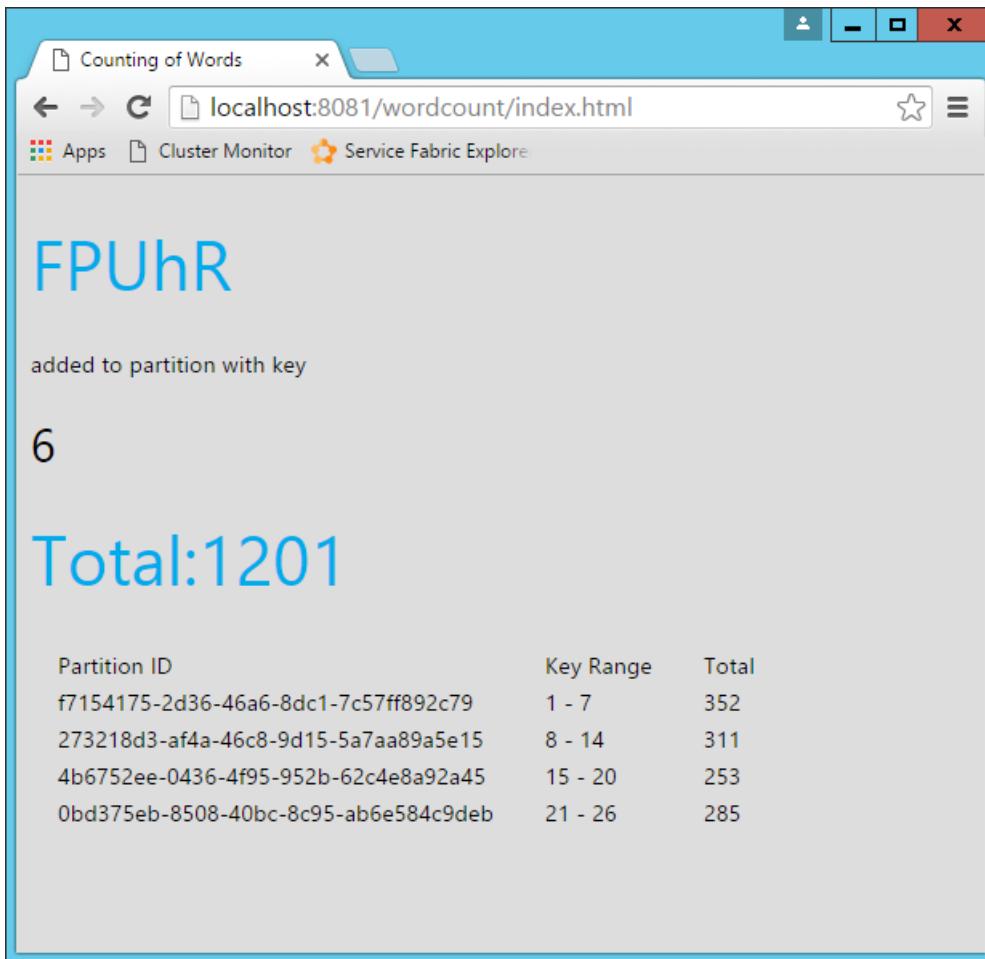
```
Get-ServiceFabricService -ApplicationName 'fabric:/WordCount'
```

```
ServiceName      : fabric:/WordCount/WordCountService
ServiceKind      : Stateful
ServiceTypeName  : WordCountServiceType
IsServiceGroup   : False
ServiceManifestVersion : 2.0.0
HasPersistedState : True
ServiceStatus    : Active
HealthState     : Ok

ServiceKind      : Stateless
ServiceName      : fabric:/WordCount/WordCountWebService
ServiceTypeName  : WordCountWebServiceType
ServiceManifestVersion : 1.0.0
HealthState     : Ok
ServiceStatus    : Active
IsServiceGroup   : False
```

This example highlights how Service Fabric manages application upgrades. It touches only the set of services (or code/configuration packages within those services) that have changed, which makes the process of upgrading faster and more reliable.

- Finally, return to the browser to observe the behavior of the new application version. As expected, the count progresses more slowly, and the first partition ends up with slightly more of the volume.



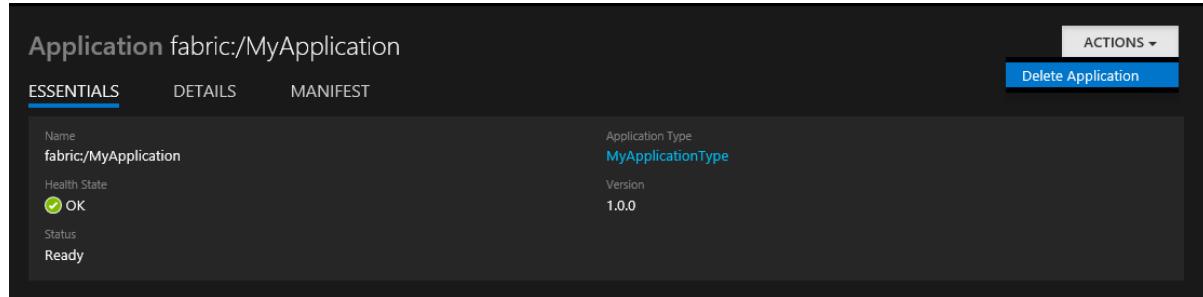
## Cleaning up

Before wrapping up, it's important to remember that the local cluster is real. Applications continue to run in the background until you remove them. Depending on the nature of your apps, a running app can take up significant resources on your machine. You have several options to manage applications and the cluster:

- To remove an individual application and all its data, run the following command:

```
Unpublish-ServiceFabricApplication -ApplicationName "fabric:/WordCount"
```

Or, delete the application from the Service Fabric Explorer **ACTIONS** menu or the context menu in the left-hand application list view.



The screenshot shows the Service Fabric Explorer interface. In the center, there's a card for the application 'fabric:/MyApplication'. On the left, under 'ESSENTIALS', it shows 'Name: fabric:/MyApplication', 'Health State: OK', and 'Status: Ready'. On the right, it shows 'Application Type: MyApplicationType', 'Version: 1.0.0'. At the top right of the card is a blue 'ACTIONS' button with a dropdown arrow, and below it is a blue 'Delete Application' button.

2. After deleting the application from the cluster, unregister versions 1.0.0 and 2.0.0 of the WordCount application type. Deletion removes the application packages, including the code and configuration, from the cluster's image store.

```
Remove-ServiceFabricApplicationType -ApplicationTypeName WordCount -ApplicationTypeVersion 2.0.0  
Remove-ServiceFabricApplicationType -ApplicationTypeName WordCount -ApplicationTypeVersion 1.0.0
```

Or, in Service Fabric Explorer, choose **Unprovision Type** for the application.

3. To shut down the cluster but keep the application data and traces, click **Stop Local Cluster** in the system tray app.
4. To delete the cluster entirely, click **Remove Local Cluster** in the system tray app. This option will result in another slow deployment the next time you press F5 in Visual Studio. Remove the local cluster only if you don't intend to use it for some time or if you need to reclaim resources.

## One-node and five-node cluster mode

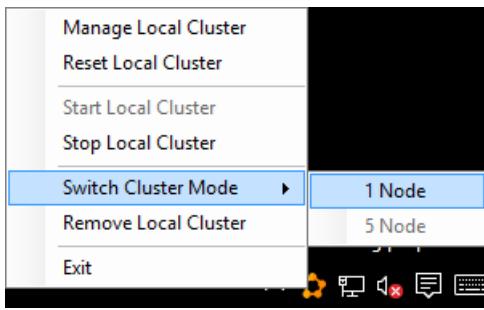
When developing applications, you often find yourself doing quick iterations of writing code, debugging, changing code, and debugging. To help optimize this process, the local cluster can run in two modes: one-node or five-node. Both cluster modes have their benefits. Five-node cluster mode enables you to work with a real cluster. You can test failover scenarios, work with more instances and replicas of your services. One-node cluster mode is optimized to do quick deployment and registration of services, to help you quickly validate code using the Service Fabric runtime.

Neither one-node cluster or five-node cluster modes are an emulator or simulator. The local development cluster runs the same platform code that is found on multi-machine clusters.

### WARNING

When you change the cluster mode, the current cluster is removed from your system and a new cluster is created. The data stored in the cluster is deleted when you change cluster mode.

To change the mode to one-node cluster, select **Switch Cluster Mode** in the Service Fabric Local Cluster Manager.



Or, change the cluster mode using PowerShell:

1. Launch a new PowerShell window as an administrator.
2. Run the cluster setup script from the SDK folder:

```
& "$ENV:ProgramFiles\Microsoft SDKs\Service Fabric\ClusterSetup\DevClusterSetup.ps1" -CreateOneNodeCluster
```

Cluster setup takes a few moments. After setup is finished, you should see output similar to:

```
PS C:\> & "$ENV:ProgramFiles\Microsoft SDKs\Service Fabric\ClusterSetup\DevClusterSetup.ps1" -CreateOneNodeCluster
WARNING: A local Service Fabric Cluster already exists on this machine and will be removed.
Do you want to continue [Y/N]?: Y
Removing cluster configuration...
Cleaning existing certificates...
Stopping all logman sessions...
Cleaning log and data folder...
Using Cluster Data Root: C:\SFDevCluster\Data
Using Cluster Log Root: C:\SFDevCluster\Log

Create node configuration succeeded
Starting service FabricHostSvc. This may take a few minutes...

Waiting for Service Fabric Cluster to be ready. This may take a few minutes...
Local Cluster ready status: 4% completed.
Local Cluster ready status: 100% completed.

Waiting for Naming Service to be ready. This may take a few minutes...
Naming Service is ready now...

Local Service Fabric Cluster created successfully.

=====
## To connect using Powershell, open an a new powershell window and connect using 'Connect-ServiceFabricCluster' command
## (without any arguments)."
## To connect using Service Fabric Explorer, run ServiceFabricExplorer and connect using 'Local/OneBox Cluster'.
## To manage using Service Fabric Local Cluster Manager (system tray app), run ServiceFabricLocalClusterManager.exe"
PS C:\>
```

## Next steps

- Now that you have deployed and upgraded some pre-built applications, you can [try building your own in Visual Studio](#).
- All the actions performed on the local cluster in this article can be performed on an [Azure cluster](#) as well.
- The upgrade that we performed in this article was basic. See the [upgrade documentation](#) to learn more about the power and flexibility of Service Fabric upgrades.

# Manage applications and services as Azure Resource Manager resources

10/6/2017 • 4 min to read • [Edit Online](#)

You can deploy applications and services onto your Service Fabric cluster via Azure Resource Manager. This means that instead of deploying and managing applications via PowerShell or CLI after having to wait for the cluster to be ready, you can now express applications and services in JSON and deploy them in the same Resource Manager template as your cluster. The process of application registration, provisioning, and deployment all happens in one step.

This is the recommended way for you to deploy any setup, governance, or cluster management applications that you require in your cluster. This includes the [Patch Orchestration Application](#), Watchdogs, or any applications that need to be running in your cluster before other applications or services are deployed.

When applicable, manage your applications as Resource Manager resources to improve:

- Audit trail: Resource Manager audits every operation and keeps a detailed *Activity Log* that can help you trace any changes made to these applications and your cluster.
- Role-based access control (RBAC): Managing access to clusters as well as applications deployed on the cluster can be done via the same Resource Manager template.
- Azure Resource Manager (via Azure portal) becomes a one-stop-shop for managing your cluster and critical application deployments.

The following snippet shows the different kinds of resources that can be managed via a template:

```
{  
  "apiVersion": "2017-07-01-preview",  
  "type": "Microsoft.ServiceFabric/clusters/applicationTypes",  
  "name": "[concat(parameters('clusterName'), '/', parameters('applicationTypeName'))]",  
  "location": "[variables('clusterLocation')]",  
},  
{  
  "apiVersion": "2017-07-01-preview",  
  "type": "Microsoft.ServiceFabric/clusters/applicationTypes/versions",  
  "name": "[concat(parameters('clusterName'), '/', parameters('applicationTypeName'), '/',  
parameters('applicationTypeVersion'))]",  
  "location": "[variables('clusterLocation')]",  
},  
{  
  "apiVersion": "2017-07-01-preview",  
  "type": "Microsoft.ServiceFabric/clusters/applications",  
  "name": "[concat(parameters('clusterName'), '/', parameters('applicationName'))]",  
  "location": "[variables('clusterLocation')]",  
},  
{  
  "apiVersion": "2017-07-01-preview",  
  "type": "Microsoft.ServiceFabric/clusters/applications/services",  
  "name": "[concat(parameters('clusterName'), '/', parameters('applicationName'), '/',  
parameters('serviceName'))]",  
  "location": "[variables('clusterLocation')]"  
}
```

## Add a new application to your Resource Manager template

1. Prepare your cluster's Resource Manager template for deployment. See [Create a Service Fabric cluster by using Azure Resource Manager](#) for more information on this.
2. Think about some of the applications you plan on deploying in the cluster. Are there any that will always be running that other applications may take dependencies on? Do you plan on deploying any cluster governance or setup applications? These sorts of applications are best managed via a Resource Manager template, as discussed above.
3. Once you have figured out what applications you want to be deployed this way, the applications have to be packaged, zipped, and put on a file share. The share needs to be accessible through a REST endpoint for Azure Resource Manager to consume during deployment.
4. In your Resource Manager template, below your cluster declaration, describe each application's properties. These properties include replica or instance count and any dependency chains between resources (other applications or services). For a list of comprehensive properties, see the [REST API Swagger Spec](#). Note that this does not replace the Application or Service manifests, but rather describes some of what is in them as part of the cluster's Resource Manager template. Here is a sample template that includes deploying a stateless service *Service1* and a stateful service *Service2* as part of *Application1*:

```
{
  "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "clusterName": {
      "type": "string",
      "defaultValue": "Cluster",
      "metadata": {
        "description": "Name of your cluster - Between 3 and 23 characters. Letters and numbers only"
      }
    },
    "applicationTypeName": {
      "type": "string",
      "defaultValue": "ApplicationType",
      "metadata": {
        "description": "The application type name"
      }
    },
    "applicationTypeVersion": {
      "type": "string",
      "defaultValue": "1",
      "metadata": {
        "description": "The application type version"
      }
    },
    "appPackageUrl": {
      "type": "string",
      "metadata": {
        "description": "The URL to the application package zip file"
      }
    },
    "applicationName": {
      "type": "string",
      "defaultValue": "Application1",
      "metadata": {
        "description": "The application name"
      }
    },
    "serviceName": {
      "type": "string",
      "defaultValue": "Service1",
      "metadata": {
        "description": "The service name"
      }
    },
    "serviceTypeName": {
      "type": "string",
      "metadata": {
        "description": "The service type name"
      }
    }
}
```

```

        },
        "defaultValue": "Service1Type",
        "metadata": {
            "description": "The service type name"
        }
    },
    "serviceName2": {
        "type": "string",
        "defaultValue": "Service2",
        "metadata": {
            "description": "The service name"
        }
    },
    "serviceTypeName2": {
        "type": "string",
        "defaultValue": "Service2Type",
        "metadata": {
            "description": "The service type name"
        }
    }
},
"variables": {
    "clusterLocation": "[resourcegroup().location]"
},
"resources": [
{
    "apiVersion": "2017-07-01-preview",
    "type": "Microsoft.ServiceFabric/clusters/applications",
    "name": "[concat(parameters('clusterName'), '/', parameters('applicationName'))]",
    "location": "[variables('clusterLocation')]",
    "dependsOn": [],
    "properties": {
        "provisioningState": "Default",
        "typeName": "[parameters('applicationTypeName')]",
        "typeVersion": "[parameters('applicationTypeVersion')]",
        "parameters": {},
        "upgradePolicy": {
            "upgradeReplicaSetCheckTimeout": "01:00:00.0",
            "forceRestart": "false",
            "rollingUpgradeMonitoringPolicy": {
                "healthCheckWaitDuration": "00:02:00.0",
                "healthCheckStableDuration": "00:05:00.0",
                "healthCheckRetryTimeout": "00:10:00.0",
                "upgradeTimeout": "01:00:00.0",
                "upgradeDomainTimeout": "00:20:00.0"
            },
            "applicationHealthPolicy": {
                "considerWarningAsError": "false",
                "maxPercentUnhealthyDeployedApplications": "50",
                "defaultServiceTypeHealthPolicy": {
                    "maxPercentUnhealthyServices": "50",
                    "maxPercentUnhealthyPartitionsPerService": "50",
                    "maxPercentUnhealthyReplicasPerPartition": "50"
                }
            }
        }
    }
},
{
    "apiVersion": "2017-07-01-preview",
    "type": "Microsoft.ServiceFabric/clusters/applications/services",
    "name": "[concat(parameters('clusterName'), '/', parameters('applicationName'), '/', parameters('serviceName'))]",
    "location": "[variables('clusterLocation')]",
    "dependsOn": [
        "[concat('Microsoft.ServiceFabric/clusters/', parameters('clusterName'), '/applications/', parameters('applicationName'))]"
    ],
    "properties": {
        "provisioningState": "Default"
    }
}
]

```

```

    "provisioningState": "Default",
    "serviceKind": "Stateless",
    "serviceTypeName": "[parameters('serviceName')]",
    "instanceCount": "-1",
    "partitionDescription": {
        "partitionScheme": "Singleton"
    },
    "correlationScheme": [],
    "serviceLoadMetrics": [],
    "servicePlacementPolicies": []
}
},
{
    "apiVersion": "2017-07-01-preview",
    "type": "Microsoft.ServiceFabric/clusters/applications/services",
    "name": "[concat(parameters('clusterName'), '/', parameters('applicationName'), '/', parameters('serviceName2'))]",
    "location": "[variables('clusterLocation')]",
    "dependsOn": [
        "[concat('Microsoft.ServiceFabric/clusters/', parameters('clusterName'), '/applications/', parameters('applicationName'))]"
    ],
    "properties": {
        "provisioningState": "Default",
        "serviceKind": "Stateful",
        "serviceTypeName": "[parameters('serviceName2')]",
        "targetReplicaSetSize": "3",
        "minReplicaSetSize": "2",
        "replicaRestartWaitDuration": "00:01:00.0",
        "quorumLossWaitDuration": "00:02:00.0",
        "standByReplicaKeepDuration": "00:00:30.0",
        "partitionDescription": {
            "partitionScheme": "UniformInt64Range",
            "count": "5",
            "lowKey": "1",
            "highKey": "26"
        },
        "hasPersistedState": "true",
        "correlationScheme": [],
        "serviceLoadMetrics": [],
        "servicePlacementPolicies": [],
        "defaultMoveCost": "Low"
    }
}
]
}

```

#### NOTE

The `apiVersion` must be set to `"2017-07-01-preview"`. This template can also be deployed independently of the cluster, as long as the cluster has already been deployed.

## 5. Deploy!

## Manage an existing application via Resource Manager

If your cluster is already up and some applications that you would like to manage as Resource Manager resources are already deployed on it, instead of removing the applications and redeploying them, you can use a PUT call using the same APIs to have the applications get acknowledged as Resource Manager resources.

## Next steps

- Use the [Service Fabric CLI](#) or [PowerShell](#) to deploy other applications to your cluster.

- Upgrade your Service Fabric cluster

# Deploy and remove applications using PowerShell

10/5/2017 • 10 min to read • [Edit Online](#)

Once an [application type has been packaged](#), it's ready for deployment into an Azure Service Fabric cluster. Deployment involves the following three steps:

1. Upload the application package to the image store
2. Register the application type
3. Create the application instance

After an application is deployed and an instance is running in the cluster, you can delete the application instance and its application type. To completely remove an application from the cluster involves the following steps:

1. Remove (or delete) the running application instance
2. Unregister the application type if you no longer need it
3. Remove the application package from the image store

If you use [Visual Studio for deploying and debugging applications](#) on your local development cluster, all the preceding steps are handled automatically through a PowerShell script. This script is found in the *Scripts* folder of the application project. This article provides background on what that script is doing so that you can perform the same operations outside of Visual Studio.

## Connect to the cluster

Before you run any PowerShell commands in this article, always start by using [Connect-ServiceFabricCluster](#) to connect to the Service Fabric cluster. To connect to the local development cluster, run the following:

```
PS C:\>Connect-ServiceFabricCluster
```

For examples of connecting to a remote cluster or cluster secured using Azure Active Directory, X509 certificates, or Windows Active Directory see [Connect to a secure cluster](#).

## Upload the application package

Uploading the application package puts it in a location that's accessible by internal Service Fabric components. If you want to verify the application package locally, use the [Test-ServiceFabricApplicationPackage](#) cmdlet.

The [Copy-ServiceFabricApplicationPackage](#) command uploads the application package to the cluster image store.

Suppose you build and package an application named *MyApplication* in Visual Studio 2015. By default, the application type name listed in the ApplicationManifest.xml is "MyApplicationType". The application package, which contains the necessary application manifest, service manifests, and code/config/data packages, is located in *C:\Users<username>\Documents\Visual Studio 2015\Projects\MyApplication\MyApplication\pkg\Debug*.

The following command lists the contents of the application package:

```

PS C:\> $path = 'C:\Users\<user>\Documents\Visual Studio
2015\Projects\MyApplication\MyApplication\pkg\Debug'
PS C:\> tree /f $path
Folder PATH listing for volume OSDisk
Volume serial number is 0459-2393
C:\USERS\USER\DOCUMENTS\VISUAL STUDIO 2015\PROJECTS\MYAPPLICATION\MYAPPLICATION\PKG\DEBUG
|   ApplicationManifest.xml
|
└── Stateless1Pkg
    |   ServiceManifest.xml
    |
    └── Code
        Microsoft.ServiceFabric.Data.dll
        Microsoft.ServiceFabric.Data.Interfaces.dll
        Microsoft.ServiceFabric.Internal.dll
        Microsoft.ServiceFabric.Internal.Strings.dll
        Microsoft.ServiceFabric.Services.dll
        ServiceFabricServiceModel.dll
        Stateless1.exe
        Stateless1.exe.config
        Stateless1.pdb
        System.Fabric.dll
        System.Fabric.Strings.dll
    |
    └── Config
        Settings.xml

```

If the application package is large and/or has many files, you can [compress it](#). The compression reduces the size and the number of files. The side effect is that registering and un-registering the application type are faster. Upload time may be slower currently, especially if you include the time to compress the package.

To compress a package, use the same [Copy-ServiceFabricApplicationPackage](#) command. Compression can be done separate from upload, by using the `SkipCopy` flag, or together with the upload operation. Applying compression on a compressed package is no-op. To uncompress a compressed package, use the same [Copy-ServiceFabricApplicationPackage](#) command with the `UncompressPackage` switch.

The following cmdlet compresses the package without copying it to the image store. The package now includes zipped files for the `Code` and `Config` packages. The application and the service manifests are not zipped, because they are needed for many internal operations (like package sharing, application type name and version extraction for certain validations). Zipping the manifests would make these operations inefficient.

```

PS C:\> Copy-ServiceFabricApplicationPackage -ApplicationPackagePath $path -CompressPackage -SkipCopy
PS C:\> tree /f $path
Folder PATH listing for volume OSDisk
Volume serial number is 0459-2393
C:\USERS\USER\DOCUMENTS\VISUAL STUDIO 2015\PROJECTS\MYAPPLICATION\MYAPPLICATION\PKG\DEBUG
|   ApplicationManifest.xml
|
└── Stateless1Pkg
    Code.zip
    Config.zip
    ServiceManifest.xml

```

For large application packages, the compression takes time. For best results, use a fast SSD drive. The compression times and the size of the compressed package also differ based on the package content. For example, here is compression statistics for some packages, which show the initial and the compressed package size, with the compression time.

INITIAL SIZE (MB)	FILE COUNT	COMPRESSION TIME	COMPRESSED PACKAGE SIZE (MB)
100	100	00:00:03.3547592	60
512	100	00:00:16.3850303	307
1024	500	00:00:32.5907950	615
2048	1000	00:01:04.3775554	1231
5012	100	00:02:45.2951288	3074

Once a package is compressed, it can be uploaded to one or multiple Service Fabric clusters as needed. The deployment mechanism is same for compressed and uncompressed packages. If the package is compressed, it is stored as such in the cluster image store and it's uncompressed on the node before the application is run.

The following example uploads the package to the image store, into a folder named "MyApplicationV1":

```
PS C:\> Copy-ServiceFabricApplicationPackage -ApplicationPackagePath $path -ApplicationPackagePathInImageStore MyApplicationV1 -TimeoutSec 1800
```

If you do not specify the `-ApplicationPackagePathInImageStore` parameter, the application package is copied into the "Debug" folder in the image store.

#### NOTE

**Copy-ServiceFabricApplicationPackage** will automatically detect the appropriate image store connection string if the PowerShell session is connected to a Service Fabric cluster. For Service Fabric versions older than 5.6, the `-ImageStoreConnectionString` argument must be explicitly provided.

```
PS C:\> Copy-ServiceFabricApplicationPackage -ApplicationPackagePath $path -ApplicationPackagePathInImageStore MyApplicationV1 -ImageStoreConnectionString (Get-ImageStoreConnectionStringFromClusterManifest(Get-ServiceFabricClusterManifest)) -TimeoutSec 1800
```

The `Get-ImageStoreConnectionStringFromClusterManifest` cmdlet, which is part of the Service Fabric SDK PowerShell module, is used to get the image store connection string. To import the SDK module, run:

```
Import-Module "$ENV:ProgramFiles\Microsoft SDKs\ServiceFabric\Tools\PSModule\ServiceFabricSDK\ServiceFabricSDK.psm1"
```

See [Understand the image store connection string](#) for supplementary information about the image store and image store connection string.

The time it takes to upload a package differs depending on multiple factors. Some of these factors are the number of files in the package, the package size, and the file sizes. The network speed between the source machine and the Service Fabric cluster also impacts the upload time. The default timeout for `Copy-ServiceFabricApplicationPackage` is 30 minutes. Depending on the described factors, you may have to increase the timeout. If you are compressing the package in the copy call, you need to also consider the compression time.

## Register the application package

The application type and version declared in the application manifest become available for use when the

application package is registered. The system reads the package uploaded in the previous step, verifies the package, processes the package contents, and copies the processed package to an internal system location.

Run the [Register-ServiceFabricApplicationType](#) cmdlet to register the application type in the cluster and make it available for deployment:

```
PS C:\> Register-ServiceFabricApplicationType MyApplicationV1
Register application type succeeded
```

"MyApplicationV1" is the folder in the image store where the application package is located. The application type with name "MyApplicationType" and version "1.0.0" (both are found in the application manifest) is now registered in the cluster.

The [Register-ServiceFabricApplicationType](#) command returns only after the system has successfully registered the application package. How long registration takes depends on the size and contents of the application package. If needed, the **-TimeoutSec** parameter can be used to supply a longer timeout (the default timeout is 60 seconds).

If you have a large application package or if you are experiencing timeouts, use the **-Async** parameter. The command returns when the cluster accepts the register command, and the processing continues as needed. The [Get-ServiceFabricApplicationType](#) command lists all successfully registered application type versions and their registration status. You can use this command to determine when the registration is done.

```
PS C:\> Get-ServiceFabricApplicationType

ApplicationTypeName      : MyApplicationType
ApplicationTypeVersion   : 1.0.0
Status                  : Available
DefaultParameters        : { "Stateless1_InstanceCount" = "-1" }
```

## Remove an application package from the image store

It's recommended that you remove the application package after the application is successfully registered. Deleting application packages from the image store frees up system resources. Keeping unused application packages consumes disk storage and leads to application performance issues.

```
PS C:\> Remove-ServiceFabricApplicationPackage -ApplicationPackagePathInImageStore MyApplicationV1
```

## Create the application

You can instantiate an application from any application type version that has been registered successfully by using the [New-ServiceFabricApplication](#) cmdlet. The name of each application must start with the "*fabric:*" scheme and must be unique for each application instance. Any default services defined in the application manifest of the target application type are also created.

```
PS C:\> New-ServiceFabricApplication fabric:/MyApp MyApplicationType 1.0.0

ApplicationName      : fabric:/MyApp
ApplicationTypeName   : MyApplicationType
ApplicationTypeVersion : 1.0.0
ApplicationParameters : {}
```

Multiple application instances can be created for any given version of a registered application type. Each application instance runs in isolation, with its own work directory and process.

To see which named apps and services are running in the cluster, run the [Get-ServiceFabricApplication](#) and [Get-ServiceFabricService](#) cmdlets:

```
PS C:\> Get-ServiceFabricApplication

ApplicationName      : fabric:/MyApp
ApplicationTypeName : MyApplicationType
ApplicationTypeVersion : 1.0.0
ApplicationStatus   : Ready
HealthState         : Ok
ApplicationParameters : {}

PS C:\> Get-ServiceFabricApplication | Get-ServiceFabricService

ServiceName      : fabric:/MyApp/Stateless1
ServiceKind       : Stateless
ServiceTypeName   : Stateless1Type
IsServiceGroup    : False
ServiceManifestVersion : 1.0.0
ServiceStatus     : Active
HealthState       : Ok
```

## Remove an application

When an application instance is no longer needed, you can permanently remove it by name using the [Remove-ServiceFabricApplication](#) cmdlet. [Remove-ServiceFabricApplication](#) automatically removes all services that belong to the application as well, permanently removing all service state.

### WARNING

This operation cannot be reversed, and application state cannot be recovered.

```
PS C:\> Remove-ServiceFabricApplication fabric:/MyApp

Confirm
Continue with this operation?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"):
Remove application instance succeeded

PS C:\> Get-ServiceFabricApplication
```

## Unregister an application type

When a particular version of an application type is no longer needed, you should unregister the application type using the [Unregister-ServiceFabricApplicationType](#) cmdlet. Unregistering unused application types releases storage space used by the image store by removing application binaries. Unregistering an application type does not remove the application package. An application type can be unregistered as long as no applications are instantiated against it and no pending application upgrades are referencing it.

Run [Get-ServiceFabricApplicationType](#) to see the application types currently registered in the cluster:

```
PS C:\> Get-ServiceFabricApplicationType

ApplicationTypeName      : MyApplicationType
ApplicationTypeVersion   : 1.0.0
Status                  : Available
DefaultParameters        : { "Stateless1_InstanceCount" = "-1" }
```

Run [Unregister-ServiceFabricApplicationType](#) to unregister a specific application type:

```
PS C:\> Unregister-ServiceFabricApplicationType MyApplicationType 1.0.0
```

## Troubleshooting

### **Copy-ServiceFabricApplicationPackage asks for an ImageStoreConnectionString**

The Service Fabric SDK environment should already have the correct defaults set up. But if needed, the ImageStoreConnectionString for all commands should match the value that the Service Fabric cluster is using. You can find the ImageStoreConnectionString in the cluster manifest, retrieved using the [Get-ServiceFabricClusterManifest](#) and [Get-ImageStoreConnectionStringFromClusterManifest](#) commands:

```
PS C:\> Get-ImageStoreConnectionStringFromClusterManifest(Get-ServiceFabricClusterManifest)
```

The **Get-ImageStoreConnectionStringFromClusterManifest** cmdlet, which is part of the Service Fabric SDK PowerShell module, is used to get the image store connection string. To import the SDK module, run:

```
Import-Module "$ENV:ProgramFiles\Microsoft SDKs\Service Fabric\Tools\PSModule\ServiceFabricSDK\ServiceFabricSDK.psm1"
```

The ImageStoreConnectionString is found in the cluster manifest:

```
<ClusterManifest xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Name="Server-Default-SingleNode" Version="1.0"
  xmlns="http://schemas.microsoft.com/2011/01/fabric">

  [...]

  <Section Name="Management">
    <Parameter Name="ImageStoreConnectionString" Value="file:D:\ServiceFabric\Data\ImageStore" />
  </Section>

  [...]
```

See [Understand the image store connection string](#) for supplementary information about the image store and image store connection string.

### **Deploy large application package**

Issue: [Copy-ServiceFabricApplicationPackage](#) times out for a large application package (order of GB). Try:

- Specify a larger timeout for [Copy-ServiceFabricApplicationPackage](#) command, with `TimeoutSec` parameter. By default, the timeout is 30 minutes.
- Check the network connection between your source machine and cluster. If the connection is slow, consider using a machine with a better network connection. If the client machine is in another region than the cluster, consider using a client machine in a closer or same region as the cluster.
- Check if you are hitting external throttling. For example, when the image store is configured to use azure

storage, upload may be throttled.

Issue: Upload package completed successfully, but [Register-ServiceFabricApplicationType](#) times out. Try:

- [Compress the package](#) before copying to the image store. The compression reduces the size and the number of files, which in turn reduces the amount of traffic and work that Service Fabric must perform. The upload operation may be slower (especially if you include the compression time), but register and unregister the application type are faster.
- Specify a larger timeout for [Register-ServiceFabricApplicationType](#) with `TimeoutSec` parameter.
- Specify `Async` switch for [Register-ServiceFabricApplicationType](#). The command returns when the cluster accepts the command and the registration of the application type continues asynchronously. For this reason, there is no need to specify a higher timeout in this case. The [Get-ServiceFabricApplicationType](#) command lists all successfully registered application type versions and their registration status. You can use this command to determine when the registration is done.

```
PS C:\> Get-ServiceFabricApplicationType

ApplicationTypeName      : MyApplicationType
ApplicationTypeVersion   : 1.0.0
Status                  : Available
DefaultParameters        : { "Stateless1_InstanceCount" = "-1" }
```

## Deploy application package with many files

Issue: [Register-ServiceFabricApplicationType](#) times out for an application package with many files (order of thousands). Try:

- [Compress the package](#) before copying to the image store. The compression reduces the number of files.
- Specify a larger timeout for [Register-ServiceFabricApplicationType](#) with `TimeoutSec` parameter.
- Specify `Async` switch for [Register-ServiceFabricApplicationType](#). The command returns when the cluster accepts the command and the registration of the application type continues asynchronously. For this reason, there is no need to specify a higher timeout in this case. The [Get-ServiceFabricApplicationType](#) command lists all successfully registered application type versions and their registration status. You can use this command to determine when the registration is done.

```
PS C:\> Get-ServiceFabricApplicationType

ApplicationTypeName      : MyApplicationType
ApplicationTypeVersion   : 1.0.0
Status                  : Available
DefaultParameters        : { "Stateless1_InstanceCount" = "-1" }
```

## Next steps

[Service Fabric application upgrade](#)

[Service Fabric health introduction](#)

[Diagnose and troubleshoot a Service Fabric service](#)

[Model an application in Service Fabric](#)

# Manage an Azure Service Fabric application by using Azure Service Fabric CLI

10/5/2017 • 4 min to read • [Edit Online](#)

Learn how to create and delete applications that are running in an Azure Service Fabric cluster.

## Prerequisites

- Install Service Fabric CLI. Then, select your Service Fabric cluster. For more information, see [Get started with Service Fabric CLI](#).
- Have a Service Fabric application package ready to be deployed. For more information about how to author and package an application, read about the [Service Fabric application model](#).

## Overview

To deploy a new application, complete these steps:

1. Upload an application package to the Service Fabric image store.
2. Provision an application type.
3. Delete the image store content.
4. Specify and create an application.
5. Specify and create services.

To remove an existing application, complete these steps:

1. Delete the application.
2. Unprovision the associated application type.

## Deploy a new application

To deploy a new application, complete the following tasks:

### Upload a new application package to the image store

Before you create an application, upload the application package to the Service Fabric image store.

For example, if your application package is in the `app_package_dir` directory, use the following commands to upload the directory:

```
sfcctl application upload --path ~/app_package_dir
```

For large application packages, you can specify the `--show-progress` option to display the progress of the upload.

### Provision the application type

When the upload is finished, provision the application. To provision the application, use the following command:

```
sfcctl application provision --application-type-build-path app_package_dir
```

The value for `application-type-build-path` is the name of the directory where you uploaded your application

package.

## Delete the application package

It's recommended that you remove the application package after the application is successfully registered. Deleting application packages from the image store frees up system resources. Keeping unused application packages consumes disk storage and leads to application performance issues.

To delete the application package from the image store, use the following command:

```
sfctl store delete --content-path app_package_dir
```

`content-path` must be the name of the directory that you uploaded when you created the application.

## Create an application from an application type

After you provision the application, use the following command to name and create your application:

```
sfctl application create --app-name fabric:/TestApp --app-type TestAppType --app-version 1.0
```

`app-name` is the name that you want to use for the application instance. You can get additional parameters from the previously provisioned application manifest.

The application name must start with the prefix `fabric:/`.

## Create services for the new application

After you have created an application, create services from the application. In the following example, we create a new stateless service from our application. The services that you can create from an application are defined in a service manifest in the previously provisioned application package.

```
sfctl service create --app-id TestApp --name fabric:/TestApp/TestSvc --service-type TestServiceType \
--stateless --instance-count 1 --singleton-scheme
```

# Verify application deployment and health

To verify everything is healthy, use the following health commands:

```
sfctl application list
sfctl service list --application-id TestApp
```

To verify that the service is healthy, use similar commands to retrieve the health of both the service and the application:

```
sfctl application health --application-id TestApp
sfctl service health --service-id TestApp/TestSvc
```

Healthy services and applications have a `HealthState` value of `ok`.

# Remove an existing application

To remove an application, complete the following tasks:

## Delete the application

To delete the application, use the following command:

```
sfctl application delete --application-id TestEdApp
```

## Unprovision the application type

After you delete the application, you can un provision the application type if you no longer need it. To un provision the application type, use the following command:

```
sfctl application un provision --application-type-name TestAppTye --application-type-version 1.0
```

The type name and type version must match the name and version in the previously provisioned application manifest.

## Upgrade application

After creating your application, you can repeat the same set of steps to provision a second version of your application. Then, with a Service Fabric application upgrade you can transition to running the second version of the application. For more information, see the documentation on [Service Fabric application upgrades](#).

To perform an upgrade, first provision the next version of the application using the same commands as before:

```
sfctl application upload --path ~/app_package_dir_2
sfctl application provision --application-type-build-path app_package_dir_2
sfctl store delete --content-path app_package_dir_2
```

It is recommended then to perform a monitored automatic upgrade, launch the upgrade by running the following command:

```
sfctl application upgrade --app-id TestApp --app-version 2.0.0 --parameters "{\"test\":\"value\"}" --mode Monitored
```

Upgrades override existing parameters with whatever set is specified. Application parameters should be passed as arguments to the upgrade command, if necessary. Application parameters should be encoded as a JSON object.

To retrieve any parameters previously specified, you can use the `sfctl application info` command.

When an application upgrade is in progress, the status can be retrieved using the

```
sfctl application upgrade-status
```

Finally, if an upgrade is in progress and needs to be canceled, you can use the

```
sfctl application upgrade-rollback
```

## Next steps

- [Service Fabric CLI basics](#)
- [Getting started with Service Fabric on Linux](#)
- [Launching a Service Fabric application upgrade](#)

# Deploy and remove applications using Visual Studio

6/27/2017 • 5 min to read • [Edit Online](#)

The Azure Service Fabric extension for Visual Studio provides an easy, repeatable, and scriptable way to publish an application to a Service Fabric cluster.

## The artifacts required for publishing

### **Deploy-FabricApplication.ps1**

This is a PowerShell script that uses a publish profile path as a parameter for publishing Service Fabric applications. Since this script is part of your application, you are welcome to modify it as necessary for your application.

### **Publish profiles**

A folder in the Service Fabric application project called **PublishProfiles** contains XML files that store essential information for publishing an application, such as:

- Service Fabric cluster connection parameters
- Path to an application parameter file
- Upgrade settings

By default, your application will include three publish profiles: Local.1Node.xml, Local.5Node.xml, and Cloud.xml. You can add more profiles by copying and pasting one of the default files.

### **Application parameter files**

A folder in the Service Fabric application project called **ApplicationParameters** contains XML files for user-specified application manifest parameter values. Application manifest files can be parameterized so that you can use different values for deployment settings. To learn more about parameterizing your application, see [Manage multiple environments in Service Fabric](#).

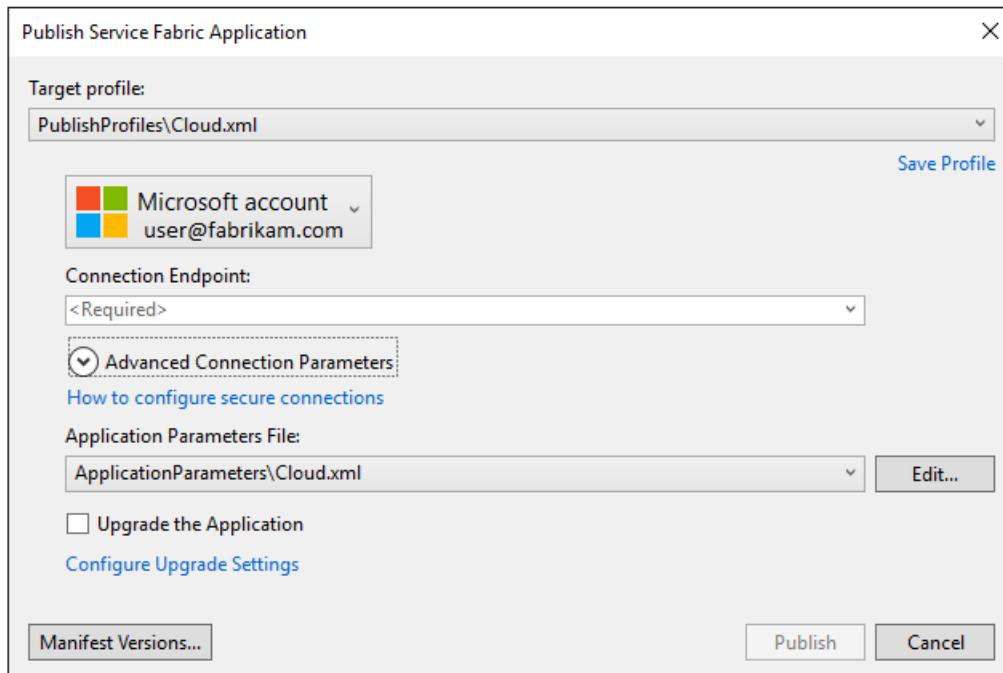
#### **NOTE**

For actor services, you should build the project first before attempting to edit the file in an editor or through the publish dialog box. This is because part of the manifest files will be generated during the build.

## To publish an application using the Publish Service Fabric Application dialog box

The following steps demonstrate how to publish an application using the **Publish Service Fabric Application** dialog box provided by the Visual Studio Service Fabric Tools.

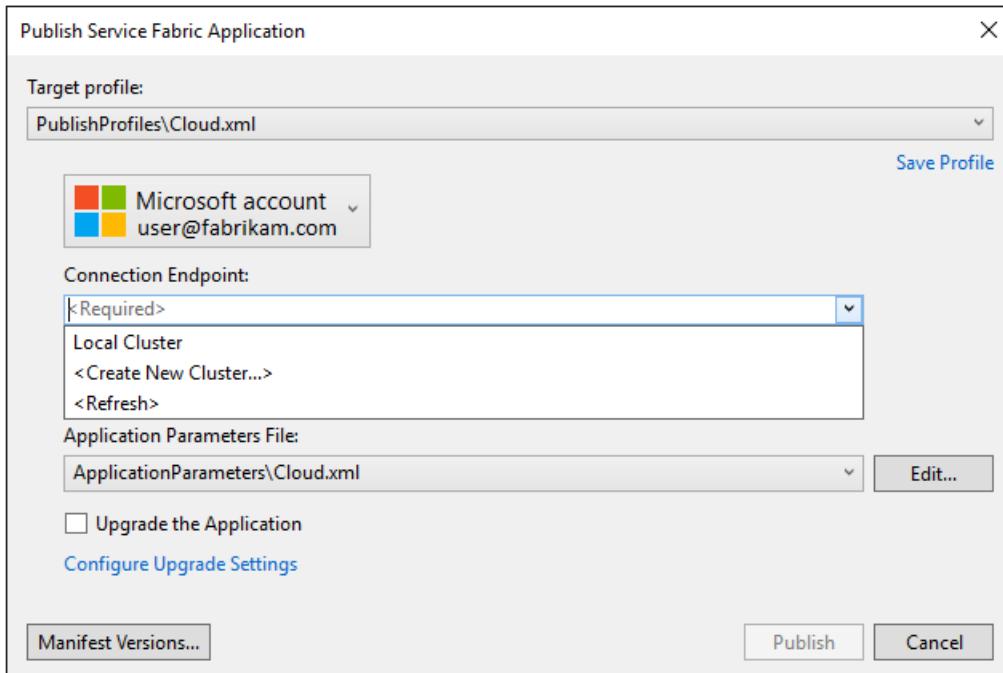
1. On the shortcut menu of the Service Fabric Application project, choose **Publish...** to view the **Publish Service Fabric Application** dialog box.



The file selected in the **Target profile** dropdown list box is where all of the settings, except **Manifest versions**, are saved. You can either reuse an existing profile or create a new one by choosing <**Manage Profiles...**> in the **Target profile** dropdown list box. When you choose a publish profile, its contents appear in the corresponding fields of the dialog box. To save your changes at any time, choose the **Save Profile** link.

2. In the **Connection endpoint** section, specify a local or remote Service Fabric cluster's publishing endpoint. To add or change the connection endpoint, click on the **Connection Endpoint** dropdown list. The list shows the available Service Fabric cluster connection endpoints to which you can publish based on your Azure subscription(s). Note that if you are not already logged in to Visual Studio, you will be prompted to do so.

Use the cluster selection dialog box to choose from the set of available subscriptions and clusters.

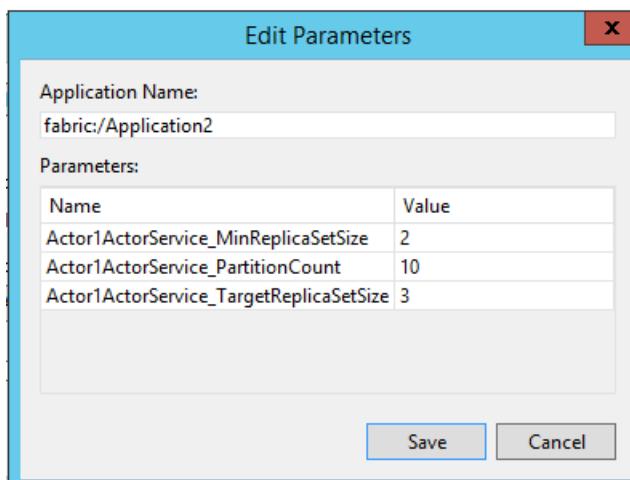


**NOTE**

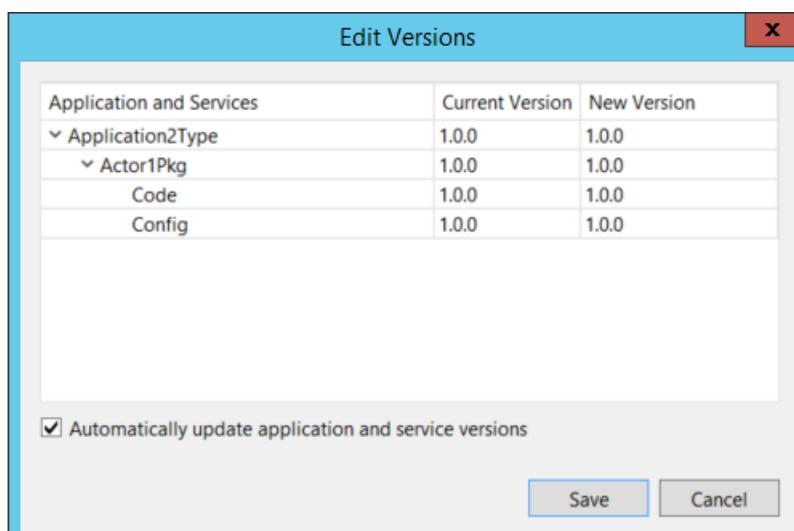
If you would like to publish to an arbitrary endpoint (such as a party cluster), see the [Publishing to an arbitrary cluster endpoint](#) section below.

Once you choose an endpoint, Visual Studio validates the connection to the selected Service Fabric cluster. If the cluster isn't secure, Visual Studio can connect to it immediately. However, if the cluster is secure, you'll need to install a certificate on your local computer before proceeding. See [How to configure secure connections](#) for more information. When you're done, choose the **OK** button. The selected cluster appears in the **Publish Service Fabric Application** dialog box.

3. In the **Application Parameter File** dropdown list box, navigate to an application parameter file. An application parameter file holds user-specified values for parameters in the application manifest file. To add or change a parameter, choose the **Edit** button. Enter or change the parameter's value in the **Parameters** grid. When you're done, choose the **Save** button.



4. Use the **Upgrade the Application** checkbox to specify whether this publish action is an upgrade. Upgrade publish actions differ from normal publish actions. See [Service Fabric Application Upgrade](#) for a list of differences. To configure upgrade settings, choose the **Configure Upgrade Settings** link. The upgrade parameter editor appears. See [Configure the upgrade of a Service Fabric application](#) to learn more about upgrade parameters.
5. Choose the **Manifest Versions...** button to view the **Edit Versions** dialog box. You need to update application and service versions for an upgrade to take place. See [Service Fabric application upgrade tutorial](#) to learn how application and service manifest versions impact an upgrade process.



If the application and service versions use semantic versioning such as 1.0.0 or numerical values in the

format of 1.0.0.0, select the **Automatically update application and service versions** option. When you choose this option, the service and application version numbers are automatically updated whenever a code, config, or data package version is updated. If you prefer to edit the versions manually, clear the checkbox to disable this feature.

**NOTE**

For all package entries to appear for an actor project, first build the project to generate the entries in the Service Manifest files.

- When you're done specifying all of the necessary settings, choose the **Publish** button to publish your application to the selected Service Fabric cluster. The settings that you specified are applied to the publish process.

## Publish to an arbitrary cluster endpoint (including party clusters)

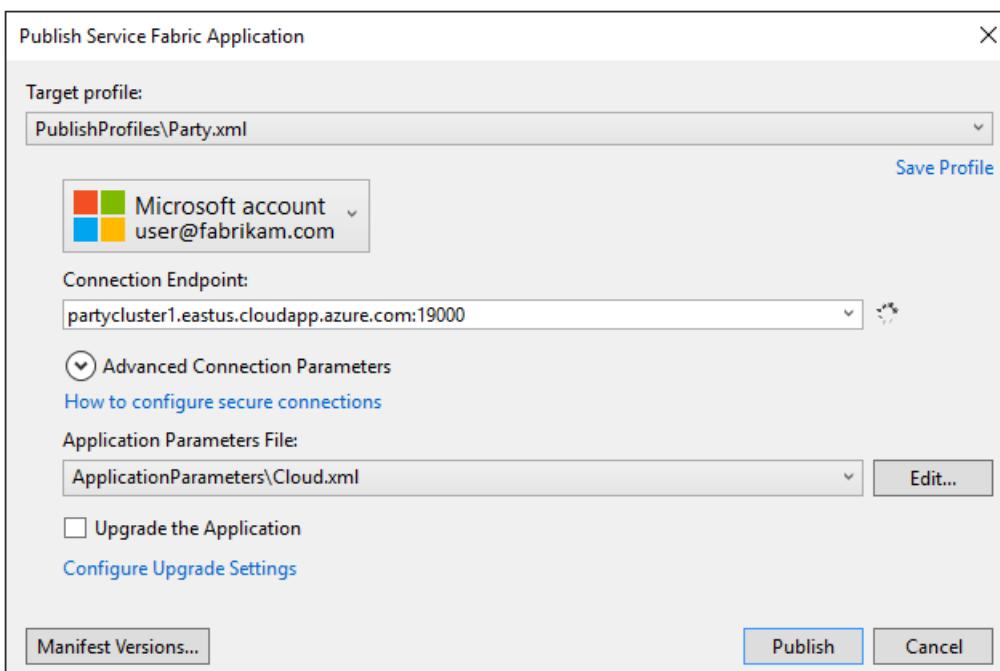
The Visual Studio publishing experience is optimized for publishing to remote clusters associated with one of your Azure subscriptions. However, it is possible to publish to arbitrary endpoints (such as Service Fabric party clusters) by directly editing the publish profile XML. As described above, three publish profiles are provided by default—**Local.1Node.xml**, **Local.5Node.xml**, and **Cloud.xml**—but you are welcome to create additional profiles for different environments. For instance, you might want to create a profile for publishing to party clusters, perhaps named **Party.xml**.

If you are connecting to an unsecured cluster, all that's required is the cluster connection endpoint, such as `partycluster1.eastus.cloudapp.azure.com:19000`. In that case, the connection endpoint in the publish profile would look something like this:

```
<ClusterConnectionParameters ConnectionEndpoint="partycluster1.eastus.cloudapp.azure.com:19000" />
```

If you are connecting to a secured cluster, you will also need to provide the details of the client certificate from the local store to be used for authentication. For more details, see [Configuring secure connections to a Service Fabric cluster](#).

Once your publish profile is set up, you can reference it in the publish dialog box as shown below.



Note that in this case, the new publish profile points to one of the default application parameter files. This is appropriate if you want to publish the same application configuration to a number of environments. By contrast, in cases where you want to have different configurations for each environment that you want to publish to, it would make sense to create a corresponding application parameter file.

## Next steps

To learn how to automate the publishing process in a continuous integration environment, see [Set up Service Fabric continuous integration](#).

# Deploy and remove applications using FabricClient

10/5/2017 • 9 min to read • [Edit Online](#)

Once an [application type has been packaged](#), it's ready for deployment into an Azure Service Fabric cluster.

Deployment involves the following three steps:

1. Upload the application package to the image store
2. Register the application type
3. Remove the application package from the image store
4. Create the application instance

After an application is deployed and an instance is running in the cluster, you can delete the application instance and its application type. To completely remove an application from the cluster involves the following steps:

1. Remove (or delete) the running application instance
2. Unregister the application type if you no longer need it

If you use [Visual Studio for deploying and debugging applications](#) on your local development cluster, all the preceding steps are handled automatically through a PowerShell script. This script is found in the *Scripts* folder of the application project. This article provides background on what that script is doing so that you can perform the same operations outside of Visual Studio.

## Connect to the cluster

Connect to the cluster by creating a [FabricClient](#) instance before you run any of the code examples in this article.

For examples of connecting to a local development cluster or a remote cluster or cluster secured using Azure Active Directory, X509 certificates, or Windows Active Directory see [Connect to a secure cluster](#). To connect to the local development cluster, run the following:

```
// Connect to the local cluster.  
FabricClient fabricClient = new FabricClient();
```

## Upload the application package

Suppose you build and package an application named *MyApplication* in Visual Studio. By default, the application type name listed in the *ApplicationManifest.xml* is "MyApplicationType". The application package, which contains the necessary application manifest, service manifests, and code/config/data packages, is located in *C:\Users\<username>\Documents\Visual Studio 2017\Projects\MyApplication\MyApplication\pkg\Debug*.

Uploading the application package puts it in a location that's accessible by the internal Service Fabric components. Service Fabric verifies the application package during the registration of the application package. However, if you want to verify the application package locally (i.e., before uploading), use the [Test-ServiceFabricApplicationPackage](#) cmdlet.

The [CopyApplicationPackage](#) API uploads the application package to the cluster image store.

If the application package is large and/or has many files, you can [compress it](#) and copy it to the image store using PowerShell. The compression reduces the size and the number of files.

See [Understand the image store connection string](#) for supplementary information about the image store and image store connection string.

## Register the application package

The application type and version declared in the application manifest become available for use when the application package is registered. The system reads the package uploaded in the previous step, verifies the package, processes the package contents, and copies the processed package to an internal system location.

The [ProvisionApplicationAsync](#) API registers the application type in the cluster and make it available for deployment.

The [GetApplicationTypeListAsync](#) API provides information about all successfully registered application types. You can use this API to determine when the registration is done.

## Remove an application package from the image store

It's recommended that you remove the application package after the application is successfully registered. Deleting application packages from the image store frees up system resources. Keeping unused application packages consumes disk storage and leads to application performance issues. Delete the application package from the image store using the [RemoveApplicationPackage](#) API.

## Create an application instance

You can instantiate an application from any application type that has been registered successfully by using the [CreateApplicationAsync](#) API. The name of each application must start with the "*fabric:*" scheme and must be unique for each application instance (within a cluster). Any default services defined in the application manifest of the target application type are also created.

Multiple application instances can be created for any given version of a registered application type. Each application instance runs in isolation, with its own working directory and set of processes.

To see which named applications and services are running in the cluster, run the [GetApplicationListAsync](#) and [GetServiceListAsync](#) APIs.

## Create a service instance

You can instantiate a service from a service type using the [CreateServiceAsync](#) API. If the service is declared as a default service in the application manifest, the service is instantiated when the application is instantiated. Calling the [CreateServiceAsync](#) API for a service that is already instantiated will return an exception of type FabricException containing an error code with a value of FabricErrorCode.ServiceAlreadyExists.

## Remove a service instance

When a service instance is no longer needed, you can remove it from the running application instance by calling the [DeleteServiceAsync](#) API.

### WARNING

This operation cannot be reversed, and service state cannot be recovered.

## Remove an application instance

When an application instance is no longer needed, you can permanently remove it by name using the

[DeleteApplicationAsync](#) API. [DeleteApplicationAsync](#) automatically removes all services that belong to the application as well, permanently removing all service state.

#### WARNING

This operation cannot be reversed, and application state cannot be recovered.

## Unregister an application type

When a particular version of an application type is no longer needed, you should unregister that particular version of the application type using the [Unregister-ServiceFabricApplicationType](#) API. Unregistering unused versions of application types releases storage space used by the image store. A version of an application type can be unregistered as long as no applications are instantiated against that version of the application type and no pending application upgrades are referencing that version of the application type.

## Troubleshooting

### **Copy-ServiceFabricApplicationPackage asks for an ImageStoreConnectionString**

The Service Fabric SDK environment should already have the correct defaults set up. But if needed, the ImageStoreConnectionString for all commands should match the value that the Service Fabric cluster is using. You can find the ImageStoreConnectionString in the cluster manifest, retrieved using the [Get-ServiceFabricClusterManifest](#) and [Get-ImageStoreConnectionStringFromClusterManifest](#) commands:

```
PS C:\> Get-ImageStoreConnectionStringFromClusterManifest(Get-ServiceFabricClusterManifest)
```

The **Get-ImageStoreConnectionStringFromClusterManifest** cmdlet, which is part of the Service Fabric SDK PowerShell module, is used to get the image store connection string. To import the SDK module, run:

```
Import-Module "$ENV:ProgramFiles\Microsoft SDKs\ServiceFabric\Tools\PSModule\ServiceFabricSDK\ServiceFabricSDK.psm1"
```

The ImageStoreConnectionString is found in the cluster manifest:

```
<ClusterManifest xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Name="Server-Default-SingleNode" Version="1.0" xmlns="http://schemas.microsoft.com/2011/01/fabric">
    [...]
    <Section Name="Management">
        <Parameter Name="ImageStoreConnectionString" Value="file:D:\ServiceFabric\Data\ImageStore" />
    </Section>
    [...]
```

See [Understand the image store connection string](#) for supplementary information about the image store and image store connection string.

### **Deploy large application package**

Issue: [CopyApplicationPackage](#) API times out for a large application package (order of GB). Try:

- Specify a larger timeout for [CopyApplicationPackage](#) method, with `timeout` parameter. By default, the timeout is 30 minutes.
- Check the network connection between your source machine and cluster. If the connection is slow, consider

using a machine with a better network connection. If the client machine is in another region than the cluster, consider using a client machine in a closer or same region as the cluster.

- Check if you are hitting external throttling. For example, when the image store is configured to use azure storage, upload may be throttled.

Issue: Upload package completed successfully, but [ProvisionApplicationAsync](#) API times out. Try:

- [Compress the package](#) before copying to the image store. The compression reduces the size and the number of files, which in turn reduces the amount of traffic and work that Service Fabric must perform. The upload operation may be slower (especially if you include the compression time), but register and un-register the application type are faster.
- Specify a larger timeout for [ProvisionApplicationAsync](#) API with `timeout` parameter.

## Deploy application package with many files

Issue: [ProvisionApplicationAsync](#) times out for an application package with many files (order of thousands). Try:

- [Compress the package](#) before copying to the image store. The compression reduces the number of files.
- Specify a larger timeout for [ProvisionApplicationAsync](#) with `timeout` parameter.

## Code example

The following example copies an application package to the image store, provisions the application type, creates an application instance, creates a service instance, removes the application instance, un-provisions the application type, and deletes the application package from the image store.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Reflection;
using System.Threading.Tasks;

using System.Fabric;
using System.Fabric.Description;
using System.Threading;

namespace ServiceFabricAppLifecycle
{
    class Program
    {
        static void Main(string[] args)
        {

            string clusterConnection = "localhost:19000";
            string appName = "fabric:/MyApplication";
            string appType = "MyApplicationType";
            string appVersion = "1.0.0";
            string serviceName = "fabric:/MyApplication/Stateless1";
            string imageStoreConnectionString = "file:C:\\\\SfDevCluster\\\\Data\\\\ImageStoreShare";
            string packagePathInImageStore = "MyApplication";
            string packagePath = "C:\\\\Users\\\\username\\\\Documents\\\\Visual Studio
2017\\\\Projects\\\\MyApplication\\\\MyApplication\\\\pkg\\\\Debug";
            string serviceType = "Stateless1Type";

            // Connect to the cluster.
            FabricClient fabricClient = new FabricClient(clusterConnection);

            // Copy the application package to a location in the image store
            try
            {
                fabricClient.ApplicationManager.CopyApplicationPackage(imageStoreConnectionString, packagePath,
                packagePathInImageStore);
```

```

        Console.WriteLine("Application package copied to {0}", packagePathInImageStore);
    }
    catch (AggregateException ae)
    {
        Console.WriteLine("Application package copy to Image Store failed: ");
        foreach (Exception ex in ae.InnerExceptions)
        {
            Console.WriteLine("HRESULT: {0} Message: {1}", ex.HResult, ex.Message);
        }
    }

    // Provision the application. "MyApplicationV1" is the folder in the image store where the application
    package is located.
    // The application type with name "MyApplicationType" and version "1.0.0" (both are found in the
    application manifest)
    // is now registered in the cluster.
    try
    {
        fabricClient.ApplicationManager.ProvisionApplicationAsync(packagePathInImageStore).Wait();

        Console.WriteLine("Provisioned application type {0}", packagePathInImageStore);
    }
    catch (AggregateException ae)
    {
        Console.WriteLine("Provision Application Type failed:");

        foreach (Exception ex in ae.InnerExceptions)
        {
            Console.WriteLine("HRESULT: {0} Message: {1}", ex.HResult, ex.Message);
        }
    }

    // Delete the application package from a location in the image store.
    try
    {
        fabricClient.ApplicationManager.RemoveApplicationPackage(imageStoreConnectionString,
    packagePathInImageStore);
        Console.WriteLine("Application package removed from {0}", packagePathInImageStore);
    }
    catch (AggregateException ae)
    {
        Console.WriteLine("Application package removal from Image Store failed: ");
        foreach (Exception ex in ae.InnerExceptions)
        {
            Console.WriteLine("HRESULT: {0} Message: {1}", ex.HResult, ex.Message);
        }
    }

    // Create the application instance.
    try
    {
        ApplicationDescription appDesc = new ApplicationDescription(new Uri(appName), appType, appVersion);
        fabricClient.ApplicationManager.CreateApplicationAsync(appDesc).Wait();
        Console.WriteLine("Created application instance of type {0}, version {1}", appType, appVersion);
    }
    catch (AggregateException ae)
    {
        Console.WriteLine("CreateApplication failed.");
        foreach (Exception ex in ae.InnerExceptions)
        {
            Console.WriteLine("HRESULT: {0} Message: {1}", ex.HResult, ex.Message);
        }
    }

    // Create the stateless service description. For stateful services, use a StatefulServiceDescription
    object.
    StatelessServiceDescription serviceDescription = new StatelessServiceDescription();
    serviceDescription.ApplicationName = new Uri(appName);
    serviceDescription.InstanceCount = 1;
}

```

```

serviceDescription.PartitionSchemeDescription = new SingletonPartitionSchemeDescription();
serviceDescription.ServiceName = new Uri(serviceName);
serviceDescription.ServiceTypeName = serviceType;

// Create the service instance. If the service is declared as a default service in the
ApplicationManifest.xml,
// the service instance is already running and this call will fail.
try
{
    fabricClient.ServiceManager.CreateServiceAsync(serviceDescription).Wait();
    Console.WriteLine("Created service instance {0}", serviceName);
}
catch (AggregateException ae)
{
    Console.WriteLine("CreateService failed.");
    foreach (Exception ex in ae.InnerExceptions)
    {
        Console.WriteLine("HRESULT: {0} Message: {1}", ex.HResult, ex.Message);
    }
}

// Delete a service instance.
try
{
    DeleteServiceDescription deleteServiceDescription = new DeleteServiceDescription(new
Uri(serviceName));

    fabricClient.ServiceManager.DeleteServiceAsync(deleteServiceDescription);
    Console.WriteLine("Deleted service instance {0}", serviceName);
}
catch (AggregateException ae)
{
    Console.WriteLine("DeleteService failed.");
    foreach (Exception ex in ae.InnerExceptions)
    {
        Console.WriteLine("HRESULT: {0} Message: {1}", ex.HResult, ex.Message);
    }
}

// Delete an application instance from the application type.
try
{
    DeleteApplicationDescription deleteApplicationDescription = new DeleteApplicationDescription(new
Uri(appName));

    fabricClient.ApplicationManager.DeleteApplicationAsync(deleteApplicationDescription).Wait();
    Console.WriteLine("Deleted application instance {0}", appName);
}
catch (AggregateException ae)
{
    Console.WriteLine("DeleteApplication failed.");
    foreach (Exception ex in ae.InnerExceptions)
    {
        Console.WriteLine("HRESULT: {0} Message: {1}", ex.HResult, ex.Message);
    }
}

// Un-provision the application type.
try
{
    fabricClient.ApplicationManager.UnprovisionApplicationAsync(appType, appVersion).Wait();
    Console.WriteLine("Un-provisioned application type {0}, version {1}", appType, appVersion);
}
catch (AggregateException ae)
{
    Console.WriteLine("Un-provision application type failed: ");
    foreach (Exception ex in ae.InnerExceptions)
    {
        Console.WriteLine("HRESULT: {0} Message: {1}", ex.HResult, ex.Message);
    }
}

```

```
        }

        Console.WriteLine("Hit enter...");
        Console.Read();
    }
}
}
```

## Next steps

[Service Fabric application upgrade](#)

[Service Fabric health introduction](#)

[Diagnose and troubleshoot a Service Fabric service](#)

[Model an application in Service Fabric](#)

# Service Fabric application upgrade using PowerShell

10/27/2017 • 7 min to read • [Edit Online](#)

The most frequently used and recommended upgrade approach is the monitored rolling upgrade. Azure Service Fabric monitors the health of the application being upgraded based on a set of health policies. Once an update domain (UD) is upgraded, Service Fabric evaluates the application health and either proceeds to the next update domain or fails the upgrade depending on the health policies.

A monitored application upgrade can be performed using the managed or native APIs, PowerShell, or REST. For instructions on performing an upgrade using Visual Studio, see [Upgrading your application using Visual Studio](#).

With Service Fabric monitored rolling upgrades, the application administrator can configure the health evaluation policy that Service Fabric uses to determine if the application is healthy. In addition, the administrator can configure the action to be taken when the health evaluation fails (for example, doing an automatic rollback.) This section walks through a monitored upgrade for one of the SDK samples that uses PowerShell. The following Microsoft Virtual Academy video also walks you through an app upgrade:



## Step 1: Build and deploy the Visual Objects sample

Build and publish the application by right-clicking on the application project, **VisualObjectsApplication**, and selecting the **Publish** command. For more information, see [Service Fabric application upgrade tutorial](#).

Alternatively, you can use PowerShell to deploy your application.

### NOTE

Before any of the Service Fabric commands may be used in PowerShell, you first need to connect to the cluster by using the `Connect-ServiceFabricCluster` cmdlet. Similarly, it is assumed that the Cluster has already been set up on your local machine. See the article on [setting up your Service Fabric development environment](#).

After building the project in Visual Studio, you can use the PowerShell command [Copy-ServiceFabricApplicationPackage](#) to copy the application package to the ImageStore. If you want to verify the app package locally, use the [Test-ServiceFabricApplicationPackage](#) cmdlet. The next step is to register the application to the Service Fabric runtime using the [Register-ServiceFabricApplicationType](#) cmdlet. The following step is to start an instance of the application by using the [New-ServiceFabricApplication](#) cmdlet. These three steps are analogous to using the **Deploy** menu item in Visual Studio. Once provisioning is completed, you should clean up the copied application package from the image store in order to reduce the resources consumed. If an application type is no longer required, it should be unregistered for the same reason. See [Deploy and remove applications using PowerShell](#) for more information.

Now, you can use [Service Fabric Explorer](#) to view the cluster and the application. The application has a web service that can be navigated to in Internet Explorer by typing <http://localhost:8081/visualobjects> in the address bar. You should see some floating visual objects moving around in the screen. Additionally, you can use [Get-ServiceFabricApplication](#) to check the application status.

## Step 2: Update the Visual Objects sample

You might notice that with the version that was deployed in Step 1, the visual objects do not rotate. Let's upgrade this application to one where the visual objects also rotate.

Select the `VisualObjects.ActorService` project within the `VisualObjects` solution, and open the `StatefulVisualObjectActor.cs` file. Within that file, navigate to the method `MoveObject`, comment out `this.State.Move()`, and uncomment `this.State.Move(true)`. This change rotates the objects after the service is upgraded.

We also need to update the `ServiceManifest.xml` file (under `PackageRoot`) of the project

**VisualObjects.ActorService**. Update the `CodePackage` and the service version to 2.0, and the corresponding lines in the `ServiceManifest.xml` file. You can use the Visual Studio *Edit Manifest Files* option after you right-click on the solution to make the manifest file changes.

After the changes are made, the manifest should look like the following (highlighted portions show the changes):

```
<ServiceManifestName="VisualObjects.ActorService" Version="2.0"
  xmlns="http://schemas.microsoft.com/2011/01/fabric" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <CodePackageName="Code" Version="2.0">
```

Now the `ApplicationManifest.xml` file (found under the **VisualObjects** project under the **VisualObjects** solution) is updated to version 2.0 of the **VisualObjects.ActorService** project. In addition, the Application version is updated to 2.0.0.0 from 1.0.0.0. The `ApplicationManifest.xml` should look like the following snippet:

```
<ApplicationManifest xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance" ApplicationTypeName="VisualObjects" ApplicationTypeVersion="2.0.0.0"
  xmlns="http://schemas.microsoft.com/2011/01/fabric">

  <ServiceManifestRef ServiceManifestName="VisualObjects.ActorService" ServiceManifestVersion="2.0" />
```

Now, build the project by selecting just the **ActorService** project, and then right-clicking and selecting the **Build** option in Visual Studio. If you select **Rebuild all**, you should update the versions for all projects, since the code would have changed. Next, let's package the updated application by right-clicking on **VisualObjectsApplication**, selecting the Service Fabric Menu, and choosing **Package**. This action creates an application package that can be deployed. Your updated application is ready to be deployed.

## Step 3: Decide on health policies and upgrade parameters

Familiarize yourself with the [application upgrade parameters](#) and the [upgrade process](#) to get a good understanding of the various upgrade parameters, time-outs, and health criterion applied. For this walkthrough, the service health evaluation criterion is set to the default (and recommended) values, which means that all services and instances should be *healthy* after the upgrade.

However, let's increase the `HealthCheckStableDuration` to 60 seconds (so that the services are healthy for at least 20 seconds before the upgrade proceeds to the next update domain). Let's also set the `UpgradeDomainTimeout` to be 1200 seconds and the `UpgradeTimeout` to be 3000 seconds.

Finally, let's also set the `UpgradeFailureAction` to rollback. This option requires Service Fabric to roll back the

application to the previous version if it encounters any issues during the upgrade. Thus, when starting the upgrade (in Step 4), the following parameters are specified:

```
FailureAction = Rollback  
HealthCheckStableDurationSec = 60  
UpgradeDomainTimeoutSec = 1200  
UpgradeTimeout = 3000
```

## Step 4: Prepare application for upgrade

Now the application is built and ready to be upgraded. If you open up a PowerShell window as an administrator and type [Get-ServiceFabricApplication](#), it should let you know that it is application type 1.0.0.0 of **VisualObjects** that's been deployed.

The application package is stored under the following relative path where you uncompressed the Service Fabric SDK - *Samples\Services\Stateful\VisualObjects\VisualObjects\obj\x64\Debug*. You should find a "Package" folder in that directory, where the application package is stored. Check the timestamps to ensure that it is the latest build (you may need to modify the paths appropriately as well).

Now let's copy the updated application package to the Service Fabric ImageStore (where the application packages are stored by Service Fabric). The parameter *ApplicationPackagePathInImageStore* informs Service Fabric where it can find the application package. We have put the updated application in "VisualObjects\_V2" with the following command (you may need to modify paths again appropriately).

```
Copy-ServiceFabricApplicationPackage -ApplicationPackagePath  
.\\Samples\\Services\\Stateful\\VisualObjects\\VisualObjects\\obj\\x64\\Debug\\Package  
-ImageStoreConnectionString fabric:ImageStore -ApplicationPackagePathInImageStore "VisualObjects\\_V2"
```

The next step is to register this application with Service Fabric, which can be performed using the [Register-ServiceFabricApplicationType](#) command:

```
Register-ServiceFabricApplicationType -ApplicationPathInImageStore "VisualObjects\\_V2"
```

If the preceding command doesn't succeed, it is likely that you need a rebuild of all services. As mentioned in Step 2, you may have to update your WebService version as well.

It's recommended that you remove the application package after the application is successfully registered. Deleting application packages from the image store frees up system resources. Keeping unused application packages consumes disk storage and leads to application performance issues.

```
Remove-ServiceFabricApplicationPackage -ApplicationPackagePathInImageStore "VisualObjects\\_V2" -  
ImageStoreConnectionString fabric:ImageStore
```

## Step 5: Start the application upgrade

Now, we're all set to start the application upgrade by using the [Start-ServiceFabricApplicationUpgrade](#) command:

```
Start-ServiceFabricApplicationUpgrade -ApplicationName fabric:/VisualObjects -ApplicationTypeVersion 2.0.0.0  
-HealthCheckStableDurationSec 60 -UpgradeDomainTimeoutSec 1200 -UpgradeTimeout 3000 -FailureAction Rollback  
-Monitored
```

The application name is the same as it was described in the *ApplicationManifest.xml* file. Service Fabric uses this

name to identify which application is getting upgraded. If you set the time-outs to be too short, you may encounter a failure message that states the problem. Refer to the troubleshooting section, or increase the time-outs.

Now, as the application upgrade proceeds, you can monitor it using Service Fabric Explorer, or by using the [Get-ServiceFabricApplicationUpgrade](#) PowerShell command:

```
Get-ServiceFabricApplicationUpgrade fabric:/VisualObjects
```

In a few minutes, the status that you got by using the preceding PowerShell command, should state that all update domains were upgraded (completed). And you should find that the visual objects in your browser window have started rotating!

You can try upgrading from version 2 to version 3, or from version 2 to version 1 as an exercise. Moving from version 2 to version 1 is also considered an upgrade. Play with time-outs and health policies to make yourself familiar with them. When you are deploying to an Azure cluster, the parameters need to be set appropriately. It is good to set the time-outs conservatively.

## Next steps

[Upgrading your application using Visual Studio](#) walks you through an application upgrade using Visual Studio.

Control how your application upgrades by using [upgrade parameters](#).

Make your application upgrades compatible by learning how to use [data serialization](#).

Learn how to use advanced functionality while upgrading your application by referring to [Advanced topics](#).

Fix common problems in application upgrades by referring to the steps in [Troubleshooting application upgrades](#).

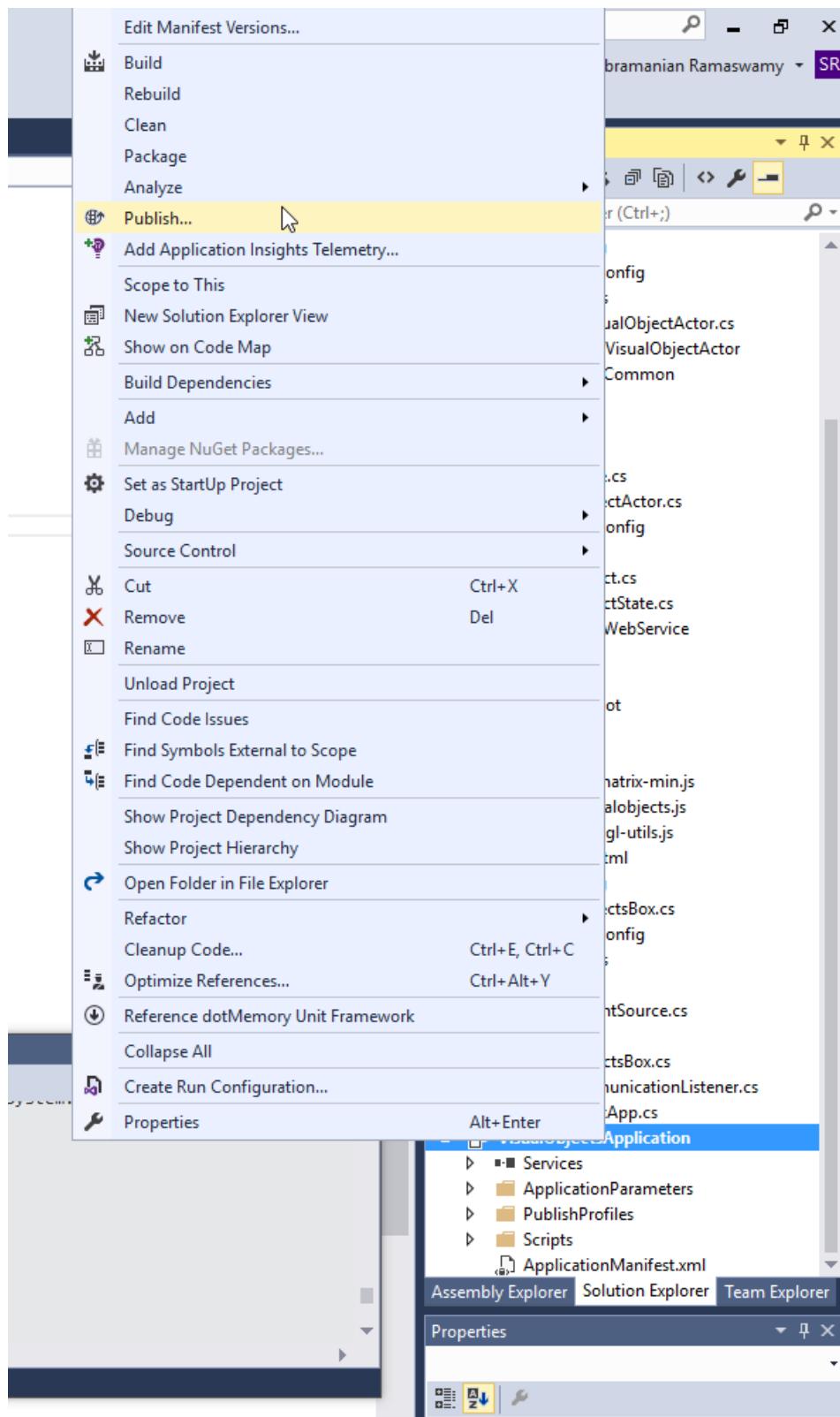
# Service Fabric application upgrade tutorial using Visual Studio

8/15/2017 • 3 min to read • [Edit Online](#)

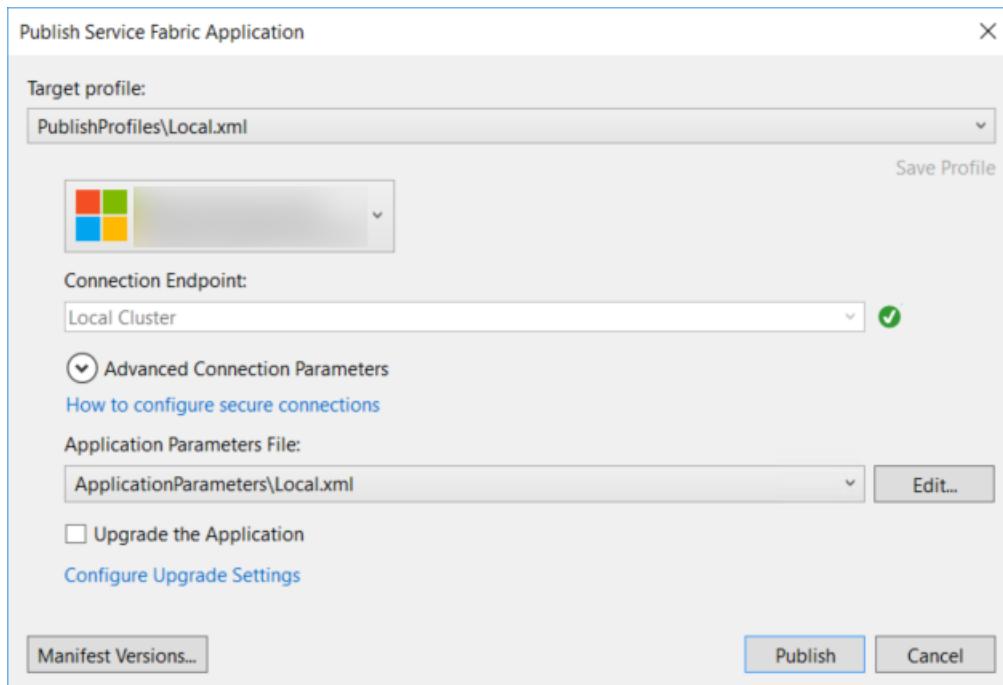
Azure Service Fabric simplifies the process of upgrading cloud applications by ensuring that only changed services are upgraded, and that application health is monitored throughout the upgrade process. It also automatically rolls back the application to the previous version upon encountering issues. Service Fabric application upgrades are *Zero Downtime*, since the application can be upgraded with no downtime. This tutorial covers how to complete a rolling upgrade from Visual Studio.

## Step 1: Build and publish the Visual Objects sample

First, download the [Visual Objects](#) application from GitHub. Then, build and publish the application by right-clicking on the application project, **VisualObjects**, and selecting the **Publish** command in the Service Fabric menu item.



Selecting **Publish** brings up a popup, and you can set the **Target profile** to **PublishProfiles\Local.xml**. The window should look like the following before you click **Publish**.



Now you can click **Publish** in the dialog box. You can use [Service Fabric Explorer](#) to view the cluster and the application. The Visual Objects application has a web service that you can go to by typing <http://localhost:8081/visualobjects/> in the address bar of your browser. You should see 10 floating visual objects moving around on the screen.

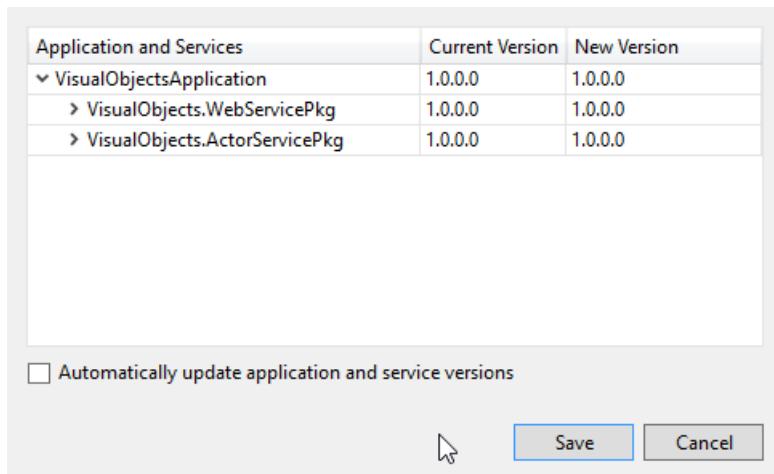
**NOTE:** If deploying to `cloud.xml` profile (Azure Service Fabric), the application should then be available at [http://{ServiceFabricName}.\(Region\).cloudapp.azure.com:8081/visualobjects/](http://{ServiceFabricName}.(Region).cloudapp.azure.com:8081/visualobjects/). Make sure you do have `8081/TCP` configured in the Load Balancer (find the Load Balancer in the same resource group as the Service Fabric instance).

## Step 2: Update the Visual Objects sample

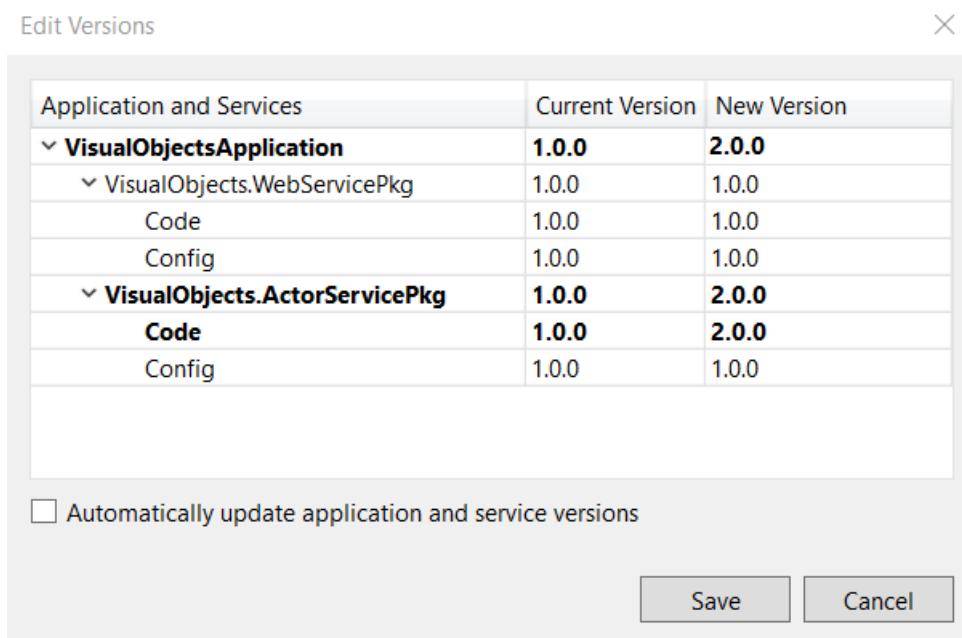
You might notice that with the version that was deployed in step 1, the visual objects do not rotate. Let's upgrade this application to one where the visual objects also rotate.

Select the `VisualObjects.ActorService` project within the `VisualObjects` solution, and open the `VisualObjectActor.cs` file. Within that file, go to the method `MoveObject`, comment out `visualObject.Move(false)`, and uncomment `visualObject.Move(true)`. This code change rotates the objects after the service is upgraded. **Now you can build (not rebuild) the solution**, which builds the modified projects. If you select `Rebuild all`, you have to update the versions for all the projects.

We also need to version our application. To make the version changes after you right-click on the `VisualObjects` project, you can use the Visual Studio **Edit Manifest Versions** option. Selecting this option brings up the dialog box for edition versions as follows:



Update the versions for the modified projects and their code packages, along with the application to version 2.0.0. After the changes are made, the manifest should look like the following (bold portions show the changes):



The Visual Studio tools can do automatic rollups of versions upon selecting **Automatically update application and service versions**. If you use [SemVer](#), you need to update the code and/or configuration package version alone if that option is selected.

Save the changes, and now check the **Upgrade the Application** box.

## Step 3: Upgrade your application

Familiarize yourself with the [application upgrade parameters](#) and the [upgrade process](#) to get a good understanding of the various upgrade parameters, time-outs, and health criterion that can be applied. For this walkthrough, the service health evaluation criterion is set to the default (unmonitored mode). You can configure these settings by selecting **Configure Upgrade Settings** and then modifying the parameters as desired.

Now we are all set to start the application upgrade by selecting **Publish**. This option upgrades your application to version 2.0.0, in which the objects rotate. Service Fabric upgrades one update domain at a time (some objects are updated first, followed by others), and the service remains accessible during the upgrade. Access to the service can be checked through your client (browser).

Now, as the application upgrade proceeds, you can monitor it with Service Fabric Explorer, by using the **Upgrades in Progress** tab under the applications.

In a few minutes, all update domains should be upgraded (completed), and the Visual Studio output window

should also state that the upgrade is completed. And you should find that *all* the visual objects in your browser window are now rotating!

You may want to try changing the versions, and moving from version 2.0.0 to version 3.0.0 as an exercise, or even from version 2.0.0 back to version 1.0.0. Play with time-outs and health policies to make yourself familiar with them. When deploying to an Azure cluster as opposed to a local cluster, the parameters used may have to differ. We recommend that you set the time-outs conservatively.

## Next steps

[Upgrading your application using PowerShell](#) walks you through an application upgrade using PowerShell.

Control how your application is upgraded by using [upgrade parameters](#).

Make your application upgrades compatible by learning how to use [data serialization](#).

Learn how to use advanced functionality while upgrading your application by referring to [Advanced topics](#).

Fix common problems in application upgrades by referring to the steps in [Troubleshooting application upgrades](#).

# Troubleshoot application upgrades

10/4/2017 • 8 min to read • [Edit Online](#)

This article covers some of the common issues around upgrading an Azure Service Fabric application and how to resolve them.

## Troubleshoot a failed application upgrade

When an upgrade fails, the output of the **Get-ServiceFabricApplicationUpgrade** command contains additional information for debugging the failure. The following list specifies how the additional information can be used:

1. Identify the failure type.
2. Identify the failure reason.
3. Isolate one or more failing components for further investigation.

This information is available when Service Fabric detects the failure regardless of whether the **FailureAction** is to roll back or suspend the upgrade.

### Identify the failure type

In the output of **Get-ServiceFabricApplicationUpgrade**, **FailureTimestampUtc** identifies the timestamp (in UTC) at which an upgrade failure was detected by Service Fabric and **FailureAction** was triggered.

**FailureReason** identifies one of three potential high-level causes of the failure:

1. UpgradeDomainTimeout - Indicates that a particular upgrade domain took too long to complete and **UpgradeDomainTimeout** expired.
2. OverallUpgradeTimeout - Indicates that the overall upgrade took too long to complete and **UpgradeTimeout** expired.
3. HealthCheck - Indicates that after upgrading an update domain, the application remained unhealthy according to the specified health policies and **HealthCheckRetryTimeout** expired.

These entries only show up in the output when the upgrade fails and starts rolling back. Further information is displayed depending on the type of the failure.

### Investigate upgrade timeouts

Upgrade timeout failures are most commonly caused by service availability issues. The output following this paragraph is typical of upgrades where service replicas or instances fail to start in the new code version. The **UpgradeDomainProgressAtFailure** field captures a snapshot of any pending upgrade work at the time of failure.

```

PS D:\temp> Get-ServiceFabricApplicationUpgrade fabric:/DemoApp

ApplicationName      : fabric:/DemoApp
ApplicationTypeName  : DemoAppType
TargetApplicationTypeVersion : v2
ApplicationParameters : {}
StartTimeStampUtc    : 4/14/2015 9:26:38 PM
FailureTimeStampUtc : 4/14/2015 9:27:05 PM
FailureReason        : UpgradeDomainTimeout
UpgradeDomainProgressAtFailure : MYUD1

   NodeName      : Node4
   UpgradePhase   : PostUpgradeSafetyCheck
   PendingSafetyChecks :
   WaitForPrimaryPlacement - PartitionId: 744c8d9f-1d26-417e-a60e-
cd48f5c098f0

   NodeName      : Node1
   UpgradePhase   : PostUpgradeSafetyCheck
   PendingSafetyChecks :
   WaitForPrimaryPlacement - PartitionId: 4b43f4d8-b26b-424e-9307-
7a7a62e79750

UpgradeState          : RollingBackCompleted
UpgradeDuration       : 00:00:46
CurrentUpgradeDomainDuration : 00:00:00
NextUpgradeDomain     :
UpgradeDomainsStatus  : { "MYUD1" = "Completed";
                           "MYUD2" = "Completed";
                           "MYUD3" = "Completed" }
UpgradeKind           : Rolling
RollingUpgradeMode    : UnmonitoredAuto
ForceRestart          : False
UpgradeReplicaSetCheckTimeout : 00:00:00

```

In this example, the upgrade failed at upgrade domain *MYUD1* and two partitions (*744c8d9f-1d26-417e-a60e-  
cd48f5c098f0* and *4b43f4d8-b26b-424e-9307-7a7a62e79750*) were stuck. The partitions were stuck because  
the runtime was unable to place primary replicas (*WaitForPrimaryPlacement*) on target nodes *Node1* and *Node4*.

The **Get-ServiceFabricNode** command can be used to verify that these two nodes are in upgrade domain  
*MYUD1*. The *UpgradePhase* says *PostUpgradeSafetyCheck*, which means that these safety checks are occurring  
after all nodes in the upgrade domain have finished upgrading. All this information points to a potential issue  
with the new version of the application code. The most common issues are service errors in the open or  
promotion to primary code paths.

An *UpgradePhase* of *PreUpgradeSafetyCheck* means there were issues preparing the upgrade domain before it  
was performed. The most common issues in this case are service errors in the close or demotion from primary  
code paths.

The current **UpgradeState** is *RollingBackCompleted*, so the original upgrade must have been performed with a  
rollback **FailureAction**, which automatically rolled back the upgrade upon failure. If the original upgrade was  
performed with a manual **FailureAction**, then the upgrade would instead be in a suspended state to allow live  
debugging of the application.

In rare cases, the **UpgradeDomainProgressAtFailure** field may be empty if the overall upgrade times out just  
as the system completes all work for the current upgrade domain. If this happens, try increasing the  
**UpgradeTimeout** and **UpgradeDomainTimeout** upgrade parameter values and retry the upgrade.

### Investigate health check failures

Health check failures can be triggered by various issues that can happen after all nodes in an upgrade domain  
finish upgrading and passing all safety checks. The output following this paragraph is typical of an upgrade  
failure due to failed health checks. The **UnhealthyEvaluations** field captures a snapshot of health checks that

failed at the time of the upgrade according to the specified [health policy](#).

```
PS D:\temp> Get-ServiceFabricApplicationUpgrade fabric:/DemoApp

ApplicationName : fabric:/DemoApp
ApplicationTypeName : DemoAppType
TargetApplicationTypeVersion : v4
ApplicationParameters : {}
StartTimestampUtc : 4/24/2015 2:42:31 AM
UpgradeState : RollingForwardPending
UpgradeDuration : 00:00:27
CurrentUpgradeDomainDuration : 00:00:27
NextUpgradeDomain : MYUD2
UpgradeDomainsStatus : { "MYUD1" = "Completed"; "MYUD2" = "Pending"; "MYUD3" = "Pending" }
UnhealthyEvaluations :
    Unhealthy services: 50% (2/4), ServiceType='PersistedServiceType',
MaxPercentUnhealthyServices=0%.
    Unhealthy service: ServiceName='fabric:/DemoApp/Svc3',
AggregatedHealthState='Error'.
        Unhealthy partitions: 100% (1/1),
MaxPercentUnhealthyPartitionsPerService=0%.
        Unhealthy partition: PartitionId='3a9911f6-a2e5-452d-89a8-09271e7e49a8', AggregatedHealthState='Error'.
            Error event: SourceId='Replica', Property='InjectedFault'.
    Unhealthy service: ServiceName='fabric:/DemoApp/Svc2',
AggregatedHealthState='Error'.
        Unhealthy partitions: 100% (1/1),
MaxPercentUnhealthyPartitionsPerService=0%.
        Unhealthy partition: PartitionId='744c8d9f-1d26-417e-a60e-cd48f5c098f0', AggregatedHealthState='Error'.
            Error event: SourceId='Replica', Property='InjectedFault'.

UpgradeKind : Rolling
RollingUpgradeMode : Monitored
FailureAction : Manual
ForceRestart : False
UpgradeReplicaSetCheckTimeout : 49710.06:28:15
HealthCheckWaitDuration : 00:00:00
HealthCheckStableDuration : 00:00:10
HealthCheckRetryTimeout : 00:00:10
UpgradeDomainTimeout : 10675199.02:48:05.4775807
UpgradeTimeout : 10675199.02:48:05.4775807
ConsiderWarningAsError :
MaxPercentUnhealthyPartitionsPerService :
MaxPercentUnhealthyReplicasPerPartition :
MaxPercentUnhealthyServices :
MaxPercentUnhealthyDeployedApplications :
ServiceTypeHealthPolicyMap :
```

Investigating health check failures first requires an understanding of the Service Fabric health model. But even without such an in-depth understanding, we can see that two services are unhealthy: *fabric:/DemoApp/Svc3* and *fabric:/DemoApp/Svc2*, along with the error health reports ("InjectedFault" in this case). In this example, two out of four services are unhealthy, which is below the default target of 0% unhealthy (*MaxPercentUnhealthyServices*).

The upgrade was suspended upon failing by specifying a **FailureAction** of manual when starting the upgrade.

This mode allows us to investigate the live system in the failed state before taking any further action.

### Recover from a suspended upgrade

With a rollback **FailureAction**, there is no recovery needed since the upgrade automatically rolls back upon failing. With a manual **FailureAction**, there are several recovery options:

1. trigger a rollback
2. Proceed through the remainder of the upgrade manually
3. Resume the monitored upgrade

The **Start-ServiceFabricApplicationRollback** command can be used at any time to start rolling back the application. Once the command returns successfully, the rollback request has been registered in the system and starts shortly thereafter.

The **Resume-ServiceFabricApplicationUpgrade** command can be used to proceed through the remainder of the upgrade manually, one upgrade domain at a time. In this mode, only safety checks are performed by the system. No more health checks are performed. This command can only be used when the *UpgradeState* shows *RollingForwardPending*, which means that the current upgrade domain has finished upgrading but the next one has not started (pending).

The **Update-ServiceFabricApplicationUpgrade** command can be used to resume the monitored upgrade with both safety and health checks being performed.

```
PS D:\temp> Update-ServiceFabricApplicationUpgrade fabric:/DemoApp -UpgradeMode Monitored

UpgradeMode          : Monitored
ForceRestart         :
UpgradeReplicaSetCheckTimeout   :
FailureAction        :
HealthCheckWaitDuration    :
HealthCheckStableDuration   :
HealthCheckRetryTimeout    :
UpgradeTimeout         :
UpgradeDomainTimeout     :
ConsiderWarningAsError   :
MaxPercentUnhealthyPartitionsPerService :
MaxPercentUnhealthyReplicasPerPartition   :
MaxPercentUnhealthyServices      :
MaxPercentUnhealthyDeployedApplications :
ServiceTypeHealthPolicyMap       :

PS D:\temp>
```

The upgrade continues from the upgrade domain where it was last suspended and use the same upgrade parameters and health policies as before. If needed, any of the upgrade parameters and health policies shown in the preceding output can be changed in the same command when the upgrade resumes. In this example, the upgrade was resumed in Monitored mode, with the parameters and the health policies unchanged.

## Further troubleshooting

### Service Fabric is not following the specified health policies

Possible Cause 1:

Service Fabric translates all percentages into actual numbers of entities (for example, replicas, partitions, and services) for health evaluation and always rounds up to whole entities. For example, if the maximum *MaxPercentUnhealthyReplicasPerPartition* is 21% and there are five replicas, then Service Fabric allows up to two unhealthy replicas (that is, `Math.Ceiling (5*0.21)`). Thus, health policies should be set accordingly.

## Possible Cause 2:

Health policies are specified in terms of percentages of total services and not specific service instances. For example, before an upgrade, if an application has four service instances A, B, C, and D, where service D is unhealthy but with little impact to the application. We want to ignore the known unhealthy service D during upgrade and set the parameter *MaxPercentUnhealthyServices* to be 25%, assuming only A, B, and C need to be healthy.

However, during the upgrade, D may become healthy while C becomes unhealthy. The upgrade would still succeed because only 25% of the services are unhealthy. However, it might result in unanticipated errors due to C being unexpectedly unhealthy instead of D. In this situation, D should be modeled as a different service type from A, B, and C. Since health policies are specified per service type, different unhealthy percentage thresholds can be applied to different services.

### I did not specify a health policy for application upgrade, but the upgrade still fails for some time-outs that I never specified

When health policies aren't provided to the upgrade request, they are taken from the *ApplicationManifest.xml* of the current application version. For example, if you're upgrading Application X from version 1.0 to version 2.0, application health policies specified for in version 1.0 are used. If a different health policy should be used for the upgrade, then the policy needs to be specified as part of the application upgrade API call. The policies specified as part of the API call only apply during the upgrade. Once the upgrade is complete, the policies specified in the *ApplicationManifest.xml* are used.

### Incorrect time-outs are specified

You may have wondered about what happens when time-outs are set inconsistently. For example, you may have an *UpgradeTimeout* that's less than the *UpgradeDomainTimeout*. The answer is that an error is returned. Errors are returned if the *UpgradeDomainTimeout* is less than the sum of *HealthCheckWaitDuration* and *HealthCheckRetryTimeout*, or if *UpgradeDomainTimeout* is less than the sum of *HealthCheckWaitDuration* and *HealthCheckStableDuration*.

### My upgrades are taking too long

The time for an upgrade to complete depends on the health checks and time-outs specified. Health checks and time-outs depend on how long it takes to copy, deploy, and stabilize the application. Being too aggressive with time-outs might mean more failed upgrades, so we recommend starting conservatively with longer time-outs.

Here's a quick refresher on how the time-outs interact with the upgrade times:

Upgrades for an upgrade domain cannot complete faster than *HealthCheckWaitDuration* + *HealthCheckStableDuration*.

Upgrade failure cannot occur faster than *HealthCheckWaitDuration* + *HealthCheckRetryTimeout*.

The upgrade time for an upgrade domain is limited by *UpgradeDomainTimeout*. If *HealthCheckRetryTimeout* and *HealthCheckStableDuration* are both non-zero and the health of the application keeps switching back and forth, then the upgrade eventually times out on *UpgradeDomainTimeout*. *UpgradeDomainTimeout* starts counting down once the upgrade for the current upgrade domain begins.

## Next steps

[Upgrading your Application Using Visual Studio](#) walks you through an application upgrade using Visual Studio.

[Upgrading your Application Using Powershell](#) walks you through an application upgrade using PowerShell.

Control how your application upgrades by using [Upgrade Parameters](#).

Make your application upgrades compatible by learning how to use [Data Serialization](#).

Learn how to use advanced functionality while upgrading your application by referring to [Advanced Topics](#).

# Service Fabric testability scenarios: Service communication

6/30/2017 • 4 min to read • [Edit Online](#)

Microservices and service-oriented architectural styles surface naturally in Azure Service Fabric. In these types of distributed architectures, componentized microservice applications are typically composed of multiple services that need to talk to each other. In even the simplest cases, you generally have at least a stateless web service and a stateful data storage service that need to communicate.

Service-to-service communication is a critical integration point of an application, because each service exposes a remote API to other services. Working with a set of API boundaries that involves I/O generally requires some care, with a good amount of testing and validation.

There are numerous considerations to make when these service boundaries are wired together in a distributed system:

- *Transport protocol.* Will you use HTTP for increased interoperability, or a custom binary protocol for maximum throughput?
- *Error handling.* How will permanent and transient errors be handled? What will happen when a service moves to a different node?
- *Timeouts and latency.* In multilayered applications, how will each service layer handle latency through the stack and to the user?

Whether you use one of the built-in service communication components provided by Service Fabric or you build your own, testing the interactions between your services is critical to ensuring resiliency in your application.

## Prepare for services to move

Service instances may move around over time. This is especially true when they are configured with load metrics for custom-tailored optimal resource balancing. Service Fabric moves your service instances to maximize their availability even during upgrades, failovers, scale-out, and other situations that occur over the lifetime of a distributed system.

As services move around in the cluster, your clients and other services should be prepared to handle two scenarios when they talk to a service:

- The service instance or partition replica has moved since the last time you talked to it. This is a normal part of a service lifecycle, and it should be expected to happen during the lifetime of your application.
- The service instance or partition replica is in the process of moving. Although failover of a service from one node to another occurs very quickly in Service Fabric, there may be a delay in availability if the communication component of your service is slow to start.

Handling these scenarios gracefully is important for a smooth-running system. To do so, keep in mind that:

- Every service that can be connected to has an *address* that it listens on (for example, HTTP or WebSockets). When a service instance or partition moves, its address endpoint changes. (It moves to a different node with a different IP address.) If you're using the built-in communication components, they will handle re-resolving service addresses for you.
- There may be a temporary increase in service latency as the service instance starts up its listener again. This depends on how quickly the service opens the listener after the service instance is moved.
- Any existing connections need to be closed and reopened after the service opens on a new node. A graceful

node shutdown or restart allows time for existing connections to be shut down gracefully.

### Test it: Move service instances

By using Service Fabric's testability tools, you can author a test scenario to test these situations in different ways:

1. Move a stateful service's primary replica.

The primary replica of a stateful service partition can be moved for any number of reasons. Use this to target the primary replica of a specific partition to see how your services react to the move in a very controlled manner.

```
PS > Move-ServiceFabricPrimaryReplica -PartitionId 6faa4ffa-521a-44e9-8351-dfca0f7e0466 -ServiceName fabric:/MyApplication/MyService
```

2. Stop a node.

When a node is stopped, Service Fabric moves all of the service instances or partitions that were on that node to one of the other available nodes in the cluster. Use this to test a situation where a node is lost from your cluster and all of the service instances and replicas on that node have to move.

You can stop a node by using the PowerShell **Stop-ServiceFabricNode** cmdlet:

```
PS > Restart-ServiceFabricNode -NodeName Node_1
```

## Maintain service availability

As a platform, Service Fabric is designed to provide high availability of your services. But in extreme cases, underlying infrastructure problems can still cause unavailability. It is important to test for these scenarios, too.

Stateful services use a quorum-based system to replicate state for high availability. This means that a quorum of replicas needs to be available to perform write operations. In rare cases, such as a widespread hardware failure, a quorum of replicas may not be available. In these cases, you will not be able to perform write operations, but you will still be able to perform read operations.

### Test it: Write operation unavailability

By using the testability tools in Service Fabric, you can inject a fault that induces quorum loss as a test. Although such a scenario is rare, it is important that clients and services that depend on a stateful service are prepared to handle situations where they cannot make write requests to it. It is also important that the stateful service itself is aware of this possibility and can gracefully communicate it to callers.

You can induce quorum loss by using the PowerShell **Invoke-ServiceFabricPartitionQuorumLoss** cmdlet:

```
PS > Invoke-ServiceFabricPartitionQuorumLoss -ServiceName fabric:/Myapplication/MyService -QuorumLossMode QuorumReplicas -QuorumLossDurationInSeconds 20
```

In this example, we set `QuorumLossMode` to `QuorumReplicas` to indicate that we want to induce quorum loss without taking down all replicas. This way, read operations are still possible. To test a scenario where an entire partition is unavailable, you can set this switch to `AllReplicas`.

## Next steps

[Learn more about testability actions](#)

[Learn more about testability scenarios](#)

# Induce controlled Chaos in Service Fabric clusters

8/11/2017 • 6 min to read • [Edit Online](#)

Large-scale distributed systems like cloud infrastructures are inherently unreliable. Azure Service Fabric enables developers to write reliable distributed services on top of an unreliable infrastructure. To write robust distributed services on top of an unreliable infrastructure, developers need to be able to test the stability of their services while the underlying unreliable infrastructure is going through complicated state transitions due to faults.

The [Fault Injection and Cluster Analysis Service](#) (also known as the Fault Analysis Service) gives developers the ability to induce faults to test their services. These targeted simulated faults, like [restarting a partition](#), can help exercise the most common state transitions. However targeted simulated faults are biased by definition and thus may miss bugs that show up only in hard-to-predict, long and complicated sequence of state transitions. For an unbiased testing, you can use Chaos.

Chaos simulates periodic, interleaved faults (both graceful and ungraceful) throughout the cluster over extended periods of time. Once you have configured Chaos with the rate and the kind of faults, you can start Chaos through C# or Powershell API to start generating faults in the cluster and in your services. You can configure Chaos to run for a specified time period (for example, for one hour), after which Chaos stops automatically, or you can call StopChaos API (C# or Powershell) to stop it at any time.

## NOTE

In its current form, Chaos induces only safe faults, which implies that in the absence of external faults a quorum loss, or data loss never occurs.

While Chaos is running, it produces different events that capture the state of the run at the moment. For example, an ExecutingFaultsEvent contains all the faults that Chaos has decided to execute in that iteration. A ValidationFailedEvent contains the details of a validation failure (health or stability issues) that was found during the validation of the cluster. You can invoke the GetChaosReport API (C# or Powershell) to get the report of Chaos runs. These events get persisted in a [reliable dictionary](#), which has a truncation policy dictated by two configurations: **MaxStoredChaosEventCount** (default value is 25000) and **StoredActionCleanupIntervalInSeconds** (default value is 3600). Every *StoredActionCleanupIntervalInSeconds* Chaos checks and all but the most recent *MaxStoredChaosEventCount* events, are purged from the reliable dictionary.

## Faults induced in Chaos

Chaos generates faults across the entire Service Fabric cluster and compresses faults that are seen in months or years into a few hours. The combination of interleaved faults with the high fault rate finds corner cases that may otherwise be missed. This exercise of Chaos leads to a significant improvement in the code quality of the service.

Chaos induces faults from the following categories:

- Restart a node
- Restart a deployed code package
- Remove a replica
- Restart a replica
- Move a primary replica (configurable)
- Move a secondary replica (configurable)

Chaos runs in multiple iterations. Each iteration consists of faults and cluster validation for the specified period. You can configure the time spent for the cluster to stabilize and for validation to succeed. If a failure is found in cluster validation, Chaos generates and persists a ValidationFailedEvent with the UTC timestamp and the failure details. For example, consider an instance of Chaos that is set to run for an hour with a maximum of three concurrent faults. Chaos induces three faults, and then validates the cluster health. It iterates through the previous step until it is explicitly stopped through the StopChaosAsync API or one-hour passes. If the cluster becomes unhealthy in any iteration (that is, it does not stabilize within the passed-in MaxClusterStabilizationTimeout), Chaos generates a ValidationFailedEvent. This event indicates that something has gone wrong and might need further investigation.

To get which faults Chaos induced, you can use GetChaosReport API (powershell or C#). The API gets the next segment of the Chaos report based on the passed-in continuation token or the passed-in time-range. You can either specify the ContinuationToken to get the next segment of the Chaos report or you can specify the time-range through StartTimeUtc and EndTimeUtc, but you cannot specify both the ContinuationToken and the time-range in the same call. When there are more than 100 Chaos events, the Chaos report is returned in segments where a segment contains no more than 100 Chaos events.

## Important configuration options

- **TimeToRun:** Total time that Chaos runs before it finishes with success. You can stop Chaos before it has run for the TimeToRun period through the StopChaos API.
- **MaxClusterStabilizationTimeout:** The maximum amount of time to wait for the cluster to become healthy before producing a ValidationFailedEvent. This wait is to reduce the load on the cluster while it is recovering. The checks performed are:
  - If the cluster health is OK
  - If the service health is OK
  - If the target replica set size is achieved for the service partition
  - That no InBuild replicas exist
- **MaxConcurrentFaults:** The maximum number of concurrent faults that are induced in each iteration. The higher the number, the more aggressive Chaos is and the failovers and the state transition combinations that the cluster goes through are also more complex.

### NOTE

Regardless how high a value *MaxConcurrentFaults* has, Chaos guarantees - in the absence of external faults - there is no quorum loss or data loss.

- **EnableMoveReplicaFaults:** Enables or disables the faults that cause the primary or secondary replicas to move. These faults are disabled by default.
- **WaitTimeBetweenIterations:** The amount of time to wait between iterations. That is, the amount of time Chaos will pause after having executed a round of faults and having finished the corresponding validation of the health of the cluster. The higher the value, the lower is the average fault injection rate.
- **WaitTimeBetweenFaults:** The amount of time to wait between two consecutive faults in a single iteration. The higher the value, the lower the concurrency of (or the overlap between) faults.
- **ClusterHealthPolicy:** Cluster health policy is used to validate the health of the cluster in between Chaos iterations. If the cluster health is in error or if an unexpected exception happens during fault execution, Chaos will wait for 30 minutes before the next health-check - to provide the cluster with some time to recuperate.
- **Context:** A collection of (string, string) type key-value pairs. The map can be used to record information about the Chaos run. There cannot be more than 100 such pairs and each string (key or value) can be at most 4095 characters long. This map is set by the starter of the Chaos run to optionally store the context about the specific

run.

## How to run Chaos

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using System.Fabric;

using System.Diagnostics;
using System.Fabric.Chaos.DataStructures;

class Program
{
    private class ChaosEventComparer : IEqualityComparer<ChaosEvent>
    {
        public bool Equals(ChaosEvent x, ChaosEvent y)
        {
            return x.TimeStampUtc.Equals(y.TimeStampUtc);
        }

        public int GetHashCode(ChaosEvent obj)
        {
            return obj.TimeStampUtc.GetHashCode();
        }
    }

    static void Main(string[] args)
    {
        var clusterConnectionString = "localhost:19000";
        using (var client = new FabricClient(clusterConnectionString))
        {
            var startTimeUtc = DateTime.UtcNow;
            var stabilizationTimeout = TimeSpan.FromSeconds(30.0);
            var timeToRun = TimeSpan.FromMinutes(60.0);
            var maxConcurrentFaults = 3;

            var parameters = new ChaosParameters(
                stabilizationTimeout,
                maxConcurrentFaults,
                true, /* EnableMoveReplicaFault */
                timeToRun);

            try
            {
                client.TestManager.StartChaosAsync(parameters).GetAwaiter().GetResult();
            }
            catch (FabricChaosAlreadyRunningException)
            {
                Console.WriteLine("An instance of Chaos is already running in the cluster.");
            }
        }

        var filter = new ChaosReportFilter(startTimeUtc, DateTime.MaxValue);

        var eventSet = new HashSet<ChaosEvent>(new ChaosEventComparer());

        while (true)
        {
            var report = client.TestManager.GetChaosReportAsync(filter).GetAwaiter().GetResult();

            foreach (var chaosEvent in report.History)
            {
                if (eventSet.Add(chaosEvent))
                {
                    Console.WriteLine(chaosEvent);
                }
            }
        }
    }
}
```

```
// When Chaos stops, a StoppedEvent is created.  
// If a StoppedEvent is found, exit the loop.  
var lastEvent = report.History.LastOrDefault();  
  
if (lastEvent is StoppedEvent)  
{  
    break;  
}  
  
Task.Delay(TimeSpan.FromSeconds(1.0)).GetAwaiter().GetResult();  
}  
}  
}  
}
```

```

$connection = "localhost:19000"
$timeToRun = 60
$maxStabilizationTimeSecs = 180
$concurrentFaults = 3
$waitTimeBetweenIterationsSec = 60

Connect-ServiceFabricCluster $connection

$events = @{}
$now = [System.DateTime]::UtcNow

Start-ServiceFabricChaos -TimeToRunMinute $timeToRun -MaxConcurrentFaults $concurrentFaults -
MaxClusterStabilizationTimeoutSec $maxStabilizationTimeSecs -EnableMoveReplicaFaults -
WaitTimeBetweenIterationsSec $waitTimeBetweenIterationsSec

while($true)
{
    $stopped = $false
    $report = Get-ServiceFabricChaosReport -StartTimeUtc $now -EndTimeUtc ([System.DateTime]::.MaxValue)

    foreach ($e in $report.History) {

        if(-Not ($events.Contains($e.TimeStampUtc.Ticks)))
        {
            $events.Add($e.TimeStampUtc.Ticks, $e)
            if($e -is [System.Fabric.Chaos.DataStructures.ValidationFailedEvent])
            {
                Write-Host -BackgroundColor White -ForegroundColor Red $e
            }
            else
            {
                if($e -is [System.Fabric.Chaos.DataStructures.StoppedEvent])
                {
                    $stopped = $true
                }

                Write-Host $e
            }
        }
    }

    if($stopped -eq $true)
    {
        break
    }

    Start-Sleep -Seconds 1
}

Stop-ServiceFabricChaos

```

# Testability actions

6/27/2017 • 8 min to read • [Edit Online](#)

In order to simulate an unreliable infrastructure, Azure Service Fabric provides you, the developer, with ways to simulate various real-world failures and state transitions. These are exposed as testability actions. The actions are the low-level APIs that cause a specific fault injection, state transition, or validation. By combining these actions, you can write comprehensive test scenarios for your services.

Service Fabric provides some common test scenarios composed of these actions. We highly recommend that you utilize these built-in scenarios, which are carefully chosen to test common state transitions and failure cases. However, actions can be used to create custom test scenarios when you want to add coverage for scenarios that are not covered by the built-in scenarios yet or that are custom tailored for your application.

C# implementations of the actions are found in the System.Fabric.dll assembly. The System Fabric PowerShell module is found in the Microsoft.ServiceFabric.Powershell.dll assembly. As part of runtime installation, the ServiceFabric PowerShell module is installed to allow for ease of use.

## Graceful vs. ungraceful fault actions

Testability actions are classified into two major buckets:

- Ungraceful faults: These faults simulate failures like machine restarts and process crashes. In such cases of failures, the execution context of process stops abruptly. This means no cleanup of the state can run before the application starts up again.
- Graceful faults: These faults simulate graceful actions like replica moves and drops triggered by load balancing. In such cases, the service gets a notification of the close and can clean up the state before exiting.

For better quality validation, run the service and business workload while inducing various graceful and ungraceful faults. Ungraceful faults exercise scenarios where the service process abruptly exits in the middle of some workflow. This tests the recovery path once the service replica is restored by Service Fabric. This will help test data consistency and whether the service state is maintained correctly after failures. The other set of failures (the graceful failures) test that the service correctly reacts to replicas being moved around by Service Fabric. This tests handling of cancellation in the RunAsync method. The service needs to check for the cancellation token being set, correctly save its state, and exit the RunAsync method.

## Testability actions list

Action	Description	Managed API	PowerShell Cmdlet	Graceful/Ungraceful Faults
CleanTestState	Removes all the test state from the cluster in case of a bad shutdown of the test driver.	CleanTestStateAsync	Remove-ServiceFabricTestState	Not applicable
InvokeDataLoss	Induces data loss into a service partition.	InvokeDataLossAsync	Invoke-ServiceFabricPartitionDataLoss	Graceful

Action	Description	Managed API	PowerShell Cmdlet	Graceful/Ungraceful Faults
InvokeQuorumLoss	Puts a given stateful service partition into quorum loss.	InvokeQuorumLossAsync	Invoke-ServiceFabricQuorumLoss	Graceful
Move Primary	Moves the specified primary replica of a stateful service to the specified cluster node.	MovePrimaryAsync	Move-ServiceFabricPrimaryReplica	Graceful
Move Secondary	Moves the current secondary replica of a stateful service to a different cluster node.	MoveSecondaryAsync	Move-ServiceFabricSecondaryReplica	Graceful
RemoveReplica	Simulates a replica failure by removing a replica from a cluster. This will close the replica and will transition it to role 'None', removing all of its state from the cluster.	RemoveReplicaAsync	Remove-ServiceFabricReplica	Graceful
RestartDeployedCodePackage	Simulates a code package process failure by restarting a code package deployed on a node in a cluster. This aborts the code package process, which will restart all the user service replicas hosted in that process.	RestartDeployedCodePackageAsync	Restart-ServiceFabricDeployedCodePackage	Ungraceful
RestartNode	Simulates a Service Fabric cluster node failure by restarting a node.	RestartNodeAsync	Restart-ServiceFabricNode	Ungraceful
RestartPartition	Simulates a datacenter blackout or cluster blackout scenario by restarting some or all replicas of a partition.	RestartPartitionAsync	Restart-ServiceFabricPartition	Graceful
RestartReplica	Simulates a replica failure by restarting a persisted replica in a cluster, closing the replica and then reopening it.	RestartReplicaAsync	Restart-ServiceFabricReplica	Graceful

Action	Description	Managed API	Powershell Cmdlet	Graceful/Ungraceful Faults
StartNode	Starts a node in a cluster that is already stopped.	StartNodeAsync	Start-ServiceFabricNode	Not applicable
StopNode	Simulates a node failure by stopping a node in a cluster. The node will stay down until StartNode is called.	StopNodeAsync	Stop-ServiceFabricNode	Ungraceful
ValidateApplication	Validates the availability and health of all Service Fabric services within an application, usually after inducing some fault into the system.	ValidateApplicationAsync	Test-ServiceFabricApplication	Not applicable
ValidateService	Validates the availability and health of a Service Fabric service, usually after inducing some fault into the system.	ValidateServiceAsync	Test-ServiceFabricService	Not applicable

## Running a testability action using PowerShell

This tutorial shows you how to run a testability action by using PowerShell. You will learn how to run a testability action against a local (one-box) cluster or an Azure cluster. Microsoft.Fabric.PowerShell.dll--the Service Fabric PowerShell module--is installed automatically when you install the Microsoft Service Fabric MSI. The module is loaded automatically when you open a PowerShell prompt.

Tutorial segments:

- [Run an action against a one-box cluster](#)
- [Run an action against an Azure cluster](#)

### Run an action against a one-box cluster

To run a testability action against a local cluster, first connect to the cluster and open the PowerShell prompt in administrator mode. Let us look at the **Restart-ServiceFabricNode** action.

```
Restart-ServiceFabricNode -NodeName Node1 -CompletionMode DoNotVerify
```

Here the action **Restart-ServiceFabricNode** is being run on a node named "Node1". The completion mode specifies that it should not verify whether the restart-node action actually succeeded. Specifying the completion mode as "Verify" will cause it to verify whether the restart action actually succeeded. Instead of directly specifying the node by its name, you can specify it via a partition key and the kind of replica, as follows:

```
Restart-ServiceFabricNode -ReplicaKindPrimary -PartitionKindNamed -PartitionKey Partition3 -CompletionMode Verify
```

```
$connection = "localhost:19000"
$nodeName = "Node1"

Connect-ServiceFabricCluster $connection
Restart-ServiceFabricNode -NodeName $nodeName -CompletionMode DoNotVerify
```

**Restart-ServiceFabricNode** should be used to restart a Service Fabric node in a cluster. This will stop the Fabric.exe process, which will restart all of the system service and user service replicas hosted on that node. Using this API to test your service helps uncover bugs along the failover recovery paths. It helps simulate node failures in the cluster.

The following screenshot shows the **Restart-ServiceFabricNode** testability command in action.

```
PS C:\windows\system32> Get-ServiceFabricNode | ft -Property NodeName,NodeUpTime,HealthState -AutoSize
NodeName NodeUpTime HealthState
-----
Node.4 00:03:28 Ok
Node.2 00:03:28 Ok
Node.1 00:03:28 Ok
Node.5 00:03:28 Ok
Node.3 00:03:28 Ok

PS C:\windows\system32> Restart-ServiceFabricNode -NodeName Node.4

NodeResult      :
    NodeName      : Node.4
    InstanceId    : 0

SelectedReplica : None

PS C:\windows\system32> Get-ServiceFabricNode | ft -Property NodeName,NodeUpTime,HealthState -AutoSize
NodeName NodeUpTime HealthState
-----
Node.4 00:00:00 Ok
Node.2 00:04:00 Ok
Node.1 00:04:00 Ok
Node.5 00:04:00 Ok
Node.3 00:04:00 Ok

PS C:\windows\system32>
```

The output of the first **Get-ServiceFabricNode** (a cmdlet from the Service Fabric PowerShell module) shows that the local cluster has five nodes: Node.1 to Node.5. After the testability action (cmdlet) **Restart-ServiceFabricNode** is executed on the node, named Node.4, we see that the node's uptime has been reset.

#### Run an action against an Azure cluster

Running a testability action (by using PowerShell) against an Azure cluster is similar to running the action against a local cluster. The only difference is that before you can run the action, instead of connecting to the local cluster, you need to connect to the Azure cluster first.

## Running a testability action using C#

To run a testability action by using C#, first you need to connect to the cluster by using FabricClient. Then obtain the parameters needed to run the action. Different parameters can be used to run the same action. Looking at the `RestartServiceFabricNode` action, one way to run it is by using the node information (node name and node instance ID) in the cluster.

```
RestartNodeAsync(nodeName, nodeInstanceId, completeMode, operationTimeout, CancellationToken.None)
```

Parameter explanation:

- **CompleteMode** specifies that the mode should not verify whether the restart action actually succeeded. Specifying the completion mode as "Verify" will cause it to verify whether the restart action actually succeeded.
- **OperationTimeout** sets the amount of time for the operation to finish before a TimeoutException exception is

thrown.

- **CancellationToken** enables a pending call to be canceled.

Instead of directly specifying the node by its name, you can specify it via a partition key and the kind of replica.

For further information, see [PartitionSelector](#) and [ReplicaSelector](#).

```

// Add a reference to System.Fabric.Testability.dll and System.Fabric.dll
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Fabric.Testability;
using System.Fabric;
using System.Threading;
using System.Numerics;

class Test
{
    public static int Main(string[] args)
    {
        string clusterConnection = "localhost:19000";
        Uri serviceName = new Uri("fabric:/samples/PersistentToDoListApp/PersistentToDoListService");
        string nodeName = "N0040";
        BigInteger nodeInstanceId = 130743013389060139;

        Console.WriteLine("Starting RestartNode test");
        try
        {
            //Restart the node by using ReplicaSelector
            RestartNodeAsync(clusterConnection, serviceName).Wait();

            //Another way to restart node is by using nodeName and nodeInstanceId
            RestartNodeAsync(clusterConnection, nodeName, nodeInstanceId).Wait();
        }
        catch (AggregateException exAgg)
        {
            Console.WriteLine("RestartNode did not complete: ");
            foreach (Exception ex in exAgg.InnerExceptions)
            {
                if (ex is FabricException)
                {
                    Console.WriteLine("HRESULT: {0} Message: {1}", ex.HResult, ex.Message);
                }
            }
            return -1;
        }

        Console.WriteLine("RestartNode completed.");
        return 0;
    }

    static async Task RestartNodeAsync(string clusterConnection, Uri serviceName)
    {
        PartitionSelector randomPartitionSelector = PartitionSelector.RandomOf(serviceName);
        ReplicaSelector primaryofReplicaSelector = ReplicaSelector.PrimaryOf(randomPartitionSelector);

        // Create FabricClient with connection and security information here
        FabricClient fabricclient = new FabricClient(clusterConnection);
        await fabricclient.FaultManager.RestartNodeAsync(primaryofReplicaSelector, CompletionMode.Verify);
    }

    static async Task RestartNodeAsync(string clusterConnection, string nodeName, BigInteger nodeInstanceId)
    {
        // Create FabricClient with connection and security information here
        FabricClient fabricclient = new FabricClient(clusterConnection);
        await fabricclient.FaultManager.RestartNodeAsync(nodeName, nodeInstanceId, CompletionMode.Verify);
    }
}

```

## PartitionSelector and ReplicaSelector

## PartitionSelector

PartitionSelector is a helper exposed in testability and is used to select a specific partition on which to perform any of the testability actions. It can be used to select a specific partition if the partition ID is known beforehand. Or, you can provide the partition key and the operation will resolve the partition ID internally. You also have the option of selecting a random partition.

To use this helper, create the PartitionSelector object and select the partition by using one of the Select\* methods. Then pass in the PartitionSelector object to the API that requires it. If no option is selected, it defaults to a random partition.

```
Uri serviceName = new Uri("fabric:/samples/InMemoryToDoListApp/InMemoryToDoListService");
Guid partitionIdGuid = new Guid("8fb7ebcc-56ee-4862-9cc0-7c6421e68829");
string partitionName = "Partition1";
Int64 partitionKeyUniformInt64 = 1;

// Select a random partition
PartitionSelector randomPartitionSelector = PartitionSelector.RandomOf(serviceName);

// Select a partition based on ID
PartitionSelector partitionSelectorById = PartitionSelector.PartitionIdOf(serviceName, partitionIdGuid);

// Select a partition based on name
PartitionSelector namedPartitionSelector = PartitionSelector.PartitionKeyOf(serviceName, partitionName);

// Select a partition based on partition key
PartitionSelector uniformIntPartitionSelector = PartitionSelector.PartitionKeyOf(serviceName,
partitionKeyUniformInt64);
```

## ReplicaSelector

ReplicaSelector is a helper exposed in testability and is used to help select a replica on which to perform any of the testability actions. It can be used to select a specific replica if the replica ID is known beforehand. In addition, you have the option of selecting a primary replica or a random secondary. ReplicaSelector derives from PartitionSelector, so you need to select both the replica and the partition on which you wish to perform the testability operation.

To use this helper, create a ReplicaSelector object and set the way you want to select the replica and the partition. You can then pass it into the API that requires it. If no option is selected, it defaults to a random replica and random partition.

```
Guid partitionIdGuid = new Guid("8fb7ebcc-56ee-4862-9cc0-7c6421e68829");
PartitionSelector partitionSelector = PartitionSelector.PartitionIdOf(serviceName, partitionIdGuid);
long replicaId = 130559876481875498;

// Select a random replica
ReplicaSelector randomReplicaSelector = ReplicaSelector.RandomOf(partitionSelector);

// Select the primary replica
ReplicaSelector primaryReplicaSelector = ReplicaSelector.PrimaryOf(partitionSelector);

// Select the replica by ID
ReplicaSelector replicaByIdSelector = ReplicaSelector.ReplicaIdOf(partitionSelector, replicaId);

// Select a random secondary replica
ReplicaSelector secondaryReplicaSelector = ReplicaSelector.RandomSecondaryOf(partitionSelector);
```

## Next steps

- [Testability scenarios](#)

- How to test your service
  - Simulate failures during service workloads
  - Service-to-service communication failures

# Simulate failures during service workloads

6/27/2017 • 3 min to read • [Edit Online](#)

The testability scenarios in Azure Service Fabric enable developers to not worry about dealing with individual faults. There are scenarios, however, where an explicit interleaving of client workload and failures might be needed. The interleaving of client workload and faults ensures that the service is actually performing some action when failure happens. Given the level of control that testability provides, these could be at precise points of the workload execution. This induction of faults at different states in the application can find bugs and improve quality.

## Sample custom scenario

This test shows a scenario that interleaves the business workload with [graceful and ungraceful failures](#). The faults should be induced in the middle of service operations or compute for best results.

Let's walk through an example of a service that exposes four workloads: A, B, C, and D. Each corresponds to a set of workflows and could be compute, storage, or a mix. For the sake of simplicity, we will abstract out the workloads in our example. The different faults executed in this example are:

- `RestartNode`: Ungraceful fault to simulate a machine restart.
- `RestartDeployedCodePackage`: Ungraceful fault to simulate service host process crashes.
- `RemoveReplica`: Graceful fault to simulate replica removal.
- `MovePrimary`: Graceful fault to simulate replica moves triggered by the Service Fabric load balancer.

```
// Add a reference to System.Fabric.Testability.dll and System.Fabric.dll.

using System;
using System.Fabric;
using System.Fabric.Testability.Scenario;
using System.Threading;
using System.Threading.Tasks;

class Test
{
    public static int Main(string[] args)
    {
        // Replace these strings with the actual version for your cluster and application.
        string clusterConnection = "localhost:19000";
        Uri applicationName = new Uri("fabric:/samples/PersistentToDoListApp");
        Uri serviceName = new Uri("fabric:/samples/PersistentToDoListApp/PersistentToDoListService");

        Console.WriteLine("Starting Workload Test...");
        try
        {
            RunTestAsync(clusterConnection, applicationName, serviceName).Wait();
        }
        catch (AggregateException ae)
        {
            Console.WriteLine("Workload Test failed: ");
            foreach (Exception ex in ae.InnerExceptions)
            {
                if (ex is FabricException)
                {
                    Console.WriteLine("HRESULT: {0} Message: {1}", ex.HResult, ex.Message);
                }
            }
        }
        return -1;
    }
}
```

```

        Console.WriteLine("Workload Test completed successfully.");
        return 0;
    }

    public enum ServiceWorkloads
    {
        A,
        B,
        C,
        D
    }

    public enum ServiceFabricFaults
    {
        RestartNode,
        RestartCodePackage,
        RemoveReplica,
        MovePrimary,
    }

    public static async Task RunTestAsync(string clusterConnection, Uri applicationName, Uri serviceName)
    {
        // Create FabricClient with connection and security information here.
        FabricClient fabricClient = new FabricClient(clusterConnection);
        // Maximum time to wait for a service to stabilize.
        TimeSpan maxServiceStabilizationTime = TimeSpan.FromSeconds(120);

        // How many loops of faults you want to execute.
        uint testLoopCount = 20;
        Random random = new Random();

        for (var i = 0; i < testLoopCount; ++i)
        {
            var workload = SelectRandomValue<ServiceWorkloads>(random);
            // Start the workload.
            var workloadTask = RunWorkloadAsync(workload);

            // While the task is running, induce faults into the service. They can be ungraceful faults like
            // RestartNode and RestartDeployedCodePackage or graceful faults like RemoveReplica or MovePrimary.
            var fault = SelectRandomValue<ServiceFabricFaults>(random);

            // Create a replica selector, which will select a primary replica from the given service to test.
            var replicaSelector = ReplicaSelector.PrimaryOf(PartitionSelector.RandomOf(serviceName));
            // Run the selected random fault.
            await RunFaultAsync(applicationName, fault, replicaSelector, fabricClient);
            // Validate the health and stability of the service.
            await fabricClient.ServiceManager.ValidateServiceAsync(serviceName, maxServiceStabilizationTime);

            // Wait for the workload to finish successfully.
            await workloadTask;
        }
    }

    private static async Task RunFaultAsync(Uri applicationName, ServiceFabricFaults fault, ReplicaSelector
selector, FabricClient client)
    {
        switch (fault)
        {
            case ServiceFabricFaults.RestartNode:
                await client.ClusterManager.RestartNodeAsync(selector, CompletionMode.Verify);
                break;
            case ServiceFabricFaults.RestartCodePackage:
                await client.ApplicationManager.RestartDeployedCodePackageAsync(applicationName, selector,
CompletionMode.Verify);
                break;
            case ServiceFabricFaults.RemoveReplica:
                await client.ServiceManager.RemoveReplicaAsync(selector, CompletionMode.Verify, false);
                break;
        }
    }
}

```

```
        case ServiceFabricFaults.MovePrimary:
            await client.ServiceManager.MovePrimaryAsync(selector.PartitionSelector);
            break;
    }
}

private static Task RunWorkloadAsync(ServiceWorkloads workload)
{
    throw new NotImplementedException();
    // This is where you trigger and complete your service workload.
    // Note that the faults induced while your service workload is running will
    // fault the primary service. Hence, you will need to reconnect to complete or check
    // the status of the workload.
}

private static T SelectRandomValue<T>(Random random)
{
    Array values = Enum.GetValues(typeof(T));
    T workload = (T)values.GetValue(random.Next(values.Length));
    return workload;
}
}
```

# Testability scenarios

6/27/2017 • 6 min to read • [Edit Online](#)

Large distributed systems like cloud infrastructures are inherently unreliable. Azure Service Fabric gives developers the ability to write services to run on top of unreliable infrastructures. In order to write high-quality services, developers need to be able to induce such unreliable infrastructure to test the stability of their services.

The Fault Analysis Service gives developers the ability to induce fault actions to test services in the presence of failures. However, targeted simulated faults will get you only so far. To take the testing further, you can use the test scenarios in Service Fabric: a chaos test and a failover test. These scenarios simulate continuous interleaved faults, both graceful and ungraceful, throughout the cluster over extended periods of time. Once a test is configured with the rate and kind of faults, it can be started through either C# APIs or PowerShell, to generate faults in the cluster and your service.

## WARNING

ChaosTestScenario is being replaced by a more resilient, service-based Chaos. Please refer to the new article [Controlled Chaos](#) for more details.

## Chaos test

The chaos scenario generates faults across the entire Service Fabric cluster. The scenario compresses faults generally seen in months or years to a few hours. The combination of interleaved faults with the high fault rate finds corner cases that are otherwise missed. This leads to a significant improvement in the code quality of the service.

### Faults simulated in the chaos test

- Restart a node
- Restart a deployed code package
- Remove a replica
- Restart a replica
- Move a primary replica (optional)
- Move a secondary replica (optional)

The chaos test runs multiple iterations of faults and cluster validations for the specified period of time. The time spent for the cluster to stabilize and for validation to succeed is also configurable. The scenario fails when you hit a single failure in cluster validation.

For example, consider a test set to run for one hour with a maximum of three concurrent faults. The test will induce three faults, and then validate the cluster health. The test will iterate through the previous step till the cluster becomes unhealthy or one hour passes. If the cluster becomes unhealthy in any iteration, i.e. it does not stabilize within a configured time, the test will fail with an exception. This exception indicates that something has gone wrong and needs further investigation.

In its current form, the fault generation engine in the chaos test induces only safe faults. This means that in the absence of external faults, a quorum or data loss will never occur.

### Important configuration options

- **TimeToRun:** Total time that the test will run before finishing with success. The test can finish earlier in lieu of a validation failure.

- **MaxClusterStabilizationTimeout**: Maximum amount of time to wait for the cluster to become healthy before failing the test. The checks performed are whether cluster health is OK, service health is OK, the target replica set size is achieved for the service partition, and no InBuild replicas exist.
- **MaxConcurrentFaults**: Maximum number of concurrent faults induced in each iteration. The higher the number, the more aggressive the test, hence resulting in more complex failovers and transition combinations. The test guarantees that in absence of external faults there will not be a quorum or data loss, irrespective of how high this configuration is.
- **EnableMoveReplicaFaults**: Enables or disables the faults that are causing the move of the primary or secondary replicas. These faults are disabled by default.
- **WaitTimeBetweenIterations**: Amount of time to wait between iterations, i.e. after a round of faults and corresponding validation.

## How to run the chaos test

C# sample

```

using System;
using System.Fabric;
using System.Fabric.Testability.Scenario;
using System.Threading;
using System.Threading.Tasks;

class Test
{
    public static int Main(string[] args)
    {
        string clusterConnection = "localhost:19000";

        Console.WriteLine("Starting Chaos Test Scenario...");
        try
        {
            RunChaosTestScenarioAsync(clusterConnection).Wait();
        }
        catch (AggregateException ae)
        {
            Console.WriteLine("Chaos Test Scenario did not complete: ");
            foreach (Exception ex in ae.InnerExceptions)
            {
                if (ex is FabricException)
                {
                    Console.WriteLine("HRESULT: {0} Message: {1}", ex.HResult, ex.Message);
                }
            }
            return -1;
        }

        Console.WriteLine("Chaos Test Scenario completed.");
        return 0;
    }

    static async Task RunChaosTestScenarioAsync(string clusterConnection)
    {
        TimeSpan maxClusterStabilizationTimeout = TimeSpan.FromSeconds(180);
        uint maxConcurrentFaults = 3;
        bool enableMoveReplicaFaults = true;

        // Create FabricClient with connection and security information here.
        FabricClient fabricClient = new FabricClient(clusterConnection);

        // The chaos test scenario should run at least 60 minutes or until it fails.
        TimeSpan timeToRun = TimeSpan.FromMinutes(60);
        ChaosTestScenarioParameters scenarioParameters = new ChaosTestScenarioParameters(
            maxClusterStabilizationTimeout,
            maxConcurrentFaults,
            enableMoveReplicaFaults
        );
    }
}

```

```

        enableMoveReplicaFaults,
        timeToRun);

    // Other related parameters:
    // Pause between two iterations for a random duration bound by this value.
    // scenarioParameters.WaitTimeBetweenIterations = TimeSpan.FromSeconds(30);
    // Pause between concurrent actions for a random duration bound by this value.
    // scenarioParameters.WaitTimeBetweenFaults = TimeSpan.FromSeconds(10);

    // Create the scenario class and execute it asynchronously.
    ChaosTestScenario chaosScenario = new ChaosTestScenario(fabricClient, scenarioParameters);

    try
    {
        await chaosScenario.ExecuteAsync(CancellationToken.None);
    }
    catch (AggregateException ae)
    {
        throw ae.InnerException;
    }
}
}

```

## PowerShell

```

$connection = "localhost:19000"
$timeToRun = 60
$maxStabilizationTimeSecs = 180
$concurrentFaults = 3
$waitTimeBetweenIterationsSec = 60

Connect-ServiceFabricCluster $connection

Invoke-ServiceFabricChaosTestScenario -TimeToRunMinute $timeToRun -MaxClusterStabilizationTimeoutSec
$maxStabilizationTimeSecs -MaxConcurrentFaults $concurrentFaults -EnableMoveReplicaFaults -
WaitTimeBetweenIterationsSec $waitTimeBetweenIterationsSec

```

## Failover test

The failover test scenario is a version of the chaos test scenario that targets a specific service partition. It tests the effect of failover on a specific service partition while leaving the other services unaffected. Once it's configured with the target partition information and other parameters, it runs as a client-side tool that uses either C# APIs or PowerShell to generate faults for a service partition. The scenario iterates through a sequence of simulated faults and service validation while your business logic runs on the side to provide a workload. A failure in service validation indicates an issue that needs further investigation.

### Faults simulated in the failover test

- Restart a deployed code package where the partition is hosted
- Remove a primary/secondary replica or stateless instance
- Restart a primary secondary replica (if a persisted service)
- Move a primary replica
- Move a secondary replica
- Restart the partition

The failover test induces a chosen fault and then runs validation on the service to ensure its stability. The failover test induces only one fault at a time, as opposed to possible multiple faults in the chaos test. If the service partition does not stabilize within the configured timeout after each fault, the test fails. The test induces only safe faults. This means that in absence of external failures, a quorum or data loss will not occur.

## Important configuration options

- **PartitionSelector:** Selector object that specifies the partition that needs to be targeted.
- **TimeToRun:** Total time that the test will run before finishing.
- **MaxServiceStabilizationTimeout:** Maximum amount of time to wait for the cluster to become healthy before failing the test. The checks performed are whether service health is OK, the target replica set size is achieved for all partitions, and no InBuild replicas exist.
- **WaitTimeBetweenFaults:** Amount of time to wait between every fault and validation cycle.

## How to run the failover test

C#

```
using System;
using System.Fabric;
using System.Fabric.Testability.Scenario;
using System.Threading;
using System.Threading.Tasks;

class Test
{
    public static int Main(string[] args)
    {
        string clusterConnection = "localhost:19000";
        Uri serviceName = new Uri("fabric:/samples/PersistentToDoListApp/PersistentToDoListService");

        Console.WriteLine("Starting Chaos Test Scenario...");
        try
        {
            RunFailoverTestScenarioAsync(clusterConnection, serviceName).Wait();
        }
        catch (AggregateException ae)
        {
            Console.WriteLine("Chaos Test Scenario did not complete: ");
            foreach (Exception ex in ae.InnerExceptions)
            {
                if (ex is FabricException)
                {
                    Console.WriteLine("HRESULT: {0} Message: {1}", ex.HResult, ex.Message);
                }
            }
        }
        return -1;
    }

    Console.WriteLine("Chaos Test Scenario completed.");
    return 0;
}

static async Task RunFailoverTestScenarioAsync(string clusterConnection, Uri serviceName)
{
    TimeSpan maxServiceStabilizationTimeout = TimeSpan.FromSeconds(180);
    PartitionSelector randomPartitionSelector = PartitionSelector.RandomOf(serviceName);

    // Create FabricClient with connection and security information here.
    FabricClient fabricClient = new FabricClient(clusterConnection);

    // The chaos test scenario should run at least 60 minutes or until it fails.
    TimeSpan timeToRun = TimeSpan.FromMinutes(60);
    FailoverTestScenarioParameters scenarioParameters = new FailoverTestScenarioParameters(
        randomPartitionSelector,
        timeToRun,
        maxServiceStabilizationTimeout);

    // Other related parameters:
    // Pause between two iterations for a random duration bound by this value.
    // scenarioParameters.WaitTimeBetweenIterations = TimeSpan.FromSeconds(30);
    // Pause between concurrent actions for a random duration bound by this value
}
```

```
// pause between concurrent actions for a random duration bound by this value.
// scenarioParameters.WaitTimeBetweenFaults = TimeSpan.FromSeconds(10);

// Create the scenario class and execute it asynchronously.
FailoverTestScenario failoverScenario = new FailoverTestScenario(fabricClient, scenarioParameters);

try
{
    await failoverScenario.ExecuteAsync(CancellationToken.None);
}
catch (AggregateException ae)
{
    throw ae.InnerException;
}
}
```

## PowerShell

```
$connection = "localhost:19000"
$timeToRun = 60
$maxStabilizationTimeSecs = 180
$waitForTimeBetweenFaultsSec = 10
$serviceName = "fabric:/SampleApp/SampleService"

Connect-ServiceFabricCluster $connection

Invoke-ServiceFabricFailoverTestScenario -TimeToRunMinute $timeToRun -MaxServiceStabilizationTimeoutSec
$maxStabilizationTimeSecs -WaitTimeBetweenFaultsSec $waitForTimeBetweenFaultsSec -ServiceName $serviceName -
PartitionKindSingleton
```

# Replacing the Start Node and Stop node APIs with the Node Transition API

6/27/2017 • 8 min to read • [Edit Online](#)

## What do the Stop Node and Start Node APIs do?

The Stop Node API (managed: [StopNodeAsync\(\)](#), PowerShell: [Stop-ServiceFabricNode](#)) stops a Service Fabric node. A Service Fabric node is process, not a VM or machine – the VM or machine will still be running. For the rest of the document "node" will mean Service Fabric node. Stopping a node puts it into a *stopped* state where it is not a member of the cluster and cannot host services, thus simulating a *down* node. This is useful for injecting faults into the system to test your application. The Start Node API (managed: [StartNodeAsync\(\)](#), PowerShell: [Start-ServiceFabricNode](#)) reverses the Stop Node API, which brings the node back to a normal state.

## Why are we replacing these?

As described earlier, a *stopped* Service Fabric node is a node intentionally targeted using the Stop Node API. A *down* node is a node that is down for any other reason (e.g. the VM or machine is off). With the Stop Node API, the system does not expose information to differentiate between *stopped* nodes and *down* nodes.

In addition, some errors returned by these APIs are not as descriptive as they could be. For example, invoking the Stop Node API on an already *stopped* node will return the error *InvalidAddress*. This experience could be improved.

Also, the duration a node is stopped for is "infinite" until the Start Node API is invoked. We've found this can cause problems and may be error-prone. For example, we've seen problems where a user invoked the Stop Node API on a node and then forgot about it. Later, it was unclear if the node was *down* or *stopped*.

## Introducing the Node Transition APIs

We've addressed these issues above in a new set of APIs. The new Node Transition API (managed: [StartNodeTransitionAsync\(\)](#)) may be used to transition a Service Fabric node to a *stopped* state, or to transition it from a *stopped* state to a normal up state. Please note that the "Start" in the name of the API does not refer to starting a node. It refers to beginning an asynchronous operation that the system will execute to transition the node to either *stopped* or started state.

### Usage

If the Node Transition API does not throw an exception when invoked, then the system has accepted the asynchronous operation, and will execute it. A successful call does not imply the operation is finished yet. To get information about the current state of the operation, call the Node Transition Progress API (managed: [GetNodeTransitionProgressAsync\(\)](#)) with the guid used when invoking Node Transition API for this operation. The Node Transition Progress API returns an `NodeTransitionProgress` object. This object's `State` property specifies the current state of the operation. If the state is "Running" then the operation is executing. If it is `Completed`, the operation finished without error. If it is `Faulted`, there was a problem executing the operation. The `Result` property's `Exception` property will indicate what the issue was. See <https://docs.microsoft.com/dotnet/api/system.fabric.testcommandprogressstate> for more information about the `State` property, and the "Sample Usage" section below for code examples.

**Differentiating between a stopped node and a down node** If a node is *stopped* using the Node Transition API, the output of a node query (managed: [GetNodeListAsync\(\)](#), PowerShell: [Get-ServiceFabricNode](#)) will show that this node has an `IsStopped` property value of true. Note this is different from the value of the `NodeStatus` property,

which will say *Down*. If the *NodeStatus* property has a value of *Down*, but *IsStopped* is false, then the node was not stopped using the Node Transition API, and is *Down* due some other reason. If the *IsStopped* property is true, and the *NodeStatus* property is *Down*, then it was stopped using the Node Transition API.

Starting a *stopped* node using the Node Transition API will return it to function as a normal member of the cluster again. The output of the node query API will show *IsStopped* as false, and *NodeStatus* as something that is not *Down* (e.g. *Up*).

**Limited Duration** When using the Node Transition API to stop a node, one of the required parameters, *stopNodeDurationInSeconds*, represents the amount of time in seconds to keep the node *stopped*. This value must be in the allowed range, which has a minimum of 600, and a maximum of 14400. After this time expires, the node will restart itself into *Up* state automatically. Refer to Sample 1 below for an example of usage.

#### WARNING

Avoid mixing Node Transition APIs and the Stop Node and Start Node APIs. The recommendation is to use the Node Transition API only. > If a node has been already been stopped using the Stop Node API, it should be started using the Start Node API first before using the > Node Transition APIs.

#### WARNING

Multiple Node Transition APIs calls cannot be made on the same node in parallel. In such a situation, the Node Transition API will > throw a FabricException with an ErrorCode property value of NodeTransitionInProgress. Once a node transition on a specific node has > been started, you should wait until the operation reaches a terminal state (Completed, Faulted, or ForceCancelled) before starting a > new transition on the same node. Parallel node transition calls on different nodes are allowed.

#### Sample Usage

**Sample 1** - The following sample uses the Node Transition API to stop a node.

```
// Helper function to get information about a node
static Node GetNodeInfo(FabricClient fc, string node)
{
    NodeList n = null;
    while (n == null)
    {
        n = fc.QueryManager.GetNodeListAsync(node).GetAwaiter().GetResult();
        Task.Delay(TimeSpan.FromSeconds(1)).GetAwaiter();
    };

    return n.FirstOrDefault();
}

static async Task WaitForStateAsync(FabricClient fc, Guid operationId, TestCommandProgressState
targetState)
{
    NodeTransitionProgress progress = null;

    do
    {
        bool exceptionObserved = false;
        try
        {
            progress = await fc.TestManager.GetNodeTransitionProgressAsync(operationId,
TimeSpan.FromMinutes(1), CancellationToken.None).ConfigureAwait(false);
        }
        catch (OperationCanceledException oce)
        {
            Console.WriteLine("Caught exception '{0}'", oce);
            exceptionObserved = true;
        }
    } while (exceptionObserved || progress.State != targetState);
}
```

```

        }

        catch (FabricTransientException fte)
        {
            Console.WriteLine("Caught exception '{0}'", fte);
            exceptionObserved = true;
        }

        if (!exceptionObserved)
        {
            Console.WriteLine("Current state of operation '{0}': {1}", operationId, progress.State);

            if (progress.State == TestCommandProgressState.Faulted)
            {
                // Inspect the progress object's Result.Exception.HResult to get the error code.
                Console.WriteLine("{0} failed with: {1}, HResult: {2}", operationId,
progress.Result.Exception, progress.Result.Exception.HResult);

                // ...additional logic as required
            }

            if (progress.State == targetState)
            {
                Console.WriteLine("Target state '{0}' has been reached", targetState);
                break;
            }
        }

        await Task.Delay(TimeSpan.FromSeconds(5)).ConfigureAwait(false);
    }
    while (true);
}

static async Task StopNodeAsync(FabricClient fc, string nodeName, int durationInSeconds)
{
    // Uses the GetNodeListAsync() API to get information about the target node
    Node n = GetNodeInfo(fc, nodeName);

    // Create a Guid
    Guid guid = Guid.NewGuid();

    // Create a NodeStopDescription object, which will be used as a parameter into StartNodeTransition
    NodeStopDescription description = new NodeStopDescription(guid, n.NodeName, n.NodeInstanceId,
durationInSeconds);

    bool wasSuccessful = false;

    do
    {
        try
        {
            // Invoke StartNodeTransitionAsync with the NodeStopDescription from above, which will stop
            // the target node. Retry transient errors.
            await fc.TestManager.StartNodeTransitionAsync(description, TimeSpan.FromMinutes(1),
CancellationToken.None).ConfigureAwait(false);
            wasSuccessful = true;
        }
        catch (OperationCanceledException oce)
        {
            // This is retryable
        }
        catch (FabricTransientException fte)
        {
            // This is retryable
        }

        // Backoff
        await Task.Delay(TimeSpan.FromSeconds(5)).ConfigureAwait(false);
    }
    while (!wasSuccessful);
}

```

```

        // Now call StartNodeTransitionProgressAsync() until hte desired state is reached.
        await WaitForStateAsync(fc, guid, TestCommandProgressState.Completed).ConfigureAwait(false);
    }
}

```

**Sample 2** - The following sample starts a *stopped* node. It uses some helper methods from the first sample.

```

static async Task StartNodeAsync(FabricClient fc, string nodeName)
{
    // Uses the GetNodeListAsync() API to get information about the target node
    Node n = GetNodeInfo(fc, nodeName);

    Guid guid = Guid.NewGuid();
    BigInteger nodeId = n.NodeInstanceId;

    // Create a NodeStartDescription object, which will be used as a parameter into StartNodeTransition
    NodeStartDescription description = new NodeStartDescription(guid, n.NodeName, nodeId);

    bool wasSuccessful = false;

    do
    {
        try
        {
            // Invoke StartNodeTransitionAsync with the NodeStartDescription from above, which will
            start the target stopped node. Retry transient errors.
            await fc.TestManager.StartNodeTransitionAsync(description, TimeSpan.FromMinutes(1),
            CancellationToken.None).ConfigureAwait(false);
            wasSuccessful = true;
        }
        catch (OperationCanceledException oce)
        {
            Console.WriteLine("Caught exception '{0}'", oce);
        }
        catch (FabricTransientException fte)
        {
            Console.WriteLine("Caught exception '{0}'", fte);
        }

        await Task.Delay(TimeSpan.FromSeconds(5)).ConfigureAwait(false);
    }
    while (!wasSuccessful);

    // Now call StartNodeTransitionProgressAsync() until hte desired state is reached.
    await WaitForStateAsync(fc, guid, TestCommandProgressState.Completed).ConfigureAwait(false);
}

```

**Sample 3** - The following sample shows incorrect usage. This usage is incorrect because the *stopDurationInSeconds* it provides is greater than the allowed range. Since StartNodeTransitionAsync() will fail with a fatal error, the operation was not accepted, and the progress API should not be called. This sample uses some helper methods from the first sample.

```

static async Task StopNodeWithOutOfRangeDurationAsync(FabricClient fc, string nodeName)
{
    Node n = GetNodeInfo(fc, nodeName);

    Guid guid = Guid.NewGuid();

    // Use an out of range value for stopDurationInSeconds to demonstrate error
    NodeStopDescription description = new NodeStopDescription(guid, n.NodeName, n.NodeInstanceId,
99999);

    try
    {
        await fc.TestManager.StartNodeTransitionAsync(description, TimeSpan.FromMinutes(1),
CancellationToken.None).ConfigureAwait(false);
    }

    catch (FabricException e)
    {
        Console.WriteLine("Caught {0}", e);
        Console.WriteLine("ErrorCode {0}", e.ErrorCode);
        // Output:
        // Caught System.Fabric.FabricException: System.Runtime.InteropServices.COMException (-
2147017629)
        // StopDurationInSeconds is out of range ---> System.Runtime.InteropServices.COMException:
Exception from HRESULT: 0x80071C63
        // << Parts of exception omitted>>
        //
        // ErrorCode InvalidDuration
    }
}

```

**Sample 4** - The following sample shows the error information that will be returned from the Node Transition Progress API when the operation initiated by the Node Transition API is accepted, but fails later while executing. In the case, it fails because the Node Transition API attempts to start a node that does not exist. This sample uses some helper methods from the first sample.

```
static async Task StartNodeWithNonexistentNodeAsync(FabricClient fc)
{
    Guid guid = Guid.NewGuid();
    BigInteger nodeId = 12345;

    // Intentionally use a nonexistent node
    NodeStartDescription description = new NodeStartDescription(guid, "NonexistentNode",
nodeInstanceId);

    bool wasSuccessful = false;

    do
    {
        try
        {
            // Invoke StartNodeTransitionAsync with the NodeStartDescription from above, which will
start the target stopped node. Retry transient errors.
            await fc.TestManager.StartNodeTransitionAsync(description, TimeSpan.FromMinutes(1),
CancellationToken.None).ConfigureAwait(false);
            wasSuccessful = true;
        }
        catch (OperationCanceledException oce)
        {
            Console.WriteLine("Caught exception '{0}'", oce);
        }
        catch (FabricTransientException fte)
        {
            Console.WriteLine("Caught exception '{0}'", fte);
        }

        await Task.Delay(TimeSpan.FromSeconds(5)).ConfigureAwait(false);

    }
    while (!wasSuccessful);

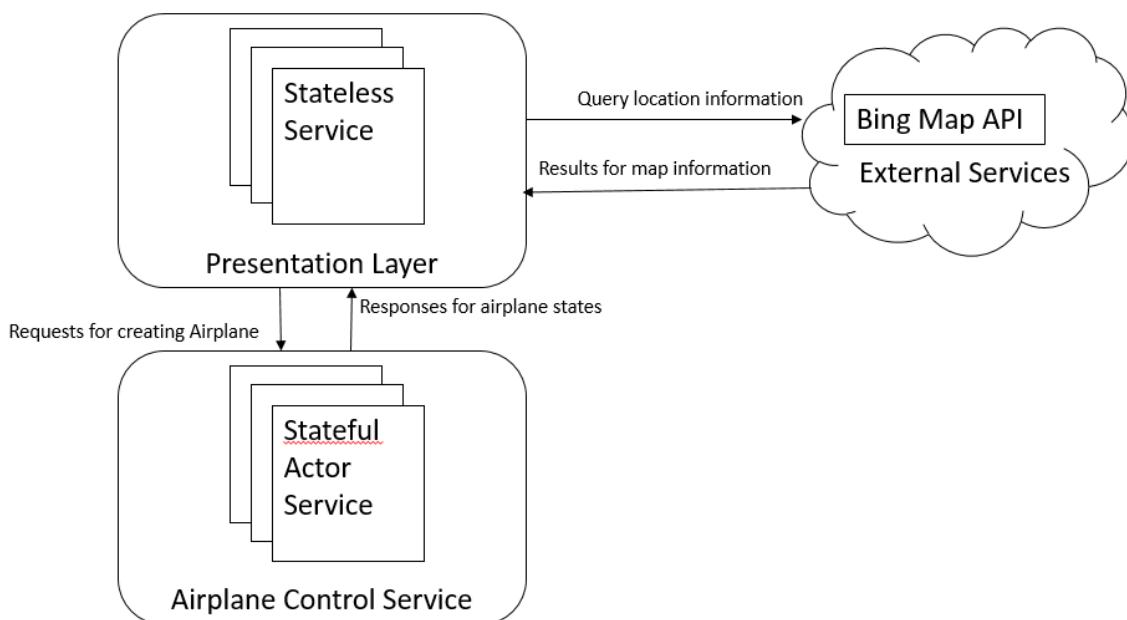
    // Now call StartNodeTransitionProgressAsync() until the desired state is reached. In this case,
it will end up in the Faulted state since the node does not exist.
    // When StartNodeTransitionProgressAsync()'s returned progress object has a State if Faulted,
inspect the progress object's Result.Exception.HResult to get the error code.
    // In this case, it will be NodeNotFound.
    await WaitForStateAsync(fc, guid, TestCommandProgressState.Faulted).ConfigureAwait(false);
}
```

# Load test your application by using Visual Studio Team Services

6/27/2017 • 5 min to read • [Edit Online](#)

This article shows how to use Microsoft Visual Studio load test features to stress test an application. It uses an Azure Service Fabric stateful service back end and a stateless service web front end. The example application used here is an airplane location simulator. You provide an airplane ID, departure time, and destination. The application's back end processes the requests, and the front end displays on a map the airplane that matches the criteria.

The following diagram illustrates the Service Fabric application that you'll be testing.



## Prerequisites

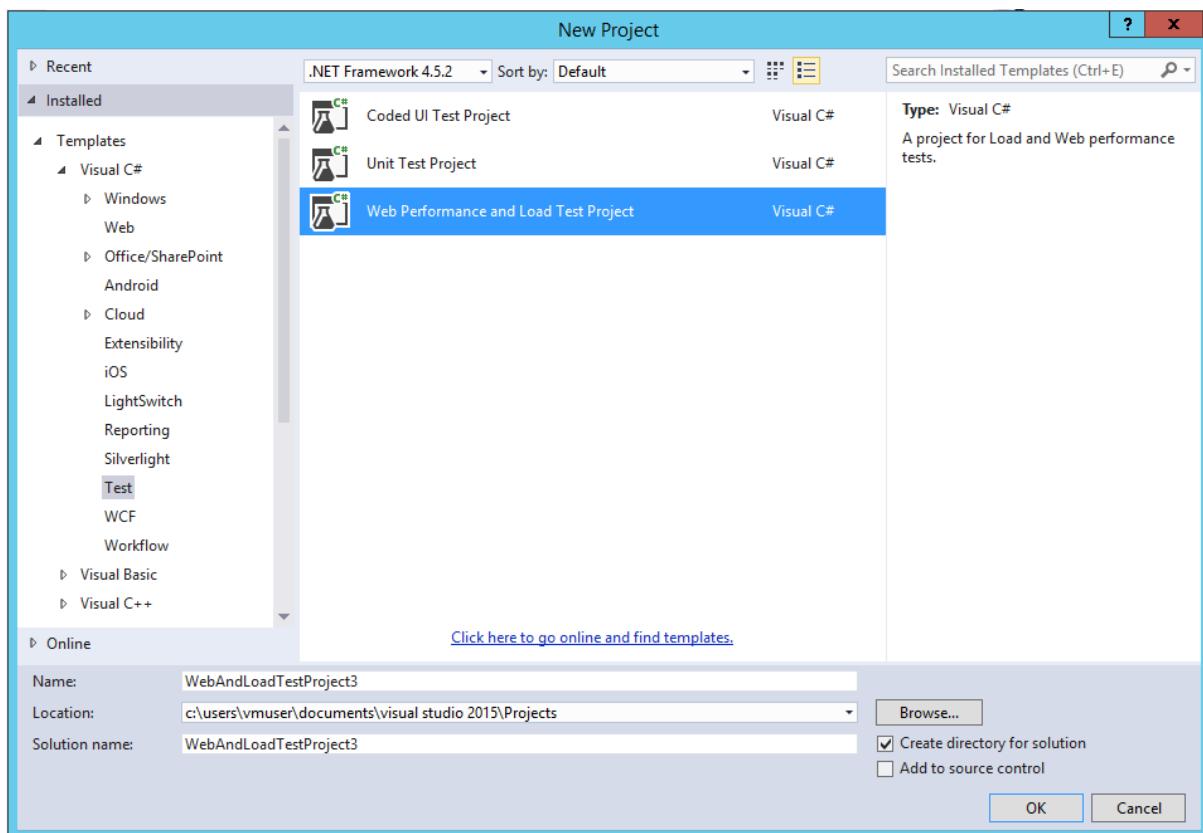
Before getting started, you need to do the following:

- Get a Visual Studio Team Services account. You can get one for free at [Visual Studio Team Services](#).
- Get and install Visual Studio 2013 or Visual Studio 2015. This article uses Visual Studio 2015 Enterprise edition, but Visual Studio 2013 and other editions should work similarly.
- Deploy your application to a staging environment. See [How to deploy applications to a remote cluster using Visual Studio](#) for information about this.
- Understand your application's usage pattern. This information is used to simulate the load pattern.
- Understand the goal for your load testing. This helps you interpret and analyze the load test results.

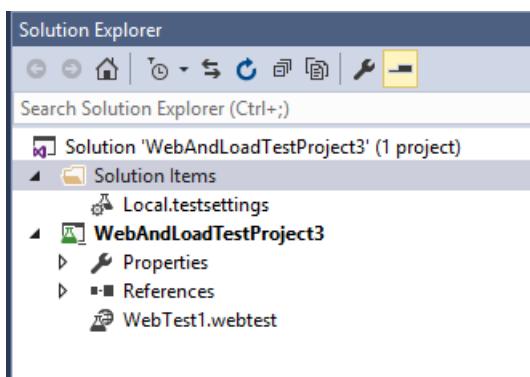
## Create and run the Web Performance and Load Test project

### Create a Web Performance and Load Test project

- Open Visual Studio 2015. Choose **File > New > Project** on the menu bar to open the **New Project** dialog box.
- Expand the **Visual C# node** and choose **Test > Web Performance and Load Test project**. Give the project a name and then choose the **OK** button.

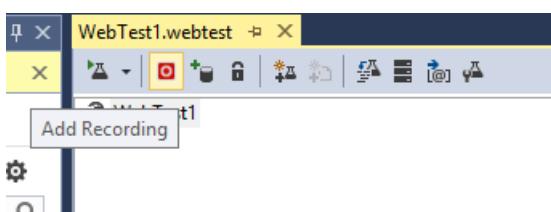


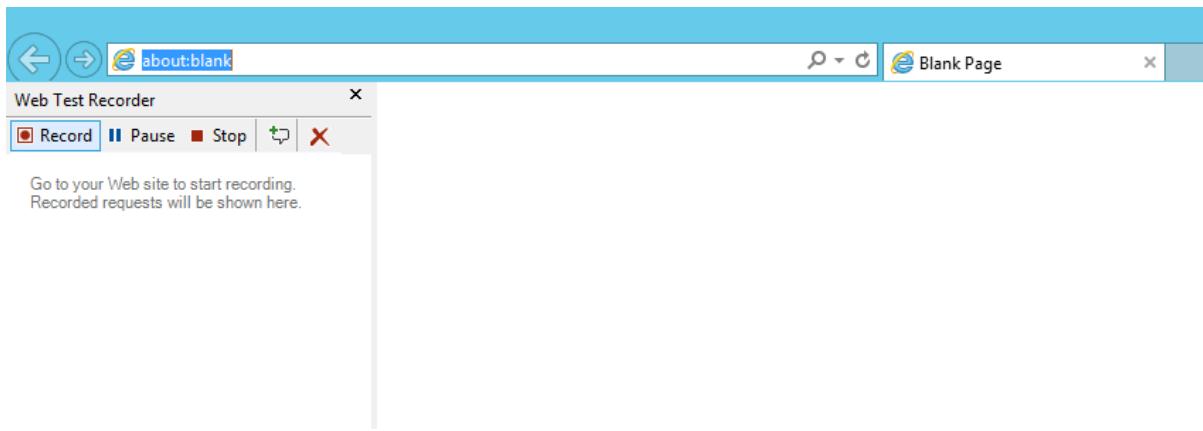
You should see a new Web Performance and Load Test project in Solution Explorer.



## Record a web performance test

1. Open the .webtest project.
2. Choose the **Add Recording** icon to start a recording session in your browser.





3. Browse to the Service Fabric application. The recording panel should show the web requests.

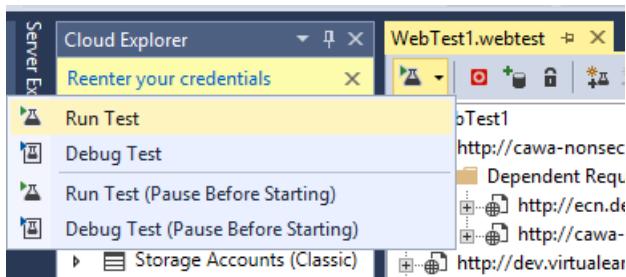
The screenshot shows the Web Test Recorder interface with a list of recorded web requests on the left. The requests are mostly to `http://cawa-nonsecure3.westus.cloudapp.azure.com:8085` and involve various API endpoints like `/api/airpla`. On the right, there is a map of the Olympic National Park area with a marker for "naimo". The "Aircraft ID" field contains "N489Y" and the "Departure airport" field is empty.

4. Perform a sequence of actions that you expect the users to perform. These actions are used as a pattern to generate the load.
5. When you're done, choose the **Stop** button to stop recording.

The screenshot shows the Web Test Recorder interface with a list of recorded web requests. A context menu is open over one of the requests, with the option "Stop this recording session" highlighted. The "Aircraft ID" field contains "1234" and the "Departure airport" field is empty.

The .webtest project in Visual Studio should have captured a series of requests. Dynamic parameters are replaced automatically. At this point, you can delete any extra, repeated dependency requests that are not part of your test scenario.

6. Save the project and then choose the **Run Test** command to run the web performance test locally and make sure everything works correctly.



## Parameterize the web performance test

You can parameterize the web performance test by converting it to a coded web performance test and then editing the code. As an alternative, you can bind the web performance test to a data list so that the test iterates through the data. See [Generate and run a coded web performance test](#) for details about how to convert the web performance test to a coded test. See [Add a data source to a web performance test](#) for information about how to bind data to a web performance test.

For this example, we'll convert the web performance test to a coded test so you can replace the airplane ID with a generated GUID and add more requests to send flights to different locations.

## Create a load test project

A load test project is composed of one or more scenarios described by the web performance test and unit test, along with additional specified load test settings. The following steps show how to create a load test project:

1. On the shortcut menu of your Web Performance and Load Test project, choose **Add > Load Test**. In the **Load Test** wizard, choose the **Next** button to configure the test settings.
2. In the **Load Pattern** section, choose whether you want a constant user load or a step load, which starts with a few users and increases the users over time.  
If you have a good estimate of the amount of user load and want to see how the current system performs, choose **Constant Load**. If your goal is to learn whether the system performs consistently under various loads, choose **Step Load**.
3. In the **Test Mix** section, choose the **Add** button and then select the test that you want to include in the load test. You can use the **Distribution** column to specify the percentage of total tests run for each test.
4. In the **Run Settings** section, specify the load test duration.

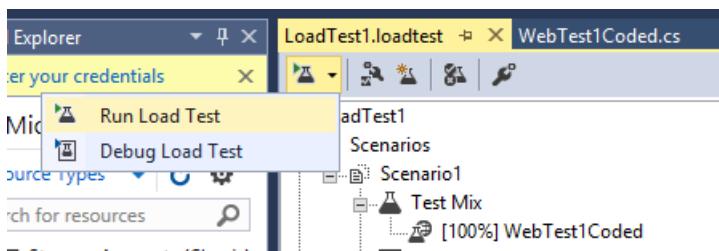
### NOTE

The **Test Iterations** option is available only when you run a load test locally using Visual Studio.

5. In the **Location** section of **Run Settings**, specify the location where load test requests are generated. The wizard may prompt you to log in to your Team Services account. Log in and then choose a geographic location. When you're done, choose the **Finish** button.
6. After the load test is created, open the .loadtest project and choose the current run setting, such as **Run Settings > Run Settings1 [Active]**. This opens the run settings in the **Properties** window.
7. In the **Results** section of the **Run Settings** properties window, the **Timing Details Storage** setting should have **None** as its default value. Change this value to **All Individual Details** to get more information on the load test results. See [Load Testing](#) for more information on how to connect to Visual Studio Team Services and run a load test.

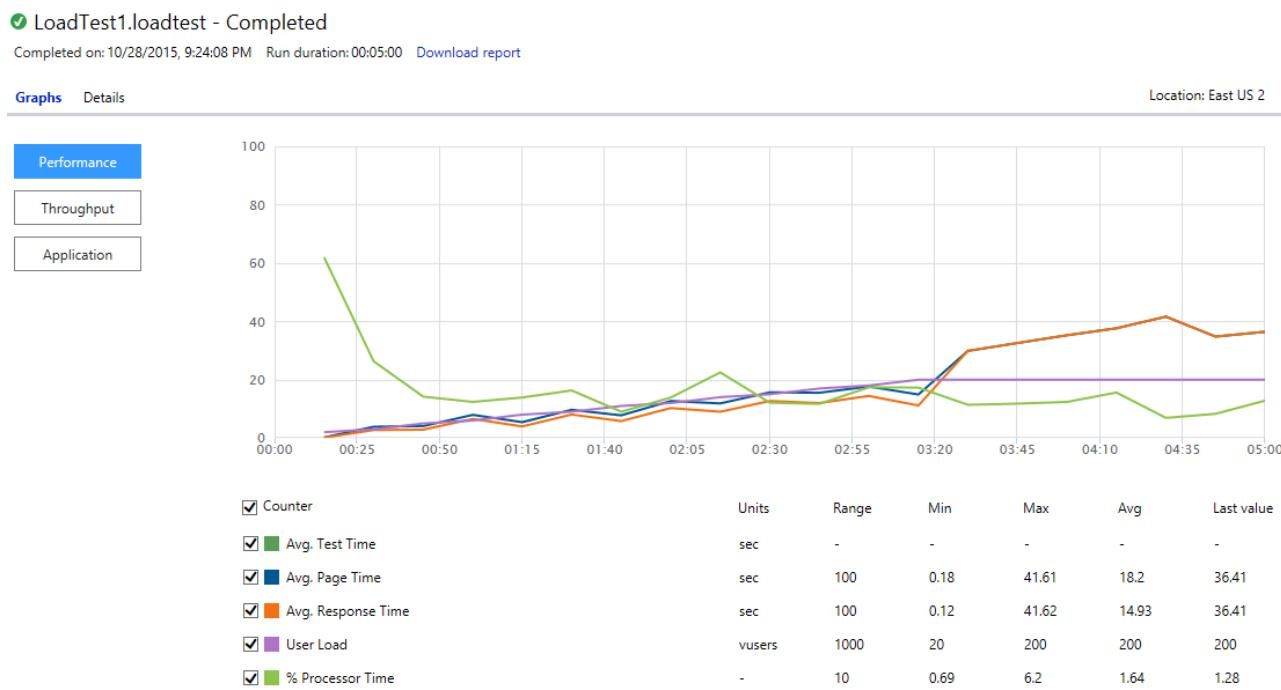
## Run the load test by using Visual Studio Team Services

Choose the **Run Load Test** command to start the test run.



## View and analyze the load test results

As the load test progresses, the performance information is graphed. You should see something similar to the following graph.



- Choose the **Download report** link near the top of the page. After the report is downloaded, choose the **View report** button.

On the **Graph** tab you can see graphs for various performance counters. On the **Summary** tab, the overall test results appear. The **Tables** tab shows the total number of passed and failed load tests.

- Choose the number links on the **Test > Failed** and the **Errors > Count** columns to see error details.

The **Detail** tab shows virtual user and test scenario information for failed requests. This data can be useful if the load test includes multiple scenarios.

See [Analyzing Load Test Results in the Graphs View of the Load Test Analyzer](#) for more information on viewing load test results.

## Automate your load test

Visual Studio Team Services Load Test provides APIs to help you manage load tests and analyze results in a Team Services account. See [Cloud Load Testing Rest APIs](#) for more information.

## Next steps

- [Monitoring and diagnosing services in a local machine development setup](#)

# Set up Service Fabric continuous integration and deployment with Visual Studio Team Services

6/27/2017 • 8 min to read • [Edit Online](#)

This article describes the steps to set up continuous integration and deployment for an Azure Service Fabric application by using Visual Studio Team Services (VSTS).

This document reflects the current procedure and is expected to change over time.

## Prerequisites

To get started, follow these steps:

1. Ensure that you have access to a Team Services account or [create one](#) yourself.
2. Ensure that you have access to a Team Services team project or [create one](#) yourself.
3. Ensure that you have a Service Fabric cluster to which you can deploy your application or create one using the [Azure portal](#), an [Azure Resource Manager template](#), or [Visual Studio](#).
4. Ensure that you have already created a Service Fabric Application (.sfproj) project. You must have a project that was created or upgraded with Service Fabric SDK 2.1 or higher (the .sfproj file should contain a ProjectVersion property value of 1.1 or higher).

### NOTE

Custom build agents are no longer required. Team Services hosted agents now come pre-installed with Service Fabric cluster management software, allowing for deployment of your applications directly from those agents.

## Configure and share your source files

The first thing you want to do is prepare a publish profile for use by the deployment process that executes within Team Services. The publish profile should be configured to target the cluster that you've previously prepared:

1. Choose a publish profile within your Application project that you want to use for your continuous integration workflow. Follow the [publish instructions](#) on how to publish an application to a remote cluster. You don't actually need to publish your application though. You can click the **Save** hyperlink in the publish dialog once you have configured things appropriately.
2. If you want your application to be upgraded for each deployment that occurs within Team Services, you want to configure the publish profile to enable upgrade. In the same publish dialog used in step 1, ensure that the **Upgrade the Application** checkbox is checked. Learn more about [configuring additional upgrade settings](#). Click the **Save** hyperlink to save the settings to the publish profile.
3. Ensure that you've saved your changes to the publish profile and cancel the publish dialog.
4. Now it's time to [share your Application project source files](#) with Team Services. Once your source files are accessible in Team Services, you can now move on to the next step of generating builds.

## Create a build definition

A Team Services build definition describes a workflow that is composed of a set of build steps that are executed sequentially. The goal of the build definition that you are creating is to produce a Service Fabric application package, and other artifacts, that can be used to deploy the application. Learn more about Team Services [build](#)

definitions.

### Create a definition from the build template

1. Open your team project in Visual Studio Team Services.
2. Select the **Build** tab.
3. Select the green + sign to create a new build definition.
4. In the dialog that opens, select **Azure Service Fabric Application** within the **Build** template category.
5. Select **Next**.
6. Select the repository and branch associated with your Service Fabric application.
7. Select the agent queue you wish to use. Hosted agents are supported.
8. Select **Create**.
9. Save the build definition and provide a name.
10. The following paragraph is a description of the build steps generated by the template:

BUILD STEP	DESCRIPTION
NuGet restore	Restores the NuGet packages for the solution.
Build solution *.sln	Builds the entire solution.
Build solution *.sfproj	Generates the Service Fabric application package that is used to deploy the application. The application package location is specified to be within the build's artifact directory.
Update Service Fabric App Versions	Updates the version values contained in the application package's manifest files to allow for upgrade support. See the <a href="#">task documentation page</a> for more information.
Copy Files	Copies the publish profile and application parameters files to the build's artifacts to be consumed for deployment.
Publish Artifact	Publishes the build's artifacts. Allows a release definition to consume the build's artifacts.

### Verify the default set of tasks

1. Verify the **Solution** input field for the **NuGet restore** and **Build solution** build steps. By default, these build steps execute upon all solution files that are contained in the associated repository. If you only want the build definition to operate on one of those solution files, you need to explicitly update the path to that file.
2. Verify the **Solution** input field for the **Package application** build step. By default, this build step assumes only one Service Fabric Application project (.sfproj) exists in the repository. If you have multiple such files in your repository and want to target only one of them for this build definition, you need to explicitly update the path to that file. If you want to package multiple Application projects in your repository, you need to create additional **Visual Studio Build** steps in the build definition that each target an Application project. You would then also need to update the **MSBuild Arguments** field for each of those build steps so that the package location is unique for each of them.
3. Verify the versioning behavior defined in the **Update Service Fabric App Versions** build step. By default, this build step appends the build number to all version values in the application package's manifest files. See the [task documentation page](#) for more information. This is useful for supporting upgrade of your application since each upgrade deployment requires different version values from the previous deployment. If you're not intending to use application upgrade in your workflow, you may consider disabling this build step. It must be disabled if your intention is to produce a build that can be used to overwrite an existing Service Fabric application. The deployment fails if the version of the application produced by the build does not match the

- version of the application in the cluster.
4. If your solution contains a .NET Core project, you must ensure that your build definition contains a build step that restores the dependencies:
    - a. Select **Add build step...**
    - b. Locate the **Command-Line** task within the Utility tab and click its Add button.
    - c. For the task's input fields, use the following values:
      - d. Tool: dotnet
      - e. Arguments: restore
      - f. Drag the task so that it is immediately after the **NuGet restore** step.
  5. Save any changes you've made to the build definition.

### Try it

Select **Queue Build** to manually start a build. Builds also triggers upon push or check-in. Once you've verified that the build is executing successfully, you can now move on to defining a release definition that deploys your application to a cluster.

## Create a release definition

A Team Services release definition describes a workflow that is composed of a set of tasks that are executed sequentially. The goal of the release definition that you are creating is to take an application package and deploy it to a cluster. When used together, the build definition and release definition can execute the entire workflow from starting with source files to ending with a running application in your cluster. Learn more about Team Services [release definitions](#).

### Create a definition from the release template

1. Open your project in Visual Studio Team Services.
2. Select the **Release** tab.
3. Select the green + sign to create a new release definition and select **Create release definition** in the menu.
4. In the dialog that opens, select **Azure Service Fabric Deployment** within the **Deployment** template category.
5. Select **Next**.
6. Select the build definition you want to use as the source of this release definition. The release definition references the artifacts that were produced by the selected build definition.
7. Check the **Continuous deployment** check box if you wish to have Team Services automatically create a new release and deploy the Service Fabric application whenever a build completes.
8. Select the agent queue you wish to use. Hosted agents are supported.
9. Select **Create**.
10. Edit the definition name by clicking the pencil icon at the top of the page.
11. Select the cluster to which your application should be deployed from the **Cluster Connection** input field of the task. The cluster connection provides the necessary information that allows the deployment task to connect to the cluster. If you do not yet have a cluster connection for your cluster, select the **Manage** hyperlink next to the field to add one. On the page that opens, perform the following steps:
  - a. Select **New Service Endpoint** and then select **Azure Service Fabric** from the menu.
  - b. Select the type of authentication being used by the cluster targeted by this endpoint.
  - c. Define a name for your connection in the **Connection Name** field. Typically, you would use the name of your cluster.
  - d. Define the client connection endpoint URL in the **Cluster Endpoint** field. Example:  
tcp://contoso.westus.cloudapp.azure.com:19000.
  - e. For Azure Active Directory credentials, define the credentials you want to use to connect to the cluster in the **Username** and **Password** fields.
  - f. For Certificate Based authentication, define the Base64 encoding of the client certificate file in the **Client**

**Certificate** field. See the help pop-up on that field for info on how to get that value. If your certificate is password-protected, define the password in the **Password** field.

- g. Confirm your changes by clicking **OK**. After navigating back to your release definition, click the refresh icon on the **Cluster Connection** field to see the endpoint you just added.

12. Save the release definition.

**NOTE**

Microsoft Accounts (for example, @hotmail.com or @outlook.com) are not supported with Azure Active Directory authentication.

The definition that is created consists of one task of type **Service Fabric Application Deployment**. See the [task documentation page](#) for more information about this task.

### Verify the template defaults

1. Verify the **Publish Profile** input field for the **Deploy Service Fabric Application** task. By default, this field references a publish profile named Cloud.xml contained in the build's artifacts. If you want to reference a different publish profile or if the build contains multiple application packages in its artifacts, you need to update the path appropriately.
2. Verify the **Application Package** input field for the **Deploy Service Fabric Application** task. By default, this field references the default application package path used in the build definition template. If you've modified the default application package path in the build definition, you need to update the path appropriately here as well.

### Try it

Select **Create Release** from the **Release** button menu to manually create a release. In the dialog that opens, select the build that you want to base the release on and then click **Create**. If you enabled continuous deployment, releases will be created automatically when the associated build definition completes a build.

## Next steps

To learn more about continuous integration with Service Fabric applications, read the following articles:

- [Team Services documentation home](#)
- [Build management in Team Services](#)
- [Release management in Team Services](#)

# Use Jenkins to build and deploy your Linux Java application

10/30/2017 • 7 min to read • [Edit Online](#)

Jenkins is a popular tool for continuous integration and deployment of your apps. Here's how to build and deploy your Azure Service Fabric application by using Jenkins.

## General prerequisites

- Have Git installed locally. You can install the appropriate Git version from [the Git downloads page](#), based on your operating system. If you are new to Git, learn more about it from the [Git documentation](#).
- Have the Service Fabric Jenkins plug-in handy. You can download it from [Service Fabric downloads](#).

## Set up Jenkins inside a Service Fabric cluster

You can set up Jenkins either inside or outside a Service Fabric cluster. The following sections show how to set it up inside a cluster while using an Azure storage account to save the state of the container instance.

### Prerequisites

1. Have a Service Fabric Linux cluster ready. A Service Fabric cluster created from the Azure portal already has Docker installed. If you are running the cluster locally, check if Docker is installed by using the command `docker info`. If it is not installed, install it accordingly by using the following commands:

```
sudo apt-get install wget
wget -qO- https://get.docker.io/ | sh
```

#### NOTE

Ensure that the 8081 port is specified as a custom endpoint on the cluster.

2. Clone the application, by using the following steps:

```
git clone https://github.com/Azure-Samples/service-fabric-java-getting-started.git
cd service-fabric-java-getting-started/Services/JenkinsDocker/
```

3. Persist the state of the Jenkins container in a file-share:

- Create an Azure storage account in the **same region** as your cluster with a name such as `sfjenkinsstorage1`.
- Create a **File Share** under the storage Account with a name such as `sfjenkins`.
- Click on **Connect** for the file-share and note the values it displays under **Connecting from Linux**, the value should look similar to the one below:  
`sh sudo mount -t cifs //sfjenkinsstorage1.file.core.windows.net/sfjenkins [mount point] -o vers=3.0,username=sfjenkinsstorage1,password=<storage_key>,dir_mode=0777,file_mode=0777`

#### NOTE

To mount cifs shares, you need to have the cifs-utils package installed in the cluster nodes.

1. Update the placeholder values in the `setupentrypoint.sh` script with the azure-storage details from step 3.

```
vi JenkinsSF/JenkinsOnSF/Code/setupentrypoint.sh
```

- Replace `[REMOTE_FILE_SHARE_LOCATION]` with the value `//sfjenkinsstorage1.file.core.windows.net/sfjenkins` from the output of the connect in step 3 above.
- Replace `[FILE_SHARE_CONNECT_OPTIONS_STRING]` with the value  
`vers=3.0,username=sfjenkinsstorage1,password=GB2NPUCQY9LDGeG9Bci5dJV91T6SrA70xrYBUsFHyueR62viMrC6NIzyQLCKNz0o7pepGfGY+vTa9gxzEtFZHw==,dir_mode=0777,file_mode=0777` from step 3 above.

2. Connect to the cluster and install the container application.

```
sfctl cluster select --endpoint http://PublicIPorFQDN:19080 # cluster connect command
bash Scripts/install.sh
```

This installs a Jenkins container on the cluster, and can be monitored by using the Service Fabric Explorer.

#### NOTE

It may take a couple of minutes for the Jenkins image to be downloaded on the cluster.

## Steps

- From your browser, go to <http://PublicIPorFQDN:8081>. It provides the path of the initial admin password required to sign in.
- Look at the Service Fabric Explorer to determine on which node the Jenkins container is running. Secure Shell (SSH) sign in to this node.  

```
sh ssh user@PublicIPorFQDN -p [port]
```
- Get the container instance ID by using `docker ps -a`.
- Secure Shell (SSH) sign in to the container, and paste the path you were shown on the Jenkins portal. For example, if in the portal it shows the path `PATH_TO_INITIAL_ADMIN_PASSWORD`, run the following:  

```
docker exec -t -i [first-four-digits-of-container-ID] /bin/bash # This takes you inside Docker shell
```

```
cat PATH_TO_INITIAL_ADMIN_PASSWORD # This displays the password value
```
- On the Jenkins Getting Started page, choose the Select plugins to install option, select the **None** checkbox, and click install.
- Create a user or select to continue as an admin.

## Set up Jenkins outside a Service Fabric cluster

You can set up Jenkins either inside or outside of a Service Fabric cluster. The following sections show how to set it up outside a cluster.

### Prerequisites

You need to have Docker installed. The following commands can be used to install Docker from the terminal:

```
sudo apt-get install wget
wget -qO- https://get.docker.io/ | sh
```

Now when you run `docker info` in the terminal, you should see in the output that the Docker service is running.

### Steps

- Pull the Service Fabric Jenkins container image: `docker pull raunakpandya/jenkins:v1`
- Run the container image: `docker run -itd -p 8080:8080 raunakpandya/jenkins:v1`
- Get the ID of the container image instance. You can list all the Docker containers with the command `docker ps -a`
- Sign in to the Jenkins portal by using the following steps:
  - `sh docker exec [first-four-digits-of-container-ID] cat /var/jenkins_home/secrets/initialAdminPassword` If container ID is 2d24a73b5964, use 2d24.
  - This password is required for signing in to the Jenkins dashboard from portal, which is <http://<HOST-IP>:8080>
  - After you sign in for the first time, you can create your own user account and use that for future purposes, or you can continue to use the administrator account. After you create a user, you need to continue with that.
- Set up GitHub to work with Jenkins, by using the steps mentioned in [Generating a new SSH key and adding it to the SSH agent](#).
  - Use the instructions provided by GitHub to generate the SSH key, and to add the SSH key to the GitHub account that is hosting the repository.
  - Run the commands mentioned in the preceding link in the Jenkins Docker shell (and not on your host).
    - To sign in to the Jenkins shell from your host, use the following commands:  

```
docker exec -t -i [first-four-digits-of-container-ID] /bin/bash
```

Ensure that the cluster or machine where the Jenkins container image is hosted has a public-facing IP. This enables the Jenkins instance to receive notifications from GitHub.

## Install the Service Fabric Jenkins plug-in from the portal

- Go to <http://PublicIPorFQDN:8081>
- From the Jenkins dashboard, select **Manage Jenkins > Manage Plugins > Advanced**. Here, you can upload a plug-in. Select **Choose file**, and then select the **serviceFabric.hpi** file, which you downloaded under prerequisites or can download [here](#). When you select **Upload**, Jenkins automatically installs the plug-in. Allow a restart if requested.

## Create and configure a Jenkins job

- Create a **new item** from dashboard.
- Enter an item name (for example, **MyJob**). Select **free-style project**, and click **OK**.
- Go the job page, and click **Configure**.
  - In the general section, select the checkbox for **GitHub project**, and specify your GitHub project URL. This URL hosts the Service Fabric Java application that you want to integrate with the Jenkins continuous integration, continuous deployment (CI/CD) flow (for example, <https://github.com/sayantancs/SFJenkins> ).
  - Under the **Source Code Management** section, select **Git**. Specify the repository URL that hosts the Service Fabric Java application that you want to integrate with the Jenkins CI/CD flow (for example, <https://github.com/sayantancs/SFJenkins.git> ). Also, you can specify here which branch to build (for example, **/master**).
- Configure your **GitHub** (which is hosting the repository) so that it is able to talk to Jenkins. Use the following steps:

- a. Go to your GitHub repository page. Go to **Settings > Integrations and Services**.
- b. Select **Add Service**, type **Jenkins**, and select the **Jenkins-GitHub plugin**.
- c. Enter your Jenkins webhook URL (by default, it should be `http://<PublicIPorFQDN>:8081/github-webhook/`). Click **add/update service**.
- d. A test event is sent to your Jenkins instance. You should see a green check by the webhook in GitHub, and your project will build.
- e. Under the **Build Triggers** section, select which build option you want. For this example, you want to trigger a build whenever some push to the repository happens. So you select **GitHub hook trigger for GITScm polling**. (Previously, this option was called **Build when a change is pushed to GitHub**.)
- f. Under the **Build section**, from the drop-down **Add build step**, select the option **Invoke Gradle Script**. In the widget that comes open the advanced menu, specify the path to **Root build script** for your application. It picks up build.gradle from the path specified and works accordingly. If you create a project named `MyActor` (using the Eclipse plug-in or Yeoman generator), the root build script should contain  `${WORKSPACE}/MyActor`. See the following screenshot for an example of what this looks like:

### Build

**Invoke Gradle script**

Invoke Gradle

Gradle Version: (Default)

Use Gradle Wrapper

Build step description:

Switches:

Tasks:

Root Build script: \${WORKSPACE}/MyActor

Build File:

Specify Gradle build file to run. Also, [some environment variables are available to the build script](#)

Force GRADLE\_USER\_HOME to use workspace

Pass job parameters as Gradle properties

**Add build step ▾**

- g. From the **Post-Build Actions** drop-down, select **Deploy Service Fabric Project**. Here you need to provide cluster details where the Jenkins compiled Service Fabric application would be deployed. You can also provide additional application details used to deploy the application. See the following screenshot for an example of what this looks like:

### Post-build Actions

**Deploy Service Fabric Project**

Select cluster type: Unsecured

Cluster public IP: 40.121.92.211

Application name: fabric:/MyActor

Application type: MyActorType

Path to Application Manifest: MyActor/MyActor/Applicati

Client Key (Secure only):

Client Cert (Secure only):

Ca-Chain (Secure Only):

**Add post-build action ▾**

#### NOTE

The cluster here could be same as the one hosting the Jenkins container application, in case you are using Service Fabric to deploy the Jenkins container image.

## Next steps

GitHub and Jenkins are now configured. Consider making some sample change in your `MyActor` project in the repository example, <https://github.com/sayantancs/SFJenkins>. Push your changes to a remote `master` branch (or any branch that you have configured to work with). This triggers the Jenkins job, `MyJob`, that you configured. It fetches the changes from GitHub, builds them, and deploys the application to the cluster endpoint you specified in post-build actions.

# Create your first Service Fabric cluster on Azure

11/2/2017 • 9 min to read • [Edit Online](#)

An [Azure Service Fabric cluster](#) is a network-connected set of virtual or physical machines into which your microservices are deployed and managed. This quickstart helps you to create a five-node cluster, running on either Windows or Linux, through [Azure PowerShell](#) or the [Azure portal](#) in just a few minutes. You can also use Azure CLI for this task.

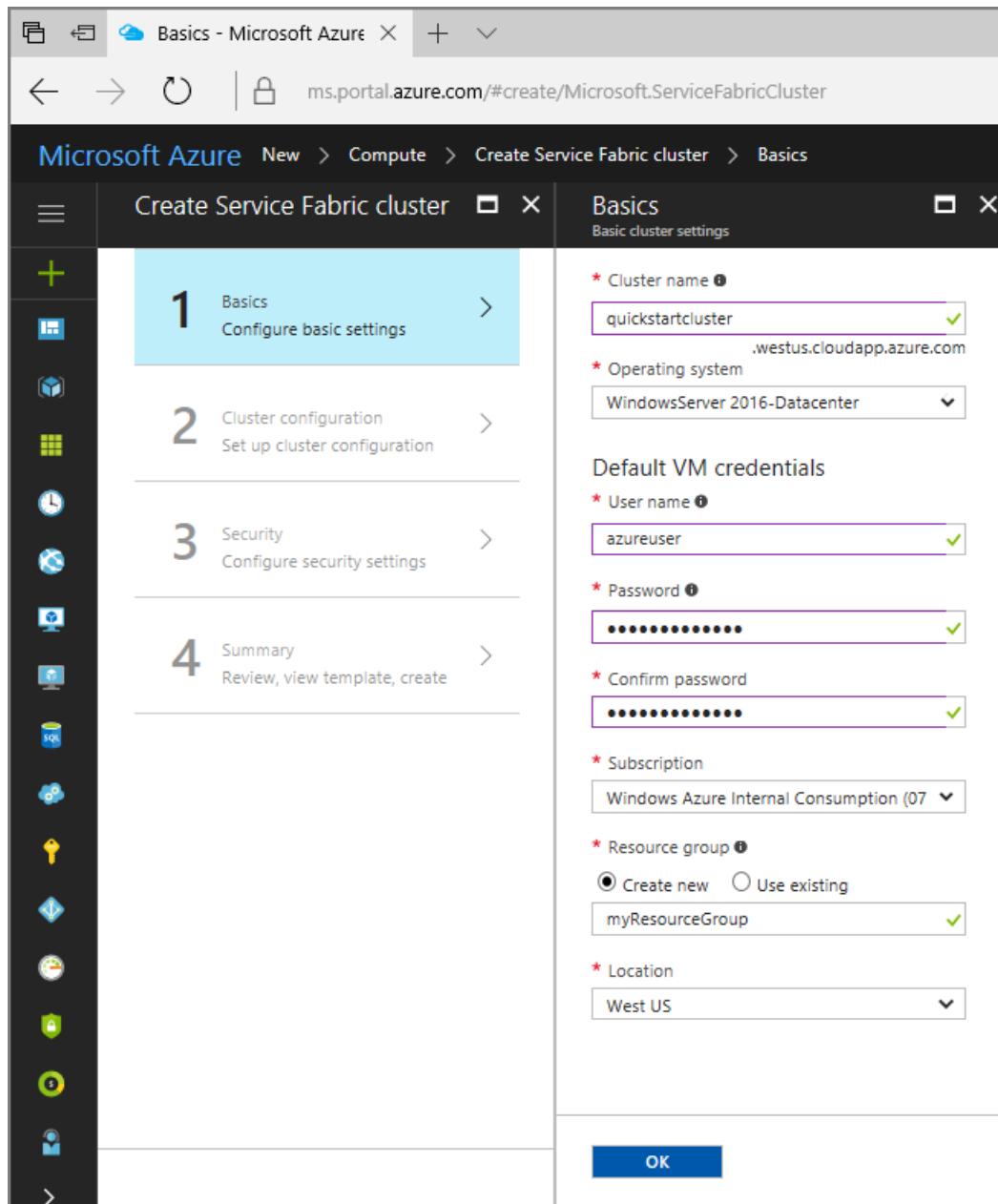
If you don't have an Azure subscription, create a [free account](#) before you begin.

## Use the Azure portal

Sign in to the Azure portal at <http://portal.azure.com>.

### Create the cluster

1. On the upper-left corner of the Azure portal, select **New**.
2. Search for **Service Fabric**, and select **Service Fabric Cluster** from the returned results. Then select **Create**.
3. Fill out the Service Fabric **Basics** form. For **Operating system**, select the version of Windows or Linux you want the cluster nodes to run. The user name and password entered here is used to sign in to the virtual machine. For **Resource group**, create a new one. A resource group is a logical container into which Azure resources are created and collectively managed. When you are done, select **OK**.



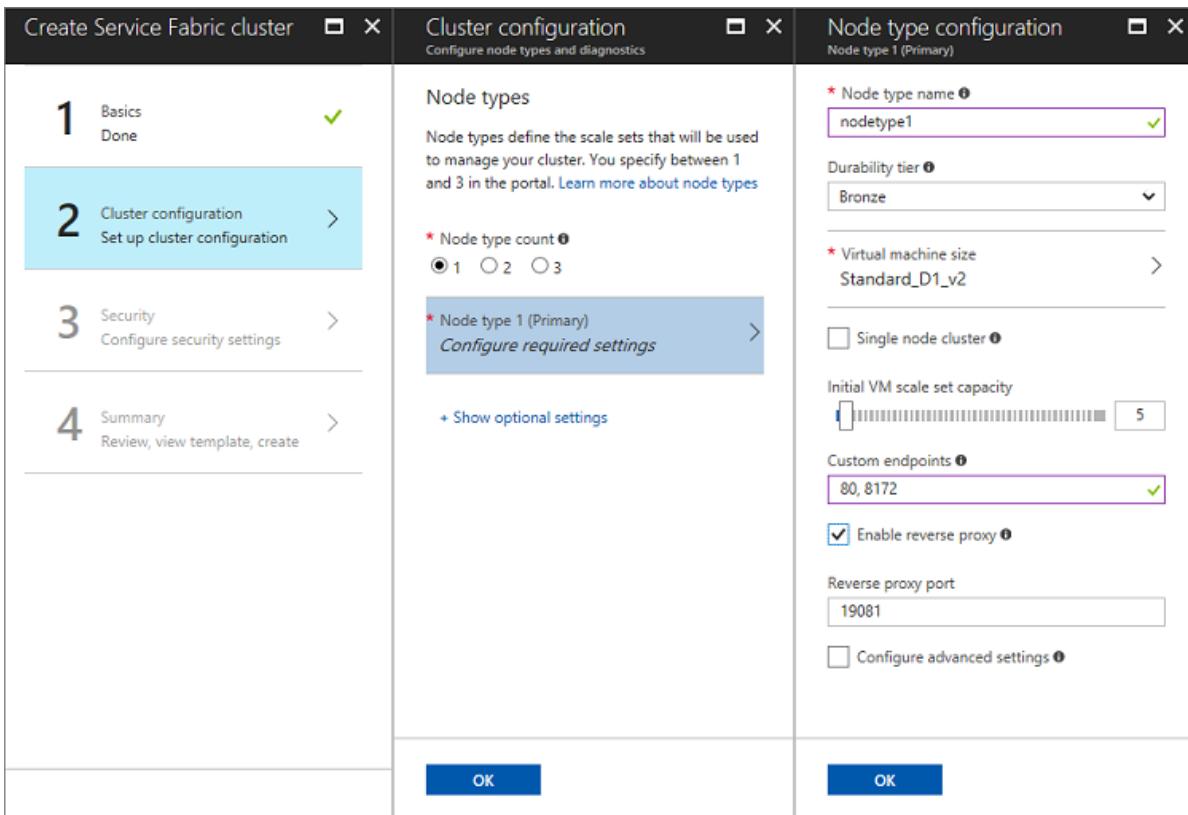
4. Fill out the **Cluster configuration** form. For **Node type count**, enter **1**.
5. Select **Node type 1 (Primary)**, and fill out the **Node type configuration** form. Enter a node type name, and set the **Durability tier** to **Bronze**. Then select a VM size.

Node types define the VM size, number of VMs, custom endpoints, and other settings for the VMs of that type. Each node type defined is set up as a separate virtual machine scale set, which is used to deploy and manage virtual machines as a set. Each node type can be scaled up or down independently, have different sets of ports open, and have different capacity metrics. The first, or primary, node type is where Service Fabric system services are hosted. This node type must have five or more VMs.

For any production deployment, [capacity planning](#) is an important step. For this quickstart, however, you aren't running applications, so select a *DS1\_v2 Standard* VM size. Select **Silver** for the [reliability tier](#), and specify an initial virtual machine scale set capacity of 5.

Custom endpoints open up ports in Azure Load Balancer so that you can connect with applications running on the cluster. Enter **80, 8172** to open up ports 80 and 8172.

Do not select the **Configure advanced settings** box, which is used for customizing TCP/HTTP management endpoints, application port ranges, [placement constraints](#), and [capacity properties](#).



Select **OK**.

- In the **Cluster configuration** form, set **Diagnostics** to **On**. For this quickstart, you do not need to enter any [fabric setting](#) properties. In **Fabric version**, select **Automatic** upgrade mode so that Microsoft automatically updates the version of the fabric code running the cluster. Set the mode to **Manual** if you want to [choose a supported version](#) to upgrade to.

Select **OK**.

- Fill out the **Security** form. For this quickstart, select **Unsecure**. Note that in general, you should create a secure cluster for production workloads. Anyone can anonymously connect to an unsecure cluster and perform management operations.

Service Fabric uses certificates to provide authentication and encryption to secure various aspects of a cluster and its applications. For more information, see [Service Fabric cluster security scenarios](#). To enable user authentication by using Azure Active Directory, or to set up certificates for application security, see [Create a cluster from a Resource Manager template](#).

Select **OK**.

- Review the summary. If you'd like to download an Azure Resource Manager template built from the settings you entered, select **Download template and parameters**. Select **Create** to create the cluster.

You can see the creation progress in the notifications. (Select the "Bell" icon near the status bar at the upper right of your screen.) If you selected **Pin to Startboard** while creating the cluster, you see **Deploying Service Fabric Cluster** pinned to the **Start** board.

### Connect to the cluster by using PowerShell

Verify that the cluster is running by connecting through PowerShell. The Service Fabric PowerShell module is installed with the [Service Fabric SDK](#). The `Connect-ServiceFabricCluster` cmdlet establishes a connection to the cluster.

```
Connect-ServiceFabricCluster -ConnectionEndpoint quickstartcluster.westus2.cloudapp.azure.com:19000
```

See [Connect to a secure cluster](#) for other examples of connecting to a cluster. After you connect to the cluster, use the [Get-ServiceFabricNode](#) cmdlet to display a list of nodes in the cluster, and status information for each node. The **HealthState** should be **OK** for each node.

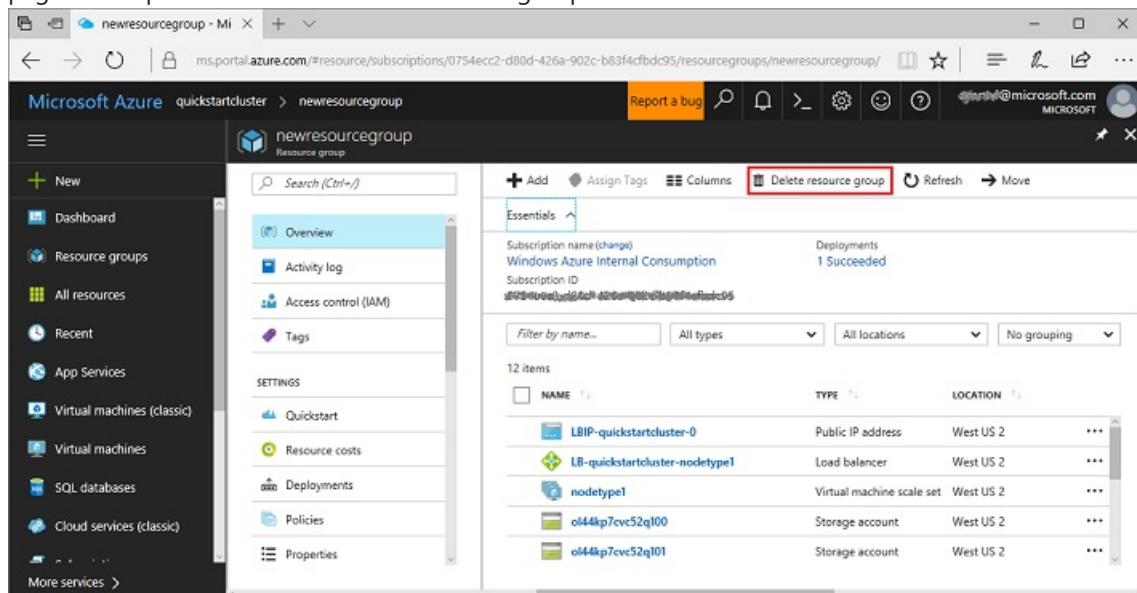
PS C:\Users\sfuser> Get-ServiceFabricNode  Format-Table							
NodeDeactivationInfo	NodeName	IpAddressOrFQDN	NodeType	CodeVersion	ConfigVersion	NodeStatus	NodeUpTime
NodeDownTime	HealthState						
00:00:00	_nodetype1_2	10.0.0.6	nodetype1	5.7.198.9494	1	Up	03:00:38
	Ok						
00:00:00	_nodetype1_1	10.0.0.5	nodetype1	5.7.198.9494	1	Up	03:00:38
	Ok						
00:00:00	_nodetype1_0	10.0.0.4	nodetype1	5.7.198.9494	1	Up	03:00:38
	Ok						
00:00:00	_nodetype1_4	10.0.0.8	nodetype1	5.7.198.9494	1	Up	03:00:38
	Ok						
00:00:00	_nodetype1_3	10.0.0.7	nodetype1	5.7.198.9494	1	Up	03:00:38
	Ok						

## Remove the cluster

A Service Fabric cluster is made up of other Azure resources, in addition to the cluster resource itself. To completely delete a Service Fabric cluster, you also need to delete all the resources it is made of. The simplest way to delete the cluster and all the resources it consumes is to delete the resource group. For other ways to delete a cluster or to delete some (but not all) of the resources in a resource group, see [Delete a cluster](#).

Delete a resource group in the Azure portal:

1. Browse to the Service Fabric cluster you want to delete.
2. On the cluster essentials page, select the **Resource Group** name.
3. On the **Resource Group Essentials** page, select **Delete resource group**. Then follow the instructions on that page to complete the deletion of the resource group.



## Use Azure PowerShell

Another way to create the cluster is to use PowerShell. Here's how:

1. Download the [Azure PowerShell module version 4.0 or higher](#) on your computer.
2. Run the [New-AzureRmServiceFabricCluster](#) cmdlet to create a five-node Service Fabric cluster, secured with an X.509 certificate. The command creates a self-signed certificate, and uploads it to a new key vault. The

certificate is also copied to a local directory. Set the -OS parameter to choose the version of Windows or Linux that runs on the cluster nodes. Customize the parameters as needed.

```
#Provide the subscription Id
$subscriptionId = 'yourSubscriptionId'

# Certificate variables.
$certypwd="Password#1234" | ConvertTo-SecureString -AsPlainText -Force
$certfolder="c:\mycertificates\"

# Variables for VM admin.
$adminuser="vmadmin"
$adminpwd="Password#1234" | ConvertTo-SecureString -AsPlainText -Force

# Variables for common values
$clusterloc="SouthCentralUS"
$clusternname = "mysfcluster"
$groupname="mysfclustergroup"
$vmsku = "Standard_D2_v2"
$vaultname = "mykeyvault"
$subname="$clusternname.$clusterloc.cloudapp.azure.com"

# Set the number of cluster nodes. Possible values: 1, 3-99
$clustersize=5

# Set the context to the subscription ID where the cluster will be created
Login-AzureRmAccount
Get-AzureRmSubscription
Select-AzureRmSubscription -SubscriptionId $subscriptionId

# Create the Service Fabric cluster.
New-AzureRmServiceFabricCluster -Name $clusternname -ResourceGroupName $groupname -Location $clusterloc ` 
-ClusterSize $clustersize -VmUserName $adminuser -VmPassword $adminpwd -CertificateSubjectName $subname ` 
-CertificatePassword $certypwd -CertificateOutputFolder $certfolder ` 
-OS WindowsServer2016DatacenterwithContainers -VmSku $vmsku -KeyVaultName $vaultname
```

The command can take anywhere from 10 minutes to 30 minutes to complete. The output has information about the certificate, the key vault where it was uploaded to, and the local folder where the certificate is copied.

3. Copy the entire output and save to a text file (you will refer to it later). Make a note of the following information from the output:

- CertificateSavedLocalPath
- CertificateThumbprint
- ManagementEndpoint
- ClientConnectionEndpointPort

### Install the certificate on your local machine

To connect to the cluster, install the certificate into the Personal (My) store of the current user.

Run the following:

```
$certypwd="Password#1234" | ConvertTo-SecureString -AsPlainText -Force
Import-PfxCertificate -Exportable -CertStoreLocation Cert:\CurrentUser\My ` 
-FilePath C:\mycertificates\<certificatename>.pfx ` 
-Password $certypwd
```

You are now ready to connect to your secure cluster.

## Connect to a secure cluster

Run the [Connect-ServiceFabricCluster](#) cmdlet to connect to a secure cluster. The certificate details must match a certificate that was used to set up the cluster.

```
Connect-ServiceFabricCluster -ConnectionEndpoint <ManagementEndpoint>:19000 `  
    -KeepAliveIntervalInSec 10 `  
    -X509Credential -ServerCertThumbprint <CertificateThumbprint> `  
    -FindType FindByThumbprint -FindValue <CertificateThumbprint> `  
    -StoreLocation CurrentUser -StoreName My
```

The following example shows sample parameters:

```
Connect-ServiceFabricCluster -ConnectionEndpoint mycluster.southcentralus.cloudapp.azure.com:19000 `  
    -KeepAliveIntervalInSec 10 `  
    -X509Credential -ServerCertThumbprint C4C1E541AD512B8065280292A8BA6079C3F26F10 `  
    -FindType FindByThumbprint -FindValue C4C1E541AD512B8065280292A8BA6079C3F26F10 `  
    -StoreLocation CurrentUser -StoreName My
```

Run the following command to check that you are connected and the cluster is healthy.

```
Get-ServiceFabricClusterHealth
```

## Remove the cluster

A cluster is made up of other Azure resources in addition to the cluster resource itself. The simplest way to delete the cluster and all the resources it consumes is to delete the resource group.

```
$groupname="mysfclustergroup"  
Remove-AzureRmResourceGroup -Name $groupname -Force
```

## Use Azure CLI

Another way to create the cluster is to use CLI. Here's how:

1. Install [Azure CLI 2.0](#) on your computer.
2. Sign in to Azure and select the subscription you want to create the cluster in.  

```
azurecli az login az account set --subscription <GUID>
```
3. Run the `az sf cluster create` command to create a five-node Service Fabric cluster, secured with an X.509 certificate. The command creates a self-signed certificate, and uploads it to a new key vault. The certificate is also copied to a local directory. Set the `-os` parameter to choose the version of Windows or Linux that runs on the cluster nodes. Customize the parameters as needed.

```

#!/bin/bash

# Variables
ResourceGroupName="aztestclustergroup"
ClusterName="aztestcluster"
Location="southcentralus"
Password="q6D7nN%6ck@6"
Subject="aztestcluster.southcentralus.cloudapp.azure.com"
VaultName="aztestkeyvault"
VaultGroupName="testvaultgroup"
VmPassword="Mypa$$word!321"
VmUserName="sfadminuser"

# Create resource groups
az group create --name $ResourceGroupName --location $Location
az group create --name $VaultGroupName --location $Location

# Create secure five node Linux cluster. Creates a key vault in a resource group
# and creates a certificate in the key vault. The certificate's subject name must match
# the domain that you use to access the Service Fabric cluster. The certificate is downloaded locally.
az sf cluster create --resource-group $ResourceGroupName --location $Location --certificate-output-
folder . \
--certificate-password $Password --certificate-subject-name $Subject --cluster-name $ClusterName \
--cluster-size 5 --os UbuntuServer1604 --vault-name $VaultName --vault-resource-group
$VaultGroupName \
--vm-password $VmPassword --vm-user-name $VmUserName

```

## Connect to the cluster

Run the following CLI command to connect to the cluster by using the certificate. When you are using a client certificate for authentication, ensure that the certificate details match a certificate deployed to the cluster nodes. Use the `--no-verify` option for a self-signed certificate.

```

az sf cluster select --endpoint https://aztestcluster.southcentralus.cloudapp.azure.com:19080 --pem
./linuxcluster201709161647.pem --no-verify

```

Run the following command to check that you are connected and the cluster is healthy.

```
az sf cluster health
```

## Connect to the nodes directly

To connect to the nodes in a Linux cluster, you can use the secure shell (SSH) command. Specify a port number from 3389 onward. For example, for the five node cluster created earlier, the commands would be as follows:

```

ssh sfadminuser@aztestcluster.southcentralus.cloudapp.azure.com -p 3389
ssh sfadminuser@aztestcluster.southcentralus.cloudapp.azure.com -p 3390
ssh sfadminuser@aztestcluster.southcentralus.cloudapp.azure.com -p 3391
ssh sfadminuser@aztestcluster.southcentralus.cloudapp.azure.com -p 3392
ssh sfadminuser@aztestcluster.southcentralus.cloudapp.azure.com -p 3393

```

## Remove the cluster

A cluster is made up of other Azure resources in addition to the cluster resource itself. The simplest way to delete the cluster and all the resources it consumes is to delete the resource group.

```

ResourceGroupName = "aztestclustergroup"
az group delete --name $ResourceGroupName

```

## Next steps

Now that you have set up a development cluster, try the following:

- [Visualize your cluster with Service Fabric Explorer](#)
- [Remove a cluster](#)
- [Deploy apps using PowerShell](#)
- [Deploy apps using CLI](#)

# Create a Service Fabric cluster in Azure using the Azure portal

10/18/2017 • 13 min to read • [Edit Online](#)

This is a step-by-step guide that walks you through the steps of setting up a secure Service Fabric cluster in Azure using the Azure portal. This guide walks you through the following steps:

- Set up Key Vault to manage keys for cluster security.
- Create a secured cluster in Azure through the Azure portal.
- Authenticate administrators using certificates.

## NOTE

For more advanced security options, such as user authentication with Azure Active Directory and setting up certificates for application security, [create your cluster using Azure Resource Manager](#).

A secure cluster is a cluster that prevents unauthorized access to management operations, which includes deploying, upgrading, and deleting applications, services, and the data they contain. An unsecure cluster is a cluster that anyone can connect to at any time and perform management operations. Although it is possible to create an unsecure cluster, it is **highly recommended to create a secure cluster**. An unsecure cluster **cannot be secured later** - a new cluster must be created.

The concepts are the same for creating secure clusters, whether the clusters are Linux clusters or Windows clusters. For more information and helper scripts for creating secure Linux clusters, please see [Creating secure clusters on Linux](#). The parameters obtained by the helper script provided can be input directly into the portal as described in the section [Create a cluster in the Azure portal](#).

## Configure Key Vault

### Log in to Azure

This guide uses [Azure PowerShell](#). When starting a new PowerShell session, log in to your Azure account and select your subscription before executing Azure commands.

Log in to your azure account:

```
Login-AzureRmAccount
```

Select your subscription:

```
Get-AzureRmSubscription  
Set-AzureRmContext -SubscriptionId <guid>
```

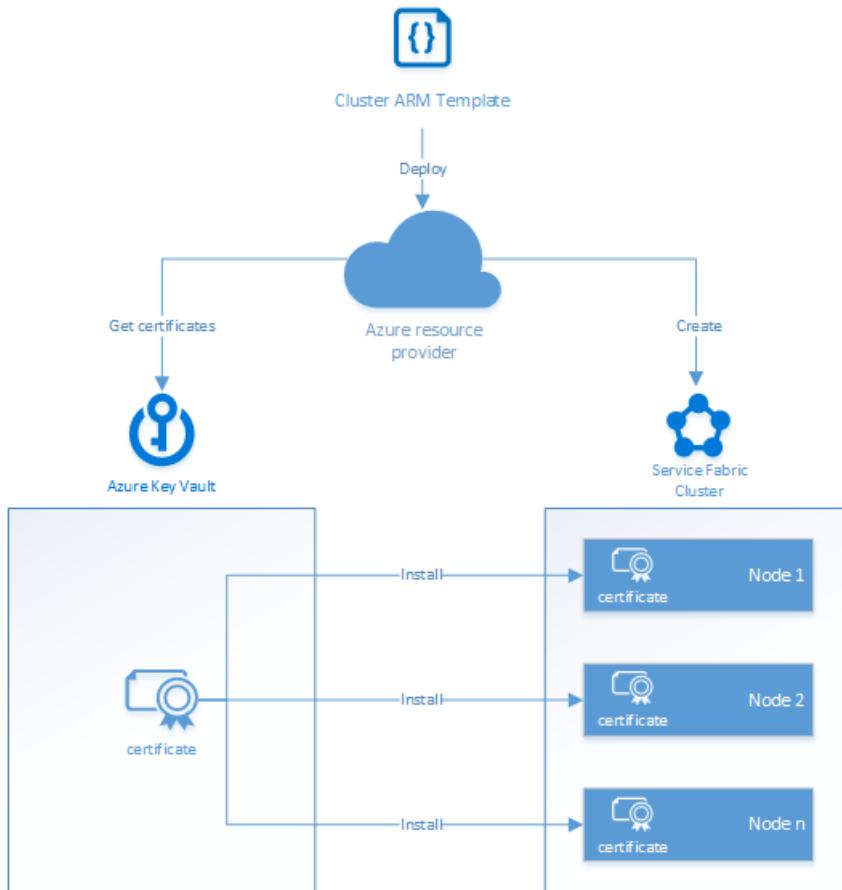
### Set up Key Vault

This part of the guide walks you through creating a Key Vault for a Service Fabric cluster in Azure and for Service Fabric applications. For a complete guide on Key Vault, see the [Key Vault getting started guide](#).

Service Fabric uses X.509 certificates to secure a cluster. Azure Key Vault is used to manage certificates for Service Fabric clusters in Azure. When a cluster is deployed in Azure, the Azure resource provider responsible

for creating Service Fabric clusters pulls certificates from Key Vault and installs them on the cluster VMs.

The following diagram illustrates the relationship between Key Vault, a Service Fabric cluster, and the Azure resource provider that uses certificates stored in Key Vault when it creates a cluster:



#### Create a Resource Group

The first step is to create a new resource group specifically for Key Vault. Putting Key Vault into its own resource group is recommended so that you can remove compute and storage resource groups - such as the resource group that has your Service Fabric cluster - without losing your keys and secrets. The resource group that has your Key Vault must be in the same region as the cluster that is using it.

```
PS C:\Users\vturecek> New-AzureRmResourceGroup -Name mycluster-keyvault -Location 'West US'
WARNING: The output object type of this cmdlet will be modified in a future release.

ResourceGroupName : mycluster-keyvault
Location        : westus
ProvisioningState : Succeeded
Tags            :
ResourceId      : /subscriptions/<guid>/resourceGroups/mycluster-keyvault
```

#### Create Key Vault

Create a Key Vault in the new resource group. The Key Vault **must be enabled for deployment** to allow the Service Fabric resource provider to get certificates from it and install on cluster nodes:

```

PS C:\Users\vturecek> New-AzureRmKeyVault -VaultName 'myvault' -ResourceGroupName 'mycluster-keyvault'
-Location 'West US' -EnabledForDeployment

Vault Name : myvault
Resource Group Name : mycluster-keyvault
Location : West US
Resource ID : /subscriptions/<guid>/resourceGroups/mycluster-
keyvault/providers/Microsoft.KeyVault/vaults/myvault
Vault URI : https://myvault.vault.azure.net
Tenant ID : <guid>
SKU : Standard
Enabled For Deployment? : False
Enabled For Template Deployment? : False
Enabled For Disk Encryption? : False
Access Policies :
    Tenant ID : <guid>
    Object ID : <guid>
    Application ID :
    Display Name :
    Permissions to Keys : get, create, delete, list, update,
import, backup, restore
    Permissions to Secrets : all

Tags :

```

If you have an existing Key Vault, you can enable it for deployment using Azure CLI:

```

> azure login
> azure account set "your account"
> azure config mode arm
> azure keyvault list
> azure keyvault set-policy --vault-name "your vault name" --enabled-for-deployment true

```

## Add certificates to Key Vault

Certificates are used in Service Fabric to provide authentication and encryption to secure various aspects of a cluster and its applications. For more information on how certificates are used in Service Fabric, see [Service Fabric cluster security scenarios](#).

### Cluster and server certificate (required)

This certificate is required to secure a cluster and prevent unauthorized access to it. It provides cluster security in a couple ways:

- **Cluster authentication:** Authenticates node-to-node communication for cluster federation. Only nodes that can prove their identity with this certificate can join the cluster.
- **Server authentication:** Authenticates the cluster management endpoints to a management client, so that the management client knows it is talking to the real cluster. This certificate also provides SSL for the HTTPS management API and for Service Fabric Explorer over HTTPS.

To serve these purposes, the certificate must meet the following requirements:

- The certificate must contain a private key.
- The certificate must be created for key exchange, exportable to a Personal Information Exchange (.pfx) file.
- The certificate's subject name must match the domain used to access the Service Fabric cluster. This is required to provide SSL for the cluster's HTTPS management endpoints and Service Fabric Explorer. You cannot obtain an SSL certificate from a certificate authority (CA) for the `.cloudapp.azure.com` domain. Acquire a custom domain name for your cluster. When you request a certificate from a CA the certificate's

subject name must match the custom domain name used for your cluster.

#### Client authentication certificates

Additional client certificates authenticate administrators for cluster management tasks. Service Fabric has two access levels: **admin** and **read-only user**. At minimum, a single certificate for administrative access should be used. For additional user-level access, a separate certificate must be provided. For more information on access roles, see [role-based access control for Service Fabric clients](#).

You do not need to upload Client authentication certificates to Key Vault to work with Service Fabric. These certificates only need to be provided to users who are authorized for cluster management.

#### NOTE

Azure Active Directory is the recommended way to authenticate clients for cluster management operations. To use Azure Active Directory, you must [create a cluster using Azure Resource Manager](#).

#### Application certificates (optional)

Any number of additional certificates can be installed on a cluster for application security purposes. Before creating your cluster, consider the application security scenarios that require a certificate to be installed on the nodes, such as:

- Encryption and decryption of application configuration values
- Encryption of data across nodes during replication

Application certificates cannot be configured when creating a cluster through the Azure portal. To configure application certificates at cluster setup time, you must [create a cluster using Azure Resource Manager](#). You can also add application certificates to the cluster after it has been created.

#### Formatting certificates for Azure resource provider use

Private key files (.pfx) can be added and used directly through Key Vault. However, the Azure resource provider requires keys to be stored in a special JSON format that includes the .pfx as a base-64 encoded string and the private key password. To accommodate these requirements, keys must be placed in a JSON string and then stored as *secrets* in Key Vault.

To make this process easier, a PowerShell module is [available on GitHub](#). Follow these steps to use the module:

1. Download the entire contents of the repo into a local directory.
2. Import the module in your PowerShell window:

```
PS C:\Users\vturecek> Import-Module  
"C:\users\vturecek\Documents\ServiceFabricRPHelpers\ServiceFabricRPHelpers.psm1"
```

The `Invoke-AddCertToKeyVault` command in this PowerShell module automatically formats a certificate private key into a JSON string and uploads it to Key Vault. Use it to add the cluster certificate and any additional application certificates to Key Vault. Repeat this step for any additional certificates you want to install in your cluster.

```

PS C:\Users\vturecek> Invoke-AddCertToKeyVault -SubscriptionId <guid> -ResourceGroupName mycluster-
keyvault -Location "West US" -VaultName myvault -CertificateName mycert -Password "<password>" -
UseExistingCertificate -ExistingPfxFilePath "C:\path\to\mycertkey.pfx"

Switching context to SubscriptionId <guid>
Ensuring ResourceGroup mycluster-keyvault in West US
WARNING: The output object type of this cmdlet will be modified in a future release.
Using existing valut myvault in West US
Reading pfx file from C:\path\to\key.pfx
Writing secret to myvault in vault myvault

Name : CertificateThumbprint
Value : <value>

Name : SourceVault
Value : /subscriptions/<guid>/resourceGroups/mycluster-
keyvault/providers/Microsoft.KeyVault/vaults/myvault

Name : CertificateURL
Value : https://myvault.vault.azure.net:443/secrets/mycert/4d087088df974e869f1c0978cb100e47

```

These are all the Key Vault prerequisites for configuring a Service Fabric cluster Resource Manager template that installs certificates for node authentication, management endpoint security and authentication, and any additional application security features that use X.509 certificates. At this point, you should now have the following setup in Azure:

- Key Vault resource group
  - Key Vault
    - Cluster server authentication certificate

</a "create-cluster-portal" >

## Create cluster in the Azure portal

**Search for the Service Fabric cluster resource**

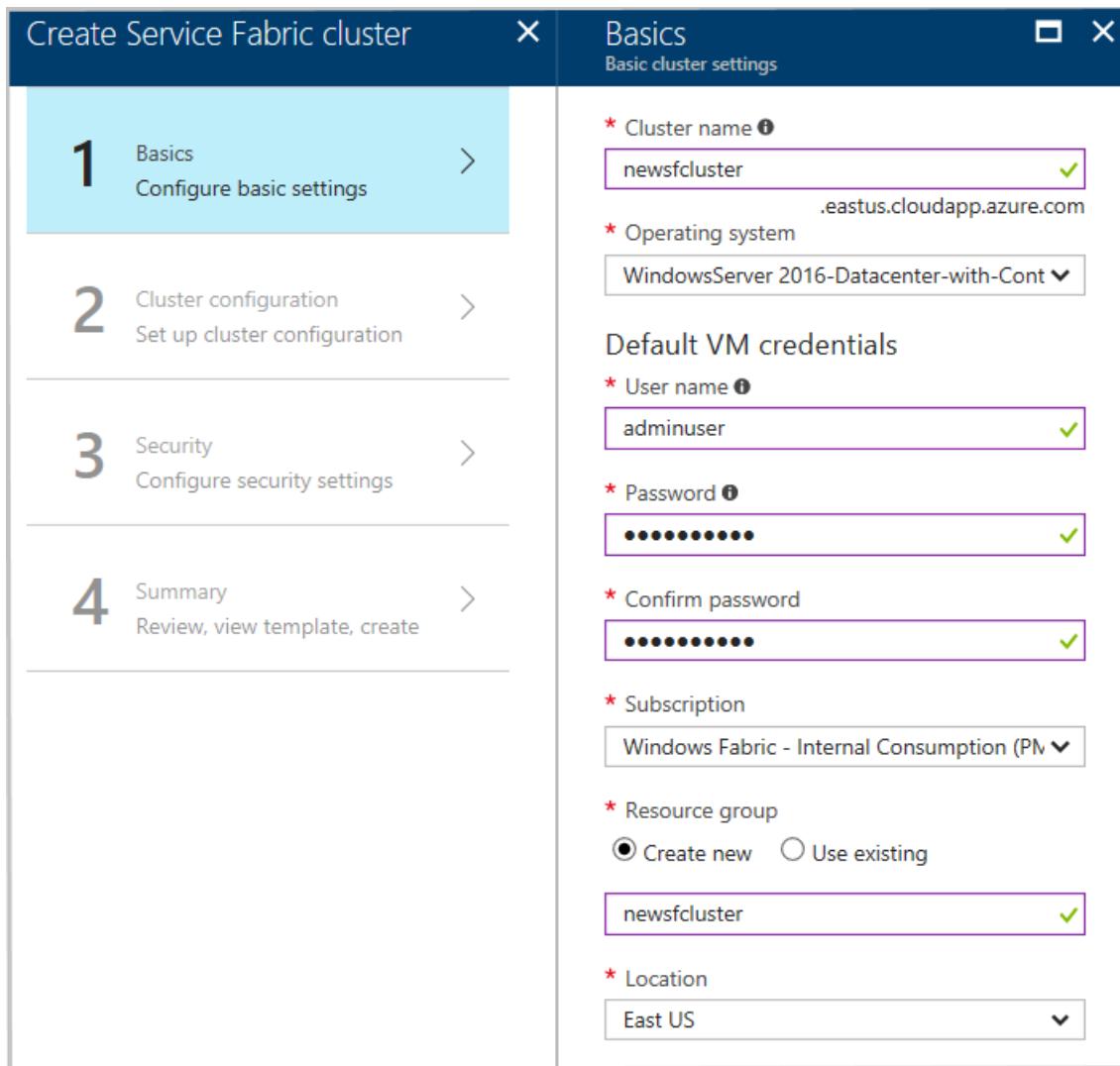
The screenshot shows the Microsoft Azure Marketplace interface. On the left, a sidebar lists various service categories like Dashboard, All resources, Resource groups, etc. A red box highlights the '+ New' button at the top of the sidebar. The main area is titled 'Everything' and contains a search bar with the query 'service fabric'. Below the search bar is a 'Results' section with a table. The table has columns for NAME, PUBLISHER, and CATEGORY. The results listed are:

NAME	PUBLISHER	CATEGORY
Service Fabric Cluster	Microsoft	Compute
Service Fabric Analytics	Microsoft	Monitoring + Manage...
Hyperledger Fabric Single Member Blockchain	Microsoft	Compute
NooBaa Hybrid S3 Archive - Community Edition	NooBaa	Compute
Corda Demo	R3	Compute
Corda Single Ledger Network	R3	Compute
Silver Peak Unity EdgeConnect v7.3	Silver Peak Systems	Compute

Below the table, there's a section titled 'Related to your search' with three items: Cloud service (Microsoft), Batch Service (Microsoft), and SQL Database (Microsoft).

1. Sign in to the [Azure portal](#).
2. Click **New** to add a new resource template. Search for the Service Fabric Cluster template in the **Marketplace** under **Everything**.
3. Select **Service Fabric Cluster** from the list.
4. Navigate to the **Service Fabric Cluster** blade, click **Create**,
5. The **Create Service Fabric cluster** blade has the following four steps.

#### 1. Basics



In the Basics blade you need to provide the basic details for your cluster.

1. Enter the name of your cluster.
2. Enter a **user name** and **password** for Remote Desktop for the VMs.
3. Make sure to select the **Subscription** that you want your cluster to be deployed to, especially if you have multiple subscriptions.
4. Create a **new resource group**. It is best to give it the same name as the cluster, since it helps in finding them later, especially when you are trying to make changes to your deployment or delete your cluster.

#### NOTE

Although you can decide to use an existing resource group, it is a good practice to create a new resource group. This makes it easy to delete clusters that you do not need.

5. Select the **region** in which you want to create the cluster. You must use the same region that your Key Vault is in.
2. **Cluster configuration**

The screenshot shows the 'Create Service Fabric cluster' wizard with four steps:

- Step 1: Basics** (Done) - Completed.
- Step 2: Cluster configuration** (Set up cluster configuration) - Selected.
- Step 3: Security** (Configure security settings)
- Step 4: Summary** (Review, view template, create)

The 'Cluster configuration' tab is active, showing the 'Node type configuration' section. It includes:

- Node types**: Describes node types as scale sets for managing the cluster. It specifies between 1 and 3 nodes.
- \* Node type count**: Radio buttons for 1, 2, or 3.
- \* Node type 1 (Primary)**: Sub-section for configuring required settings, including:
  - Diagnostics**: Options for application log storage (On or Off) and Application Insights key (optional).
  - Add on features**: Includes DNS service (checked) and repair manager.
  - Fabric version**: Options for fabric upgrade mode (Automatic or Manual) and fabric version (default).
  - Custom fabric settings**: Option to enter fabric setting properties.
- Node type configuration** (Node type 1 (Primary)) settings:
  - \* Node type name**: nt1
  - Durability tier**: Silver
  - Virtual machine size**: Standard\_D1\_v2
  - Single node cluster**: Unchecked
  - Initial VM scale set capacity**: Set to 5
  - Custom endpoints**: 80, 81
  - Enable reverse proxy**: Checked
  - Reverse proxy port**: 19081
  - Configure advanced settings**: Unchecked

Configure your cluster nodes. Node types define the VM sizes, the number of VMs, and their properties. Your cluster can have more than one node type, but the primary node type (the first one that you define on the portal) must have at least five VMs, as this is the node type where Service Fabric system services are placed. Do not configure **Placement Properties** because a default placement property of "NodeType" is added automatically.

#### NOTE

A common scenario for multiple node types is an application that contains a front-end service and a back-end service. You want to put the front-end service on smaller VMs (VM sizes like D2) with ports open to the Internet, but you want to put the back-end service on larger VMs (with VM sizes like D4, D6, D15, and so on) with no Internet-facing ports open.

1. Choose a name for your node type (1 to 12 characters containing only letters and numbers).
2. The minimum **size** of VMs for the primary node type is driven by the **durability** tier you choose for the cluster. The default for the durability tier is bronze. For more information on durability, see [how to choose the Service Fabric cluster reliability and durability](#).
3. Select the VM size and pricing tier. D-series VMs have SSD drives and are highly recommended for stateful applications. Do not use any VM SKU that has partial cores or have less than 7 GB of available disk capacity.
4. The minimum **number** of VMs for the primary node type is driven by the **reliability** tier you choose. The default for the reliability tier is Silver. For more information on reliability, see [how to choose the Service Fabric cluster reliability and durability](#).
5. Choose the number of VMs for the node type. You can scale up or down the number of VMs in a node type later on, but on the primary node type, the minimum is driven by the reliability level that you have chosen.

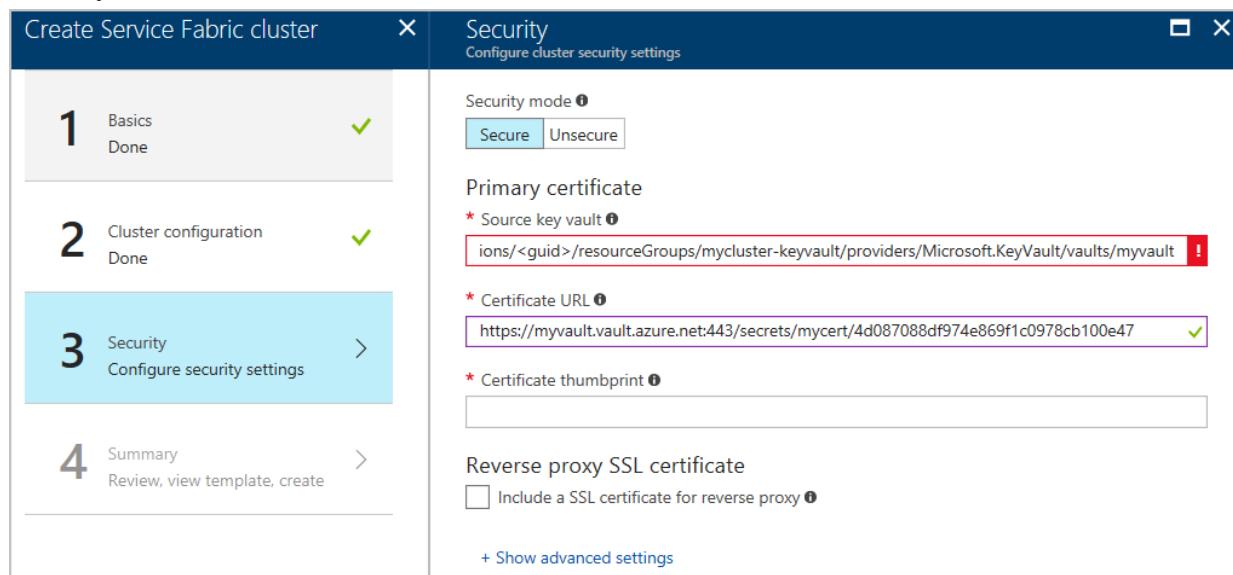
Other node types can have a minimum of 1 VM.

6. Configure custom endpoints. This field allows you to enter a comma-separated list of ports that you want to expose through the Azure Load Balancer to the public Internet for your applications. For example, if you plan to deploy a web application to your cluster, enter "80" here to allow traffic on port 80 into your cluster. For more information on endpoints, see [communicating with applications](#)
7. Configure cluster **diagnostics**. By default, diagnostics are enabled on your cluster to assist with troubleshooting issues. If you want to disable diagnostics change the **Status** toggle to **Off**. Turning off diagnostics is **not** recommended.
8. Select the Fabric upgrade mode you want set your cluster to. Select **Automatic**, if you want the system to automatically pick up the latest available version and try to upgrade your cluster to it. Set the mode to **Manual**, if you want to choose a supported version.

#### NOTE

We support only clusters that are running supported versions of service Fabric. By selecting the **Manual** mode, you are taking on the responsibility to upgrade your cluster to a supported version. For more details on the Fabric upgrade mode see the [service-fabric-cluster-upgrade document](#).

### 3. Security



The screenshot shows the 'Create Service Fabric cluster' wizard with four steps:

- Step 1: Basics** (Done) - Completed successfully.
- Step 2: Cluster configuration** (Done) - Completed successfully.
- Step 3: Security** (Configure security settings) - This is the current step. It includes fields for Primary certificate (Source key vault, Certificate URL, Certificate thumbprint), Reverse proxy SSL certificate (checkbox for including a certificate), and advanced settings.
- Step 4: Summary** (Review, view template, create) - Summary step.

The final step is to provide certificate information to secure the cluster using the Key Vault and certificate information created earlier.

- Populate the primary certificate fields with the output obtained from uploading the **cluster certificate** to Key Vault using the `Invoke-AddCertToKeyVault` PowerShell command.

```
Name : CertificateThumbprint
Value : <value>

Name : SourceVault
Value : /subscriptions/<guid>/resourceGroups/mycluster-
keyvault/providers/Microsoft.KeyVault/vaults/myvault

Name : CertificateURL
Value : https://myvault.vault.azure.net:443/secrets/mycert/4d087088df974e869f1c0978cb100e47
```

- Check the **Configure advanced settings** box to enter client certificates for **admin client** and **read-only client**. In these fields, enter the thumbprint of your admin client certificate and the thumbprint of your read-only user client certificate, if applicable. When administrators attempt to connect to the cluster, they are

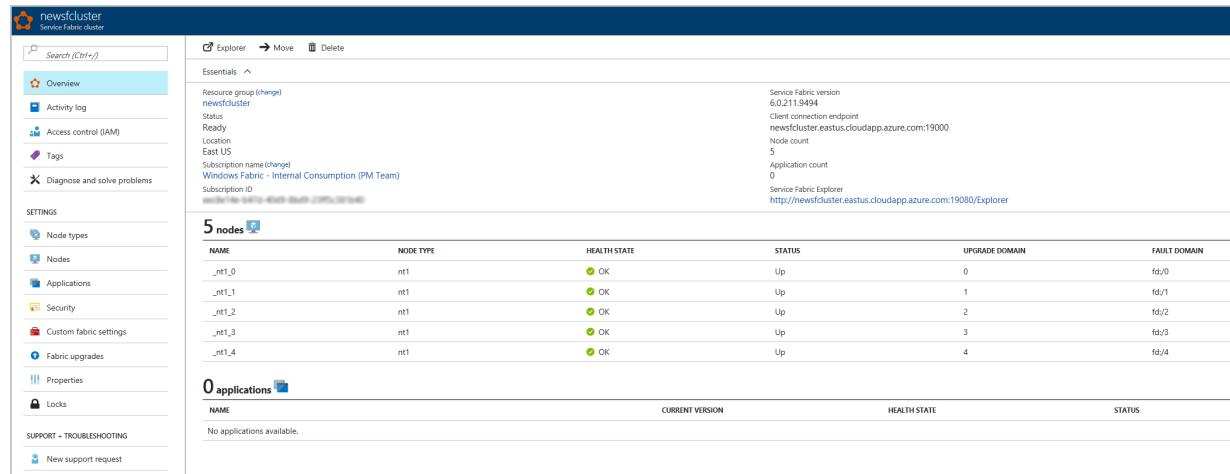
granted access only if they have a certificate with a thumbprint that matches the thumbprint values entered here.

#### 4. Summary

To complete the cluster creation, click **Summary** to see the configurations that you have provided, or download the Azure Resource Manager template that that used to deploy your cluster. After you have provided the mandatory settings, the **OK** button becomes green and you can start the cluster creation process by clicking it.

You can see the creation progress in the notifications. (Click the "Bell" icon near the status bar at the upper right of your screen.) If you clicked **Pin to Startboard** while creating the cluster, you will see **Deploying Service Fabric Cluster** pinned to the **Start** board.

#### View your cluster status



The screenshot shows the Azure portal's Service Fabric cluster management interface. On the left, there's a sidebar with navigation links like Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. The main content area has tabs for Explorer, Move, and Delete. Under the Explorer tab, there's a summary section with details such as Resource group (changed) newscluster, Status Ready, Location East US, Subscription name (changed) Windows Fabric - Internal Consumption (PM Team), and Subscription ID. Below this is a table titled '5 nodes' showing five nodes named \_nt1\_0 through \_nt1\_4, all in OK health state and Up status, assigned to upgrade domain 0 through 4 and fault domain fd/0 through fd/4. There's also a section for '0 applications' with a note that no applications are available. At the bottom, there's a 'New support request' link.

Once your cluster is created, you can inspect your cluster in the portal:

1. Go to **Browse** and click **Service Fabric Clusters**.
2. Locate your cluster and click it.
3. You can now see the details of your cluster in the dashboard, including the cluster's public endpoint and a link to Service Fabric Explorer.

The **Node Monitor** section on the cluster's dashboard blade indicates the number of VMs that are healthy and not healthy. You can find more details about the cluster's health at [Service Fabric health model introduction](#).

#### NOTE

Service Fabric clusters require a certain number of nodes to be up always to maintain availability and preserve state – referred to as "maintaining quorum". Therefore, it is typically not safe to shut down all machines in the cluster unless you have first performed a [full backup of your state](#).

## Remote connect to a Virtual Machine Scale Set instance or a cluster node

Each of the NodeTypes you specify in your cluster results in a Virtual Machine Scale Set getting set-up.

## Next steps

At this point, you have a secure cluster using certificates for management authentication. Next, [connect to your cluster](#) and learn how to [manage application secrets](#). Also, learn about [Service Fabric support options](#).

# Create a Service Fabric cluster by using Azure Resource Manager

9/25/2017 • 20 min to read • [Edit Online](#)

This step-by-step guide walks you through setting up a secure Azure Service Fabric cluster in Azure by using Azure Resource Manager. We acknowledge that the article is long. Nevertheless, unless you are already thoroughly familiar with the content, be sure to follow each step carefully.

The guide covers the following procedures:

- Setting up an Azure key vault to upload certificates for cluster and application security
- Creating a secured cluster in Azure by using Azure Resource Manager
- Authenticating users by using Azure Active Directory (Azure AD) for cluster management

A secure cluster is a cluster that prevents unauthorized access to management operations. This includes deploying, upgrading, and deleting applications, services, and the data they contain. An unsecure cluster is a cluster that anyone can connect to at any time and perform management operations. Although it is possible to create an unsecure cluster, we highly recommend that you create a secure cluster from the outset. Because an unsecure cluster cannot be secured later, a new cluster must be created.

The concept of creating secure clusters is the same, whether they are Linux or Windows clusters. For more information and helper scripts for creating secure Linux clusters, see [Creating secure clusters on Linux](#).

## Sign in to your Azure account

This guide uses [Azure PowerShell](#). When you start a new PowerShell session, sign in to your Azure account and select your subscription before you execute Azure commands.

Sign in to your Azure account:

```
Login-AzureRmAccount
```

Select your subscription:

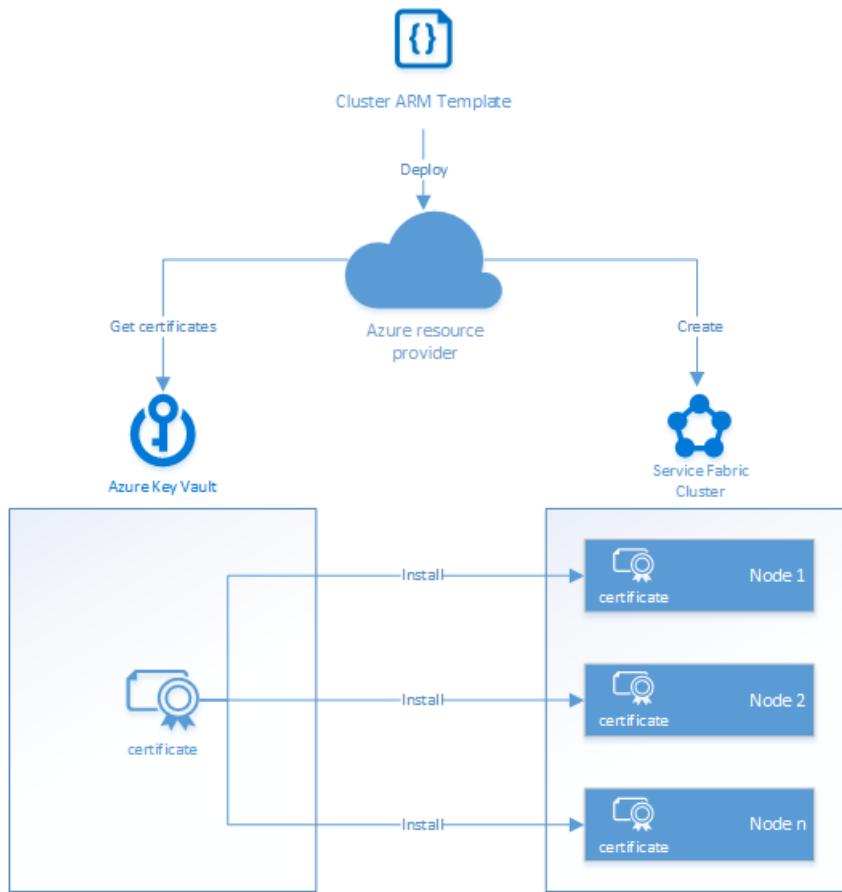
```
Get-AzureRmSubscription  
Set-AzureRmContext -SubscriptionId <guid>
```

## Set up a key vault

This section discusses creating a key vault for a Service Fabric cluster in Azure and for Service Fabric applications. For a complete guide to Azure Key Vault, refer to the [Key Vault getting started guide](#).

Service Fabric uses X.509 certificates to secure a cluster and provide application security features. You use Key Vault to manage certificates for Service Fabric clusters in Azure. When a cluster is deployed in Azure, the Azure resource provider that's responsible for creating Service Fabric clusters pulls certificates from Key Vault and installs them on the cluster VMs.

The following diagram illustrates the relationship between Azure Key Vault, a Service Fabric cluster, and the Azure resource provider that uses certificates stored in a key vault when it creates a cluster:



### Create a resource group

The first step is to create a resource group specifically for your key vault. We recommend that you put the key vault into its own resource group. This action lets you remove the compute and storage resource groups, including the resource group that contains your Service Fabric cluster, without losing your keys and secrets. The resource group that contains your key vault *must be in the same region* as the cluster that is using it.

If you plan to deploy clusters in multiple regions, we suggest that you name the resource group and the key vault in a way that indicates which region it belongs to.

```
New-AzureRmResourceGroup -Name westus-mykeyvault -Location 'West US'
```

The output should look like this:

```
WARNING: The output object type of this cmdlet is going to be modified in a future release.
```

```

ResourceGroupName : westus-mykeyvault
Location        : westus
ProvisioningState : Succeeded
Tags            :
ResourceId      : /subscriptions/<guid>/resourceGroups/westus-mykeyvault
  
```

### Create a key vault in the new resource group

The key vault *must be enabled for deployment* to allow the compute resource provider to get certificates from it and install it on virtual machine instances:

```
New-AzureRmKeyVault -VaultName 'mywestusvault' -ResourceGroupName 'westus-mykeyvault' -Location 'West US' -EnabledForDeployment
```

The output should look like this:

```
Vault Name : mywestusvault
Resource Group Name : westus-mykeyvault
Location : West US
Resource ID : /subscriptions/<guid>/resourceGroups/westus-
mykeyvault/providers/Microsoft.KeyVault/vaults/mywestusvault
Vault URI : https://mywestusvault.vault.azure.net
Tenant ID : <guid>
SKU : Standard
Enabled For Deployment? : False
Enabled For Template Deployment? : False
Enabled For Disk Encryption? : False
Access Policies :
    Tenant ID : <guid>
    Object ID : <guid>
    Application ID :
    Display Name :
    Permissions to Keys : get, create, delete, list, update,
import, backup, restore
    Permissions to Secrets : all

Tags :
```

## Use an existing key vault

To use an existing key vault, you *must enable it for deployment* to allow the compute resource provider to get certificates from it and install it on cluster nodes:

```
Set-AzureRmKeyVaultAccessPolicy -VaultName 'ContosoKeyVault' -EnabledForDeployment
```

## Add certificates to your key vault

Certificates are used in Service Fabric to provide authentication and encryption to secure various aspects of a cluster and its applications. For more information on how certificates are used in Service Fabric, see [Service Fabric cluster security scenarios](#).

### Cluster and server certificate (required)

This certificate is required to secure a cluster and prevent unauthorized access to it. It provides cluster security in two ways:

- Cluster authentication: Authenticates node-to-node communication for cluster federation. Only nodes that can prove their identity with this certificate can join the cluster.
- Server authentication: Authenticates the cluster management endpoints to a management client, so that the management client knows it is talking to the real cluster. This certificate also provides an SSL for the HTTPS management API and for Service Fabric Explorer over HTTPS.

To serve these purposes, the certificate must meet the following requirements:

- The certificate must contain a private key.

- The certificate must be created for key exchange, which is exportable to a Personal Information Exchange (.pfx) file.
- The certificate's subject name must match the domain that you use to access the Service Fabric cluster. This matching is required to provide an SSL for the cluster's HTTPS management endpoints and Service Fabric Explorer. You cannot obtain an SSL certificate from a certificate authority (CA) for the .cloudapp.azure.com domain. You must obtain a custom domain name for your cluster. When you request a certificate from a CA, the certificate's subject name must match the custom domain name that you use for your cluster.

### **Application certificates (optional)**

Any number of additional certificates can be installed on a cluster for application security purposes. Before creating your cluster, consider the application security scenarios that require a certificate to be installed on the nodes, such as:

- Encryption and decryption of application configuration values.
- Encryption of data across nodes during replication.

### **Formatting certificates for Azure resource provider use**

You can add and use private key files (.pfx) directly through your key vault. However, the Azure compute resource provider requires keys to be stored in a special JavaScript Object Notation (JSON) format. The format includes the .pfx file as a base 64-encoded string and the private key password. To accommodate these requirements, the keys must be placed in a JSON string and then stored as "secrets" in the key vault.

To make this process easier, a [PowerShell module is available on GitHub](#). To use the module, do the following:

1. Download the entire contents of the repo into a local directory.
2. Go to the local directory.
3. Import the ServiceFabricRPHelpers module in your PowerShell window:

```
Import-Module "C:\..\ServiceFabricRPHelpers\ServiceFabricRPHelpers.psm1"
```

The `Invoke-AddCertToKeyVault` command in this PowerShell module automatically formats a certificate private key into a JSON string and uploads it to the key vault. Use the command to add the cluster certificate and any additional application certificates to the key vault. Repeat this step for any additional certificates you want to install in your cluster.

### **Uploading an existing certificate**

```
Invoke-AddCertToKeyVault -SubscriptionId <guid> -ResourceGroupName westus-mykeyvault -Location "West US"
-VaultName mywestusvault -CertificateName mycert -Password "<password>" -UseExistingCertificate -
ExistingPfxFilePath "C:\path\to\mycertkey.pfx"
```

If you get an error, such as the one shown here, it usually means that you have a resource URL conflict. To resolve the conflict, change the key vault name.

```
Set-AzureKeyVaultSecret : The remote name could not be resolved: 'westuskv.vault.azure.net'
At C:\Users\chackdan\Documents\GitHub\Service-
Fabric\Scripts\ServiceFabricRPHelpers\ServiceFabricRPHelpers.psm1:440 char:11
+ $secret = Set-AzureKeyVaultSecret -VaultName $VaultName -Name $Certif ...
+           ~~~~~
+ CategoryInfo          : CloseError: (:) [Set-AzureKeyVaultSecret], WebException
+ FullyQualifiedErrorId : Microsoft.Azure.Commands.KeyVault.SetAzureKeyVaultSecret
```

After the conflict is resolved, the output should look like this:

```
Switching context to SubscriptionId <guid>
Ensuring ResourceGroup westus-mykeyvault in West US
WARNING: The output object type of this cmdlet is going to be modified in a future release.
Using existing value mywestusvault in West US
Reading pfx file from C:\path\to\key.pfx
Writing secret to mywestusvault in vault mywestusvault

Name  : CertificateThumbprint
Value : E21DBC64B183B5BF355C34C46E03409FEEAEF58D

Name  : SourceVault
Value : /subscriptions/<guid>/resourceGroups/westus-
mykeyvault/providers/Microsoft.KeyVault/vaults/mywestusvault

Name  : CertificateURL
Value : https://mywestusvault.vault.azure.net:443/secrets/mycert/4d087088df974e869f1c0978cb100e47
```

#### NOTE

You need the three preceding strings, CertificateThumbprint, SourceVault, and CertificateURL, to set up a secure Service Fabric cluster and to obtain any application certificates that you might be using for application security. If you do not save the strings, it can be difficult to retrieve them by querying the key vault later.

#### Creating a self-signed certificate and uploading it to the key vault

If you have already uploaded your certificates to the key vault, skip this step. This step is for generating a new self-signed certificate and uploading it to your key vault. After you change the parameters in the following script and then run it, you should be prompted for a certificate password.

```
$ResourceGroup = "chackowestuskv"
$VName = "chackokv2"
$SubID = "6c653126-e4ba-42cd-a1dd-f7bf96ae7a47"
$locationRegion = "westus"
$newCertName = "chackotestcertificate1"
$dnsName = "www.mycluster.westus.mydomain.com" #The certificate's subject name must match the domain used
to access the Service Fabric cluster.
$localCertPath = "C:\MyCertificates" # location where you want the .PFX to be stored

Invoke-AddCertToKeyVault -SubscriptionId $SubID -ResourceGroupName $ResourceGroup -Location
$locationRegion -VaultName $VName -CertificateName $newCertName -CreateSelfSignedCertificate -DnsName
$dnsName -OutputPath $localCertPath
```

If you get an error, such as the one shown here, it usually means that you have a resource URL conflict. To resolve the conflict, change the key vault name, RG name, and so forth.

```
Set-AzureKeyVaultSecret : The remote name could not be resolved: 'westuskv.vault.azure.net'
At C:\Users\chackdan\Documents\GitHub\Service-
Fabric\Scripts\ServiceFabricRPHelpers\ServiceFabricRPHelpers.ps1:440 char:11
+ $secret = Set-AzureKeyVaultSecret -VaultName $VaultName -Name $Certif ...
+           ~~~~~
+ CategoryInfo          : CloseError: () [Set-AzureKeyVaultSecret], WebException
+ FullyQualifiedErrorId : Microsoft.Azure.Commands.KeyVault.SetAzureKeyVaultSecret
```

After the conflict is resolved, the output should look like this:

```

PS C:\Users\chackdan\Documents\GitHub\Service-Fabric\Scripts\ServiceFabricRPHelpers> Invoke-
AddCertToKeyVault -SubscriptionId $SubID -ResourceGroupName $ResourceGroup -Location $locationRegion -
VaultName $VName -CertificateName $newCertName -Password $certPassword -CreateSelfSignedCertificate -
DnsName $dnsName -OutputPath $localCertPath
Switching context to SubscriptionId 6c343126-e4ba-52cd-a1dd-f8bf96ae7a47
Ensuring ResourceGroup chackowestuskv in westus
WARNING: The output object type of this cmdlet will be modified in a future release.
Creating new vault westuskv1 in westus
Creating new self signed certificate at C:\MyCertificates\chackonewcertificate1.pfx
Reading pfx file from C:\MyCertificates\chackonewcertificate1.pfx
Writing secret to chackonewcertificate1 in vault westuskv1

Name : CertificateThumbprint
Value : 96BB3CC234F9D43C25D4B547sd8DE7B569F413EE

Name : SourceVault
Value : /subscriptions/6c653126-e4ba-52cd-a1dd-
f8bf96ae7a47/resourceGroups/chackowestuskv/providers/Microsoft.KeyVault/vaults/westuskv1

Name : CertificateURL
Value :
https://westuskv1.vault.azure.net:443/secrets/chackonewcertificate1/ee247291e45d405b8c8bbf81782d12bd

```

#### **NOTE**

You need the three preceding strings, CertificateThumbprint, SourceVault, and CertificateURL, to set up a secure Service Fabric cluster and to obtain any application certificates that you might be using for application security. If you do not save the strings, it can be difficult to retrieve them by querying the key vault later.

At this point, you should have the following elements in place:

- The key vault resource group.
- The key vault and its URL (called SourceVault in the preceding PowerShell output).
- The cluster server authentication certificate and its URL in the key vault.
- The application certificates and their URLs in the key vault.

## Set up Azure Active Directory for client authentication

Azure AD enables organizations (known as tenants) to manage user access to applications. Applications are divided into those with a web-based sign-in UI and those with a native client experience. In this article, we assume that you have already created a tenant. If you have not, start by reading [How to get an Azure Active Directory tenant](#).

A Service Fabric cluster offers several entry points to its management functionality, including the web-based [Service Fabric Explorer](#) and [Visual Studio](#). As a result, you create two Azure AD applications to control access to the cluster, one web application and one native application.

To simplify some of the steps involved in configuring Azure AD with a Service Fabric cluster, we have created a set of Windows PowerShell scripts.

#### **NOTE**

You must complete the following steps before you create the cluster. Because the scripts expect cluster names and endpoints, the values should be planned and not values that you have already created.

1. [Download the scripts](#) to your computer.

2. Right-click the zip file, select **Properties**, select the **Unblock** check box, and then click **Apply**.
3. Extract the zip file.
4. Run `SetupApplications.ps1`, and provide the TenantId, ClusterName, and WebApplicationReplyUrl as parameters. For example:

```
.\SetupApplications.ps1 -TenantId '690ec069-8200-4068-9d01-5aaf188e557a' -ClusterName 'mycluster' -WebApplicationReplyUrl 'https://mycluster.westus.cloudapp.azure.com:19080/Explorer/index.html'
```

You can find your TenantId by executing the PowerShell command `Get-AzureSubscription`. Executing this command displays the TenantId for every subscription.

ClusterName is used to prefix the Azure AD applications that are created by the script. It does not need to match the actual cluster name exactly. It is intended only to make it easier to map Azure AD artifacts to the Service Fabric cluster that they're being used with.

WebApplicationReplyUrl is the default endpoint that Azure AD returns to your users after they finish signing in. Set this endpoint as the Service Fabric Explorer endpoint for your cluster, which by default is:

`https://<cluster_domain>:19080/Explorer`

You are prompted to sign in to an account that has administrative privileges for the Azure AD tenant. After you sign in, the script creates the web and native applications to represent your Service Fabric cluster. If you look at the tenant's applications in the [Azure classic portal](#), you should see two new entries:

- *ClusterName\_Cluster*
- *ClusterName\_Client*

The script prints the JSON required by the Azure Resource Manager template when you create the cluster in the next section, so it's a good idea to keep the PowerShell window open.

```
"azureActiveDirectory": {
  "tenantId": "<guid>",
  "clusterApplication": "<guid>",
  "clientApplication": "<guid>"
},
```

## Create a Service Fabric cluster Resource Manager template

In this section, the outputs of the preceding PowerShell commands are used in a Service Fabric cluster Resource Manager template.

Sample Resource Manager templates are available in the [Azure quick-start template gallery on GitHub](#). These templates can be used as a starting point for your cluster template.

### Create the Resource Manager template

This guide uses the [5-node secure cluster](#) example template and template parameters. Download `azuredeploy.json` and `azuredeploy.parameters.json` to your computer and open both files in your favorite text editor.

### Add certificates

You add certificates to a cluster Resource Manager template by referencing the key vault that contains the certificate keys. We recommend that you place the key-vault values in a Resource Manager template parameters file. Doing so keeps the Resource Manager template file reusable and free of values specific to a

deployment.

#### Add all certificates to the virtual machine scale set osProfile

Every certificate that's installed in the cluster must be configured in the osProfile section of the scale set resource (Microsoft.Compute/virtualMachineScaleSets). This action instructs the resource provider to install the certificate on the VMs. This installation includes both the cluster certificate and any application security certificates that you plan to use for your applications:

```
{  
    "apiVersion": "2016-03-30",  
    "type": "Microsoft.Compute/virtualMachineScaleSets",  
    ...  
    "properties": {  
        ...  
        "osProfile": {  
            ...  
            "secrets": [  
                {  
                    "sourceVault": {  
                        "id": "[parameters('sourceVaultValue')]"  
                    },  
                    "vaultCertificates": [  
                        {  
                            "certificateStore": "[parameters('clusterCertificateStorevalue')]",  
                            "certificateUrl": "[parameters('clusterCertificateUrlValue')]"  
                        },  
                        {  
                            "certificateStore": "[parameters('applicationCertificateStorevalue')]",  
                            "certificateUrl": "[parameters('applicationCertificateUrlValue')]"  
                        },  
                        ...  
                    ]  
                }  
            ]  
        }  
    }  
}
```

#### Configure the Service Fabric cluster certificate

The cluster authentication certificate must be configured in both the Service Fabric cluster resource (Microsoft.ServiceFabric/clusters) and the Service Fabric extension for virtual machine scale sets in the virtual machine scale set resource. This arrangement allows the Service Fabric resource provider to configure it for use for cluster authentication and server authentication for management endpoints.

**Virtual machine scale set resource:**

```
{
  "apiVersion": "2016-03-30",
  "type": "Microsoft.Compute/virtualMachineScaleSets",
  ...
  "properties": {
    ...
    "virtualMachineProfile": {
      "extensionProfile": {
        "extensions": [
          {
            "name": "[concat('ServiceFabricNodeVmExt', '_vmNodeType0Name')]",
            "properties": {
              ...
              "settings": {
                ...
                "certificate": {
                  "thumbprint": "[parameters('clusterCertificateThumbprint')]",
                  "x509StoreName": "[parameters('clusterCertificateStoreValue')]"
                },
                ...
              }
            }
          ]
        }
      }
    }
  }
}
```

#### Service Fabric resource:

```
{
  "apiVersion": "2016-03-01",
  "type": "Microsoft.ServiceFabric/clusters",
  "name": "[parameters('clusterName')]",
  "location": "[parameters('clusterLocation')]",
  "dependsOn": [
    "[concat('Microsoft.Storage/storageAccounts/', variables('supportLogStorageAccountName'))]"
  ],
  "properties": {
    "certificate": {
      "thumbprint": "[parameters('clusterCertificateThumbprint')]",
      "x509StoreName": "[parameters('clusterCertificateStoreValue')]"
    },
    ...
  }
}
```

#### Insert Azure AD configuration

The Azure AD configuration that you created earlier can be inserted directly into your Resource Manager template. However, we recommended that you first extract the values into a parameters file to keep the Resource Manager template reusable and free of values specific to a deployment.

```
{
  "apiVersion": "2016-03-01",
  "type": "Microsoft.ServiceFabric/clusters",
  "name": "[parameters('clusterName')]",
  ...
  "properties": {
    "certificate": {
      "thumbprint": "[parameters('clusterCertificateThumbprint')]",
      "x509StoreName": "[parameters('clusterCertificateStorevalue')]"
    },
    ...
    "azureActiveDirectory": {
      "tenantId": "[parameters('aadTenantId')]",
      "clusterApplication": "[parameters('aadClusterApplicationId')]",
      "clientApplication": "[parameters('aadClientApplicationId')]"
    },
    ...
  }
}
```

## <a "configure-arm" >Configure Resource Manager template parameters

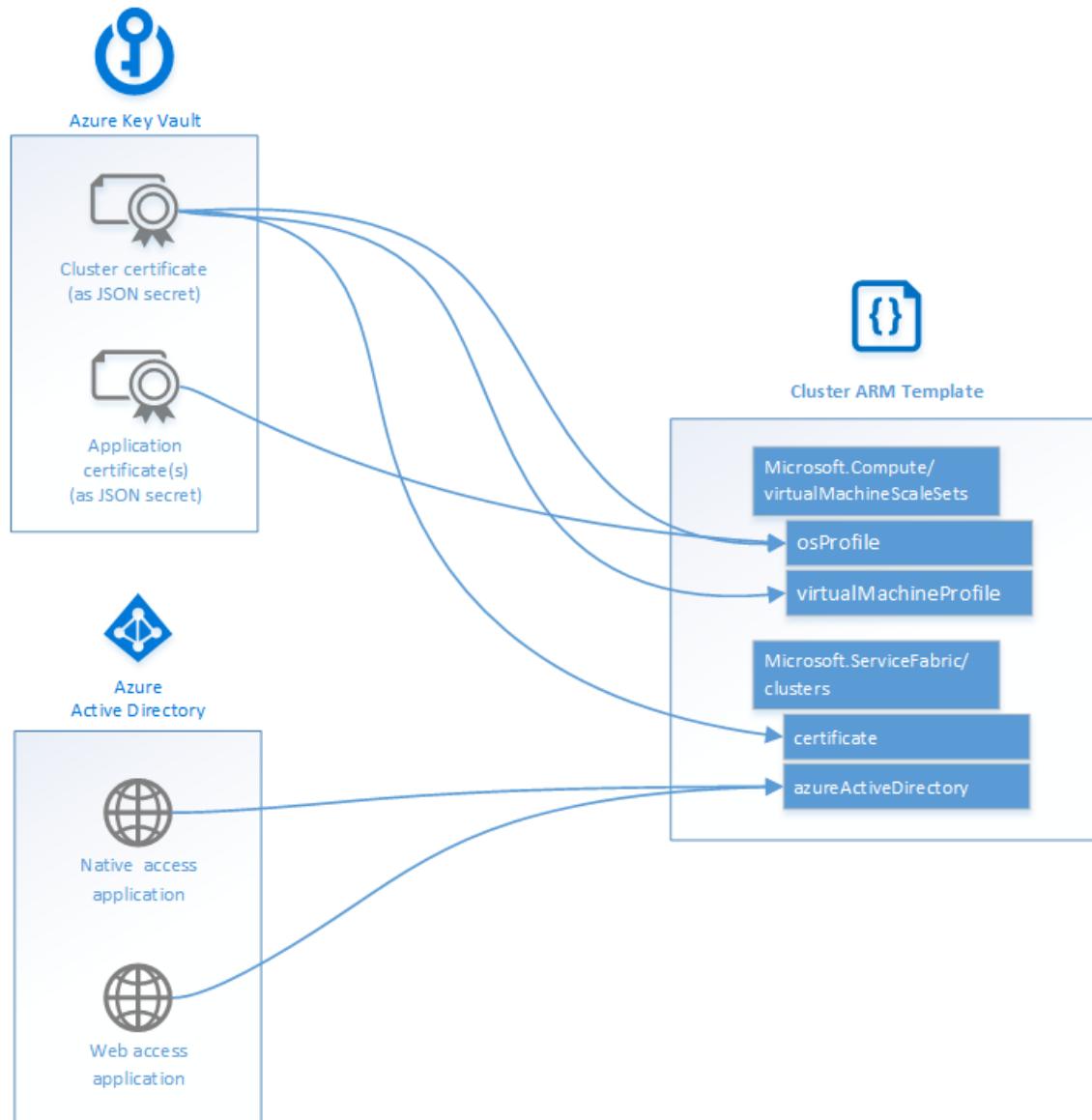
Finally, use the output values from the key vault and Azure AD PowerShell commands to populate the parameters file:

```
{
  "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    ...
    "clusterCertificateStoreValue": {
      "value": "My"
    },
    "clusterCertificateThumbprint": {
      "value": "<thumbprint>"
    },
    "clusterCertificateUrlValue": {
      "value":
"https://myvault.vault.azure.net:443/secrets/myclustercert/4d087088df974e869f1c0978cb100e47"
    },
    "applicationCertificateStorevalue": {
      "value": "My"
    },
    "applicationCertificateUrlValue": {
      "value":
"https://myvault.vault.azure.net:443/secrets/myapplicationcert/2e035058ae274f869c4d0348ca100f08"
    },
    "sourceVaultvalue": {
      "value": "/subscriptions/<guid>/resourceGroups/mycluster-
keyvault/providers/Microsoft.KeyVault/vaults/myvault"
    },
    "aadTenantId": {
      "value": "<guid>"
    },
    "aadClusterApplicationId": {
      "value": "<guid>"
    },
    "aadClientApplicationId": {
      "value": "<guid>"
    },
    ...
  }
}
```

At this point, you should have the following elements in place:

- Key vault resource group
  - Key vault
  - Cluster server authentication certificate
  - Data encipherment certificate
- Azure Active Directory tenant
  - Azure AD application for web-based management and Service Fabric Explorer
  - Azure AD application for native client management
  - Users and their assigned roles
- Service Fabric cluster Resource Manager template
  - Certificates configured through key vault
  - Azure Active Directory configured

The following diagram illustrates where your key vault and Azure AD configuration fit into your Resource Manager template.



## Create the cluster

You are now ready to create the cluster by using [Azure resource template deployment](#).

### Test it

Use the following PowerShell command to test your Resource Manager template with a parameters file:

```
Test-AzureRmResourceGroupDeployment -ResourceGroupName "myresourcegroup" -TemplateFile .\azuredeploy.json  
-TemplateParameterFile .\azuredeploy.parameters.json
```

### Deploy it

If the Resource Manager template test passes, use the following PowerShell command to deploy your Resource Manager template with a parameters file:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName "myresourcegroup" -TemplateFile .\azuredeploy.json  
-TemplateParameterFile .\azuredeploy.parameters.json
```

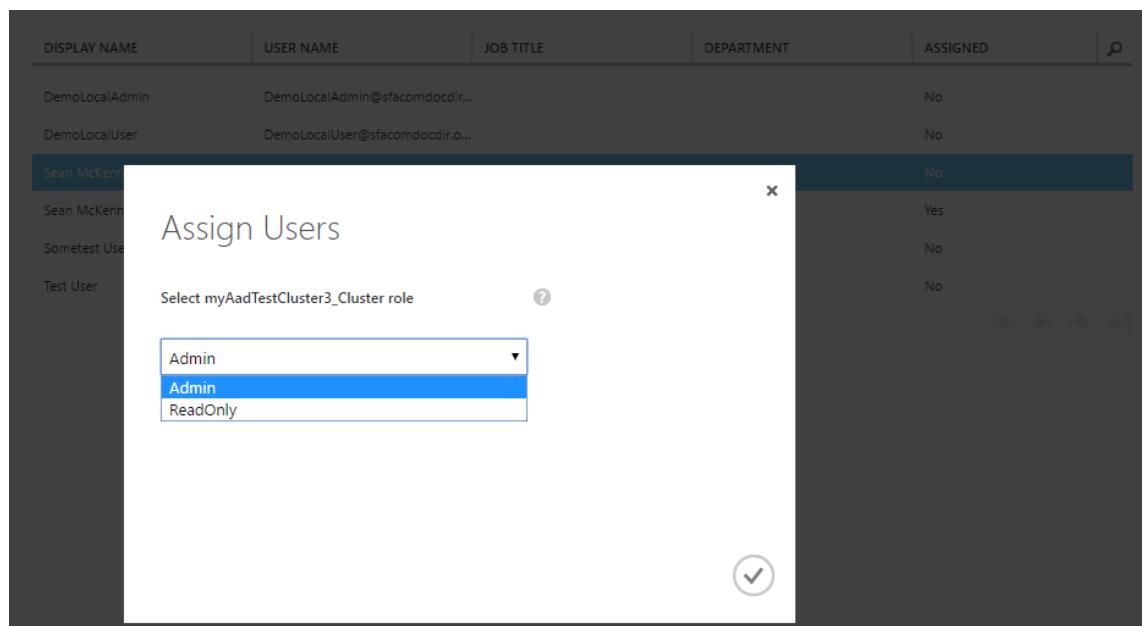
## Assign users to roles

After you have created the applications to represent your cluster, assign your users to the roles supported by Service Fabric: read-only and admin. You can assign the roles by using the [Azure classic portal](#).

1. In the Azure portal, go to your tenant, and then select **Applications**.
2. Select the web application, which has a name like `myTestCluster_Cluster`.
3. Click the **Users** tab.
4. Select a user to assign, and then click the **Assign** button at the bottom of the screen.



5. Select the role to assign to the user.



### NOTE

For more information about roles in Service Fabric, see [Role-based access control for Service Fabric clients](#).

## Create secure clusters on Linux

To make the process easier, we have provided a [helper script](#). Before you use this helper script, ensure that you already have Azure command-line interface (CLI) installed, and it is in your path. Make sure that the script has permissions to execute by running `chmod +x cert_helper.py` after downloading it. The first step is to sign

in to your Azure account by using CLI with the `azure login` command. After signing in to your Azure account, use the helper script with your CA signed certificate, as the following command shows:

```
./cert_helper.py [-h] CERT_TYPE [-ifile INPUT_CERT_FILE] [-sub SUBSCRIPTION_ID] [-rgname  
RESOURCE_GROUP_NAME] [-kv KEY_VAULT_NAME] [-sname CERTIFICATE_NAME] [-l LOCATION] [-p PASSWORD]
```

The `-ifile` parameter can take a .pfx file or a .pem file as input, with the certificate type (pfx or pem, or ss if it is a self-signed certificate). The parameter `-h` prints out the help text.

This command returns the following three strings as the output:

- `SourceVaultID`, which is the ID for the new KeyVault ResourceGroup it created for you
- `CertificateUrl` for accessing the certificate
- `CertificateThumbprint`, which is used for authentication

The following example shows how to use the command:

```
./cert_helper.py pfx -sub "ffffffff-ffff-ffff-ffff-ffffffffffff" -rgname "mykvrg" -kv "mykevname" -ifile  
"/home/test/cert.pfx" -sname "mycert" -l "East US" -p "pfxtest"
```

Executing the preceding command gives you the three strings as follows:

```
SourceVault: /subscriptions/ffffffff-ffff-ffff-ffff-  
ffffffffffff/resourceGroups/mykvrg/providers/Microsoft.KeyVault/vaults/mykvname  
CertificateUrl: https://myvault.vault.azure.net/secrets/mycert/00000000000000000000000000000000  
CertificateThumbprint: 0xfffffffffffffffffffffffffffff
```

The certificate's subject name must match the domain that you use to access the Service Fabric cluster. This match is required to provide an SSL for the cluster's HTTPS management endpoints and Service Fabric Explorer. You cannot obtain an SSL certificate from a CA for the `.cloudapp.azure.com` domain. You must obtain a custom domain name for your cluster. When you request a certificate from a CA, the certificate's subject name must match the custom domain name that you use for your cluster.

These subject names are the entries you need to create a secure Service Fabric cluster (without Azure AD), as described at [Configure Resource Manager template parameters](#). You can connect to the secure cluster by following the instructions for [authenticating client access to a cluster](#). Linux clusters do not support Azure AD authentication. You can assign admin and client roles as described in the [Assign roles to users](#) section. When you specify admin and client roles for a Linux cluster, you have to provide certificate thumbprints for authentication. You do not provide the subject name, because no chain validation or revocation is being performed.

If you want to use a self-signed certificate for testing, you can use the same script to generate one. You can then upload the certificate to your key vault by providing the flag `ss` instead of providing the certificate path and certificate name. For example, see the following command for creating and uploading a self-signed certificate:

```
./cert_helper.py ss -rgname "mykvrg" -sub "ffffffff-ffff-ffff-ffff-ffffffffffff" -kv "mykevname" -sname  
"mycert" -l "East US" -p "selftest" -subj "mytest.eastus.cloudapp.net"
```

This command returns the same three strings: `SourceVault`, `CertificateUrl`, and `CertificateThumbprint`. You can then use the strings to create both a secure Linux cluster and a location where the self-signed certificate is placed. You need the self-signed certificate to connect to the cluster. You can connect to the secure cluster by following the instructions for [authenticating client access to a cluster](#).

The certificate's subject name must match the domain that you use to access the Service Fabric cluster. This match is required to provide an SSL for the cluster's HTTPS management endpoints and Service Fabric Explorer. You cannot obtain an SSL certificate from a CA for the `.cloudapp.azure.com` domain. You must obtain a custom domain name for your cluster. When you request a certificate from a CA, the certificate's subject name must match the custom domain name that you use for your cluster.

You can fill the parameters from the helper script in the Azure portal, as described in the [Create a cluster in the Azure portal](#) section.

## Next steps

At this point, you have a secure cluster with Azure Active Directory providing management authentication. Next, [connect to your cluster](#) and learn how to [manage application secrets](#).

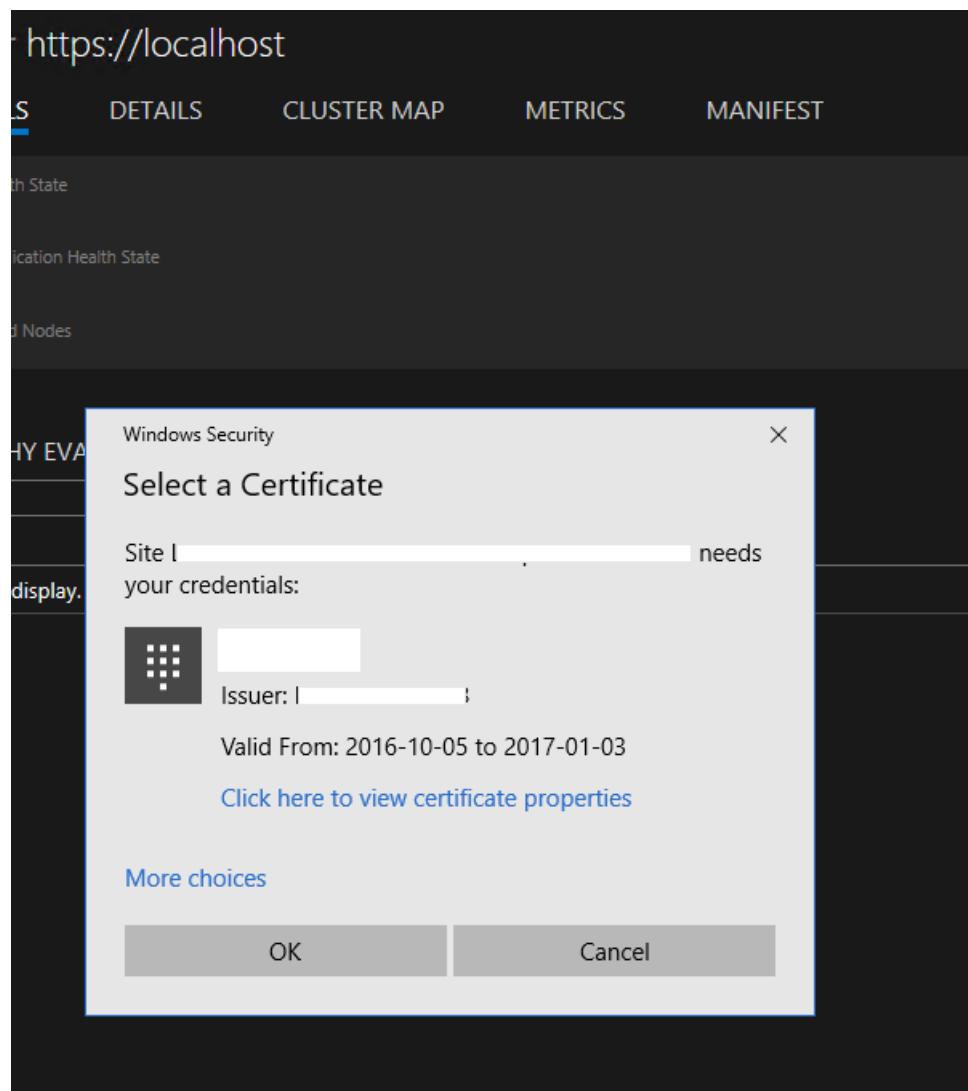
## Troubleshoot setting up Azure Active Directory for client authentication

If you run into an issue while you're setting up Azure AD for client authentication, review the potential solutions in this section.

### Service Fabric Explorer prompts you to select a certificate

#### Problem

After you sign in successfully to Azure AD in Service Fabric Explorer, the browser returns to the home page but a message prompts you to select a certificate.



#### **Reason**

The user isn't assigned a role in the Azure AD cluster application. Thus, Azure AD authentication fails on Service Fabric cluster. Service Fabric Explorer falls back to certificate authentication.

#### **Solution**

Follow the instructions for setting up Azure AD, and assign user roles. Also, we recommend that you turn on "User assignment required to access app," as `SetupApplications.ps1` does.

### **Connection with PowerShell fails with an error: "The specified credentials are invalid"**

#### **Problem**

When you use PowerShell to connect to the cluster by using "AzureActiveDirectory" security mode, after you sign in successfully to Azure AD, the connection fails with an error: "The specified credentials are invalid."

#### **Solution**

This solution is the same as the preceding one.

### **Service Fabric Explorer returns a failure when you sign in: "AADSTS50011"**

#### **Problem**

When you try to sign in to Azure AD in Service Fabric Explorer, the page returns a failure: "AADSTS50011: The reply address <url> does not match the reply addresses configured for the application: <guid>."

## Sign In

Sorry, but we're having trouble signing you in.

We received a bad request.

Additional technical information:

Correlation ID: e3a88cc6-92f2-47df-9220-fd92a752d3c5

Timestamp: 2016-09-09 23:12:53Z

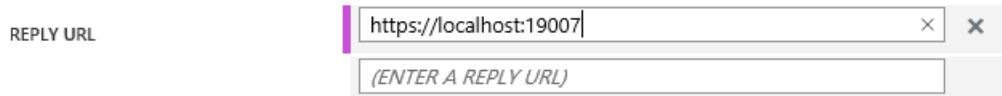
AADSTS50011: The reply address 'https://localhost:19007/Explorer/index.html' does not match the reply addresses configured for the application: 'e0f1e177-401f-4977-9807-5956b013a91a'. More details: not specified

#### **Reason**

The cluster (web) application that represents Service Fabric Explorer attempts to authenticate against Azure AD, and as part of the request it provides the redirect return URL. But the URL is not listed in the Azure AD application **REPLY URL** list.

#### **Solution**

On the **Configure** tab of the cluster (web) application, add the URL of Service Fabric Explorer to the **REPLY URL** list or replace one of the items in the list. When you have finished, save your change.



## Connect the cluster by using Azure AD authentication via PowerShell

To connect the Service Fabric cluster, use the following PowerShell command example:

```
Connect-ServiceFabricCluster -ConnectionEndpoint <endpoint> -KeepAliveIntervalInSec 10 -  
AzureActiveDirectory -ServerCertThumbprint <thumbprint>
```

To learn about the `Connect-ServiceFabricCluster` cmdlet, see [Connect-ServiceFabricCluster](#).

## Can I reuse the same Azure AD tenant in multiple clusters?

Yes. But remember to add the URL of Service Fabric Explorer to your cluster (web) application. Otherwise, Service Fabric Explorer doesn't work.

## Why do I still need a server certificate while Azure AD is enabled?

FabricClient and FabricGateway perform a mutual authentication. During Azure AD authentication, Azure AD integration provides a client identity to the server, and the server certificate is used to verify the server identity. For more information about Service Fabric certificates, see [X.509 certificates and Service Fabric](#).

# Scale a Service Fabric cluster in or out using auto-scale rules

10/5/2017 • 6 min to read • [Edit Online](#)

Virtual machine scale sets are an Azure compute resource that you can use to deploy and manage a collection of virtual machines as a set. Every node type that is defined in a Service Fabric cluster is set up as a separate Virtual Machine scale set. Each node type can then be scaled in or out independently, have different sets of ports open, and can have different capacity metrics. Read more about it in the [Service Fabric nodetypes](#) document. Since the Service Fabric node types in your cluster are made of Virtual Machine scale sets at the backend, you need to set up auto-scale rules for each node type/Virtual Machine scale set.

## NOTE

Your subscription must have enough cores to add the new VMs that make up this cluster. There is no model validation currently, so you get a deployment time failure, if any of the quota limits are hit.

## Choose the node type/Virtual Machine scale set to scale

Currently, you are not able to specify the auto-scale rules for Virtual Machine scale sets using the portal, so let us use Azure PowerShell (1.0+) to list the node types and then add auto-scale rules to them.

To get the list of Virtual Machine scale set that make up your cluster, run the following cmdlets:

```
Get-AzureRmResource -ResourceGroupName <RGname> -ResourceType Microsoft.Compute/VirtualMachineScaleSets  
Get-AzureRmVmss -ResourceGroupName <RGname> -VMScaleSetName <Virtual Machine scale set name>
```

## Set auto-scale rules for the node type/Virtual Machine scale set

If your cluster has multiple node types, then repeat this for each node types/Virtual Machine scale sets that you want to scale (in or out). Take into account the number of nodes that you must have before you set up auto-scaling. The minimum number of nodes that you must have for the primary node type is driven by the reliability level you have chosen. Read more about [reliability levels](#).

## NOTE

Scaling down the primary node type to less than the minimum number make the cluster unstable or bring it down. This could result in data loss for your applications and for the system services.

Currently the auto-scale feature is not driven by the loads that your applications may be reporting to Service Fabric. So at this time the auto-scale you get is purely driven by the performance counters that are emitted by each of the Virtual Machine scale set instances.

Follow these instructions [to set up auto-scale for each Virtual Machine scale set](#).

**NOTE**

In a scale down scenario, unless your node type has a durability level of Gold or Silver you need to call the [Remove-ServiceFabricNodeState cmdlet](#) with the appropriate node name.

## Manually add VMs to a node type/Virtual Machine scale set

Follow the sample/instructions in the [quick start template gallery](#) to change the number of VMs in each Nodetype.

**NOTE**

Adding of VMs takes time, so do not expect the additions to be instantaneous. So plan to add capacity well in time, to allow for over 10 minutes before the VM capacity is available for the replicas/ service instances to get placed.

## Manually remove VMs from the primary node type/Virtual Machine scale set

**NOTE**

The service fabric system services run in the Primary node type in your cluster. So should never shut down or scale down the number of instances in that node types less than what the reliability tier warrants. Refer to [the details on reliability tiers here](#).

You need to execute the following steps one VM instance at a time. This allows for the system services (and your stateful services) to be shut down gracefully on the VM instance you are removing and new replicas created on other nodes.

1. Run [Disable-ServiceFabricNode](#) with intent 'RemoveNode' to disable the node you're going to remove (the highest instance in that node type).
2. Run [Get-ServiceFabricNode](#) to make sure that the node has indeed transitioned to disabled. If not, wait until the node is disabled. You cannot hurry this step.
3. Follow the sample/instructions in the [quick start template gallery](#) to change the number of VMs by one in that Nodetype. The instance removed is the highest VM instance.
4. Repeat steps 1 through 3 as needed, but never scale down the number of instances in the primary node types less than what the reliability tier warrants. Refer to [the details on reliability tiers here](#).

## Manually remove VMs from the non-primary node type/Virtual Machine scale set

**NOTE**

For a stateful service, you need a certain number of nodes to be always up to maintain availability and preserve state of your service. At the very minimum, you need the number of nodes equal to the target replica set count of the partition/service.

You need to execute the following steps one VM instance at a time. This allows for the system services (and your stateful services) to be shut down gracefully on the VM instance you are removing and new replicas created elsewhere.

1. Run [Disable-ServiceFabricNode](#) with intent 'RemoveNode' to disable the node you're going to remove (the highest instance in that node type).
2. Run [Get-ServiceFabricNode](#) to make sure that the node has indeed transitioned to disabled. If not wait till the

node is disabled. You cannot hurry this step.

3. Follow the sample/instructions in the [quick start template gallery](#) to change the number of VMs by one in that Nodetype. This will now remove the highest VM instance.
4. Repeat steps 1 through 3 as needed, but never scale down the number of instances in the primary node types less than what the reliability tier warrants. Refer to [the details on reliability tiers here](#).

## Behaviors you may observe in Service Fabric Explorer

When you scale up a cluster the Service Fabric Explorer will reflect the number of nodes (Virtual Machine scale set instances) that are part of the cluster. However, when you scale a cluster down you will see the removed node/VM instance displayed in an unhealthy state unless you call [Remove-ServiceFabricNodeState cmd](#) with the appropriate node name.

Here is the explanation for this behavior.

The nodes listed in Service Fabric Explorer are a reflection of what the Service Fabric system services (FM specifically) knows about the number of nodes the cluster had/has. When you scale the Virtual Machine scale set down, the VM was deleted but FM system service still thinks that the node (that was mapped to the VM that was deleted) will come back. So Service Fabric Explorer continues to display that node (though the health state may be error or unknown).

In order to make sure that a node is removed when a VM is removed, you have two options:

- 1) Choose a durability level of Gold or Silver for the node types in your cluster, which gives you the infrastructure integration. Which will then automatically remove the nodes from our system services (FM) state when you scale down. Refer to [the details on durability levels here](#)
- 2) Once the VM instance has been scaled down, you need to call the [Remove-ServiceFabricNodeState cmdlet](#).

### NOTE

Service Fabric clusters require a certain number of nodes to be up at all the time in order to maintain availability and preserve state - referred to as "maintaining quorum." So, it is typically unsafe to shut down all the machines in the cluster unless you have first performed a [full backup of your state](#).

## Next steps

Read the following to also learn about planning cluster capacity, upgrading a cluster, and partitioning services:

- [Plan your cluster capacity](#)
- [Cluster upgrades](#)
- [Partition stateful services for maximum scale](#)

# Scale a Service Fabric cluster programmatically

10/25/2017 • 6 min to read • [Edit Online](#)

Fundamentals of scaling a Service Fabric cluster in Azure are covered in documentation on [cluster scaling](#). That article covers how Service Fabric clusters are built on top of virtual machine scale sets and can be scaled either manually or with auto-scale rules. This document looks at programmatic methods of coordinating Azure scaling operations for more advanced scenarios.

## Reasons for programmatic scaling

In many scenarios, scaling manually or via auto-scale rules are good solutions. In other scenarios, though, they may not be the right fit. Potential drawbacks to these approaches include:

- Manually scaling requires you to log in and explicitly request scaling operations. If scaling operations are required frequently or at unpredictable times, this approach may not be a good solution.
- When auto-scale rules remove an instance from a virtual machine scale set, they do not automatically remove knowledge of that node from the associated Service Fabric cluster unless the node type has a durability level of Silver or Gold. Because auto-scale rules work at the scale set level (rather than at the Service Fabric level), auto-scale rules can remove Service Fabric nodes without shutting them down gracefully. This rude node removal will leave 'ghost' Service Fabric node state behind after scale-in operations. An individual (or a service) would need to periodically clean up removed node state in the Service Fabric cluster.
  - A node type with a durability level of Gold or Silver automatically cleans up removed nodes, so no additional clean-up is needed.
- Although there are [many metrics](#) supported by auto-scale rules, it is still a limited set. If your scenario calls for scaling based on some metric not covered in that set, then auto-scale rules may not be a good option.

Based on these limitations, you may wish to implement more customized automatic scaling models.

## Scaling APIs

Azure APIs exist which allow applications to programmatically work with virtual machine scale sets and Service Fabric clusters. If existing auto-scale options don't work for your scenario, these APIs make it possible to implement custom scaling logic.

One approach to implementing this 'home-made' auto-scaling functionality is to add a new stateless service to the Service Fabric application to manage scaling operations. Within the service's `RunAsync` method, a set of triggers can determine if scaling is required (including checking parameters such as maximum cluster size and scaling cooldowns).

The API used for virtual machine scale set interactions (both to check the current number of virtual machine instances and to modify it) is the [fluent Azure Management Compute library](#). The fluent compute library provides an easy-to-use API for interacting with virtual machine scale sets.

To interact with the Service Fabric cluster itself, use `System.Fabric.FabricClient`.

Of course, the scaling code doesn't need to run as a service in the cluster to be scaled. Both `IAzure` and `FabricClient` can connect to their associated Azure resources remotely, so the scaling service could easily be a console application or Windows service running from outside the Service Fabric application.

## Credential management

One challenge of writing a service to handle scaling is that the service must be able to access virtual machine scale set resources without an interactive login. Accessing the Service Fabric cluster is easy if the scaling service is modifying its own Service Fabric application, but credentials are needed to access the scale set. To log in, you can use a [service principal](#) created with the [Azure CLI 2.0](#).

A service principal can be created with the following steps:

1. Log in to the Azure CLI (`az login`) as a user with access to the virtual machine scale set
2. Create the service principal with `az ad sp create-for-rbac`
  - a. Make note of the appId (called 'client ID' elsewhere), name, password, and tenant for later use.
  - b. You will also need your subscription ID, which can be viewed with `az account list`

The fluent compute library can log in using these credentials as follows (note that core fluent Azure types like `IAzure` are in the [Microsoft.Azure.Management.Fluent](#) package):

```
var credentials = new AzureCredentials(new ServicePrincipalLoginInformation {
    ClientId = AzureClientId,
    ClientSecret =
        AzureClientKey }, AzureTenantId, AzureEnvironment.AzureGlobalCloud);
IAzure AzureClient = Azure.Authenticate(credentials).WithSubscription(AzureSubscriptionId);

if (AzureClient?.SubscriptionId == AzureSubscriptionId)
{
    ServiceEventSource.Current.ServiceMessage(Context, "Successfully logged into Azure");
}
else
{
    ServiceEventSource.Current.ServiceMessage(Context, "ERROR: Failed to login to Azure");
}
```

Once logged in, scale set instance count can be queried via

```
AzureClient.VirtualMachineScaleSets.GetById(ScaleSetId).Capacity .
```

## Scaling out

Using the fluent Azure compute SDK, instances can be added to the virtual machine scale set with just a few calls -

```
var scaleSet = AzureClient.VirtualMachineScaleSets.GetById(ScaleSetId);
var newCapacity = (int)Math.Min(MaximumNodeCount, scaleSet.Capacity + 1);
scaleSet.Update().WithCapacity(newCapacity).Apply();
```

Alternatively, virtual machine scale set size can also be managed with PowerShell cmdlets. [Get-AzureRmVmss](#) can retrieve the virtual machine scale set object. The current capacity is available through the `.sku.capacity` property. After changing the capacity to the desired value, the virtual machine scale set in Azure can be updated with the [Update-AzureRmVmss](#) command.

As when adding a node manually, adding a scale set instance should be all that's needed to start a new Service Fabric node since the scale set template includes extensions to automatically join new instances to the Service Fabric cluster.

## Scaling in

Scaling in is similar to scaling out. The actual virtual machine scale set changes are practically the same. But, as was discussed previously, Service Fabric only automatically cleans up removed nodes with a durability of Gold or Silver. So, in the Bronze-durability scale-in case, it's necessary to interact with the Service Fabric cluster to shut down the node to be removed and then to remove its state.

Preparing the node for shutdown involves finding the node to be removed (the most recently added node) and deactivating it. For non-seed nodes, newer nodes can be found by comparing `NodeInstanceId`.

```
using (var client = new FabricClient())
{
    var mostRecentLiveNode = (await client.QueryManager.GetNodeListAsync())
        .Where(n => n.NodeType.Equals(NodeTypeToScale, StringComparison.OrdinalIgnoreCase))
        .Where(n => n.NodeStatus == System.Fabric.Query.NodeStatus.Up)
        .OrderByDescending(n => n.NodeInstanceId)
        .FirstOrDefault();
```

Seed nodes are different and don't necessarily follow the convention that greater instance IDs are removed first.

Once the node to be removed is found, it can be deactivated and removed using the same `FabricClient` instance and the `IAzure` instance from earlier.

```
var scaleSet = AzureClient.VirtualMachineScaleSets.GetById(ScaleSetId);

// Remove the node from the Service Fabric cluster
ServiceEventSource.Current.ServiceMessage(Context, $"Disabling node {mostRecentLiveNode.NodeName}");
await client.ClusterManager.DeactivateNodeAsync(mostRecentLiveNode.NodeName,
NodeDeactivationIntent.RemoveNode);

// Wait (up to a timeout) for the node to gracefully shutdown
var timeout = TimeSpan.FromMinutes(5);
var waitStart = DateTime.Now;
while ((mostRecentLiveNode.NodeStatus == System.Fabric.Query.NodeStatus.Up || mostRecentLiveNode.NodeStatus ==
System.Fabric.Query.NodeStatus.Disabling) &&
DateTime.Now - waitStart < timeout)
{
    mostRecentLiveNode = (await client.QueryManager.GetNodeListAsync()).FirstOrDefault(n => n.NodeName ==
mostRecentLiveNode.NodeName);
    await Task.Delay(10 * 1000);
}

// Decrement VMSS capacity
var newCapacity = (int)Math.Max(MinimumNodeCount, scaleSet.Capacity - 1); // Check min count

scaleSet.Update().WithCapacity(newCapacity).Apply();
```

As with scaling out, PowerShell cmdlets for modifying virtual machine scale set capacity can also be used here if a scripting approach is preferable. Once the virtual machine instance is removed, Service Fabric node state can be removed.

```
await client.ClusterManager.RemoveNodeStateAsync(mostRecentLiveNode.NodeName);
```

## Potential drawbacks

As demonstrated in the preceding code snippets, creating your own scaling service provides the highest degree of control and customizability over your application's scaling behavior. This can be useful for scenarios requiring precise control over when or how an application scales in or out. However, this control comes with a tradeoff of code complexity. Using this approach means that you need to own scaling code, which is non-trivial.

How you should approach Service Fabric scaling depends on your scenario. If scaling is uncommon, the ability to add or remove nodes manually is probably sufficient. For more complex scenarios, auto-scale rules and SDKs exposing the ability to scale programmatically offer powerful alternatives.

## Next steps

To get started implementing your own auto-scaling logic, familiarize yourself with the following concepts and useful APIs:

- [Scaling manually or with auto-scale rules](#)
- [Fluent Azure Management Libraries for .NET](#) (useful for interacting with a Service Fabric cluster's underlying virtual machine scale sets)
- [System.Fabric.FabricClient](#) (useful for interacting with a Service Fabric cluster and its nodes)

# Upgrade an Azure Service Fabric cluster

8/11/2017 • 10 min to read • [Edit Online](#)

For any modern system, designing for upgradability is key to achieving long-term success of your product. An Azure Service Fabric cluster is a resource that you own, but is partly managed by Microsoft. This article describes what is managed automatically and what you can configure yourself.

## Controlling the fabric version that runs on your Cluster

You can set your cluster to receive automatic fabric upgrades as they are released by Microsoft or you can select a supported fabric version you want your cluster to be on.

You do this by setting the "upgradeMode" cluster configuration on the portal or using Resource Manager at the time of creation or later on a live cluster

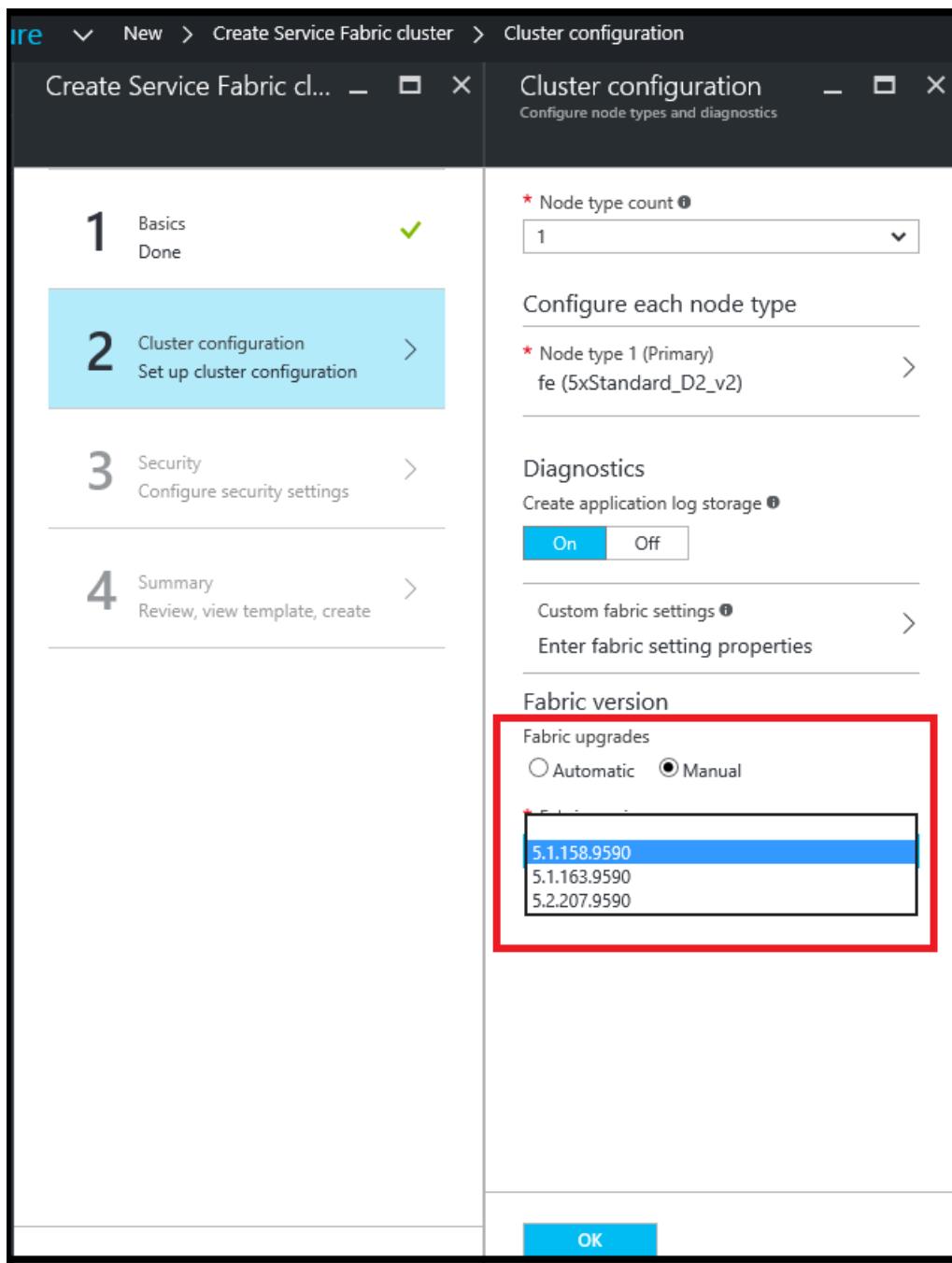
### NOTE

Make sure to keep your cluster running a supported fabric version always. As and when we announce the release of a new version of service fabric, the previous version is marked for end of support after a minimum of 60 days from that date. The new releases are announced [on the service fabric team blog](#). The new release is available to choose then.

14 days prior to the expiry of the release your cluster is running, a health event is generated that puts your cluster into a warning health state. The cluster remains in a warning state until you upgrade to a supported fabric version.

### Setting the upgrade mode via portal

You can set the cluster to automatic or manual when you are creating the cluster.



You can set the cluster to automatic or manual when on a live cluster, using the manage experience.

#### **Upgrading to a new version on a cluster that is set to Manual mode via portal.**

To upgrade to a new version, all you need to do is select the available version from the dropdown and save. The Fabric upgrade gets kicked off automatically. The cluster health policies (a combination of node health and the health all the applications running in the cluster) are adhered to during the upgrade.

If the cluster health policies are not met, the upgrade is rolled back. Scroll down this document to read more on how to set those custom health policies.

Once you have fixed the issues that resulted in the rollback, you need to initiate the upgrade again, by following the same steps as before.

The screenshot shows the Azure portal interface for managing Service Fabric clusters. On the left, a sidebar lists 'Subscriptions' and 'Service Fabric clusters'. The main area shows details for the cluster 'chackodemo02 - Fabric upgrades'. The 'Fabric upgrades' section is highlighted with a red box. It displays the 'Current fabric version' as 5.2.207.9590. Under 'Fabric upgrades', there are two radio buttons: 'Automatic' (selected) and 'Manual'. A dropdown menu for 'Fabric version' is set to 5.2.207.9590. Below this, there is a checkbox for 'Advanced upgrade settings'. The left sidebar also includes sections for Overview, Activity log, Access control (IAM), Tags, Node types, Nodes, Applications, Security, Custom fabric settings, Properties, Locks, and Automation script.

### Setting the upgrade mode via a Resource Manager template

Add the "upgradeMode" configuration to the Microsoft.ServiceFabric/clusters resource definition and set the "clusterCodeVersion" to one of the supported fabric versions as shown below and then deploy the template. The valid values for "upgradeMode" are "Manual" or "Automatic"

```

    "apiVersion": "2016-09-01",
    "type": "Microsoft.ServiceFabric/clusters",
    "name": "[parameters('clusterName')]",
    "location": "[parameters('clusterLocation')]",
    "dependsOn": [
        "[concat('Microsoft.Storage/storageAccounts/', parameters('supportLogStorageAccountName'))]"
    ],
    "properties": {
        "clientCertificateCommonNames": [],
        "clientCertificateThumbprints": [],
        "clusterCodeVersion": "5.3.121.9494", [Red Box]
        "clusterState": "Default",
        "diagnosticsStorageAccountConfig": {
            "blobEndpoint": "[reference(concat('Microsoft.Storage/storageAccounts/', parameters('supportLogStorageAccountName')))]",
            "protectedAccountKeyName": "StorageAccountKey1",
            "queueEndpoint": "[reference(concat('Microsoft.Storage/storageAccounts/', parameters('supportLogStorageAccountName')))]",
            "storageAccountName": "[parameters('supportLogStorageAccountName')]",
            "tableEndpoint": "[reference(concat('Microsoft.Storage/storageAccounts/', parameters('supportLogStorageAccountName')))]"
        },
        "fabricSettings": [],
        "managementEndpoint": "[concat('http://', reference(concat(parameters('lbIPName'), parameters('lbIPPort'))))]",
        "nodeTypes": [
            {
                "name": "[parameters('vmNodeType0Name')]",
                "applicationPorts": {
                    "endPort": "[parameters('nt0applicationEndPort')]",
                    "startPort": "[parameters('nt0applicationStartPort')]"
                },
                "clientConnectionEndpointPort": "[parameters('nt0fabricTcpGatewayPort')]",
                "durabilityLevel": "Bronze",
                "ephemeralPorts": {
                    "endPort": "[parameters('nt0ephemeralEndPort')]",
                    "startPort": "[parameters('nt0ephemeralStartPort')]"
                },
                "httpGatewayEndpointPort": "[parameters('nt0fabricHttpGatewayPort')]",
                "isPrimary": true,
                "vmInstanceCount": "[parameters('nt0InstanceCount')]"
            }
        ],
        "provisioningState": "Default",
        "reliabilityLevel": "Silver",
        "upgradeMode": "Manual", [Red Box]
        "vmImage": "Windows"
    }
}

```

#### **Upgrading to a new version on a cluster that is set to Manual mode via a Resource Manager template.**

When the cluster is in Manual mode, to upgrade to a new version, change the "clusterCodeVersion" to a supported version and deploy it. The deployment of the template, kicks off the Fabric upgrade gets kicked off automatically. The cluster health policies (a combination of node health and the health all the applications running in the cluster) are adhered to during the upgrade.

If the cluster health policies are not met, the upgrade is rolled back. Scroll down this document to read more on how to set those custom health policies.

Once you have fixed the issues that resulted in the rollback, you need to initiate the upgrade again, by following the same steps as before.

#### **Get list of all available version for all environments for a given subscription**

Run the following command, and you should get an output similar to this.

"supportExpiryUtc" tells you when a given release is expiring or has expired. The latest release does not have a valid date - it has a value of "9999-12-31T23:59:59.9999999", which just means that the expiry date is not yet set.

GET

```
https://<endpoint>/subscriptions/{{subscriptionId}}/providers/Microsoft.ServiceFabric/locations/{{location}}/clusterVersions?api-version=2016-09-01
```

Example: <https://management.azure.com/subscriptions/1857f442-3bce-4b96-ad95-627f76437a67/providers/Microsoft.ServiceFabric/locations/eastus/clusterVersions?api-version=2016-09-01>

Output:

```
{
    "value": [
        {
            "id": "subscriptions/35349203-a0b3-405e-8a23-9f1450984307/providers/Microsoft.ServiceFabric/environments/Windows/clusterVersions/5.0.1427.9490",
            "name": "5.0.1427.9490",
            "type": "Microsoft.ServiceFabric/environments/clusterVersions",
            "properties": {
                "codeVersion": "5.0.1427.9490",
                "supportExpiryUtc": "2016-11-26T23:59:59.999999",
                "environment": "Windows"
            }
        },
        {
            "id": "subscriptions/35349203-a0b3-405e-8a23-9f1450984307/providers/Microsoft.ServiceFabric/environments/Windows/clusterVersions/4.0.1427.9490",
            "name": "5.1.1427.9490",
            "type": "Microsoft.ServiceFabric/environments/clusterVersions",
            "properties": {
                "codeVersion": "5.1.1427.9490",
                "supportExpiryUtc": "9999-12-31T23:59:59.999999",
                "environment": "Windows"
            }
        },
        {
            "id": "subscriptions/35349203-a0b3-405e-8a23-9f1450984307/providers/Microsoft.ServiceFabric/environments/Windows/clusterVersions/4.4.1427.9490",
            "name": "4.4.1427.9490",
            "type": "Microsoft.ServiceFabric/environments/clusterVersions",
            "properties": {
                "codeVersion": "4.4.1427.9490",
                "supportExpiryUtc": "9999-12-31T23:59:59.999999",
                "environment": "Linux"
            }
        }
    ]
}
```

## Fabric upgrade behavior when the cluster Upgrade Mode is Automatic

Microsoft maintains the fabric code and configuration that runs in an Azure cluster. We perform automatic monitored upgrades to the software on an as-needed basis. These upgrades could be code, configuration, or both. To make sure that your application suffers no impact or minimal impact due to these upgrades, we perform the upgrades in the following phases:

### Phase 1: An upgrade is performed by using all cluster health policies

During this phase, the upgrades proceed one upgrade domain at a time, and the applications that were running in the cluster continue to run without any downtime. The cluster health policies (a combination of node health and the health all the applications running in the cluster) are adhered to during the upgrade.

If the cluster health policies are not met, the upgrade is rolled back. Then an email is sent to the owner of the subscription. The email contains the following information:

- Notification that we had to roll back a cluster upgrade.

- Suggested remedial actions, if any.
- The number of days (n) until we execute Phase 2.

We try to execute the same upgrade a few more times in case any upgrades failed for infrastructure reasons. After the n days from the date the email was sent, we proceed to Phase 2.

If the cluster health policies are met, the upgrade is considered successful and marked complete. This can happen during the initial upgrade or any of the upgrade reruns in this phase. There is no email confirmation of a successful run. This is to avoid sending you too many emails--receiving an email should be seen as an exception to normal. We expect most of the cluster upgrades to succeed without impacting your application availability.

### **Phase 2: An upgrade is performed by using default health policies only**

The health policies in this phase are set in such a way that the number of applications that were healthy at the beginning of the upgrade remains the same for the duration of the upgrade process. As in Phase 1, the Phase 2 upgrades proceed one upgrade domain at a time, and the applications that were running in the cluster continue to run without any downtime. The cluster health policies (a combination of node health and the health all the applications running in the cluster) are adhered to for the duration of the upgrade.

If the cluster health policies in effect are not met, the upgrade is rolled back. Then an email is sent to the owner of the subscription. The email contains the following information:

- Notification that we had to roll back a cluster upgrade.
- Suggested remedial actions, if any.
- The number of days (n) until we execute Phase 3.

We try to execute the same upgrade a few more times in case any upgrades failed for infrastructure reasons. A reminder email is sent a couple of days before n days are up. After the n days from the date the email was sent, we proceed to Phase 3. The emails we send you in Phase 2 must be taken seriously and remedial actions must be taken.

If the cluster health policies are met, the upgrade is considered successful and marked complete. This can happen during the initial upgrade or any of the upgrade reruns in this phase. There is no email confirmation of a successful run.

### **Phase 3: An upgrade is performed by using aggressive health policies**

These health policies in this phase are geared towards completion of the upgrade rather than the health of the applications. Very few cluster upgrades end up in this phase. If your cluster gets to this phase, there is a good chance that your application becomes unhealthy and/or lose availability.

Similar to the other two phases, Phase 3 upgrades proceed one upgrade domain at a time.

If the cluster health policies are not met, the upgrade is rolled back. We try to execute the same upgrade a few more times in case any upgrades failed for infrastructure reasons. After that, the cluster is pinned, so that it will no longer receive support and/or upgrades.

An email with this information is sent to the subscription owner, along with the remedial actions. We do not expect any clusters to get into a state where Phase 3 has failed.

If the cluster health policies are met, the upgrade is considered successful and marked complete. This can happen during the initial upgrade or any of the upgrade reruns in this phase. There is no email confirmation of a successful run.

## **Cluster configurations that you control**

In addition to the ability to set the cluster upgrade mode, Here are the configurations that you can change on a live cluster.

## Certificates

You can add new or delete certificates for the cluster and client via the portal easily. Refer to [this document](#) for detailed instructions

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with 'QUICK ACCESS' and 'SETTINGS' sections. Under 'SETTINGS', 'Security' is selected. In the main area, the 'chackosecure2 - Security' blade is open, showing 'Cluster certificates' and 'Admin client' sections. A 'Add certificate...' blade is open on the right, titled 'chackosecure2'. It has a 'Certificate type' dropdown set to 'Secondary certificate thumbprint' and a 'Certificate thumbprint' input field which is highlighted with a red box.

## Application ports

You can change application ports by changing the Load Balancer resource properties that are associated with the node type. You can use the portal, or you can use Resource Manager PowerShell directly.

To open a new port on all VMs in a node type, do the following:

1. Add a new probe to the appropriate load balancer.

If you deployed your cluster by using the portal, the load balancers are named "LB-name of the Resource group-NodeTypename", one for each node type. Since the load balancer names are unique only within a resource group, it is best if you search for them under a specific resource group.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with 'QUICK ACCESS' and 'SETTINGS' sections. Under 'SETTINGS', 'Load balancers' is selected. In the main area, the 'LB-chackosecure5-nt1vm - Probes' blade is open, showing a list of existing probes. An 'Add probe' blade is open on the right, titled 'LB-chackosecure5-nt1vm'. It has fields for 'Name' (set to 'MyNewAppProbe'), 'Protocol' (set to 'HTTP'), 'Port' (set to '8087'), 'Path' (set to '/'), 'Interval' (set to '5 seconds'), and 'Unhealthy threshold' (set to '2 consecutive failures').

2. Add a new rule to the load balancer.

Add a new rule to the same load balancer by using the probe that you created in the previous step.

The screenshot shows the Azure portal interface for managing load balancing rules. On the left, a sidebar lists resources: Subscription ID (6c653126-e4ba-42cd-a1dd-f7bf96ae7a47), Location (West US), TYPE (Virtual machine, Load balancer, Public IP address, Virtual network, Service Fabric, Storage account, Storage account), LOCATION (West US), and SETTINGS (Overview, Activity log, Access control (IAM), Tags, Load balancing rules, Probes, IP address, Backend pools, Inbound NAT rules, Properties, Locks, Automation script). The 'Load balancing rules' section is selected. In the main pane, a table lists existing rules: AppPortLBRule1, AppPortLBRule2, LBHttpRule, and LBRule. A red box highlights the '+ Add' button. To the right, a detailed configuration form is shown for creating a new rule:

- Name:** MyNewAppProbeRule
- Protocol:** TCP
- Port:** 8087
- Backend port:** 8087
- Backend pool:** LoadBalancerBEAddressPool
- Probe:** MyNewAppProbe (HTTP:8087)
- Session persistence:** None
- Idle timeout (minutes):** 4
- Floating IP (direct server return):** Enabled

## Placement properties

For each of the node types, you can add custom placement properties that you want to use in your applications. NodeType is a default property that you can use without adding it explicitly.

### NOTE

For details on the use of placement constraints, node properties, and how to define them, refer to the section "Placement Constraints and Node Properties" in the Service Fabric Cluster Resource Manager Document on [Describing Your Cluster](#).

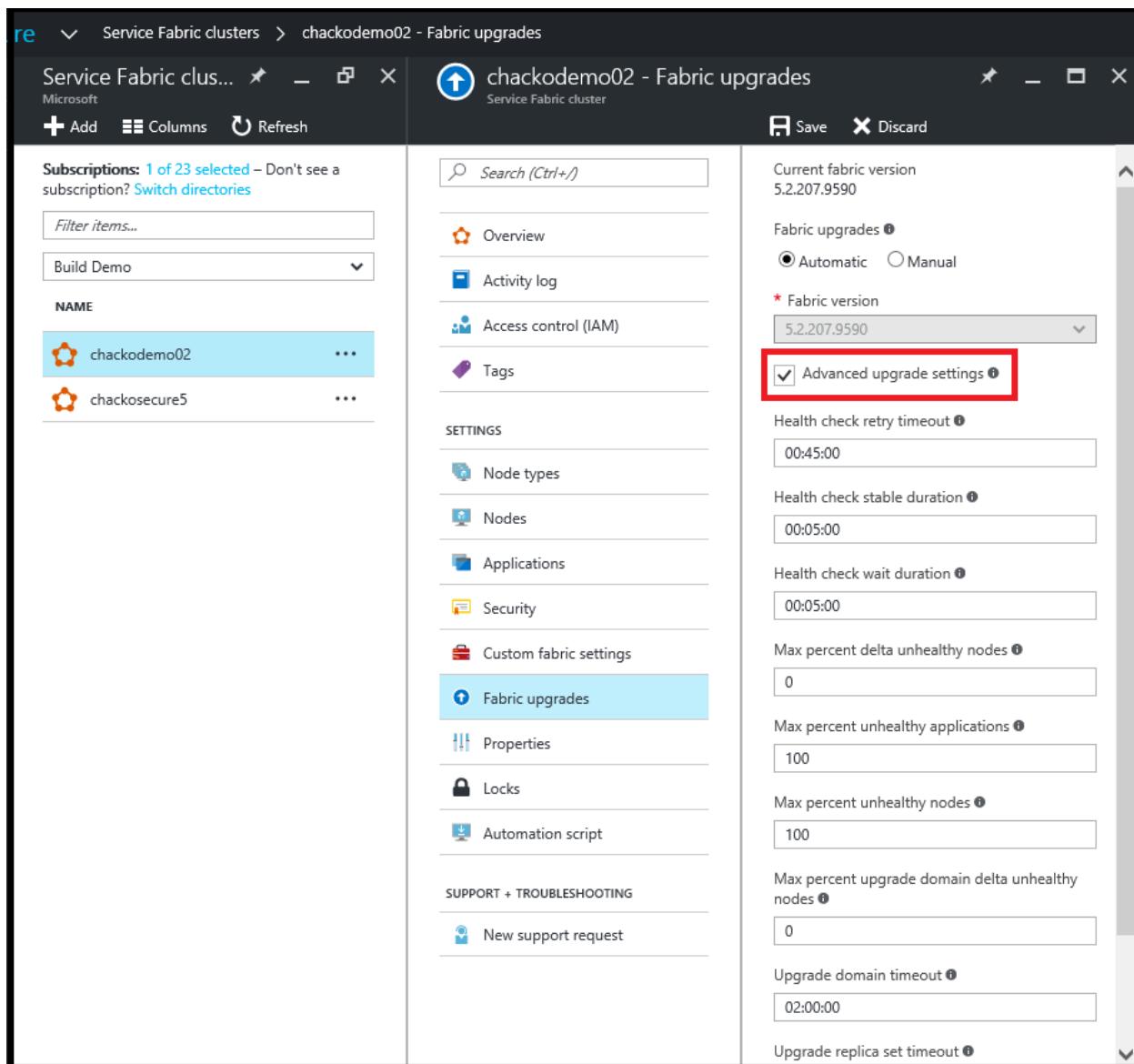
## Capacity metrics

For each of the node types, you can add custom capacity metrics that you want to use in your applications to report load. For details on the use of capacity metrics to report load, refer to the Service Fabric Cluster Resource Manager Documents on [Describing Your Cluster](#) and [Metrics and Load](#).

## Fabric upgrade settings - Health polices

You can specify custom health policies for fabric upgrade. If you have set your cluster to Automatic fabric upgrades, then these policies get applied to the Phase-1 of the automatic fabric upgrades. If you have set your cluster for Manual fabric upgrades, then these policies get applied each time you select a new version triggering the system to kick off the fabric upgrade in your cluster. If you do not override the policies, the defaults are used.

You can specify the custom health policies or review the current settings under the "fabric upgrade" blade, by selecting the advanced upgrade settings. Review the following picture on how to.



## Customize Fabric settings for your cluster

Refer to [service fabric cluster fabric settings](#) on what and how you can customize them.

## OS patches on the VMs that make up the cluster

Refer to [Patch Orchestration Application](#) which can be deployed on your cluster to install patches from Windows Update in an orchestrated manner, keeping the services available all the time.

## OS upgrades on the VMs that make up the cluster

If you must upgrade the OS image on the virtual machines of the cluster, you must do it one VM at a time. You are responsible for this upgrade--there is currently no automation for this.

## Next steps

- Learn how to customize some of the [service fabric cluster fabric settings](#)
- Learn how to [scale your cluster in and out](#)
- Learn about [application upgrades](#)

# Role-based access control for Service Fabric clients

8/15/2017 • 2 min to read • [Edit Online](#)

Azure Service Fabric supports two different access control types for clients that are connected to a Service Fabric cluster: administrator and user. Access control allows the cluster administrator to limit access to certain cluster operations for different groups of users, making the cluster more secure.

**Administrators** have full access to management capabilities (including read/write capabilities). By default, **users** only have read access to management capabilities (for example, query capabilities), and the ability to resolve applications and services.

You specify the two client roles (administrator and client) at the time of cluster creation by providing separate certificates for each. See [Service Fabric cluster security](#) for details on setting up a secure Service Fabric cluster.

## Default access control settings

The administrator access control type has full access to all the FabricClient APIs. It can perform any reads and writes on the Service Fabric cluster, including the following operations:

### Application and service operations

- **CreateService**: service creation
- **CreateServiceFromTemplate**: service creation from template
- **UpdateService**: service updates
- **DeleteService**: service deletion
- **ProvisionApplicationType**: application type provisioning
- **CreateApplication**: application creation
- **DeleteApplication**: application deletion
- **UpgradeApplication**: starting or interrupting application upgrades
- **UnprovisionApplicationType**: application type unprovisioning
- **MoveNextUpgradeDomain**: resuming application upgrades with an explicit update domain
- **ReportUpgradeHealth**: resuming application upgrades with the current upgrade progress
- **ReportHealth**: reporting health
- **PredeployPackageToNode**: predeployment API
- **CodePackageControl**: restarting code packages
- **RecoverPartition**: recovering a partition
- **RecoverPartitions**: recovering partitions
- **RecoverServicePartitions**: recovering service partitions
- **RecoverSystemPartitions**: recovering system service partitions

### Cluster operations

- **ProvisionFabric**: MSI and/or cluster manifest provisioning
- **UpgradeFabric**: starting cluster upgrades
- **UnprovisionFabric**: MSI and/or cluster manifest unprovisioning
- **MoveNextFabricUpgradeDomain**: resuming cluster upgrades with an explicit update domain
- **ReportFabricUpgradeHealth**: resuming cluster upgrades with the current upgrade progress
- **StartInfrastructureTask**: starting infrastructure tasks
- **FinishInfrastructureTask**: finishing infrastructure tasks

- **InvokeInfrastructureCommand**: infrastructure task management commands
- **ActivateNode**: activating a node
- **DeactivateNode**: deactivating a node
- **DeactivateNodesBatch**: deactivating multiple nodes
- **RemoveNodeDeactivations**: reverting deactivation on multiple nodes
- **GetNodeDeactivationStatus**: checking deactivation status
- **NodeStateRemoved**: reporting node state removed
- **ReportFault**: reporting fault
- **FileContent**: image store client file transfer (external to cluster)
- **FileDownload**: image store client file download initiation (external to cluster)
- **InternalList**: image store client file list operation (internal)
- **Delete**: image store client delete operation
- **Upload**: image store client upload operation
- **NodeControl**: starting, stopping, and restarting nodes
- **MoveReplicaControl**: moving replicas from one node to another

### Miscellaneous operations

- **Ping**: client pings
- **Query**: all queries allowed
- **NameExists**: naming URI existence checks

The user access control type is, by default, limited to the following operations:

- **EnumerateSubnames**: naming URI enumeration
- **EnumerateProperties**: naming property enumeration
- **PropertyReadBatch**: naming property read operations
- **GetServiceDescription**: long-poll service notifications and reading service descriptions
- **ResolveService**: complaint-based service resolution
- **ResolveNameOwner**: resolving naming URI owner
- **ResolvePartition**: resolving system services
- **ServiceNotifications**: event-based service notifications
- **GetUpgradeStatus**: polling application upgrade status
- **GetFabricUpgradeStatus**: polling cluster upgrade status
- **InvokeInfrastructureQuery**: querying infrastructure tasks
- **List**: image store client file list operation
- **ResetPartitionLoad**: resetting load for a failover unit
- **ToggleVerboseServicePlacementHealthReporting**: toggling verbose service placement health reporting

The admin access control also has access to the preceding operations.

## Changing default settings for client roles

In the cluster manifest file, you can provide admin capabilities to the client if needed. You can change the defaults by going to the **Fabric Settings** option during [cluster creation](#), and providing the preceding settings in the **name**, **admin**, **user**, and **value** fields.

## Next steps

[Service Fabric cluster security](#)

[Service Fabric cluster creation](#)

# Customize Service Fabric cluster settings and Fabric Upgrade policy

11/2/2017 • 64 min to read • [Edit Online](#)

This document tells you how to customize the various fabric settings and the fabric upgrade policy for your Service Fabric cluster. You can customize them through the [Azure portal](#) or using an Azure Resource Manager template.

## NOTE

Not all settings are available in the portal. In case a setting listed below is not available via the portal customize it using an Azure Resource Manager template.

## Customize cluster settings using Resource Manager templates

The steps below illustrate how to add a new setting *MaxDiskQuotaInMB* to the *Diagnostics* section.

1. Go to <https://resources.azure.com>
2. Navigate to your subscription by expanding **subscriptions** -> **resource groups** -> **Microsoft.ServiceFabric** -> **<Your Cluster Name>**
3. In the top right corner, select **Read/Write**.
4. Select **Edit** and update the `fabricSettings` JSON element and add a new element:

```
{  
    "name": "Diagnostics",  
    "parameters": [  
        {  
            "name": "MaxDiskQuotaInMB",  
            "value": "65536"  
        }  
    ]  
}
```

The following is a list of Fabric settings that you can customize, organized by section.

### Section Name: Diagnostics

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ConsumerInstances	String	Dynamic	The list of DCA consumer instances.
ProducerInstances	String	Dynamic	The list of DCA producer instances.
AppEtwTraceDeletionAgeInDays	Int, default is 3	Dynamic	Number of days after which we delete old ETL files containing application ETW traces.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
AppDiagnosticStoreAccessRequiresImpersonation	Bool, default is true	Dynamic	Whether or not impersonation is required when accessing diagnostic stores on behalf of the application.
MaxDiskQuotaInMB	Int, default is 65536	Dynamic	Disk quota in MB for Windows Fabric log files.
DiskFullSafetySpaceInMB	Int, default is 1024	Dynamic	Remaining disk space in MB to protect from use by DCA.
ApplicationLogsFormatVersion	Int, default is 0	Dynamic	Version for application logs format. Supported values are 0 and 1. Version 1 includes more fields from the ETW event record than version 0.
ClusterId	String	Dynamic	The unique id of the cluster. This is generated when the cluster is created.
EnableTelemetry	Bool, default is true	Dynamic	This is going to enable or disable telemetry.
EnableCircularTraceSession	Bool, default is false	Static	Flag indicates whether circular trace sessions should be used.

#### Section Name: Trace/Etw

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
Level	Int, default is 4	Dynamic	Trace etw level can take values 1, 2, 3, 4. To be supported you must keep the trace level at 4

#### Section Name: PerformanceCounterLocalStore

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
isEnabled	Bool, default is true	Dynamic	Flag indicates whether performance counter collection on local node is enabled.
SamplingIntervalInSeconds	Int, default is 60	Dynamic	Sampling interval for performance counters being collected.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
Counters	String	Dynamic	Comma-separated list of performance counters to collect.
MaxCounterBinaryFileSizeInMB	Int, default is 1	Dynamic	Maximum size (in MB) for each performance counter binary file.
NewCounterBinaryFileCreateonIntervalInMinutes	Int, default is 10	Dynamic	Maximum interval (in seconds) after which a new performance counter binary file is created.

### Section Name: Setup

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
FabricDataRoot	String	Not Allowed	Service Fabric data root directory. Default for Azure is d:\svcfab
FabricLogRoot	String	Not Allowed	Service fabric log root directory. This is where SF logs and traces are placed.
ServiceRunAsAccountName	String	Not Allowed	The account name under which to run fabric host service.
SkipFirewallConfiguration	Bool, default is false	Not Allowed	Specifies if firewall settings need to be set by the system or not. This applies only if you are using windows firewall. If you are using third party firewalls, then you must open the ports for the system and applications to use
NodesToBeRemoved	string, default is ""	Dynamic	The nodes which should be removed as part of configuration upgrade. (Only for Standalone Deployments)
ContainerNetworkSetup	bool, default is FALSE	Static	Whether to set up a container network.
ContainerNetworkName	string, default is L""	Static	The network name to use when setting up a container network.

### Section Name: TransactionalReplicator

Parameter	Allowed Values	Upgrade Policy	Guidance or Short Description
MaxCopyQueueSize	Uint, default is 16384	Static	This is the maximum value defines the initial size for the queue which maintains replication operations. Note that it must be a power of 2. If during runtime the queue grows to this size operation will be throttled between the primary and secondary replicators.
BatchAcknowledgementInterval	Time in seconds, default is 0.015	Static	Specify timespan in seconds. Determines the amount of time that the replicator waits after receiving an operation before sending back an acknowledgement. Other operations received during this time period will have their acknowledgements sent back in a single message-> reducing network traffic but potentially reducing the throughput of the replicator.
MaxReplicationMessageSize	Uint, default is 52428800	Static	Maximum message size of replication operations. Default is 50MB.
ReplicatorAddress	string, default is "localhost:0"	Static	The endpoint in form of a string -'IP:Port' which is used by the Windows Fabric Replicator to establish connections with other replicas in order to send/receive operations.
InitialPrimaryReplicationQueueSize	Uint, default is 64	Static	This value defines the initial size for the queue which maintains the replication operations on the primary. Note that it must be a power of 2.
MaxPrimaryReplicationQueueSize	Uint, default is 8192	Static	This is the maximum number of operations that could exist in the primary replication queue. Note that it must be a power of 2.
MaxPrimaryReplicationQueueMemorySize	Uint, default is 0	Static	This is the maximum value of the primary replication queue in bytes.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
InitialSecondaryReplicationQueueSize	Uint, default is 64	Static	This value defines the initial size for the queue which maintains the replication operations on the secondary. Note that it must be a power of 2.
MaxSecondaryReplicationQueueSize	Uint, default is 16384	Static	This is the maximum number of operations that could exist in the secondary replication queue. Note that it must be a power of 2.
MaxSecondaryReplicationQueueMemorySize	Uint, default is 0	Static	This is the maximum value of the secondary replication queue in bytes.
SecondaryClearAcknowledgedOperations	Bool, default is false	Static	Bool which controls if the operations on the secondary replicator are cleared once they are acknowledged to the primary(flushed to the disk). Settings this to TRUE can result in additional disk reads on the new primary, while catching up replicas after a failover.
MaxMetadataSizeInKB	Int, default is 4	Not Allowed	Maximum size of the log stream metadata.
MaxRecordSizeInKB	Uint, default is 1024	Not Allowed	Maximum size of a log stream record.
CheckpointThresholdInMB	Int, default is 50	Static	A checkpoint will be initiated when the log usage exceeds this value.
MaxAccumulatedBackupLogSizeInMB	Int, default is 800	Static	Max accumulated size (in MB) of backup logs in a given backup log chain. An incremental backup requests will fail if the incremental backup would generate a backup log that would cause the accumulated backup logs since the relevant full backup to be larger than this size. In such cases, user is required to take a full backup.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
MaxWriteQueueDepthInKB	Int, default is 0	Not Allowed	Int for maximum write queue depth that the core logger can use as specified in kilobytes for the log that is associated with this replica. This value is the maximum number of bytes that can be outstanding during core logger updates. It may be 0 for the core logger to compute an appropriate value or a multiple of 4.
SharedLogId	String	Not Allowed	Shared log identifier. This is a guid and should be unique for each shared log.
SharedLogPath	String	Not Allowed	Path to the shared log. If this value is empty then the default shared log is used.
SlowApiMonitoringDuration	Time in seconds, default is 300	Static	Specify duration for api before warning health event is fired.
MinLogSizeInMB	Int, default is 0	Static	Minimum size of the transactional log. The log will not be allowed to truncate to a size below this setting. 0 indicates that the replicator will determine the minimum log size according to other settings. Increasing this value increases the possibility of doing partial copies and incremental backups since chances of relevant log records being truncated is lowered.

### Section Name: FabricClient

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
NodeAddresses	string, default is ""	Static	A collection of addresses (connection strings) on different nodes that can be used to communicate with the the Naming Service. Initially the Client connects selecting one of the addresses randomly. If more than one connection string is supplied and a connection fails because of a communication or timeout error; the Client switches to use the next address sequentially. See the Naming Service Address retry section for details on retries semantics.
ConnectionInitializationTime out	Time in seconds, default is 2	Dynamic	Specify timespan in seconds. Connection timeout interval for each time client tries to open a connection to the gateway.
PartitionLocationCacheLimit	Int, default is 100000	Static	Number of partitions cached for service resolution (set to 0 for no limit).
ServiceChangePollInterval	Time in seconds, default is 120	Dynamic	Specify timespan in seconds. The interval between consecutive polls for service changes from the client to the gateway for registered service change notifications callbacks.
KeepAliveIntervalInSeconds	Int, default is 20	Static	The interval at which the FabricClient transport sends keep-alive messages to the gateway. For 0; keepAlive is disabled. Must be a positive value.
HealthOperationTimeout	Time in seconds, default is 120	Dynamic	Specify timespan in seconds. The timeout for a report message sent to Health Manager.
HealthReportSendInterval	Time in seconds, default is 30	Dynamic	Specify timespan in seconds. The interval at which reporting component sends accumulated health reports to Health Manager.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
HealthReportRetrySendInterval	Time in seconds, default is 30	Dynamic	Specify timespan in seconds. The interval at which reporting component re-sends accumulated health reports to Health Manager.
RetryBackoffInterval	Time in seconds, default is 3	Dynamic	Specify timespan in seconds. The back-off interval before retrying the operation.
MaxFileSenderThreads	Uint, default is 10	Static	The max number of files that are transferred in parallel.

#### Section Name: Common

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
PerfMonitorInterval	Time in seconds, default is 1	Dynamic	Specify timespan in seconds. Performance monitoring interval. Setting to 0 or negative value disables monitoring.

#### Section Name: HealthManager

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
EnableApplicationTypeHealthEvaluation	Bool, default is false	Static	Cluster health evaluation policy: enable per application type health evaluation.

#### Section Name: NodeDomainIds

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
UpgradeDomainId	string, default is ""	Static	Describes the upgrade domain a node belongs to.
PropertyGroup	NodeFaultDomainIdCollection	Static	Describes the fault domains a node belongs to. The fault domain is defined through a URI that describes the location of the node in the datacenter. Fault Domain URLs are of the format fd:/fd/ followed by a URI path segment.

#### Section Name: NodeProperties

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
PropertyGroup	NodePropertyCollectionMap	Static	A collection of string key-value pairs for node properties.

### Section Name: NodeCapacities

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
PropertyGroup	NodeCapacityCollectionMap	Static	A collection of node capacities for different metrics.

### Section Name: FabricNode

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
StateTraceInterval	Time in seconds, default is 300	Static	Specify timespan in seconds. The interval for tracing node status on each node and up nodes on FM/FMM.
StartApplicationPortRange	Int, default is 0	Static	Start of the application ports managed by hosting subsystem. Required if EndpointFilteringEnabled is true in Hosting.
EndApplicationPortRange	Int, default is 0	Static	End (no inclusive) of the application ports managed by hosting subsystem. Required if EndpointFilteringEnabled is true in Hosting.
ClusterX509StoreName	string, default is "My"	Dynamic	Name of X.509 certificate store that contains cluster certificate for securing intra-cluster communication.
ClusterX509FindType	string, default is "FindByThumbprint"	Dynamic	Indicates how to search for cluster certificate in the store specified by ClusterX509StoreName Supported values: "FindByThumbprint"; "FindBySubjectName" With "FindBySubjectName"; when there are multiple matches; the one with the furthest expiration is used.
ClusterX509FindValue	string, default is ""	Dynamic	Search filter value used to locate cluster certificate.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ClusterX509FindValueSecondary	string, default is ""	Dynamic	Search filter value used to locate cluster certificate.
ServerAuthX509StoreName	string, default is "My"	Dynamic	Name of X.509 certificate store that contains server certificate for entree service.
ServerAuthX509FindType	string, default is "FindByThumbprint"	Dynamic	Indicates how to search for server certificate in the store specified by ServerAuthX509StoreName Supported value: FindByThumbprint; FindBySubjectName.
ServerAuthX509FindValue	string, default is ""	Dynamic	Search filter value used to locate server certificate.
ServerAuthX509FindValueSecondary	string, default is ""	Dynamic	Search filter value used to locate server certificate.
ClientAuthX509StoreName	string, default is "My"	Dynamic	Name of the X.509 certificate store that contains certificate for default admin role FabricClient.
ClientAuthX509FindType	string, default is "FindByThumbprint"	Dynamic	Indicates how to search for certificate in the store specified by ClientAuthX509StoreName Supported value: FindByThumbprint; FindBySubjectName.
ClientAuthX509FindValue	string, default is ""	Dynamic	Search filter value used to locate certificate for default admin role FabricClient.
ClientAuthX509FindValueSecondary	string, default is ""	Dynamic	Search filter value used to locate certificate for default admin role FabricClient.
UserRoleClientX509StoreName	string, default is "My"	Dynamic	Name of the X.509 certificate store that contains certificate for default user role FabricClient.
UserRoleClientX509FindType	string, default is "FindByThumbprint"	Dynamic	Indicates how to search for certificate in the store specified by UserRoleClientX509StoreName Supported value: FindByThumbprint; FindBySubjectName.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
UserRoleClientX509FindValue	string, default is ""	Dynamic	Search filter value used to locate certificate for default user role FabricClient.
UserRoleClientX509FindValueSecondary	string, default is ""	Dynamic	Search filter value used to locate certificate for default user role FabricClient.

#### Section Name: Paas

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ClusterId	string, default is ""	Not Allowed	X509 certificate store used by fabric for configuration protection.

#### Section Name: FabricHost

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
StopTimeout	Time in seconds, default is 300	Dynamic	Specify timespan in seconds. The timeout for hosted service activation; deactivation and upgrade.
StartTimeout	Time in seconds, default is 300	Dynamic	Specify timespan in seconds. Time out for fabricactivationmanager startup.
ActivationRetryBackoffInterval	Time in seconds, default is 5	Dynamic	Specify timespan in seconds. Backoff interval on every activation failure;On every continuous activation failure the system will retry the activation for up to the MaxActivationFailureCount. The retry interval on every try is a product of continuous activation failure and the activation back-off interval.
ActivationMaxRetryInterval	Time in seconds, default is 300	Dynamic	Specify timespan in seconds. Max retry interval for Activation. On every continuous failure the retry interval is calculated as Min(ActivationMaxRetryInterval; Continuous Failure Count * ActivationRetryBackoffInterval).

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ActivationMaxFailureCount	Int, default is 10	Dynamic	This is the maximum count for which system will retry failed activation before giving up.
EnableServiceFabricAutomaticUpdates	Bool, default is false	Dynamic	This is to enable fabric automatic update via Windows Update.
EnableServiceFabricBaseUpdate	Bool, default is false	Dynamic	This is to enable base update for server.
EnableRestartManagement	Bool, default is false	Dynamic	This is to enable server restart.

### Section Name: FailoverManager

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
UserReplicaRestartWaitDuration	Time in seconds, default is 60.0 * 30	Dynamic	Specify timespan in seconds. When a persisted replica goes down; Windows Fabric waits for this duration for the replica to come back up before creating new replacement replicas (which would require a copy of the state).
QuorumLossWaitDuration	Time in seconds, default is MaxValue	Dynamic	Specify timespan in seconds. This is the max duration for which we allow a partition to be in a state of quorum loss. If the partition is still in quorum loss after this duration; the partition is recovered from quorum loss by considering the down replicas as lost. Note that this can potentially incur data loss.
UserStandByReplicaKeepDuration	Time in seconds, default is 3600.0 * 24 * 7	Dynamic	Specify timespan in seconds. When a persisted replica come back from a down state; it may have already been replaced. This timer determines how long the FM will keep the standby replica before discarding it.
UserMaxStandByReplicaCount	Int, default is 1	Dynamic	The default max number of StandBy replicas that the system keeps for user services.

Parameter	Allowed Values	Upgrade Policy	Guidance or Short Description
ExpectedClusterSize	int, default is 1	Dynamic	When the cluster is initially started up; the FM will wait for this many nodes to report themselves up before it begins placing other services; including the system services like naming. Increasing this value increases the time it takes a cluster to start up; but prevents the early nodes from becoming overloaded and also the additional moves that will be necessary as more nodes come online. This value should generally be set to some small fraction of the initial cluster size.
ClusterPauseThreshold	int, default is 1	Dynamic	If the number of nodes in system go below this value then placement; load balancing; and failover is stopped.
TargetReplicaSetSize	int, default is 7	Not Allowed	This is the target number of FM replicas that Windows Fabric will maintain. A higher number results in higher reliability of the FM data; with a small performance tradeoff.
MinReplicaSetSize	int, default is 3	Not Allowed	This is the minimum replica set size for the FM. If the number of active FM replicas drops below this value; the FM will reject changes to the cluster until at least the min number of replicas is recovered
ReplicaRestartWaitDuration	TimeSpan, default is Common::TimeSpan::FromSeconds(60.0 * 30)	Not Allowed	Specify timespan in seconds. This is the ReplicaRestartWaitDuration for the FMService
StandByReplicaKeepDuration	Timespan, default is Common::TimeSpan::FromSeconds(3600.0 * 24 * 7)	Not Allowed	Specify timespan in seconds. This is the StandByReplicaKeepDuration for the FMService
PlacementConstraints	string, default is L""	Not Allowed	Any placement constraints for the failover manager replicas

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ExpectedNodeFabricUpgradeDuration	TimeSpan, default is Common:: TimeSpan::FromSeconds(60.0 * 30)	Dynamic	Specify timespan in seconds. This is the expected duration for a node to be upgraded during Windows Fabric upgrade.
ExpectedReplicaUpgradeDuration	TimeSpan, default is Common:: TimeSpan::FromSeconds(60.0 * 30)	Dynamic	Specify timespan in seconds. This is the expected duration for all the replicas to be upgraded on a node during application upgrade.
ExpectedNodeDeactivationDuration	TimeSpan, default is Common:: TimeSpan::FromSeconds(60.0 * 30)	Dynamic	Specify timespan in seconds. This is the expected duration for a node to complete deactivation in.
IsSingletonReplicaMoveAllowedDuringUpgrade	bool, default is TRUE	Dynamic	If set to true; replicas with a target replica set size of 1 will be permitted to move during upgrade.
ReconfigurationTimeLimit	TimeSpan, default is Common:: TimeSpan::FromSeconds(300)	Dynamic	Specify timespan in seconds. The time limit for reconfiguration; after which a warning health report will be initiated
BuildReplicaTimeLimit	TimeSpan, default is Common:: TimeSpan::FromSeconds(3600)	Dynamic	Specify timespan in seconds. The time limit for building a stateful replica; after which a warning health report will be initiated
CreateInstanceTimeLimit	TimeSpan, default is Common:: TimeSpan::FromSeconds(300)	Dynamic	Specify timespan in seconds. The time limit for creating a stateless instance; after which a warning health report will be initiated
PlacementTimeLimit	TimeSpan, default is Common:: TimeSpan::FromSeconds(600)	Dynamic	Specify timespan in seconds. The time limit for reaching target replica count; after which a warning health report will be initiated

## Section Name: NamingService

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
-----------	----------------	----------------	-------------------------------

Parameter	Allowed Values	Upgrade Policy	Guidance or Short Description
TargetReplicaSetSize	Int, default is 7	Not Allowed	The number of replica sets for each partition of the Naming Service store. Increasing the number of replica sets increases the level of reliability for the information in the Naming Service Store; decreasing the change that the information will be lost as a result of node failures; at a cost of increased load on Windows Fabric and the amount of time it takes to perform updates to the naming data.
MinReplicaSetSize	Int, default is 3	Not Allowed	The minimum number of Naming Service replicas required to write into to complete an update. If there are fewer replicas than this active in the system the Reliability System denies updates to the Naming Service Store until replicas are restored. This value should never be more than the TargetReplicaSetSize.
ReplicaRestartWaitDuration	Time in seconds, default is (60.0 * 30)	Not Allowed	Specify timespan in seconds. When a Naming Service replica goes down; this timer starts. When it expires the FM will begin to replace the replicas which are down (it does not yet consider them lost).
QuorumLossWaitDuration	Time in seconds, default is MaxValue	Not Allowed	Specify timespan in seconds. When a Naming Service gets into quorum loss; this timer starts. When it expires the FM will consider the down replicas as lost; and attempt to recover quorum. Not that this may result in data loss.
StandByReplicaKeepDuration	Time in seconds, default is 3600.0 * 2	Not Allowed	Specify timespan in seconds. When a Naming Service replica come back from a down state; it may have already been replaced. This timer determines how long the FM will keep the standby replica before discarding it.

Parameter	Allowed Values	Upgrade Policy	Guidance or Short Description
PlacementConstraints	string, default is ""	Not Allowed	Placement constraint for the Naming Service.
ServiceDescriptionCacheLimit	Int, default is 0	Static	The maximum number of entries maintained in the LRU service description cache at the Naming Store Service (set to 0 for no limit).
RepairInterval	Time in seconds, default is 5	Static	Specify timespan in seconds. Interval in which the naming inconsistency repair between the authority owner and name owner will start.
MaxNamingServiceHealthReports	Int, default is 10	Dynamic	The maximum number of slow operations that Naming store service reports unhealthy at one time. If 0; all slow operations are sent.
MaxMessageSize	Int, default is 4*1024*1024	Static	The maximum message size for client node communication when using naming. DOS attack alleviation; default value is 4MB.
MaxFileOperationTimeout	Time in seconds, default is 30	Dynamic	Specify timespan in seconds. The maximum timeout allowed for file store service operation. Requests specifying a larger timeout will be rejected.
MaxOperationTimeout	Time in seconds, default is 600	Dynamic	Specify timespan in seconds. The maximum timeout allowed for client operations. Requests specifying a larger timeout will be rejected.
MaxClientConnections	Int, default is 1000	Dynamic	The maximum allowed number of client connections per gateway.
ServiceNotificationTimeout	Time in seconds, default is 30	Dynamic	Specify timespan in seconds. The timeout used when delivering service notifications to the client.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
MaxOutstandingNotificationsPerClient	Int, default is 1000	Dynamic	The maximum number of outstanding notifications before a client registration is forcibly closed by the gateway.
MaxIndexedEmptyPartitions	Int, default is 1000	Dynamic	The maximum number of empty partitions that will remain indexed in the notification cache for synchronizing reconnecting clients. Any empty partitions above this number will be removed from the index in ascending lookup version order. Reconnecting clients can still synchronize and receive missed empty partition updates; but the synchronization protocol becomes more expensive.
GatewayServiceDescriptionCacheLimit	Int, default is 0	Static	The maximum number of entries maintained in the LRU service description cache at the Naming Gateway (set to 0 for no limit).
PartitionCount	Int, default is 3	Not Allowed	The number of partitions of the Naming Service store to be created. Each partition owns a single partition key that corresponds to its index; so partition keys [0; PartitionCount) exist. Increasing the number of Naming Service partitions increases the scale that the Naming Service can perform at by decreasing the average amount of data held by any backing replica set; at a cost of increased utilization of resources (since PartitionCount*ReplicaSetSize service replicas must be maintained).

### Section Name: RunAs

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
RunAsAccountName	string, default is ""	Dynamic	Indicates the RunAs account name. This is only needed for "DomainUser" or "ManagedServiceAccount" account type. Valid values are "domain\user" or "user@domain".
RunAsAccountType	string, default is ""	Dynamic	Indicates the RunAs account type. This is needed for any RunAs section Valid values are "DomainUser/NetworkService/ManagedServiceAccount/LocalSystem".
RunAsPassword	string, default is ""	Dynamic	Indicates the RunAs account password. This is only needed for "DomainUser" account type.

#### Section Name: RunAs\_Fabric

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
RunAsAccountName	string, default is ""	Dynamic	Indicates the RunAs account name. This is only needed for "DomainUser" or "ManagedServiceAccount" account type. Valid values are "domain\user" or "user@domain".
RunAsAccountType	string, default is ""	Dynamic	Indicates the RunAs account type. This is needed for any RunAs section Valid values are "LocalUser/DomainUser/NetworkService/ManagedServiceAccount/LocalSystem".
RunAsPassword	string, default is ""	Dynamic	Indicates the RunAs account password. This is only needed for "DomainUser" account type.

#### Section Name: RunAs\_HttpGateway

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
RunAsAccountName	string, default is ""	Dynamic	Indicates the RunAs account name. This is only needed for "DomainUser" or "ManagedServiceAccount" account type. Valid values are "domain\user" or "user@domain".
RunAsAccountType	string, default is ""	Dynamic	Indicates the RunAs account type. This is needed for any RunAs section Valid values are "LocalUser/DomainUser/NetworkService/ManagedServiceAccount/LocalSystem".
RunAsPassword	string, default is ""	Dynamic	Indicates the RunAs account password. This is only needed for "DomainUser" account type.

#### Section Name: RunAs\_DCA

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
RunAsAccountName	string, default is ""	Dynamic	Indicates the RunAs account name. This is only needed for "DomainUser" or "ManagedServiceAccount" account type. Valid values are "domain\user" or "user@domain".
RunAsAccountType	string, default is ""	Dynamic	Indicates the RunAs account type. This is needed for any RunAs section Valid values are "LocalUser/DomainUser/NetworkService/ManagedServiceAccount/LocalSystem".
RunAsPassword	string, default is ""	Dynamic	Indicates the RunAs account password. This is only needed for "DomainUser" account type.

#### Section Name: HttpGateway

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
isEnabled	Bool, default is false	Static	Enables/Disables the HttpGateway. HttpGateway is disabled by default.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ActiveListeners	Uint, default is 50	Static	Number of reads to post to the http server queue. This controls the number of concurrent requests that can be satisfied by the HttpGateway.
MaxEntityBodySize	Uint, default is 4194304	Dynamic	Gives the maximum size of the body that can be expected from an http request. Default value is 4MB. Httpgateway will fail a request if it has a body of size > this value. Minimum read chunk size is 4096 bytes. So this has to be >= 4096.
HttpGatewayHealthReportSendInterval	Time in seconds, default is 30	Static	Specify timespan in seconds. The interval at which the Http Gateway sends accumulated health reports to the Health Manager.

### Section Name: KtlLogger

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
AutomaticMemoryConfiguration	Int, default is 1	Dynamic	Flag that indicates if the memory settings should be automatically and dynamically configured. If zero then the memory configuration settings are used directly and do not change based on system conditions. If one then the memory settings are configured automatically and may change based on system conditions.
WriteBufferMemoryPoolMinimumInKB	Int, default is 8388608	Dynamic	The number of KB to initially allocate for the write buffer memory pool. Use 0 to indicate no limit Default should be consistent with SharedLogSizeInMB below.
WriteBufferMemoryPoolMaximumInKB	Int, default is 0	Dynamic	The number of KB to allow the write buffer memory pool to grow up to. Use 0 to indicate no limit.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
MaximumDestagingWriteOutstandingInKB	Int, default is 0	Dynamic	The number of KB to allow the shared log to advance ahead of the dedicated log. Use 0 to indicate no limit.
SharedLogPath	string, default is ""	Static	Path and file name to location to place shared log container. Use "" for using default path under fabric data root.
SharedLogId	string, default is ""	Static	Unique guid for shared log container. Use "" if using default path under fabric data root.
SharedLogSizeInMB	Int, default is 8192	Static	The number of MB to allocate in the shared log container.

### Section Name: ApplicationGateway/Http

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
.IsEnabled	Bool, default is false	Static	Enables/Disables the HttpApplicationGateway. HttpApplicationGateway is disabled by default and this config needs to be set to enable it.
NumberOfParallelOperations	Uint, default is 5000	Static	Number of reads to post to the http server queue. This controls the number of concurrent requests that can be satisfied by the HttpGateway.
DefaultHttpRequestTimeout	Time in seconds. default is 120	Dynamic	Specify timespan in seconds. Gives the default request timeout for the http requests being processed in the http app gateway.
ResolveServiceBackoffInterval	Time in seconds, default is 5	Dynamic	Specify timespan in seconds. Gives the default back-off interval before retrying a failed resolve service operation.
BodyChunkSize	Uint, default is 16384	Dynamic	Gives the size of for the chunk in bytes used to read the body.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
GatewayAuthCredentialType	string, default is "None"	Static	Indicates the type of security credentials to use at the http app gateway endpoint Valid values are "None/X509".
GatewayX509CertificateStoreName	string, default is "My"	Dynamic	Name of X.509 certificate store that contains certificate for http app gateway.
GatewayX509CertificateFindType	string, default is "FindByThumbprint"	Dynamic	Indicates how to search for certificate in the store specified by GatewayX509CertificateStoreName Supported value: FindByThumbprint; FindBySubjectName.
GatewayX509CertificateFindValue	string, default is ""	Dynamic	Search filter value used to locate the http app gateway certificate. This certificate is configured on the https endpoint and can also be used to verify the identity of the app if needed by the services. FindValue is looked up first; and if that does not exist; FindValueSecondary is looked up.
GatewayX509CertificateFindValueSecondary	string, default is ""	Dynamic	Search filter value used to locate the http app gateway certificate. This certificate is configured on the https endpoint and can also be used to verify the identity of the app if needed by the services. FindValue is looked up first; and if that does not exist; FindValueSecondary is looked up.
HttpRequestConnectTimeout	TimeSpan, default is Common::TimeSpan::FromSeconds(5)	Dynamic	Specify timespan in seconds. Gives the connect timeout for the http requests being sent from the http app gateway.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
RemoveServiceResponseHeaders	string, default is L"Date; Server"	Static	Semi colon/ comma-separated list of response headers that will be removed from the service response; before forwarding it to the client. If this is set to empty string; pass all the headers returned by the service as-is. i.e do not overwrite the Date and Server
ApplicationCertificateValidationPolicy	string, default is L"None"	Static	ApplicationCertificateValidationPolicy: None: Do not validate server certificate; succeed the request. ServiceCertificateThumbprints: Refer to config ServiceCertificateThumbprints for the comma-separated list of thumbprints of the remote certs that the reverse proxy can trust. ServiceCommonNameAndIssuer: Refer to config ServiceCommonNameAndIssuer for the subject name and issuer thumbprint of the remote certs that the reverse proxy can trust.
ServiceCertificateThumbprints	string, default is L""	Dynamic	
CrlCheckingFlag	uint, default is 0x40000000	Dynamic	Flags for application/service certificate chain validation; e.g. CRL checking 0x10000000 CERT_CHAIN_REVOCATION_CHECK_END_CERT 0x20000000 CERT_CHAIN_REVOCATION_CHECK_CHAIN 0x40000000 CERT_CHAIN_REVOCATION_CHECK_CHAIN_EXCLUDE_ROOT 0x80000000 CERT_CHAIN_REVOCATION_CHECK_CACHE_ONLY Setting to 0 disables CRL checking Full list of supported values is documented by dwFlags of CertGetCertificateChain: <a href="http://msdn.microsoft.com/library/windows/desktop/aa376078(v=vs.85).aspx">http://msdn.microsoft.com/library/windows/desktop/aa376078(v=vs.85).aspx</a>

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
IgnoreCrlOfflineError	bool, default is TRUE	Dynamic	Whether to ignore CRL offline error for application/service certificate verification.
SecureOnlyMode	bool, default is FALSE	Dynamic	SecureOnlyMode: true: Reverse Proxy will only forward to services that publish secure endpoints. false: Reverse Proxy can forward requests to secure/non-secure endpoints.
ForwardClientCertificate	bool, default is FALSE	Dynamic	

#### Section Name: ApplicationGateway/Http/ServiceCommonNameAndIssuer

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
PropertyGroup	X509NameMap, default is None	Dynamic	

#### Section Name: Management

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ImageStoreConnectionString	SecureString	Static	Connection string to the Root for ImageStore.
ImageStoreMinimumTransferBPS	Int, default is 1024	Dynamic	The minimum transfer rate between the cluster and ImageStore. This value is used to determine the timeout when accessing the external ImageStore. Change this value only if the latency between the cluster and ImageStore is high to allow more time for the cluster to download from the external ImageStore.
AzureStorageMaxWorkerThreads	Int, default is 25	Dynamic	The maximum number of worker threads in parallel.
AzureStorageMaxConnections	Int, default is 5000	Dynamic	The maximum number of concurrent connections to azure storage.
AzureStorageOperationTimeout	Time in seconds, default is 6000	Dynamic	Specify timespan in seconds. Time out for xstore operation to complete.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ImageCachingEnabled	Bool, default is true	Static	This configuration allows us to enable or disable caching.
DisableChecksumValidation	Bool, default is false	Static	This configuration allows us to enable or disable checksum validation during application provisioning.
DisableServerSideCopy	Bool, default is false	Static	This configuration enables or disables server-side copy of application package on the ImageStore during application provisioning.

### Section Name: HealthManager/ClusterHealthPolicy

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ConsiderWarningAsError	Bool, default is false	Static	Cluster health evaluation policy: warnings are treated as errors.
MaxPercentUnhealthyNodes	Int, default is 0	Static	Cluster health evaluation policy: maximum percent of unhealthy nodes allowed for the cluster to be healthy.
MaxPercentUnhealthyApplications	Int, default is 0	Static	Cluster health evaluation policy: maximum percent of unhealthy applications allowed for the cluster to be healthy.

### Section Name: HealthManager/ClusterUpgradeHealthPolicy

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
MaxPercentDeltaUnhealthyNodes	int, default is 10	Static	Cluster upgrade health evaluation policy: maximum percent of delta unhealthy nodes allowed for the cluster to be healthy
MaxPercentUpgradeDomainDeltaUnhealthyNodes	int, default is 15	Static	Cluster upgrade health evaluation policy: maximum percent of delta of unhealthy nodes in an upgrade domain allowed for the cluster to be healthy

### Section Name: FaultAnalysisService

Parameter	Allowed Values	Upgrade Policy	Guidance or Short Description
TargetReplicaSetSize	Int, default is 0	Not Allowed	NOT_PLATFORM_UNIX_START The TargetReplicaSetSize for FaultAnalysisService.
MinReplicaSetSize	Int, default is 0	Not Allowed	The MinReplicaSetSize for FaultAnalysisService.
ReplicaRestartWaitDuration	Time in seconds, default is 60 minutes	Static	Specify timespan in seconds. The ReplicaRestartWaitDuration for FaultAnalysisService.
QuorumLossWaitDuration	Time in seconds, default is MaxValue	Static	Specify timespan in seconds. The QuorumLossWaitDuration for FaultAnalysisService.
StandByReplicaKeepDuration	Time in seconds, default is (60247) minutes	Static	Specify timespan in seconds. The StandByReplicaKeepDuration for FaultAnalysisService.
PlacementConstraints	string, default is ""	Static	The PlacementConstraints for FaultAnalysisService.
StoredActionCleanupIntervalInSeconds	Int, default is 3600	Static	This is how often the store will be cleaned up. Only actions in a terminal state; and that completed at least CompletedActionKeepDurationInSeconds ago will be removed.
CompletedActionKeepDurationInSeconds	Int, default is 604800	Static	This is approximately how long to keep actions that are in a terminal state. This also depends on StoredActionCleanupIntervalInSeconds; since the work to cleanup is only done on that interval. 604800 is 7 days.
StoredChaosEventCleanupIntervalInSeconds	Int, default is 3600	Static	This is how often the store will be audited for cleanup; if the number of events is more than 30000; the cleanup will kick in.
DataLossCheckWaitDurationInSeconds	int, default is 25	Static	The total amount of time; in seconds; that the system will wait for data loss to happen. This is internally used when the StartPartitionDataLossAsync() api is called.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
DataLossCheckPollIntervalInSeconds	int, default is 5	Static	This is the time between the checks the system performs while waiting for data loss to happen. The number of times the data loss number will be checked per internal iteration is DataLossCheckWaitDurationInSeconds/this.
ReplicaDropWaitDurationInSeconds	int, default is 600	Static	This parameter is used when the data loss api is called. It controls how long the system will wait for a replica to get dropped after remove replica is internally invoked on it.

## Section Name: FileStoreService

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
NamingOperationTimeout	Time in seconds, default is 60	Dynamic	Specify timespan in seconds. The timeout for performing naming operation.
QueryOperationTimeout	Time in seconds, default is 60	Dynamic	Specify timespan in seconds. The timeout for performing query operation.
MaxCopyOperationThreads	Uint, default is 0	Dynamic	The maximum number of parallel files that secondary can copy from primary. '0' == number of cores.
MaxFileOperationThreads	Uint, default is 100	Static	The maximum number of parallel threads allowed to perform FileOperations (Copy/Move) in the primary. '0' == number of cores.
MaxStoreOperations	Uint, default is 4096	Static	The maximum number of parallel store transaction operations allowed on primary. '0' == number of cores.
MaxRequestProcessingThreads	Uint, default is 200	Static	The maximum number of parallel threads allowed to process requests in the primary. '0' == number of cores.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
MaxSecondaryFileCopyFailureThreshold	Uint, default is 25	Dynamic	The maximum number of file copy retries on the secondary before giving up.
AnonymousAccessEnabled	Bool, default is true	Static	Enable/Disable anonymous access to the FileStoreService shares.
PrimaryAccountType	string, default is ""	Static	The primary AccountType of the principal to ACL the FileStoreService shares.
PrimaryAccountUserName	string, default is ""	Static	The primary account Username of the principal to ACL the FileStoreService shares.
PrimaryAccountUserPassword	SecureString, default is empty	Static	The primary account password of the principal to ACL the FileStoreService shares.
PrimaryAccountNTLMPasswordSecret	SecureString, default is empty	Static	The password secret which used as seed to generated same password when using NTLM authentication.
PrimaryAccountNTLMX509StoreLocation	string, default is "LocalMachine"	Static	The store location of the X509 certificate used to generate HMAC on the PrimaryAccountNTLMPasswordSecret when using NTLM authentication.
PrimaryAccountNTLMX509StoreName	string, default is "MY"	Static	The store name of the X509 certificate used to generate HMAC on the PrimaryAccountNTLMPasswordSecret when using NTLM authentication.
PrimaryAccountNTLMX509Thumbprint	string, default is ""	Static	The thumbprint of the X509 certificate used to generate HMAC on the PrimaryAccountNTLMPasswordSecret when using NTLM authentication.
SecondaryAccountType	string, default is ""	Static	The secondary AccountType of the principal to ACL the FileStoreService shares.
SecondaryAccountUserName	string, default is ""	Static	The secondary account Username of the principal to ACL the FileStoreService shares.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
SecondaryAccountUserPassword	SecureString, default is empty	Static	The secondary account password of the principal to ACL the FileStoreService shares.
SecondaryAccountNTLMPasswordSecret	SecureString, default is empty	Static	The password secret which used as seed to generated same password when using NTLM authentication.
SecondaryAccountNTLMX509StoreLocation	string, default is "LocalMachine"	Static	The store location of the X509 certificate used to generate HMAC on the SecondaryAccountNTLMPasswordSecret when using NTLM authentication.
SecondaryAccountNTLMX509StoreName	string, default is "MY"	Static	The store name of the X509 certificate used to generate HMAC on the SecondaryAccountNTLMPasswordSecret when using NTLM authentication.
SecondaryAccountNTLMX509Thumbprint	string, default is ""	Static	The thumbprint of the X509 certificate used to generate HMAC on the SecondaryAccountNTLMPasswordSecret when using NTLM authentication.
CommonNameNtlmPasswordSecret	SecureString, default is Common::SecureString(L "")	Static	The password secret which used as seed to generated same password when using NTLM authentication
CommonName1Ntlmx509StoreLocation	string, default is L"LocalMachine"	Static	The store location of the X509 certificate used to generate HMAC on the CommonName1NtlmPasswordSecret when using NTLM authentication
CommonName1Ntlmx509StoreName	string, default is L"MY"	Static	The store name of the X509 certificate used to generate HMAC on the CommonName1NtlmPasswordSecret when using NTLM authentication
CommonName1Ntlmx509CommonName	string, default is L""	Static	The common name of the X509 certificate used to generate HMAC on the CommonName1NtlmPasswordSecret when using NTLM authentication

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
CommonName2Ntlmx509StoreLocation	string, default is L"LocalMachine"	Static	The store location of the X509 certificate used to generate HMAC on the CommonName2NtlmPasswordSecret when using NTLM authentication
CommonName2Ntlmx509StoreName	string, default is L"MY"	Static	The store name of the X509 certificate used to generate HMAC on the CommonName2NtlmPasswordSecret when using NTLM authentication
CommonName2Ntlmx509CommonName	string, default is L""	Static	The common name of the X509 certificate used to generate HMAC on the CommonName2NtlmPasswordSecret when using NTLM authentication

### Section Name: ImageStoreService

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
Enabled	Bool, default is false	Static	The Enabled flag for ImageStoreService. Default: false
TargetReplicaSetSize	Int, default is 7	Not Allowed	The TargetReplicaSetSize for ImageStoreService.
MinReplicaSetSize	Int, default is 3	Not Allowed	The MinReplicaSetSize for ImageStoreService.
ReplicaRestartWaitDuration	Time in seconds, default is 60.0 * 30	Static	Specify timespan in seconds. The ReplicaRestartWaitDuration for ImageStoreService.
QuorumLossWaitDuration	Time in seconds, default is MaxValue	Static	Specify timespan in seconds. The QuorumLossWaitDuration for ImageStoreService.
StandByReplicaKeepDuration	Time in seconds, default is 3600.0 * 2	Static	Specify timespan in seconds. The StandByReplicaKeepDuration for ImageStoreService.
PlacementConstraints	string, default is ""	Static	The PlacementConstraints for ImageStoreService.

### Section Name: ImageStoreClient

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ClientUploadTimeout	Time in seconds, default is 1800	Dynamic	Specify timespan in seconds. Time out value for top-level upload request to Image Store Service.
ClientCopyTimeout	Time in seconds, default is 1800	Dynamic	Specify timespan in seconds. Time out value for top-level copy request to Image Store Service.
ClientDownloadTimeout	Time in seconds, default is 1800	Dynamic	Specify timespan in seconds. Time out value for top-level download request to Image Store Service.
ClientListTimeout	Time in seconds, default is 600	Dynamic	Specify timespan in seconds. Time out value for top-level list request to Image Store Service.
ClientDefaultTimeout	Time in seconds, default is 180	Dynamic	Specify timespan in seconds. Time out value for all non-upload/non-download requests (e.g. exists; delete) to Image Store Service.

#### Section Name: TokenValidationService

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
Providers	string, default is "DSTS"	Static	Comma separated list of token validation providers to enable (valid providers are: DSTS; AAD). Currently only a single provider can be enabled at any time.

#### Section Name: UpgradeOrchestrationService

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
TargetReplicaSetSize	Int, default is 0	Not Allowed	The TargetReplicaSetSize for UpgradeOrchestrationService.
MinReplicaSetSize	Int, default is 0	Not Allowed	The MinReplicaSetSize for UpgradeOrchestrationService.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ReplicaRestartWaitDuration	Time in seconds, default is 60 minutes	Static	Specify timespan in seconds. The ReplicaRestartWaitDuration for UpgradeOrchestrationService.
QuorumLossWaitDuration	Time in seconds, default is MaxValue	Static	Specify timespan in seconds. The QuorumLossWaitDuration for UpgradeOrchestrationService.
StandByReplicaKeepDuration	Time in seconds, default is 60247 minutes	Static	Specify timespan in seconds. The StandByReplicaKeepDuration for UpgradeOrchestrationService.
PlacementConstraints	string, default is ""	Static	The PlacementConstraints for UpgradeOrchestrationService.
AutoupgradeEnabled	Bool, default is true	Static	Automatic polling and upgrade action based on a goal-state file.
UpgradeApprovalRequired	Bool, default is false	Static	Setting to make code upgrade require administrator approval before proceeding.

### Section Name: UpgradeService

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
PlacementConstraints	string, default is ""	Not Allowed	The PlacementConstraints for Upgrade service.
TargetReplicaSetSize	Int, default is 3	Not Allowed	The TargetReplicaSetSize for UpgradeService.
MinReplicaSetSize	Int, default is 2	Not Allowed	The MinReplicaSetSize for UpgradeService.
CoordinatorType	string, default is "WUTest"	Not Allowed	The CoordinatorType for UpgradeService.
BaseUrl	string, default is ""	Static	BaseUrl for UpgradeService.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ClusterId	string, default is ""	Static	ClusterId for UpgradeService.
X509StoreName	string, default is "My"	Dynamic	X509StoreName for UpgradeService.
X509StoreLocation	string, default is ""	Dynamic	X509StoreLocation for UpgradeService.
X509FindType	string, default is ""	Dynamic	X509FindType for UpgradeService.
X509FindValue	string, default is ""	Dynamic	X509FindValue for UpgradeService.
X509SecondaryFindValue	string, default is ""	Dynamic	X509SecondaryFindValue for UpgradeService.
OnlyBaseUpgrade	Bool, default is false	Dynamic	OnlyBaseUpgrade for UpgradeService.
TestCabFolder	string, default is ""	Static	TestCabFolder for UpgradeService.

## Section Name: Security

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ClusterCredentialType	string, default is L"None"	Not Allowed	Indicates the type of security credentials to use in order to secure the cluster. Valid values are "None/X509/Windows"
ServerAuthCredentialType	string, default is L"None"	Static	Indicates the type of security credentials to use in order to secure the communication between FabricClient and the Cluster. Valid values are "None/X509/Windows"
ClientRoleEnabled	bool, default is FALSE	Static	Indicates if client role is enabled; when set to true; clients are assigned roles based on their identities. For V2; enabling this means client not in AdminClientCommonNames /AdminClientIdentities can only execute read-only operations.

Parameter	Allowed Values	Upgrade Policy	Guidance or Short Description
ClusterCertThumbprints	string, default is L""	Dynamic	Thumbprints of certificates allowed to join the cluster; a comma-separated name list.
ServerCertThumbprints	string, default is L""	Dynamic	Thumbprints of server certificates used by cluster to talk to clients; clients use this to authenticate the cluster. It is a comma-separated name list.
ClientCertThumbprints	string, default is L""	Dynamic	Thumbprints of certificates used by clients to talk to the cluster; cluster uses this to authorize incoming connection. It is a comma-separated name list.
AdminClientCertThumbprints	string, default is L""	Dynamic	Thumbprints of certificates used by clients in admin role. It is a comma-separated name list.
CrlCheckingFlag	uint, default is 0x40000000	Dynamic	Default certificate chain validation flag; may be overridden by component-specific flag; e.g. Federation/X509CertChainFlags 0x10000000 CERT_CHAIN_REVOCATION_CHECK_END_CERT 0x20000000 CERT_CHAIN_REVOCATION_CHECK_CHAIN 0x40000000 CERT_CHAIN_REVOCATION_CHECK_CHAIN_EXCLUDE_ROOT 0x80000000 CERT_CHAIN_REVOCATION_CHECK_CACHE_ONLY Setting to 0 disables CRL checking. Full list of supported values is documented by dwFlags of CertGetCertificateChain: <a href="http://msdn.microsoft.com/library/windows/desktop/aa376078(v=vs.85).aspx">http://msdn.microsoft.com/library/windows/desktop/aa376078(v=vs.85).aspx</a>
IgnoreCrlOfflineError	bool, default is FALSE	Dynamic	Whether to ignore CRL offline error when server-side verifies incoming client certificates

Parameter	Allowed Values	Upgrade Policy	Guidance or Short Description
IgnoreSrvCrlOfflineError	bool, default is TRUE	Dynamic	Whether to ignore CRL offline error when client side verifies incoming server certificates; default to true. Attacks with revoked server certificates require compromising DNS; harder than with revoked client certificates.
CrlDisablePeriod	TimeSpan, default is Common::TimeSpan::FromMinutes(15)	Dynamic	Specify timespan in seconds. How long CRL checking is disabled for a given certificate after encountering offline error; if CRL offline error can be ignored.
CrlOfflineHealthReportTtl	TimeSpan, default is Common::TimeSpan::FromMinutes(1440)	Dynamic	Specify timespan in seconds.
CertificateHealthReportingInterval	TimeSpan, default is Common::TimeSpan::FromSeconds(3600 * 8)	Static	Specify timespan in seconds. Specify interval for certificate health reporting; default to 8 hours; setting to 0 disables certificate health reporting
CertificateExpirySafetyMargin	TimeSpan, default is Common::TimeSpan::FromMinutes(43200)	Static	Specify timespan in seconds. Safety margin for certificate expiration; certificate health report status changes from OK to Warning when expiration is closer than this. Default is 30 days.
ClientClaimAuthEnabled	bool, default is FALSE	Static	Indicates if claim-based authentication is enabled on clients; setting this true implicitly sets ClientRoleEnabled.
ClientClaims	string, default is L""	Dynamic	All possible claims expected from clients for connecting to gateway. This is a 'OR' list: ClaimsEntry
AdminClientClaims	string, default is L""	Dynamic	All possible claims expected from admin clients; the same format as ClientClaims; this list internally gets added to ClientClaims; so no need to also add the same entries to ClientClaims.

Parameter	Allowed Values	Upgrade Policy	Guidance or Short Description
ClusterSpn	string, default is L""	Not Allowed	Service principal name of the cluster; when fabric runs as a single domain user (gMSA/domain user account). It is the SPN of lease listeners and listeners in fabric.exe: federation listeners; internal replication listeners; runtime service listener and naming gateway listener. This should be left empty when fabric runs as machine accounts; in which case connecting side compute listener SPN from listener transport address.
ClusterIdentities	string, default is L""	Dynamic	Windows identities of cluster nodes; used for cluster membership authorization. It is a comma-separated list; each entry is a domain account name or group name
ClientIdentities	string, default is L""	Dynamic	Windows identities of FabricClient; naming gateway uses this to authorize incoming connections. It is a comma-separated list; each entry is a domain account name or group name. For convenience; the account that runs fabric.exe is automatically allowed; so are group ServiceFabricAllowedUsers and ServiceFabricAdministrators.
AdminClientIdentities	string, default is L""	Dynamic	Windows identities of fabric clients in admin role; used to authorize privileged fabric operations. It is a comma-separated list; each entry is a domain account name or group name. For convenience; the account that runs fabric.exe is automatically assigned admin role; so is group ServiceFabricAdministrators.
AADTenantId	string, default is L""	Static	Tenant ID (GUID)

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
AADClusterApplication	string, default is L""	Static	Web API application name or ID representing the cluster
AADClientApplication	string, default is L""	Static	Native Client application name or ID representing Fabric Clients
X509Folder	string, default is /var/lib/waagent	Static	Folder where X509 certificates and private keys are located
FabricHostSpn	string, default is L""	Static	Service principal name of FabricHost; when fabric runs as a single domain user (gMSA/domain user account) and FabricHost runs under machine account. It is the SPN of IPC listener for FabricHost; which by default should be left empty since FabricHost runs under machine account
DisableFirewallRuleForPublicProfile	bool, default is TRUE	Static	Indicates if firewall rule should not be enabled for public profile
DisableFirewallRuleForPrivateProfile	bool, default is TRUE	Static	Indicates if firewall rule should not be enabled for private profile
DisableFirewallRuleForDomainProfile	bool, default is TRUE	Static	Indicates if firewall rule should not be enabled for domain profile
SettingsX509StoreName	string, default is L"MY"	Dynamic	X509 certificate store used by fabric for configuration protection

#### Section Name: Security/AdminClientX509Names

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
PropertyGroup	X509NameMap, default is None	Dynamic	

#### Section Name: Security/ClientX509Names

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
PropertyGroup	X509NameMap, default is None	Dynamic	

### **Section Name: Security/ClusterX509Names**

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
PropertyGroup	X509NameMap, default is None	Dynamic	

### **Section Name: Security/ServerX509Names**

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
PropertyGroup	X509NameMap, default is None	Dynamic	

### **Section Name: Security/ClientAccess**

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
CreateName	string, default is "Admin"	Dynamic	Security configuration for Naming URI creation.
DeleteName	string, default is "Admin"	Dynamic	Security configuration for Naming URI deletion.
PropertyWriteBatch	string, default is "Admin"	Dynamic	Security configurations for Naming property write operations.
CreateService	string, default is "Admin"	Dynamic	Security configuration for service creation.
CreateServiceFromTemplate	string, default is "Admin"	Dynamic	Security configuration for service creation from template.
UpdateService	string, default is "Admin"	Dynamic	Security configuration for service updates.
DeleteService	string, default is "Admin"	Dynamic	Security configuration for service deletion.
ProvisionApplicationType	string, default is "Admin"	Dynamic	Security configuration for application type provisioning.
CreateApplication	string, default is "Admin"	Dynamic	Security configuration for application creation.
DeleteApplication	string, default is "Admin"	Dynamic	Security configuration for application deletion.
UpgradeApplication	string, default is "Admin"	Dynamic	Security configuration for starting or interrupting application upgrades.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
RollbackApplicationUpgrade	string, default is "Admin"	Dynamic	Security configuration for rolling back application upgrades.
UnprovisionApplicationType	string, default is "Admin"	Dynamic	Security configuration for application type unprovisioning.
MoveNextUpgradeDomain	string, default is "Admin"	Dynamic	Security configuration for resuming application upgrades with an explicit Upgrade Domain.
ReportUpgradeHealth	string, default is "Admin"	Dynamic	Security configuration for resuming application upgrades with the current upgrade progress.
ReportHealth	string, default is "Admin"	Dynamic	Security configuration for reporting health.
ProvisionFabric	string, default is "Admin"	Dynamic	Security configuration for MSI and/or Cluster Manifest provisioning.
UpgradeFabric	string, default is "Admin"	Dynamic	Security configuration for starting cluster upgrades.
RollbackFabricUpgrade	string, default is "Admin"	Dynamic	Security configuration for rolling back cluster upgrades.
UnprovisionFabric	string, default is "Admin"	Dynamic	Security configuration for MSI and/or Cluster Manifest unprovisioning.
MoveNextFabricUpgradeDo main	string, default is "Admin"	Dynamic	Security configuration for resuming cluster upgrades with an explicitly Upgrade Domain.
ReportFabricUpgradeHealth	string, default is "Admin"	Dynamic	Security configuration for resuming cluster upgrades with the current upgrade progress.
StartInfrastructureTask	string, default is "Admin"	Dynamic	Security configuration for starting infrastructure tasks.
FinishInfrastructureTask	string, default is "Admin"	Dynamic	Security configuration for finishing infrastructure tasks.
ActivateNode	string, default is "Admin"	Dynamic	Security configuration for activation a node.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
DeactivateNode	string, default is "Admin"	Dynamic	Security configuration for deactivating a node.
DeactivateNodesBatch	string, default is "Admin"	Dynamic	Security configuration for deactivating multiple nodes.
RemoveNodeDeactivations	string, default is "Admin"	Dynamic	Security configuration for reverting deactivation on multiple nodes.
GetNodeDeactivationStatus	string, default is "Admin"	Dynamic	Security configuration for checking deactivation status.
NodeStateRemoved	string, default is "Admin"	Dynamic	Security configuration for reporting node state removed.
RecoverPartition	string, default is "Admin"	Dynamic	Security configuration for recovering a partition.
RecoverPartitions	string, default is "Admin"	Dynamic	Security configuration for recovering partitions.
RecoverServicePartitions	string, default is "Admin"	Dynamic	Security configuration for recovering service partitions.
RecoverSystemPartitions	string, default is "Admin"	Dynamic	Security configuration for recovering system service partitions.
ReportFault	string, default is "Admin"	Dynamic	Security configuration for reporting fault.
InvokeInfrastructureCommand	string, default is "Admin"	Dynamic	Security configuration for infrastructure task management commands.
FileContent	string, default is "Admin"	Dynamic	Security configuration for image store client file transfer (external to cluster).
FileDownload	string, default is "Admin"	Dynamic	Security configuration for image store client file download initiation (external to cluster).
InternalList	string, default is "Admin"	Dynamic	Security configuration for image store client file list operation (internal).
Delete	string, default is "Admin"	Dynamic	Security configurations for image store client delete operation.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
Upload	string, default is "Admin"	Dynamic	Security configuration for image store client upload operation.
GetStagingLocation	string, default is "Admin"	Dynamic	Security configuration for image store client staging location retrieval.
GetStoreLocation	string, default is "Admin"	Dynamic	Security configuration for image store client store location retrieval.
NodeControl	string, default is "Admin"	Dynamic	Security configuration for starting; stopping; and restarting nodes.
CodePackageControl	string, default is "Admin"	Dynamic	Security configuration for restarting code packages.
UnreliableTransportControl	string, default is "Admin"	Dynamic	Unreliable Transport for adding and removing behaviors.
MoveReplicaControl	string, default is "Admin"	Dynamic	Move replica.
PredeployPackageToNode	string, default is "Admin"	Dynamic	Predeployment api.
StartPartitionDataLoss	string, default is "Admin"	Dynamic	Induces data loss on a partition.
StartPartitionQuorumLoss	string, default is "Admin"	Dynamic	Induces quorum loss on a partition.
StartPartitionRestart	string, default is "Admin"	Dynamic	Simultaneously restarts some or all the replicas of a partition.
CancelTestCommand	string, default is "Admin"	Dynamic	Cancels a specific TestCommand - if it is in flight.
StartChaos	string, default is "Admin"	Dynamic	Starts Chaos - if it is not already started.
StopChaos	string, default is "Admin"	Dynamic	Stops Chaos - if it has been started.
StartNodeTransition	string, default is "Admin"	Dynamic	Security configuration for starting a node transition.
StartClusterConfigurationUpgrade	string, default is "Admin"	Dynamic	Induces StartClusterConfigurationUpgrade on a partition.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
GetUpgradesPendingApproval	string, default is "Admin"	Dynamic	Induces GetUpgradesPendingApproval on a partition.
StartApprovedUpgrades	string, default is "Admin"	Dynamic	Induces StartApprovedUpgrades on a partition.
Ping	string, default is "Admin  User"	Dynamic	Security configuration for client pings.
Query	string, default is "Admin  User"	Dynamic	Security configuration for queries.
NameExists	string, default is "Admin  User"	Dynamic	Security configuration for Naming URI existence checks.
EnumerateSubnames	string, default is "Admin  User"	Dynamic	Security configuration for Naming URI enumeration.
EnumerateProperties	string, default is "Admin  User"	Dynamic	Security configuration for Naming property enumeration.
PropertyReadBatch	string, default is "Admin  User"	Dynamic	Security configuration for Naming property read operations.
GetServiceDescription	string, default is "Admin  User"	Dynamic	Security configuration for long-poll service notifications and reading service descriptions.
ResolveService	string, default is "Admin  User"	Dynamic	Security configuration for complaint-based service resolution.
ResolveNameOwner	string, default is "Admin  User"	Dynamic	Security configuration for resolving Naming URI owner.
ResolvePartition	string, default is "Admin  User"	Dynamic	Security configuration for resolving system services.
ServiceNotifications	string, default is "Admin  User"	Dynamic	Security configuration for event-based service notifications.
PrefixResolveService	string, default is "Admin  User"	Dynamic	Security configuration for complaint-based service prefix resolution.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
GetUpgradeStatus	string, default is "Admin  User"	Dynamic	Security configuration for polling application upgrade status.
GetFabricUpgradeStatus	string, default is "Admin  User"	Dynamic	Security configuration for polling cluster upgrade status.
InvokeInfrastructureQuery	string, default is "Admin  User"	Dynamic	Security configuration for querying infrastructure tasks.
List	string, default is "Admin  User"	Dynamic	Security configuration for image store client file list operation.
ResetPartitionLoad	string, default is "Admin  User"	Dynamic	Security configuration for reset load for a failoverUnit.
ToggleVerboseServicePlacementHealthReporting	string, default is "Admin  User"	Dynamic	Security configuration for Toggling Verbose ServicePlacement HealthReporting.
GetPartitionDataLossProgress	string, default is "Admin  User"	Dynamic	Fetches the progress for an invoke data loss api call.
GetPartitionQuorumLossProgress	string, default is "Admin  User"	Dynamic	Fetches the progress for an invoke quorum loss api call.
GetPartitionRestartProgress	string, default is "Admin  User"	Dynamic	Fetches the progress for a restart partition api call.
GetChaosReport	string, default is "Admin  User"	Dynamic	Fetches the status of Chaos within a given time range.
GetNodeTransitionProgress	string, default is "Admin  User"	Dynamic	Security configuration for getting progress on a node transition command.
GetClusterConfigurationUpgradeStatus	string, default is "Admin  User"	Dynamic	Induces GetClusterConfigurationUpgradeStatus on a partition.
GetClusterConfiguration	string, default is "Admin  User"	Dynamic	Induces GetClusterConfiguration on a partition.
CreateComposeDeployment	string, default is L"Admin"	Dynamic	Creates a compose deployment described by compose files
DeleteComposeDeployment	string, default is L"Admin"	Dynamic	Deletes the compose deployment

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
UpgradeComposeDeployment	string, default is L"Admin"	Dynamic	Upgrades the compose deployment
ResolveSystemService	string, default is L"Admin  User"	Dynamic	Security configuration for resolving system services
GetUpgradeOrchestrationServiceState	string, default is L"Admin"	Dynamic	Induces GetUpgradeOrchestrationServiceState on a partition
SetUpgradeOrchestrationServiceState	string, default is L"Admin"	Dynamic	Induces SetUpgradeOrchestrationServiceState on a partition

## Section Name: ReconfigurationAgent

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ApplicationUpgradeMaxReplicaCloseDuration	Time in seconds, default is 900	Dynamic	Specify timespan in seconds. The duration for which the system will wait before terminating service hosts that have replicas that are stuck in close during Application Upgrade.
ServiceApiHealthDuration	Time in seconds, default is 30 minutes	Dynamic	Specify timespan in seconds. ServiceApiHealthDuration defines how long do we wait for a service API to run before we report it unhealthy.
ServiceReconfigurationApiHealthDuration	Time in seconds, default is 30	Dynamic	Specify timespan in seconds. ServiceReconfigurationApiHealthDuration defines how long do we wait for a service API to run before we report unhealthy. This applies to API calls that impact availability.
PeriodicApiSlowTraceInterval	Time in seconds, default is 5 minutes	Dynamic	Specify timespan in seconds. PeriodicApiSlowTraceInterval defines the interval over which slow API calls will be retraced by the API monitor.
NodeDeactivationMaxReplicaCloseDuration	Time in seconds, default is 900	Dynamic	Specify timespan in seconds. The duration for which the system will wait before terminating service hosts that have replicas that are stuck in close during node deactivation.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
FabricUpgradeMaxReplicaCloseDuration	Time in seconds, default is 900	Dynamic	Specify timespan in seconds. The duration for which the system will wait before terminating service hosts that have replicas that are stuck in close during fabric upgrade.
GracefulReplicaShutdownMaxDuration	TimeSpan, default is Common::TimeSpan::FromSeconds(120)	Dynamic	Specify timespan in seconds. The duration for which the system will wait before terminating service hosts that have replicas that are stuck in close. If this value is set to 0, replicas will not be instructed to close.
ReplicaChangeRoleFailureRestartThreshold	int, default is 10	Dynamic	Integer. Specify the number of API failures during primary promotion after which auto-mitigation action (replica restart) will be applied.
ReplicaChangeRoleFailureWarningReportThreshold	int, default is 2147483647	Dynamic	Integer. Specify the number of API failures during primary promotion after which warning health report will be raised.

### Section Name: PlacementAndLoadBalancing

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
TraceCRMReasons	Bool, default is true	Dynamic	Specifies whether to trace reasons for CRM issued movements to the operational events channel.
ValidatePlacementConstraint	Bool, default is true	Dynamic	Specifies whether or not the PlacementConstraint expression for a service is validated when a service's ServiceDescription is updated.
PlacementConstraintValidationCacheSize	Int, default is 10000	Dynamic	Limits the size of the table used for quick validation and caching of Placement Constraint Expressions.
VerboseHealthReportLimit	Int, default is 20	Dynamic	Defines the number of times a replica has to go unplaced before a health warning is reported for it (if verbose health reporting is enabled).

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ConstraintViolationHealthReportLimit	Int, default is 50	Dynamic	Defines the number of times constraint violating replica has to be persistently unfixed before diagnostics are conducted and health reports are emitted.
DetailedConstraintViolationHealthReportLimit	Int, default is 200	Dynamic	Defines the number of times constraint violating replica has to be persistently unfixed before diagnostics are conducted and detailed health reports are emitted.
DetailedVerboseHealthReportLimit	Int, default is 200	Dynamic	Defines the number of times an unplaced replica has to be persistently unplaced before detailed health reports are emitted.
ConsecutiveDroppedMovementsHealthReportLimit	Int, default is 20	Dynamic	Defines the number of consecutive times that ResourceBalancer-issued Movements are dropped before diagnostics are conducted and health warnings are emitted. Negative: No Warnings Emitted under this condition.
DetailedNodeListLimit	Int, default is 15	Dynamic	Defines the number of nodes per constraint to include before truncation in the Unplaced Replica reports.
DetailedPartitionListLimit	Int, default is 15	Dynamic	Defines the number of partitions per diagnostic entry for a constraint to include before truncation in Diagnostics.
DetailedDiagnosticsInfoListLimit	Int, default is 15	Dynamic	Defines the number of diagnostic entries (with detailed information) per constraint to include before truncation in Diagnostics.
PLBRefreshGap	Time in seconds, default is 1	Dynamic	Specify timespan in seconds. Defines the minimum amount of time that must pass before PLB refreshes state again.

Parameter	Allowed Values	Upgrade Policy	Guidance or Short Description
MinPlacementInterval	Time in seconds, default is 1	Dynamic	Specify timespan in seconds. Defines the minimum amount of time that must pass before two consecutive placement rounds.
MinConstraintCheckInterval	Time in seconds, default is 1	Dynamic	Specify timespan in seconds. Defines the minimum amount of time that must pass before two consecutive constraint check rounds.
MinLoadBalancingInterval	Time in seconds, default is 5	Dynamic	Specify timespan in seconds. Defines the minimum amount of time that must pass before two consecutive balancing rounds.
BalancingDelayAfterNodeDown	Time in seconds, default is 120	Dynamic	Specify timespan in seconds. Do not start balancing activities within this period after a node down event.
BalancingDelayAfterNewNode	Time in seconds, default is 120	Dynamic	Specify timespan in seconds. Do not start balancing activities within this period after adding a new node.
ConstraintFixPartialDelayAfterNodeDown	Time in seconds, default is 120	Dynamic	Specify timespan in seconds. Do not Fix FaultDomain and UpgradeDomain constraint violations within this period after a node down event.
ConstraintFixPartialDelayAfterNewNode	Time in seconds, default is 120	Dynamic	Specify timespan in seconds. Do not Fix FaultDomain and UpgradeDomain constraint violations within this period after adding a new node.
GlobalMovementThrottleThreshold	Uint, default is 1000	Dynamic	Maximum number of movements allowed in the Balancing Phase in the past interval indicated by GlobalMovementThrottleCountingInterval.
GlobalMovementThrottleThresholdForPlacement	Uint, default is 0	Dynamic	Maximum number of movements allowed in Placement Phase in the past interval indicated by GlobalMovementThrottleCountingInterval. 0 indicates no limit.

Parameter	Allowed Values	Upgrade Policy	Guidance or Short Description
GlobalMovementThrottleThresholdForBalancing	Uint, default is 0	Dynamic	Maximum number of movements allowed in Balancing Phase in the past interval indicated by GlobalMovementThrottleCountingInterval. 0 indicates no limit.
GlobalMovementThrottleCountingInterval	Time in seconds, default is 600	Static	Specify timespan in seconds. Indicate the length of the past interval for which to track per domain replica movements (used along with GlobalMovementThrottleThreshold). Can be set to 0 to ignore global throttling altogether.
MovementPerPartitionThrottleThreshold	Uint, default is 50	Dynamic	No balancing-related movement will occur for a partition if the number of balancing related movements for replicas of that partition has reached or exceeded MovementPerFailoverUnitThrottleThreshold in the past interval indicated by MovementPerPartitionThrottleCountingInterval.
MovementPerPartitionThrottleCountingInterval	Time in seconds, default is 600	Static	Specify timespan in seconds. Indicate the length of the past interval for which to track replica movements for each partition (used along with MovementPerPartitionThrottleThreshold).
PlacementSearchTimeout	Time in seconds, default is 0.5	Dynamic	Specify timespan in seconds. When placing services; search for at most this long before returning a result.
UseMoveCostReports	Bool, default is false	Dynamic	Instructs the LB to ignore the cost element of the scoring function; resulting potentially large number of moves for better balanced placement.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
PreventTransientOvercommit	Bool, default is false	Dynamic	Determines should PLB immediately count on resources that will be freed up by the initiated moves. By default; PLB can initiate move out and move in on the same node which can create transient overcommit. Setting this parameter to true will prevent those kinds of overcommits and on-demand defrag (aka placementWithMove) will be disabled.
InBuildThrottlingEnabled	Bool, default is false	Dynamic	Determine whether the in-build throttling is enabled.
InBuildThrottlingAssociatedMetric	string, default is ""	Static	The associated metric name for this throttling.
InBuildThrottlingGlobalMaxValue	Int, default is 0	Dynamic	The maximal number of in-build replicas allowed globally.
SwapPrimaryThrottlingEnabled	Bool, default is false	Dynamic	Determine whether the swap-primary throttling is enabled.
SwapPrimaryThrottlingAssociatedMetric	string, default is ""	Static	The associated metric name for this throttling.
SwapPrimaryThrottlingGlobal.MaxValue	Int, default is 0	Dynamic	The maximal number of swap-primary replicas allowed globally.
PlacementConstraintPriority	Int, default is 0	Dynamic	Determines the priority of placement constraint: 0: Hard; 1: Soft; negative: Ignore.
PreferredLocationConstraintPriority	Int, default is 2	Dynamic	Determines the priority of preferred location constraint: 0: Hard; 1: Soft; 2: Optimization; negative: Ignore
CapacityConstraintPriority	Int, default is 0	Dynamic	Determines the priority of capacity constraint: 0: Hard; 1: Soft; negative: Ignore.
AffinityConstraintPriority	Int, default is 0	Dynamic	Determines the priority of affinity constraint: 0: Hard; 1: Soft; negative: Ignore.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
FaultDomainConstraintPriority	Int, default is 0	Dynamic	Determines the priority of fault domain constraint: 0: Hard; 1: Soft; negative: Ignore.
UpgradeDomainConstraintPriority	Int, default is 1	Dynamic	Determines the priority of upgrade domain constraint: 0: Hard; 1: Soft; negative: Ignore.
ScaleoutCountConstraintPriority	Int, default is 0	Dynamic	Determines the priority of scaleout count constraint: 0: Hard; 1: Soft; negative: Ignore.
ApplicationCapacityConstraintPriority	Int, default is 0	Dynamic	Determines the priority of capacity constraint: 0: Hard; 1: Soft; negative: Ignore.
MoveParentToFixAffinityViolation	Bool, default is false	Dynamic	Setting which determines if parent replicas can be moved to fix affinity constraints.
MoveExistingReplicaForPlacement	Bool, default is true	Dynamic	Setting which determines if to move existing replica during placement.
UseSeparateSecondaryLoad	Bool, default is true	Dynamic	Setting which determines if use different secondary load.
PlaceChildWithoutParent	Bool, default is true	Dynamic	Setting which determines if child service replica can be placed if no parent replica is up.
PartiallyPlaceServices	Bool, default is true	Dynamic	Determines if all service replicas in cluster will be placed "all or nothing" given limited suitable nodes for them.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
InterruptBalancingForAllFailoverUnitUpdates	Bool, default is false	Dynamic	<p>Determines if any type of failover unit update should interrupt fast or slow balancing run. With specified "false" balancing run will be interrupted if FailoverUnit: is created/deleted; has missing replicas; changed primary replica location or changed number of replicas.</p> <p>Balancing run will NOT be interrupted in other cases - if FailoverUnit: has extra replicas; changed any replica flag; changed only partition version or any other case.</p>
GlobalMovementThrottleThresholdPercentage	double, default is 0	Dynamic	<p>Maximum number of total movements allowed in Balancing and Placement phases (expressed as percentage of total number of replicas in the cluster) in the past interval indicated by GlobalMovementThrottleCountingInterval. 0 indicates no limit. If both this and GlobalMovementThrottleThreshold are specified; then more conservative limit is used.</p>
GlobalMovementThrottleThresholdPercentageForBalancing	double, default is 0	Dynamic	<p>Maximum number of movements allowed in Balancing Phase (expressed as percentage of total number of replicas in PLB) in the past interval indicated by GlobalMovementThrottleCountingInterval. 0 indicates no limit. If both this and GlobalMovementThrottleThresholdForBalancing are specified; then more conservative limit is used.</p>

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
AutoDetectAvailableResources	bool, default is TRUE	Static	This config will trigger auto detection of available resources on node (CPU and Memory) When this config is set to true - we will read real capacities and correct them if user specified bad node capacities or didn't define them at all If this config is set to false - we will trace a warning that user specified bad node capacities; but we will not correct them; meaning that user wants to have the capacities specified as > than the node really has or if capacities are undefined; it will assume unlimited capacity

### Section Name: Hosting

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ServiceTypeRegistrationTimeout	Time in Seconds, default is 300	Dynamic	Maximum time allowed for the ServiceType to be registered with fabric
ServiceTypeDisableFailureThreshold	Whole number, default is 1	Dynamic	This is the threshold for the failure count after which FailoverManager (FM) is notified to disable the service type on that node and try a different node for placement.
ActivationRetryBackoffInterval	Time in Seconds, default is 5	Dynamic	Backoff interval on every activation failure; On every continuous activation failure, the system retries the activation for up to the MaxActivationFailureCount. The retry interval on every try is a product of continuous activation failure and the activation back-off interval.

Parameter	Allowed Values	Upgrade Policy	Guidance or Short Description
ActivationMaxRetryInterval	Time in seconds, default is 300	Dynamic	On every continuous activation failure, the system retries the activation for up to ActivationMaxFailureCount. ActivationMaxRetryInterval specifies Wait time interval before retry after every activation failure
ActivationMaxFailureCount	Whole number, default is 10	Dynamic	Number of times system retries failed activation before giving up
ActivationTimeout	TimeSpan, default is Common:: TimeSpan::FromSeconds(180)	Dynamic	Specify timespan in seconds. The timeout for application activation; deactivation and upgrade.
ApplicationHostCloseTimeout	TimeSpan, default is Common:: TimeSpan::FromSeconds(120)	Dynamic	Specify timespan in seconds. When Fabric exit is detected in a self activated processes; FabricRuntime closes all of the replicas in the user's host (applicationhost) process. This is the timeout for the close operation.
ApplicationUpgradeTimeout	TimeSpan, default is Common:: TimeSpan::FromSeconds(360)	Dynamic	Specify timespan in seconds. The timeout for application upgrade. If the timeout is less than the "ActivationTimeout" deployer will fail.
CreateFabricRuntimeTimeout	TimeSpan, default is Common:: TimeSpan::FromSeconds(120)	Dynamic	Specify timespan in seconds. The timeout value for the sync FabricCreateRuntime call
DeploymentMaxFailureCount	int, default is 20	Dynamic	Application deployment will be retried for DeploymentMaxFailureCount times before failing the deployment of that application on the node.
DeploymentMaxRetryInterval	TimeSpan, default is Common:: TimeSpan::FromSeconds(3600)	Dynamic	Specify timespan in seconds. Max retry interval for the deployment. On every continuous failure the retry interval is calculated as Min(DeploymentMaxRetryInterval; Continuous Failure Count * DeploymentRetryBackoffInterval)

Parameter	Allowed Values	Upgrade Policy	Guidance or Short Description
DeploymentRetryBackoffInterval	TimeSpan, default is Common:: TimeSpan::FromSeconds(10)	Dynamic	Specify timespan in seconds. Back-off interval for the deployment failure. On every continuous deployment failure the system will retry the deployment for up to the MaxDeploymentFailureCount. The retry interval is a product of continuous deployment failure and the deployment backoff interval.
EnableActivateNoWindow	bool, default is FALSE	Dynamic	The activated process is created in the background without any console.
EnableProcessDebugging	bool, default is FALSE	Dynamic	Enables launching application hosts under debugger
EndpointProviderEnabled	bool, default is FALSE	Static	Enables management of Endpoint resources by Fabric. Requires specification of start and end application port range in FabricNode.
FabricContainerAppsEnabled	bool, default is FALSE	Static	
FirewallPolicyEnabled	bool, default is FALSE	Static	Enables opening firewall ports for Endpoint resources with explicit ports specified in ServiceManifest
GetCodePackageActivationContextTimeout	TimeSpan, default is Common:: TimeSpan::FromSeconds(120)	Dynamic	Specify timespan in seconds. The timeout value for the CodePackageActivationContext calls. This is not applicable to ad-hoc services.
IPProviderEnabled	bool, default is FALSE	Static	Enables management of IP addresses.
NTLMAuthenticationEnabled	bool, default is FALSE	Static	Enables support for using NTLM by the code packages that are running as other users so that the processes across machines can communicate securely.

Parameter	Allowed Values	Upgrade Policy	Guidance or Short Description
NTLMAuthenticationPasswordSecret	SecureString, default is Common::SecureString(L "")	Static	Is an encrypted has that is used to generate the password for NTLM users. Has to be set if NTLMAuthenticationEnabled is true. Validated by the deployer.
NTLMSecurityUsersByX509CommonNamesRefreshInterval	TimeSpan, default is Common::TimeSpan::FromMinutes(3)	Dynamic	Specify timespan in seconds. Environment-specific settings The periodic interval at which Hosting scans for new certificates to be used for FileStoreService NTLM configuration.
NTLMSecurityUsersByX509CommonNamesRefreshTimeout	TimeSpan, default is Common::TimeSpan::FromMinutes(4)	Dynamic	Specify timespan in seconds. The timeout for configuring NTLM users using certificate common names. The NTLM users are needed for FileStoreService shares.
RegisterCodePackageHostTimeout	TimeSpan, default is Common::TimeSpan::FromSeconds(120)	Dynamic	Specify timespan in seconds. The timeout value for the FabricRegisterCodePackage Host sync call. This is applicable for only multi code package application hosts like FWP
RequestTimeout	TimeSpan, default is Common::TimeSpan::FromSeconds(30)	Dynamic	Specify timespan in seconds. This represents the timeout for communication between the user's application host and Fabric process for various hosting related operations such as factory registration; runtime registration.
RunAsPolicyEnabled	bool, default is FALSE	Static	Enables running code packages as local user other than the user under which fabric process is running. In order to enable this policy Fabric must be running as SYSTEM or as user who has SeAssignPrimaryTokenPrivilege.
ServiceFactoryRegistrationTimeout	TimeSpan, default is Common::TimeSpan::FromSeconds(120)	Dynamic	Specify timespan in seconds. The timeout value for the sync Register(Stateless/Stateful)ServiceFactory call

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ServiceTypeDisableGraceInterval	TimeSpan, default is Common::TimeSpan::FromSeconds(30)	Dynamic	Specify timespan in seconds. Time interval after which the service type can be disabled

### Section Name: Federation

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
LeaseDuration	Time in seconds, default is 30	Dynamic	Duration that a lease lasts between a node and its neighbors.
LeaseDurationAcrossFaultDomain	Time in seconds, default is 30	Dynamic	Duration that a lease lasts between a node and its neighbors across fault domains.

### Section Name: ClusterManager

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
UpgradeStatusPollInterval	Time in seconds, default is 60	Dynamic	The frequency of polling for application upgrade status. This value determines the rate of update for any GetApplicationUpgradeProgress call
UpgradeHealthCheckInterval	Time in seconds, default is 60	Dynamic	The frequency of health status checks during a monitored application upgrades
FabricUpgradeStatusPollInterval	Time in seconds, default is 60	Dynamic	The frequency of polling for Fabric upgrade status. This value determines the rate of update for any GetFabricUpgradeProgress call
FabricUpgradeHealthCheckInterval	Time in seconds, default is 60	Dynamic	The frequency of health status check during a monitored Fabric upgrade
InfrastructureTaskProcessingInterval	Time in seconds, default is 10	Dynamic	Specify timespan in seconds. The processing interval used by the infrastructure task processing state machine.
TargetReplicaSetSize	Int, default is 7	Not Allowed	The TargetReplicaSetSize for ClusterManager.
MinReplicaSetSize	Int, default is 3	Not Allowed	The MinReplicaSetSize for ClusterManager.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ReplicaRestartWaitDuration	Time in seconds, default is (60.0 * 30)	Not Allowed	Specify timespan in seconds. The ReplicaRestartWaitDuration for ClusterManager.
QuorumLossWaitDuration	Time in seconds, default is MaxValue	Not Allowed	Specify timespan in seconds. The QuorumLossWaitDuration for ClusterManager.
StandByReplicaKeepDuration	Time in seconds, default is (3600.0 * 2)	Not Allowed	Specify timespan in seconds. The StandByReplicaKeepDuration for ClusterManager.
PlacementConstraints	string, default is ""	Not Allowed	The PlacementConstraints for ClusterManager.
SkipRollbackUpdateDefaultService	Bool, default is false	Dynamic	The CM will skip reverting updated default services during application upgrade rollback.
EnableDefaultServicesUpgrade	Bool, default is false	Dynamic	Enable upgrading default services during application upgrade. Default service descriptions would be overwritten after upgrade.
InfrastructureTaskHealthCheckWaitDuration	Time in seconds, default is 0	Dynamic	Specify timespan in seconds. The amount of time to wait before starting health checks after post-processing an infrastructure task.
InfrastructureTaskHealthCheckStableDuration	Time in seconds, default is 0	Dynamic	Specify timespan in seconds. The amount of time to observe consecutive passed health checks before post-processing of an infrastructure task finishes successfully. Observing a failed health check will reset this timer.
InfrastructureTaskHealthCheckRetryTimeout	Time in seconds, default is 60	Dynamic	Specify timespan in seconds. The amount of time to spend retrying failed health checks while post-processing an infrastructure task. Observing a passed health check will reset this timer.

Parameter	Allowed Values	Upgrade Policy	Guidance or Short Description
ImageBuilderTimeoutBuffer	Time in seconds, default is 3	Dynamic	Specify timespan in seconds. The amount of time to allow for Image Builder specific timeout errors to return to the client. If this buffer is too small; then the client times out before the server and gets a generic timeout error.
MinOperationTimeout	Time in seconds, default is 60	Dynamic	Specify timespan in seconds. The minimum global timeout for internally processing operations on ClusterManager.
MaxOperationTimeout	Time in seconds, default is MaxValue	Dynamic	Specify timespan in seconds. The maximum global timeout for internally processing operations on ClusterManager.
MaxTimeoutRetryBuffer	Time in seconds, default is 600	Dynamic	Specify timespan in seconds. The maximum operation timeout when internally retrying due to timeouts is + . Additional timeout is added in increments of MinOperationTimeout.
MaxCommunicationTimeout	Time in seconds, default is 600	Dynamic	Specify timespan in seconds. The maximum timeout for internal communications between ClusterManager and other system services (i.e.; Naming Service; Failover Manager and etc.). This timeout should be smaller than global MaxOperationTimeout (as there might be multiple communications between system components for each client operation).
MaxDataMigrationTimeout	Time in seconds, default is 600	Dynamic	Specify timespan in seconds. The maximum timeout for data migration recovery operations after a Fabric upgrade has taken place.
MaxOperationRetryDelay	Time in seconds, default is 5	Dynamic	Specify timespan in seconds. The maximum delay for internal retries when failures are encountered.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ReplicaSetCheckTimeoutRollbackOverride	Time in seconds, default is 1200	Dynamic	Specify timespan in seconds. If ReplicaSetCheckTimeout is set to the maximum value of DWORD; then it's overridden with the value of this config for the purposes of rollback. The value used for roll-forward is never overridden.
ImageBuilderJobQueueThrottle	Int, default is 10	Dynamic	Thread count throttle for Image Builder proxy job queue on application requests.
MaxExponentialOperationRetryDelay	TimeSpan, default is Common::TimeSpan::FromSeconds(30)	Dynamic	Specify timespan in seconds. The maximum exponential delay for internal retries when failures are encountered repeatedly

### Section Name: DefragmentationEmptyNodeDistributionPolicy

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
PropertyGroup	KeyIntegerValueMap, default is None	Dynamic	Specifies the policy defragmentation follows when emptying nodes. For a given metric 0 indicates that SF should try to defragment nodes evenly across UDs and FDs; 1 indicates only that the nodes must be defragmented

### Section Name: DefragmentationMetrics

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
PropertyGroup	KeyBoolValueMap, default is None	Dynamic	Determines the set of metrics that should be used for defragmentation and not for load balancing.

### Section Name: DefragmentationMetricsPercentOrNumberOfEmptyNodesTriggeringThreshold

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
PropertyGroup	KeyDoubleValueMap, default is None	Dynamic	Determines the number of free nodes which are needed to consider cluster defragmented by specifying either percent in range [0.0 - 1.0) or number of empty nodes as number >= 1.0

#### Section Name: DnsService

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
IsEnabled	bool, default is FALSE	Static	
InstanceCount	int, default is -1	Static	

#### Section Name: MetricActivityThresholds

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
PropertyGroup	KeyIntegerValueMap, default is None	Dynamic	Determines the set of MetricActivityThresholds for the metrics in the cluster. Balancing will work if maxNodeLoad is greater than MetricActivityThresholds. For defrag metrics it defines the amount of load equal to or below which Service Fabric will consider the node empty

#### Section Name: MetricBalancingThresholds

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
PropertyGroup	KeyDoubleValueMap, default is None	Dynamic	Determines the set of MetricBalancingThresholds for the metrics in the cluster. Balancing will work if maxNodeLoad/minNodeLoad is greater than MetricBalancingThresholds. Defragmentation will work if maxNodeLoad/minNodeLoad in at least one FD or UD is smaller than MetricBalancingThresholds.

#### Section Name: NodeBufferPercentage

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
PropertyGroup	KeyDoubleValueMap, default is None	Dynamic	Node capacity percentage per metric name; used as a buffer in order to keep some free place on a node for the failover case.

## Section Name: Replication

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
MaxCopyQueueSize	uint, default is 1024	Static	This is the maximum value defines the initial size for the queue which maintains replication operations. Note that it must be a power of 2. If during runtime the queue grows to this size operation will be throttled between the primary and secondary replicators.
BatchAcknowledgementInterval	TimeSpan, default is Common::TimeSpan::FromMilliseconds(15)	Static	Specify timespan in seconds. Determines the amount of time that the replicator waits after receiving an operation before sending back an acknowledgement. Other operations received during this time period will have their acknowledgements sent back in a single message-> reducing network traffic but potentially reducing the throughput of the replicator.
MaxReplicationMessageSize	uint, default is 52428800	Static	Maximum message size of replication operations. Default is 50MB.
ReplicatorAddress	string, default is L"localhost:0"	Static	The endpoint in form of a string -'IP:Port' which is used by the Windows Fabric Replicator to establish connections with other replicas in order to send/receive operations.
ReplicatorListenAddress	string, default is L"localhost:0"	Static	The endpoint in form of a string -'IP:Port' which is used by the Windows Fabric Replicator to receive operations from other replicas.

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ReplicatorPublishAddress	string, default is L"localhost:0"	Static	The endpoint in form of a string -'IP:Port' which is used by the Windows Fabric Replicator to send operations to other replicas.
MaxPrimaryReplicationQueueSize	uint, default is 1024	Static	This is the maximum number of operations that could exist in the primary replication queue. Note that it must be a power of 2.
MaxPrimaryReplicationQueueMemorySize	uint, default is 0	Static	This is the maximum value of the primary replication queue in bytes.
MaxSecondaryReplicationQueueSize	uint, default is 2048	Static	This is the maximum number of operations that could exist in the secondary replication queue. Note that it must be a power of 2.
MaxSecondaryReplicationQueueMemorySize	uint, default is 0	Static	This is the maximum value of the secondary replication queue in bytes.
QueueHealthMonitoringInterval	TimeSpan, default is Common::TimeSpan::FromSeconds(30)	Static	Specify timespan in seconds. This value determines the time period used by the Replicator to monitor any warning/error health events in the replication operation queues. A value of '0' disables health monitoring
QueueHealthWarningAtUsagePercent	uint, default is 80	Static	This value determines the replication queue usage(in percentage) after which we report warning about high queue usage. We do so after a grace interval of QueueHealthMonitoringInterval. If the queue usage falls below this percentage in the grace interval
RetryInterval	TimeSpan, default is Common::TimeSpan::FromSeconds(5)	Static	Specify timespan in seconds. When an operation is lost or rejected this timer determines how often the replicator will retry sending the operation.

## Section Name: Transport

PARAMETER	ALLOWED VALUES	UPGRADE POLICY	GUIDANCE OR SHORT DESCRIPTION
ResolveOption	string, default is L"unspecified"	Static	Determines how FQDN is resolved. Valid values are "unspecified/ipv4/ipv6".
ConnectionOpenTimeout	TimeSpan, default is Common::TimeSpan::FromSeconds(60)	Static	Specify timespan in seconds. Time out for connection setup on both incoming and accepting side (including security negotiation in secure mode)

## Next steps

Read these articles for more information on cluster management:

[Add, Roll over, remove certificates from your Azure cluster](#)

# Open ports for a Service Fabric cluster

9/7/2017 • 2 min to read • [Edit Online](#)

The load balancer deployed with your Azure Service Fabric cluster directs traffic to your app running on a node. If you change your app to use a different port, you must expose that port (or route a different port) in the Azure Load Balancer.

When you deployed your Service Fabric cluster to Azure, a load balancer was automatically created for you. If you do not have a load balancer, see [Configure an Internet-facing load balancer](#).

## Configure service fabric

Your Service Fabric application **ServiceManifest.xml** config file defines the endpoints your application expects to use. After the config file has been updated to define an endpoint, the load balancer must be updated to expose that (or a different) port. For more information on how to create the service fabric endpoint, see [Setup an Endpoint](#).

## Create a load balancer rule

A Load Balancer rule opens up an internet-facing port and forwards traffic to the internal node's port used by your application. If you do not have a load balancer, see [Configure an Internet-facing load balancer](#).

To create a Load Balancer rule, you need to collect the following information:

- Load balancer name.
- Resource group of the load balancer and service fabric cluster.
- External port.
- Internal port.

## Azure CLI

It only takes a single command to create a load balancer rule with the **Azure CLI**. You just need to know both the name of the load balancer and resource group to create a new rule.

### NOTE

If you need to determine the name of the load balancer, use this command to quickly get a list of all load balancers and the associated resource groups.

```
az network lb list --query "[].{ResourceGroup: resourceGroup, Name: name}"
```

```
az network lb rule create --backend-port 40000 --frontend-port 39999 --protocol Tcp --lb-name LB-svcfab3 -g svcfab_cli -n my-app-rule
```

The Azure CLI command has a few parameters that are described in the following table:

PARAMETER	DESCRIPTION
--backend-port	The port the Service Fabric application is listening to.

PARAMETER	DESCRIPTION
--frontend-port	The port the load balancer exposes for external connections.
-lb-name	The name of the load balancer to change.
-g	The resource group that has both the load balancer and Service Fabric cluster.
-n	The desired name of the rule.

#### NOTE

For more information on how to create a load balancer with the Azure CLI, see [Create a load balancer with the Azure CLI](#).

## PowerShell

PowerShell is a little more complicated than the Azure CLI. Follow these conceptual steps to create a rule:

1. Get the load balancer from Azure.
2. Create a rule.
3. Add the rule to the load balancer.
4. Update the load balancer.

#### NOTE

If you need to determine the name of the load balancer, use this command to quickly get a list of all load balancers and associated resource groups.

```
Get-AzureRmLoadBalancer | Select Name, ResourceGroupName
```

```
# Get the load balancer
$lb = Get-AzureRmLoadBalancer -Name LB-svcfab3 -ResourceGroupName svcfab_cli

# Create the rule based on information from the load balancer.
$lbrule = New-AzureRmLoadBalancerRuleConfig -Name my-app-rule7 -Protocol Tcp -FrontendPort 39990 -BackendPort 40009 `

# Add the rule to the load balancer
$lb.LoadBalancingRules.Add($lbrule)

# Update the load balancer on Azure
$lb | Set-AzureRmLoadBalancer
```

Regarding the `New-AzureRmLoadBalancerRuleConfig` command, the `-FrontendPort` represents the port the load balancer exposes for external connections, and the `-BackendPort` represents the port the service fabric app is listening to.

**NOTE**

For more information on how to create a load balancer with PowerShell, see [Create a load balancer with PowerShell](#).

## Next steps

Learn more about [networking in Service Fabric](#).

# Add or remove certificates for a Service Fabric cluster in Azure

6/27/2017 • 8 min to read • [Edit Online](#)

It is recommended that you familiarize yourself with how Service Fabric uses X.509 certificates and be familiar with the [Cluster security scenarios](#). You must understand what a cluster certificate is and what is used for, before you proceed further.

Service fabric lets you specify two cluster certificates, a primary and a secondary, when you configure certificate security during cluster creation, in addition to client certificates. Refer to [creating an azure cluster via portal](#) or [creating an azure cluster via Azure Resource Manager](#) for details on setting them up at create time. If you specify only one cluster certificate at create time, then that is used as the primary certificate. After cluster creation, you can add a new certificate as a secondary.

## NOTE

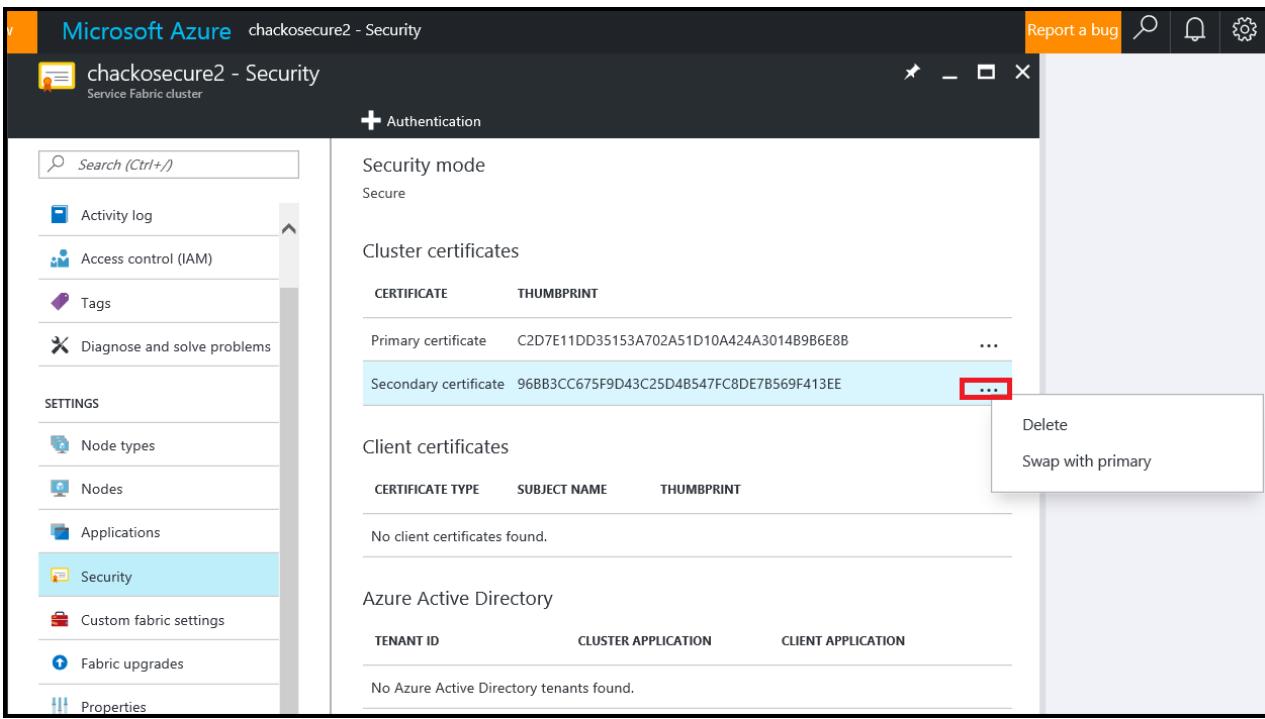
For a secure cluster, you will always need at least one valid (not revoked and not expired) cluster certificate (primary or secondary) deployed (if not, the cluster stops functioning). 90 days before all valid certificates reach expiration, the system generates a warning trace and also a warning health event on the node. There is currently no email or any other notification that service fabric sends out on this topic.

## Add a secondary cluster certificate using the portal

Secondary cluster certificate cannot be added through the Azure portal. You have to use Azure powershell for that. The process is outlined later in this document.

## Swap the cluster certificates using the portal

After you have successfully deployed a secondary cluster certificate, if you want to swap the primary and secondary, then navigate to the Security blade, and select the 'Swap with primary' option from the context menu to swap the secondary cert with the primary cert.



## Remove a cluster certificate using the portal

For a secure cluster, you will always need at least one valid (not revoked and not expired) certificate (primary or secondary) deployed if not, the cluster stops functioning.

To remove a secondary certificate from being used for cluster security, Navigate to the Security blade and select the 'Delete' option from the context menu on the secondary certificate.

If your intent is to remove the certificate that is marked primary, then you will need to swap it with the secondary first, and then delete the secondary after the upgrade has completed.

## Add a secondary certificate using Resource Manager Powershell

These steps assume that you are familiar with how Resource Manager works and have deployed atleast one Service Fabric cluster using a Resource Manager template, and have the template that you used to set up the cluster handy. It is also assumed that you are comfortable using JSON.

### NOTE

If you are looking for a sample template and parameters that you can use to follow along or as a starting point, then download it from this [git-repo](#).

### Edit your Resource Manager template

For ease of following along, sample 5-VM-1-NodeTypes-Secure\_Step2.JSON contains all the edits we will be making. the sample is available at [git-repo](#).

### Make sure to follow all the steps

**Step 1:** Open up the Resource Manager template you used to deploy you Cluster. (If you have downloaded the sample from the above repo, then Use 5-VM-1-NodeTypes-Secure\_Step1.JSON to deploy a secure cluster and then open up that template).

**Step 2:** Add **two new parameters** "secCertificateThumbprint" and "secCertificateUrlValue" of type "string" to the parameter section of your template. You can copy the following code snippet and add it to the template.

Depending on the source of your template, you may already have these defined, if so move to the next step.

```

"secCertificateThumbprint": {
    "type": "string",
    "metadata": {
        "description": "Certificate Thumbprint"
    }
},
"secCertificateUrlValue": {
    "type": "string",
    "metadata": {
        "description": "Refers to the location URL in your key vault where the certificate was uploaded, it is  
should be in the format of https://<name of the vault>.vault.azure.net:443/secrets/<exact location>"
    }
},

```

**Step 3:** Make changes to the **Microsoft.ServiceFabric/clusters** resource - Locate the "Microsoft.ServiceFabric/clusters" resource definition in your template. Under properties of that definition, you will find "Certificate" JSON tag, which should look something like the following JSON snippet:

```

"properties": {
    "certificate": {
        "thumbprint": "[parameters('certificateThumbprint')]",
        "x509StoreName": "[parameters('certificateStoreValue')]"
    }
}

```

Add a new tag "thumbprintSecondary" and give it a value "[parameters('secCertificateThumbprint')]".

So now the resource definition should look like the following (depending on your source of the template, it may not be exactly like the snippet below).

```

"properties": {
    "certificate": {
        "thumbprint": "[parameters('certificateThumbprint')]",
        "thumbprintSecondary": "[parameters('secCertificateThumbprint')]",
        "x509StoreName": "[parameters('certificateStoreValue')]"
    }
}

```

If you want to **rollover the cert**, then specify the new cert as primary and moving the current primary as secondary. This results in the rollover of your current primary certificate to the new certificate in one deployment step.

```

"properties": {
    "certificate": {
        "thumbprint": "[parameters('secCertificateThumbprint')]",
        "thumbprintSecondary": "[parameters('certificateThumbprint')]",
        "x509StoreName": "[parameters('certificateStoreValue')]"
    }
}

```

**Step 4:** Make changes to **all** the **Microsoft.Compute/virtualMachineScaleSets** resource definitions - Locate the Microsoft.Compute/virtualMachineScaleSets resource definition. Scroll to the "publisher": "Microsoft.Azure.ServiceFabric", under "virtualMachineProfile".

In the service fabric publisher settings, you should see something like this.

```

"virtualMachineProfile": {
    "extensionProfile": {
        "extensions": [
            {
                "name": "[concat('ServiceFabricNodeVmExt', '_vmNodeType0Name')]",
                "properties": {
                    "type": "ServiceFabricNode",
                    "autoUpgradeMinorVersion": false,
                    "protectedSettings": {
                        "StorageAccountKey1": "[listKeys(resourceId('Microsoft.Storage/StorageAccountKey2'))]",
                        "StorageAccountKey2": "[listKeys(resourceId('Microsoft.Storage/StorageAccountKey1'))]"
                    },
                    "publisher": "Microsoft.Azure.ServiceFabric",
                    "settings": {
                        "clusterEndpoint": "[reference(parameters('clusterName')).clusterEndpoint]",
                        "nodeTypeRef": "[variables('vmNodeType0Name')]",
                        "dataPath": "D:\\\\SvcFab",
                        "durabilityLevel": "Bronze",
                        "nicPrefixOverride": "[variables('subnet0Prefix')]",
                        "certificate": {
                            "thumbprint": "[parameters('certificateThumbprint')]",
                            "x509StoreName": "[parameters('certificateStoreValue')]"
                        }
                    }
                }
            }
        ]
    }
}

```

Add the new cert entries to it

```

"certificateSecondary": {
    "thumbprint": "[parameters('secCertificateThumbprint')]",
    "x509StoreName": "[parameters('certificateStoreValue')]"
}
},

```

The properties should now look like this

```

"virtualMachineProfile": {
    "extensionProfile": {
        "extensions": [
            {
                "name": "[concat('ServiceFabricNodeVmExt', '_vmNodeType0Name')]",
                "properties": {
                    "type": "ServiceFabricNode",
                    "autoUpgradeMinorVersion": false,
                    "protectedSettings": {
                        "StorageAccountKey1": "[listKeys(resourceId('Microsoft.Storage/StorageAccountKey2'))]",
                        "StorageAccountKey2": "[listKeys(resourceId('Microsoft.Storage/StorageAccountKey1'))]"
                    },
                    "publisher": "Microsoft.Azure.ServiceFabric",
                    "settings": {
                        "clusterEndpoint": "[reference(parameters('clusterName')).clusterEndpoint]",
                        "nodeTypeRef": "[variables('vmNodeType0Name')]",
                        "dataPath": "D:\\\\SvcFab",
                        "durabilityLevel": "Bronze",
                        "nicPrefixOverride": "[variables('subnet0Prefix')]",
                        "certificate": {
                            "thumbprint": "[parameters('certificateThumbprint')]",
                            "x509StoreName": "[parameters('certificateStoreValue')]"
                        }
                    }
                }
            }
        ]
    }
}

"certificateSecondary": {
    "thumbprint": "[parameters('secCertificateThumbprint')]",
    "x509StoreName": "[parameters('certificateStoreValue')]"
}
}

```

If you want to **rollover the cert**, then specify the new cert as primary and moving the current primary as secondary. This results in the rollover of your current certificate to the new certificate in one deployment step.

```

    "certificate": {
        "thumbprint": "[parameters('secCertificateThumbprint')]",
        "x509StoreName": "[parameters('certificateStoreValue')]"
    },
    "certificateSecondary": {
        "thumbprint": "[parameters('certificateThumbprint')]",
        "x509StoreName": "[parameters('certificateStoreValue')]"
    }
},

```

The properties should now look like this

```

},
"virtualMachineProfile": {
    "extensionProfile": {
        "extensions": [
            {
                "name": "[concat('ServiceFabricNodeVmExt', '_vmNodeType0Name')]",
                "properties": {
                    "type": "ServiceFabricNode",
                    "autoUpgradeMinorVersion": false,
                    "protectedSettings": {
                        "StorageAccountKey1": "[listKeys(resourceId('Microsoft.Storage/storageAccounts', variables('storageAccountName')), '2015-06-15').keys[0].value]",
                        "StorageAccountKey2": "[listKeys(resourceId('Microsoft.Storage/storageAccounts', variables('storageAccountName')), '2015-06-15').keys[1].value]"
                    },
                    "publisher": "Microsoft.Azure.ServiceFabric",
                    "settings": {
                        "clusterEndpoint": "[reference(parameters('clusterName')).clusterEndpoint]",
                        "nodeTypeRef": "[variables('vmNodeType0Name')]",
                        "dataPath": "D:\\\\SvcFab",
                        "durabilityLevel": "Bronze",
                        "nicPrefixOverride": "[variables('subnet0Prefix')]",
                        "certificate": {
                            "thumbprint": "[parameters('secCertificateThumbprint')]",
                            "x509StoreName": "[parameters('certificateStoreValue')]"
                        },
                        "certificateSecondary": {
                            "thumbprint": "[parameters('certificateThumbprint')]",
                            "x509StoreName": "[parameters('certificateStoreValue')]"
                        }
                    }
                }
            }
        ]
    }
},

```

**Step 5:** Make Changes to **all** the **Microsoft.Compute/virtualMachineScaleSets** resource definitions - Locate the Microsoft.Compute/virtualMachineScaleSets resource definition. Scroll to the "vaultCertificates":, under "OSProfile". it should look something like this.

```

        "osProfile": {
            "adminPassword": "[parameters('adminPassword')]",
            "adminUsername": "[parameters('adminUsername')]",
            "computerNamePrefix": "[variables('vmNodeType0Name')]",
            "secrets": [
                {
                    "sourceVault": {
                        "id": "[parameters('sourceVaultValue')]"
                    }
                },
                "vaultCertificates": [
                    {
                        "certificateStore": "[parameters('certificateStoreValue')]",
                        "certificateUrl": "[parameters('certificateUrlValue')]"
                    }
                ],

```

Add the secCertificateUrlValue to it. use the following snippet:

```

    },
    "certificateStore": "[parameters('certificateStoreValue')]",
    "certificateUrl": "[parameters('secCertificateUrlValue')]"
}

```

Now the resulting Json should look something like this.

```

"osProfile": {
    "adminPassword": "[parameters('adminPassword')]",
    "adminUsername": "[parameters('adminUsername')]",
    "computernamePrefix": "[variables('vmNodeType0Name')]",
    "secrets": [
        {
            "sourceVault": {
                "id": "[parameters('sourceVaultValue')]"
            },
            "vaultCertificates": [
                {
                    "certificateStore": "[parameters('certificateStoreValue')]",
                    "certificateUrl": "[parameters('certificateUrlValue')]"
                },
                {
                    "certificateStore": "[parameters('certificateStoreValue')]",
                    "certificateUrl": "[parameters('secCertificateUrlValue')]"
                }
            ]
        }
    ]
}

```

#### **NOTE**

Make sure that you have repeated steps 4 and 5 for all the Nodetypes/Microsoft.Compute/virtualMachineScaleSets resource definitions in your template. If you miss one of them, the certificate will not get installed on that VMSS and you will have unpredictable results in your cluster, including the cluster going down (if you end up with no valid certificates that the cluster can use for security). So please double check, before proceeding further.

#### **Edit your template file to reflect the new parameters you added above**

If you are using the sample from the [git-repo](#) to follow along, you can start to make changes in The sample 5-VM-1-NodeTypes-Secure.paramters\_Step2.JSON

Edit your Resource Manager Template parameter File, add the two new parameters for secCertificateThumbprint and secCertificateUrlValue.

```

"secCertificateThumbprint": {
    "value": "thumbprint value"
},
"secCertificateUrlValue": {
    "value": "Refers to the location URL in your key vault where the certificate was uploaded, it is should be in the format of https://<name of the vault>.vault.azure.net:443/secrets/<exact location>"
},

```

#### **Deploy the template to Azure**

- You are now ready to deploy your template to Azure. Open an Azure PS version 1+ command prompt.
- Log in to your Azure Account and select the specific azure subscription. This is an important step for folks who have access to more than one azure subscription.

```

Login-AzureRmAccount
Select-AzureRmSubscription -SubscriptionId <Subscription ID>

```

Test the template prior to deploying it. Use the same Resource Group that your cluster is currently deployed to.

```
Test-AzureRmResourceGroupDeployment -ResourceGroupName <Resource Group that your cluster is currently deployed to> -TemplateFile <PathToTemplate>
```

Deploy the template to your resource group. Use the same Resource Group that your cluster is currently deployed to. Run the New-AzureRmResourceGroupDeployment command. You do not need to specify the mode, since the default value is **incremental**.

#### NOTE

If you set Mode to Complete, you can inadvertently delete resources that are not in your template. So do not use it in this scenario.

```
New-AzureRmResourceGroupDeployment -Name ExampleDeployment -ResourceGroupName <Resource Group that your cluster is currently deployed to> -TemplateFile <PathToTemplate>
```

Here is a filled out example of the same powershell.

```
$ResourceGroup2 = "chackosecure5"
$templatefile = "C:\GitHub\Service-Fabric\ARM Templates\Cert Rollover Sample\5-VM-1-NodeTypes-Secure_Step2.json"
$templateparmfile = "C:\GitHub\Service-Fabric\ARM Templates\Cert Rollover Sample\5-VM-1-NodeTypes-Secure.parameters_Step2.json"

New-AzureRmResourceGroupDeployment -ResourceGroupName $ResourceGroup2 -TemplateParameterFile $templateparmfile -TemplateUri $templatefile -clusterName $ResourceGroup2
```

Once the deployment is complete, connect to your cluster using the new Certificate and perform some queries. If you are able to do. Then you can delete the old certificate.

If you are using a self-signed certificate, do not forget to import them into your local TrustedPeople cert store.

```
##### Set up the certs on your local box
Import-PfxCertificate -Exportable -CertStoreLocation Cert:\CurrentUser\TrustedPeople -FilePath c:\MyCertificates\chackdanTestCertificate9.pfx -Password (ConvertTo-SecureString -String abcd123 -AsPlainText -Force)
Import-PfxCertificate -Exportable -CertStoreLocation Cert:\CurrentUser\My -FilePath c:\MyCertificates\chackdanTestCertificate9.pfx -Password (ConvertTo-SecureString -String abcd123 -AsPlainText -Force)
```

For quick reference here is the command to connect to a secure cluster

```
$ClusterName= "chackosecure5.westus.cloudapp.azure.com:19000"
$CertThumbprint= "70EF5E22ADB649799DA3C8B6A6BF7SD1D630F8F3"

Connect-serviceFabricCluster -ConnectionEndpoint $ClusterName -KeepAliveIntervalInSec 10 ` -X509Credential ` -ServerCertThumbprint $CertThumbprint ` -FindType FindByThumbprint ` -FindValue $CertThumbprint ` -StoreLocation CurrentUser ` -StoreName My
```

For quick reference here is the command to get cluster health

## Deploying Application certificates to the cluster.

You can use the same steps as outlined in Steps 5 above to have the certificates deployed from a keyvault to the Nodes. you just need define and use different parameters.

## Adding or removing Client certificates

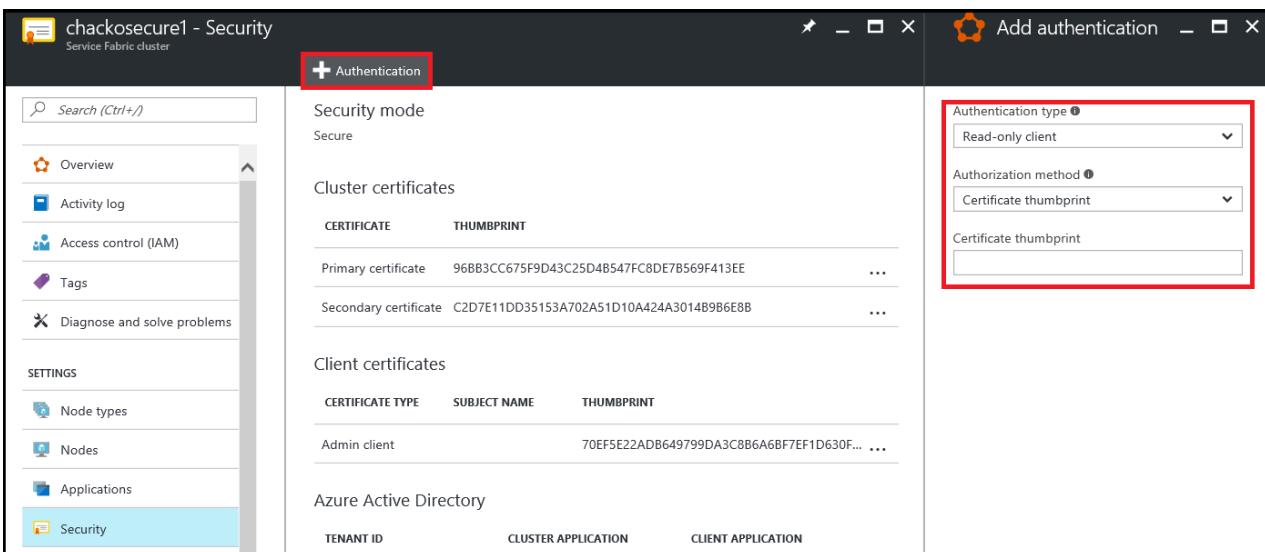
In addition to the cluster certificates, you can add client certificates to perform management operations on a service fabric cluster.

You can add two kinds of client certificates - Admin or Read-only. These then can be used to control access to the admin operations and Query operations on the cluster. By default, the cluster certificates are added to the allowed Admin certificates list.

you can specify any number of client certificates. Each addition/deletion results in a configuration update to the service fabric cluster

### Adding client certificates - Admin or Read-Only via portal

1. Navigate to the Security blade, and select the '+ Authentication' button on top of the security blade.
2. On the 'Add Authentication' blade, choose the 'Authentication Type' - 'Read-only client' or 'Admin client'
3. Now choose the Authorization method. This indicates to Service Fabric whether it should look up this certificate by using the subject name or the thumbprint. In general, it is not a good security practice to use the authorization method of subject name.



### Deletion of Client Certificates - Admin or Read-Only using the portal

To remove a secondary certificate from being used for cluster security, Navigate to the Security blade and select the 'Delete' option from the context menu on the specific certificate.

## Next steps

Read these articles for more information on cluster management:

- [Service Fabric Cluster upgrade process and expectations from you](#)
- [Setup role-based access for clients](#)

# Delete a Service Fabric cluster on Azure and the resources it uses

7/11/2017 • 3 min to read • [Edit Online](#)

A Service Fabric cluster is made up of many other Azure resources in addition to the cluster resource itself. So to completely delete a Service Fabric cluster you also need to delete all the resources it is made of. You have two options: Either delete the resource group that the cluster is in (which deletes the cluster resource and any other resources in the resource group) or specifically delete the cluster resource and its associated resources (but not other resources in the resource group).

## NOTE

Deleting the cluster resource **does not** delete all the other resources that your Service Fabric cluster is composed of.

## Delete the entire resource group (RG) that the Service Fabric cluster is in

This is the easiest way to ensure that you delete all the resources associated with your cluster, including the resource group. You can delete the resource group using PowerShell or through the Azure portal. If your resource group has resources that are not related to Service fabric cluster, then you can delete specific resources.

### Delete the resource group using Azure PowerShell

You can also delete the resource group by running the following Azure PowerShell cmdlets. Make sure Azure PowerShell 1.0 or greater is installed on your computer. If you have not done this before, follow the steps outlined in [How to install and Configure Azure PowerShell](#).

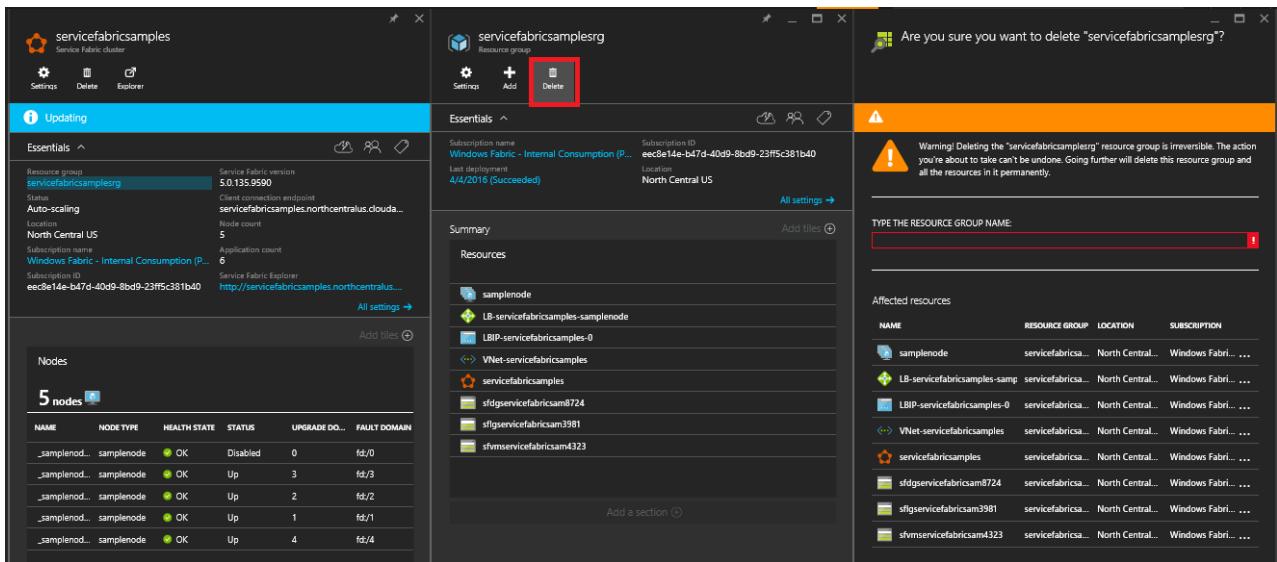
Open a PowerShell window and run the following PS cmdlets:

```
Login-AzureRmAccount  
Remove-AzureRmResourceGroup -Name <name of ResourceGroup> -Force
```

You will get a prompt to confirm the deletion if you did not use the *-Force* option. On confirmation the RG and all the resources it contains are deleted.

### Delete a resource group in the Azure portal

1. Login to the [Azure portal](#).
2. Navigate to the Service Fabric cluster you want to delete.
3. Click on the Resource Group name on the cluster essentials page.
4. This brings up the **Resource Group Essentials** page.
5. Click **Delete**.
6. Follow the instructions on that page to complete the deletion of the resource group.



## Delete the cluster resource and the resources it uses, but not other resources in the resource group

If your resource group has only resources that are related to the Service Fabric cluster you want to delete, then it is easier to delete the entire resource group. If you want to selectively delete the resources one-by-one in your resource group, then follow these steps.

If you deployed your cluster using the portal or using one of the Service Fabric Resource Manager templates from the template gallery, then all the resources that the cluster uses are tagged with the following two tags. You can use them to decide which resources you want to delete.

**Tag#1:** Key = `clusterName`, Value = 'name of the cluster'

**Tag#2:** Key = `resourceName`, Value = `ServiceFabric`

### Delete specific resources in the Azure portal

1. Login to the [Azure portal](#).
2. Navigate to the Service Fabric cluster you want to delete.
3. Go to **All settings** on the essentials blade.
4. Click on **Tags** under **Resource Management** in the settings blade.
5. Click on one of the **Tags** in the tags blade to get a list of all the resources with that tag.

The screenshot shows the Azure portal interface for a Service Fabric cluster named 'servicefabricsamples'. On the left, the 'Nodes' section displays 5 nodes with their status and upgrade details. On the right, the 'Tags' section shows two entries: 'clusterName : servicefabricsamples' and 'resourceType : Service Fabric'. Both of these entries are highlighted with red boxes.

- Once you have the list of tagged resources, click on each of the resources and delete them.

The screenshot shows the 'Tags' blade for the cluster 'servicefabricsamples'. It lists the tag 'clusterName : servicefabricsamples'. Below this, the 'Subscriptions' blade is open, showing a list of resources with their names and subscriptions. All the resource names listed are highlighted with red boxes: 'samplenode', 'LBIP-servicefabricsamples-0', 'VNet-servicefabricsamples', 'servicefabricsamples', 'sfdgservicefabricsam8724', 'sflgservicefabricsam3981', and 'sfvmservicefabricsam4323'. The entire 'Subscriptions' blade area is also highlighted with a large red box.

### Delete the resources using Azure PowerShell

You can delete the resources one-by-one by running the following Azure PowerShell cmdlets. Make sure Azure PowerShell 1.0 or greater is installed on your computer. If you have not done this before, follow the steps outlined in [How to install and Configure Azure PowerShell](#).

Open a PowerShell window and run the following PS cmdlets:

```
Login-AzureRmAccount
```

For each of the resources you want to delete, run the following:

```
Remove-AzureRmResource -ResourceName "<name of the Resource>" -ResourceType "<Resource Type>" -  
ResourceGroupName "<name of the resource group>" -Force
```

To delete the cluster resource, run the following:

```
Remove-AzureRmResource -ResourceName "<name of the Resource>" -ResourceType "Microsoft.ServiceFabric/clusters"  
-ResourceGroupName "<name of the resource group>" -Force
```

## Next steps

Read the following to also learn about upgrading a cluster and partitioning services:

- [Learn about cluster upgrades](#)
- [Learn about partitioning stateful services for maximum scale](#)

# Plan and prepare your Service Fabric Standalone cluster deployment

Perform the following steps before you create your cluster.

## Plan your cluster infrastructure

You are about to create a Service Fabric cluster on machines you "own", so you can decide what kinds of failures you want the cluster to survive. For example, do you need separate power lines or Internet connections supplied to these machines? In addition, consider the physical security of these machines. Where are the machines located and who needs access to them? After you make these decisions, you can logically map the machines to various fault domains (see next step). The infrastructure planning for production clusters is more involved than for test clusters.

## Determine the number of fault domains and upgrade domains

A *fault domain (FD)* is a physical unit of failure and is directly related to the physical infrastructure in the data centers. A fault domain consists of hardware components (computers, switches, networks, and more) that share a single point of failure. Although there is no 1:1 mapping between fault domains and racks, loosely speaking, each rack can be considered a fault domain.

When you specify FDs in ClusterConfig.json, you can choose the name for each FD. Service Fabric supports hierarchical FDs, so you can reflect your infrastructure topology in them. For example, the following FDs are valid:

- "faultDomain": "fd:/Room1/Rack1/Machine1"
- "faultDomain": "fd:/FD1"
- "faultDomain": "fd:/Room1/Rack1/PDU1/M1"

An *upgrade domain (UD)* is a logical unit of nodes. During Service Fabric orchestrated upgrades (either an application upgrade or a cluster upgrade), all nodes in a UD are taken down to perform the upgrade while nodes in other UDs remain available to serve requests. The firmware upgrades you perform on your machines do not honor UDs, so you must do them one machine at a time.

The simplest way to think about these concepts is to consider FDs as the unit of unplanned failure and UDs as the unit of planned maintenance.

When you specify UDs in ClusterConfig.json, you can choose the name for each UD. For example, the following names are valid:

- "upgradeDomain": "UD0"
- "upgradeDomain": "UD1A"
- "upgradeDomain": "DomainRed"
- "upgradeDomain": "Blue"

For more detailed information on FDs and UDs, see [Describing a Service Fabric cluster](#).

A cluster in production should span at least three FDs in order to be supported in a production environment, if you have full control over the maintenance and management of the nodes, i.e. you are responsible for updating and replacing machines. For clusters running in environments (i.e. Amazon Web Services VM instances) where you do

not have full control over the machines, you should have a minimum of five FDs in your cluster. Each FD can have one or more nodes. This is to prevent issues caused by machine upgrades and updates which, depending on their timing, can interfere with the running of applications and services in clusters.

## Determine the initial cluster size

Generally, the number of nodes in your cluster is determined based on your business needs, i.e., how many services and containers will be running on the cluster and how many resources do you need to sustain your workloads. For production clusters, we recommend having at least 5 nodes in your cluster, spanning 5 FDs. However, as described above, if you have full control over your nodes and can span three FDs, then three nodes should also do the job.

Test clusters running stateful workloads should have three nodes, whereas test clusters only running stateless workloads only need one node. It should also be noted that for development purposes, you can have more than one node on a given machine. In a production environment however, Service Fabric supports only one node per physical or virtual machine.

## Prepare the machines that will serve as nodes

Here are some recommended specs for each machine that you want to add to the cluster:

- A minimum of 16 GB of RAM
- A minimum of 40 GB available disk space
- A 4 core or greater CPU
- Connectivity to a secure network or networks for all machines
- Windows Server 2012 R2 or Windows Server 2016
- [.NET Framework 4.5.1 or higher](#), full install
- [Windows PowerShell 3.0](#)
- The [RemoteRegistry service](#) should be running on all the machines

The cluster administrator deploying and configuring the cluster must have [administrator privileges](#) on each of the machines. You cannot install Service Fabric on a domain controller.

## Download the Service Fabric standalone package for Windows Server

[Download Link - Service Fabric Standalone Package - Windows Server](#) and unzip the package, either to a deployment machine that is not part of the cluster, or to one of the machines that will be a part of your cluster.

## Modify cluster configuration

To create a standalone cluster you have to create a standalone cluster configuration ClusterConfig.json file, which describes the specification of the cluster. You can base the configuration file on the templates found at the below link.

[Standalone Cluster Configurations](#)

For details on the sections in this file, see [Configuration settings for standalone Windows cluster](#).

Open one of the ClusterConfig.json files from the package you downloaded and modify the following settings:

CONFIGURATION SETTING	DESCRIPTION
-----------------------	-------------

CONFIGURATION SETTING	DESCRIPTION
<b>NodeTypes</b>	<p>Node types allow you to separate your cluster nodes into various groups. A cluster must have at least one NodeType. All nodes in a group have the following common characteristics:</p> <ul style="list-style-type: none"> <li><b>Name</b> - This is the node type name.</li> <li><b>Endpoint Ports</b> - These are various named end points (ports) that are associated with this node type. You can use any port number that you wish, as long as they do not conflict with anything else in this manifest and are not already in use by any other application running on the machine/VM.</li> <li><b>Placement Properties</b> - These describe properties for this node type that you use as placement constraints for the system services or your services. These properties are user-defined key/value pairs that provide extra meta data for a given node. Examples of node properties would be whether the node has a hard drive or graphics card, the number of spindles in its hard drive, cores, and other physical properties.</li> <li><b>Capacities</b> - Node capacities define the name and amount of a particular resource that a particular node has available for consumption. For example, a node may define that it has capacity for a metric called "MemoryInMb" and that it has 2048 MB available by default. These capacities are used at runtime to ensure that services that require particular amounts of resources are placed on the nodes that have those resources available in the required amounts.</li> <li><b>IsPrimary</b> - If you have more than one NodeType defined ensure that only one is set to primary with the value <i>true</i>, which is where the system services run. All other node types should be set to the value <i>false</i></li> </ul>
<b>Nodes</b>	<p>These are the details for each of the nodes that are part of the cluster (node type, node name, IP address, fault domain, and upgrade domain of the node). The machines you want the cluster to be created on need to be listed here with their IP addresses.</p> <p>If you use the same IP address for all the nodes, then a one-box cluster is created, which you can use for testing purposes. Do not use One-box clusters for deploying production workloads.</p>

After the cluster configuration has had all settings configured to the environment, it can be tested against the cluster environment (step 7).

## Environment setup

When a cluster administrator configures a Service Fabric standalone cluster, the environment needs to be set up with the following criteria:

1. The user creating the cluster should have administrator-level security privileges to all machines that are listed as nodes in the cluster configuration file.
2. Machine from which the cluster is created, as well as each cluster node machine must:
  3. Have Service Fabric SDK uninstalled
  4. Have Service Fabric runtime uninstalled
  5. Have the Windows Firewall service (mpssvc) enabled
  6. Have the Remote Registry Service (remoteregistry) enabled
  7. Have file sharing (SMB) enabled
  8. Have necessary ports opened, based on cluster configuration ports

9. Have necessary ports opened for Windows SMB and Remote Registry service: 135, 137, 138, 139, and 445
10. Have network connectivity to one another
11. None of the cluster node machines should be a Domain Controller.
12. If the cluster to be deployed is a secure cluster, validate the necessary security prerequisites are in place, and are configured correctly against the configuration.
13. If the cluster machines are not internet-accessible, set the following in the cluster configuration:
14. Disable telemetry: Under *properties* set "*enableTelemetry*": *false*
15. Disable automatic Fabric version downloading & notifications that the current cluster version is nearing end of support: Under *properties* set "*fabricClusterAutoupgradeEnabled*": *false*
16. Alternatively, if network internet access is limited to white-listed domains, the domains below are required for automatic upgrade: go.microsoft.com download.microsoft.com
17. Set appropriate Service Fabric antivirus exclusions:

#### **ANTIVIRUS EXCLUDED DIRECTORIES**

Program Files\Microsoft Service Fabric

FabricDataRoot (from cluster configuration)

FabricLogRoot (from cluster configuration)

#### **ANTIVIRUS EXCLUDED PROCESSES**

Fabric.exe

FabricHost.exe

FabricInstallerService.exe

FabricSetup.exe

FabricDeployer.exe

ImageBuilder.exe

FabricGateway.exe

FabricDCA.exe

FabricFAS.exe

FabricUOS.exe

FabricRM.exe

FileStoreService.exe

## **Validate environment using TestConfiguration script**

The TestConfiguration.ps1 script can be found in the standalone package. It is used as a Best Practices Analyzer to validate some of the criteria above and should be used as a sanity check to validate whether a cluster can be

deployed on a given environment. If there is any failure, refer to the list under [Environment Setup](#) for troubleshooting.

This script can be run on any machine that has administrator access to all the machines that are listed as nodes in the cluster configuration file. The machine that this script is run on does not have to be part of the cluster.

```
PS C:\temp\Microsoft.Azure.ServiceFabric.WindowsServer> .\TestConfiguration.ps1 -ClusterConfigFilePath  
.\\ClusterConfig.Unsecure.DevCluster.json  
Trace folder already exists. Traces will be written to existing trace folder:  
C:\\temp\\Microsoft.Azure.ServiceFabric.WindowsServer\\DeploymentTraces  
Running Best Practices Analyzer...  
Best Practices Analyzer completed successfully.
```

```
LocalAdminPrivilege      : True  
IsJsonValid             : True  
IsCabValid              : True  
RequiredPortsOpen        : True  
RemoteRegistryAvailable : True  
FirewallAvailable       : True  
RpcCheckPassed          : True  
NoConflictingInstallations : True  
FabricInstallable       : True  
Passed                  : True
```

Currently this configuration testing module does not validate the security configuration so this has to be done independently.

#### NOTE

We are continually making improvements to make this module more robust, so if there is a faulty or missing case which you believe isn't currently caught by TestConfiguration, please let us know through our [support channels](#).

## Next steps

- [Create a standalone cluster running on Windows Server](#)

# Create your first Service Fabric standalone cluster

8/22/2017 • 3 min to read • [Edit Online](#)

You can create a Service Fabric standalone cluster on any virtual machines or computers running Windows Server 2012 R2 or Windows Server 2016, on-premises or in the cloud. This quickstart helps you to create a development standalone cluster in just a few minutes. When you're finished, you have a three-node cluster running on a single computer that you can deploy apps to.

## Before you begin

Service Fabric provides a setup package to create Service Fabric standalone clusters. [Download the setup package](#). Unzip the setup package to a folder on the computer or virtual machine where you are setting up the development cluster. The contents of the setup package are described in detail [here](#).

The cluster administrator deploying and configuring the cluster must have administrator privileges on the computer. You cannot install Service Fabric on a domain controller.

## Validate the environment

The *TestConfiguration.ps1* script in the standalone package is used as a best practices analyzer to validate whether a cluster can be deployed on a given environment. [Deployment preparation](#) lists the pre-requisites and environment requirements. Run the script to verify if you can create the development cluster:

```
.\TestConfiguration.ps1 -ClusterConfigFilePath .\ClusterConfig.Unsecure.DevCluster.json
```

## Create the cluster

Several sample cluster configuration files are installed with the setup package.

*ClusterConfig.Unsecure.DevCluster.json* is the simplest cluster configuration: an unsecure, three-node cluster running on a single computer. Other config files describe single or multi-machine clusters secured with X.509 certificates or Windows security. You don't need to modify any of the default config settings for this tutorial, but look through the config file and get familiar with the settings. The **nodes** section describes the three nodes in the cluster: name, IP address, [node type](#), [fault domain](#), and [upgrade domain](#). The **properties** section defines the [security](#), [reliability level](#), [diagnostics collection](#), and [types of nodes](#) for the cluster.

This cluster is unsecure. Anyone can connect anonymously and perform management operations, so production clusters should always be secured using X.509 certificates or Windows security. Security is only configured at cluster creation time and it is not possible to enable security after the cluster is created. Read [Secure a cluster](#) to learn more about Service Fabric cluster security.

To create the three-node development cluster, run the *CreateServiceFabricCluster.ps1* script from an administrator PowerShell session:

```
.\CreateServiceFabricCluster.ps1 -ClusterConfigFilePath .\ClusterConfig.Unsecure.DevCluster.json -AcceptEULA
```

The Service Fabric runtime package is automatically downloaded and installed at time of cluster creation.

## Connect to the cluster

Your three-node development cluster is now running. The ServiceFabric PowerShell module is installed with the runtime. You can verify that the cluster is running from the same computer or from a remote computer with the Service Fabric runtime. The [Connect-ServiceFabricCluster](#) cmdlet establishes a connection to the cluster.

```
Connect-ServiceFabricCluster -ConnectionEndpoint localhost:19000
```

See [Connect to a secure cluster](#) for other examples of connecting to a cluster. After connecting to the cluster, use the [Get-ServiceFabricNode](#) cmdlet to display a list of nodes in the cluster and status information for each node.

**HealthState** should be OK for each node.

```
PS C:\temp\Microsoft.Azure.ServiceFabric.WindowsServer> Get-ServiceFabricNode |Format-Table
```

	NodeDeactivationInfo	NodeName	IpAddressOrFQDN	NodeType	CodeVersion	ConfigVersion	NodeStatus	NodeUpTime	NodeDownTime	HealthState
00:00:00		vm2	localhost	NodeType2	5.6.220.9494	0	Up	00:03:38		OK
00:00:00		vm1	localhost	NodeType1	5.6.220.9494	0	Up	00:03:38		OK
00:00:00		vm0	localhost	NodeType0	5.6.220.9494	0	Up	00:02:43		OK
00:00:00										

## Visualize the cluster using Service Fabric explorer

[Service Fabric Explorer](#) is a good tool for visualizing your cluster and managing applications. Service Fabric Explorer is a service that runs in the cluster, which you access using a browser by navigating to <http://localhost:19080/Explorer>.

The cluster dashboard provides an overview of your cluster, including a summary of application and node health. The node view shows the physical layout of the cluster. For a given node, you can inspect which applications have code deployed on that node.

The screenshot shows the Service Fabric Explorer interface. The top navigation bar includes back and forward buttons, a search bar with the URL 'http://localhost:19080/Explorer', and icons for home, star, gear, and smiley face. The main header says 'Service Fabric Explorer'. On the left, there's a sidebar with 'Microsoft Azure' and 'Service Fabric Explorer' tabs, and buttons for 'OK', 'Warning', and 'Error'. The sidebar also has a 'Search Cluster' input field. The main content area is titled 'Nodes' and contains a table with the following data:

Name	Address	Node Type	Upgrade Domain	Fault Domain	Health State	Status
vm0	localhost	NodeType0	UD0	fd/dc1/r0	OK	Up
vm1	localhost	NodeType1	UD1	fd/dc2/r0	OK	Up
vm2	localhost	NodeType2	UD2	fd/dc3/r0	OK	Up

## Remove the cluster

To remove a cluster, run the *RemoveServiceFabricCluster.ps1* PowerShell script from the package folder and pass in the path to the JSON configuration file. You can optionally specify a location for the log of the deletion.

```
# Removes Service Fabric cluster nodes from each computer in the configuration file.  
.\\RemoveServiceFabricCluster.ps1 -ClusterConfigFilePath .\\ClusterConfig.Unsecure.DevCluster.json -Force
```

To remove the Service Fabric runtime from the computer, run the following PowerShell script from the package folder.

```
# Removes Service Fabric from the current computer.  
.\\CleanFabric.ps1
```

## Next steps

Now that you have set up a development standalone cluster, try the following articles:

- [Set up a multi-machine standalone cluster](#) and enable security.
- [Deploy apps using PowerShell](#)

# Create a standalone cluster running on Windows Server

8/11/2017 • 6 min to read • [Edit Online](#)

You can use Azure Service Fabric to create Service Fabric clusters on any virtual machines or computers running Windows Server. This means you can deploy and run Service Fabric applications in any environment that contains a set of interconnected Windows Server computers, be it on premises or with any cloud provider. Service Fabric provides a setup package to create Service Fabric clusters called the standalone Windows Server package.

This article walks you through the steps for creating a Service Fabric standalone cluster.

## NOTE

This standalone Windows Server package is commercially available and may be used for production deployments. This package may contain new Service Fabric features that are in "Preview". Scroll down to "[Preview features included in this package](#)." section for the list of the preview features. You can [download a copy of the EULA](#) now.

## Get support for the Service Fabric for Windows Server package

- Ask the community about the Service Fabric standalone package for Windows Server in the [Azure Service Fabric forum](#).
- Open a ticket for [Professional Support for Service Fabric](#). Learn more about Professional Support from Microsoft [here](#).
- You can also get support for this package as a part of [Microsoft Premier Support](#).
- For more details, please see [Azure Service Fabric support options](#).
- To collect logs for support purposes, run the [Service Fabric Standalone Log collector](#).

## Download the Service Fabric for Windows Server package

To create the cluster, use the Service Fabric for Windows Server package (Windows Server 2012 R2 and newer) found here:

[Download Link - Service Fabric Standalone Package - Windows Server](#)

Find details on contents of the package [here](#).

The Service Fabric runtime package is automatically downloaded at time of cluster creation. If deploying from a machine not connected to the internet, please download the runtime package out of band from here:

[Download Link - Service Fabric Runtime - Windows Server](#)

Find Standalone Cluster Configuration samples at:

[Standalone Cluster Configuration Samples](#)

## Create the cluster

Service Fabric can be deployed to a one-machine development cluster by using the `ClusterConfig.Unsecure.DevCluster.json` file included in [Samples](#).

Unpack the standalone package to your machine, copy the sample config file to the local machine, then run the `CreateServiceFabricCluster.ps1` script through an administrator PowerShell session, from the standalone package

folder:

### Step 1A: Create an unsecured local development cluster

```
.\CreateServiceFabricCluster.ps1 -ClusterConfigFilePath .\ClusterConfig.Unsecure.DevCluster.json -AcceptEULA
```

See the Environment Setup section at [Plan and prepare your cluster deployment](#) for troubleshooting details.

If you're finished running development scenarios, you can remove the Service Fabric cluster from the machine by referring to steps in section "[Remove a cluster](#)".

### Step 1B: Create a multi-machine cluster

After you have gone through the planning and preparation steps detailed at the below link, you are ready to create your production cluster using your cluster configuration file.

[Plan and prepare your cluster deployment](#)

1. Validate the configuration file you have written by running the *TestConfiguration.ps1* script from the standalone package folder:

```
.\TestConfiguration.ps1 -ClusterConfigFilePath .\ClusterConfig.json
```

You should see output like below. If the bottom field "Passed" is returned as "True", sanity checks have passed and the cluster looks to be deployable based on the input configuration.

```
Trace folder already exists. Traces will be written to existing trace folder:  
C:\temp\Microsoft.Azure.ServiceFabric.WindowsServer\DeploymentTraces  
Running Best Practices Analyzer...  
Best Practices Analyzer completed successfully.  
  
LocalAdminPrivilege      : True  
IsJsonValid              : True  
IsCabValid               : True  
RequiredPortsOpen         : True  
RemoteRegistryAvailable   : True  
FirewallAvailable        : True  
RpcCheckPassed           : True  
NoConflictingInstallations : True  
FabricInstallable        : True  
Passed                  : True
```

2. Create the cluster: Run the *CreateServiceFabricCluster.ps1* script to deploy the Service Fabric cluster across each machine in the configuration.

```
.\CreateServiceFabricCluster.ps1 -ClusterConfigFilePath .\ClusterConfig.json -AcceptEULA
```

#### NOTE

Deployment traces are written to the VM/machine on which you ran the *CreateServiceFabricCluster.ps1* PowerShell script. These can be found in the subfolder DeploymentTraces, based in the directory from which the script was run. To see if Service Fabric was deployed correctly to a machine, find the installed files in the FabricDataRoot directory, as detailed in the cluster configuration file FabricSettings section (by default c:\ProgramData\SF). As well, FabricHost.exe and Fabric.exe processes can be seen running in Task Manager.

### Step 1C: Create an offline (internet-disconnected) cluster

The Service Fabric runtime package is automatically downloaded at cluster creation. When deploying a cluster to

machines not connected to the internet, you will need to download the Service Fabric runtime package separately, and provide the path to it at cluster creation. The runtime package can be downloaded separately, from another machine connected to the internet, at [Download Link - Service Fabric Runtime - Windows Server](#). Copy the runtime package to where you are deploying the offline cluster from, and create the cluster by running `CreateServiceFabricCluster.ps1` with the `-FabricRuntimePackagePath` parameter included, as shown below:

```
.\CreateServiceFabricCluster.ps1 -ClusterConfigFilePath .\ClusterConfig.json -FabricRuntimePackagePath  
.\\MicrosoftAzureServiceFabric.cab
```

where `.\ClusterConfig.json` and `.\MicrosoftAzureServiceFabric.cab` are the paths to the cluster configuration and the runtime .cab file respectively.

## Step 2: Connect to the cluster

To connect to a secure cluster, see [Service fabric connect to secure cluster](#).

To connect to an unsecure cluster, run the following PowerShell command:

```
Connect-ServiceFabricCluster -ConnectionEndpoint <*IPAddressofaMachine*>:<Client connection end point port>
```

Example:

```
Connect-ServiceFabricCluster -ConnectionEndpoint 192.13.123.2345:19000
```

## Step 3: Bring up Service Fabric Explorer

Now you can connect to the cluster with Service Fabric Explorer either directly from one of the machines with <http://localhost:19080/Explorer/index.html> or remotely with <http://<IPAddressofaMachine>:19080/Explorer/index.html>.

## Add and remove nodes

You can add or remove nodes to your standalone Service Fabric cluster as your business needs change. See [Add or Remove nodes to a Service Fabric standalone cluster](#) for detailed steps.

## Remove a cluster

To remove a cluster, run the `RemoveServiceFabricCluster.ps1` PowerShell script from the package folder and pass in the path to the JSON configuration file. You can optionally specify a location for the log of the deletion.

This script can be run on any machine that has administrator access to all the machines that are listed as nodes in the cluster configuration file. The machine that this script is run on does not have to be part of the cluster.

```
# Removes Service Fabric from each machine in the configuration  
.\\RemoveServiceFabricCluster.ps1 -ClusterConfigFilePath .\\ClusterConfig.json -Force
```

```
# Removes Service Fabric from the current machine  
.\\CleanFabric.ps1
```

## Telemetry data collected and how to opt out of it

As a default, the product collects telemetry on the Service Fabric usage to improve the product. The Best Practice Analyzer that runs as a part of the setup checks for connectivity to <https://vortex.data.microsoft.com/collect/v1>. If

it is not reachable, the setup fails unless you opt out of telemetry.

1. The telemetry pipeline tries to upload the following data to <https://vortex.data.microsoft.com/collect/v1> once every day. It is a best-effort upload and has no impact on the cluster functionality. The telemetry is only sent from the node that runs the failover manager primary. No other nodes send out telemetry.
2. The telemetry consists of the following:
  - Number of services
  - Number of ServiceTypes
  - Number of Applications
  - Number of ApplicationUpgrades
  - Number of FailoverUnits
  - Number of InBuildFailoverUnits
  - Number of UnhealthyFailoverUnits
  - Number of Replicas
  - Number of InBuildReplicas
  - Number of StandByReplicas
  - Number of OfflineReplicas
  - CommonQueueLength
  - QueryQueueLength
  - FailoverUnitQueueLength
  - CommitQueueLength
  - Number of Nodes
  - IsContextComplete: True/False
  - ClusterId: This is a GUID randomly generated for each cluster
  - ServiceFabricVersion
  - IP address of the virtual machine or machine from which the telemetry is uploaded

To disable telemetry, add the following to *properties* in your cluster config: *enableTelemetry: false*.

## Preview features included in this package

None.

### NOTE

Starting with the new [GA version of the standalone cluster for Windows Server \(version 5.3.204.x\)](#), you can upgrade your cluster to future releases, manually or automatically. Refer to [Upgrade a standalone Service Fabric cluster version](#) document for details.

## Next steps

- [Deploy and remove applications using PowerShell](#)
- [Configuration settings for standalone Windows cluster](#)
- [Add or remove nodes to a standalone Service Fabric cluster](#)
- [Upgrade a standalone Service Fabric cluster version](#)
- [Create a standalone Service Fabric cluster with Azure VMs running Windows](#)
- [Secure a standalone cluster on Windows using Windows security](#)
- [Secure a standalone cluster on Windows using X509 certificates](#)

# Secure a standalone cluster on Windows by using X.509 certificates

10/30/2017 • 10 min to read • [Edit Online](#)

This article describes how to secure communication between the various nodes of your standalone Windows cluster. It also describes how to authenticate clients that connect to this cluster by using X.509 certificates. Authentication ensures that only authorized users can access the cluster and the deployed applications and perform management tasks. Certificate security should be enabled on the cluster when the cluster is created.

For more information on cluster security, such as node-to-node security, client-to-node security, and role-based access control, see [Cluster security scenarios](#).

## Which certificates do you need?

To start with, [download the Service Fabric for Windows Server package](#) to one of the nodes in your cluster. In the downloaded package, you find a ClusterConfig.X509.MultiMachine.json file. Open the file, and review the section for security under the properties section:

```
"security": {
    "metadata": "The Credential type X509 indicates this cluster is secured by using X509 certificates. The thumbprint format is d5 ec 42 3b 79 cb e5 07 fd 83 59 3c 56 b9 d5 31 24 25 42 64.",
    "ClusterCredentialType": "X509",
    "ServerCredentialType": "X509",
    "CertificateInformation": {
        "ClusterCertificate": {
            "Thumbprint": "[Thumbprint]",
            "ThumbprintSecondary": "[Thumbprint]",
            "X509StoreName": "My"
        },
        "ClusterCertificateCommonNames": {
            "CommonNames": [
                {
                    "CertificateCommonName": "[CertificateCommonName]",
                    "CertificateIssuerThumbprint": "[Thumbprint1,Thumbprint2,Thumbprint3,...]"
                }
            ],
            "X509StoreName": "My"
        },
        "ServerCertificate": {
            "Thumbprint": "[Thumbprint]",
            "ThumbprintSecondary": "[Thumbprint]",
            "X509StoreName": "My"
        },
        "ServerCertificateCommonNames": {
            "CommonNames": [
                {
                    "CertificateCommonName": "[CertificateCommonName]",
                    "CertificateIssuerThumbprint": "[Thumbprint1,Thumbprint2,Thumbprint3,...]"
                }
            ],
            "X509StoreName": "My"
        },
        "ClientCertificateThumbprints": [
            {
                "CertificateThumbprint": "[Thumbprint]",
                "IsAdmin": false
            },
            {
                "CertificateThumbprint": "[Thumbprint]",
                "IsAdmin": true
            }
        ],
        "ClientCertificateCommonNames": [
            {
                "CertificateCommonName": "[CertificateCommonName]",
                "CertificateIssuerThumbprint": "[Thumbprint]",
                "IsAdmin": true
            }
        ],
        "ReverseProxyCertificate": {
            "Thumbprint": "[Thumbprint]",
            "ThumbprintSecondary": "[Thumbprint]",
            "X509StoreName": "My"
        },
        "ReverseProxyCertificateCommonNames": {
            "CommonNames": [
                {
                    "CertificateCommonName": "[CertificateCommonName]"
                }
            ],
            "X509StoreName": "My"
        }
    }
},
```

This section describes the certificates that you need to secure your standalone Windows cluster. If you specify a cluster certificate, set the value of `ClusterCredentialType` to `X509`. If you specify a server certificate for outside connections, set the `ServerCredentialType` to `X509`. Although not mandatory, we recommend that you have both of these certificates for a properly secured cluster. If you set these values to `X509`, you also must specify the corresponding certificates or Service Fabric throws an exception. In some scenarios, you might want to specify only the `ClientCertificateThumbprints` or the `ReverseProxyCertificate`. In those scenarios, you don't need to set `ClusterCredentialType` or `ServerCredentialType` to `X509`.

**NOTE**

A [thumbprint](#) is the primary identity of a certificate. To find out the thumbprint of the certificates that you create, see [Retrieve a thumbprint of a certificate](#).

The following table lists the certificates that you need on your cluster setup:

CERTIFICATE INFORMATION SETTING	DESCRIPTION
<code>ClusterCertificate</code>	Recommended for a test environment. This certificate is required to secure the communication between the nodes on a cluster. You can use two different certificates, a primary and a secondary, for upgrade. Set the thumbprint of the primary certificate in the <code>Thumbprint</code> section and that of the secondary in the <code>ThumbprintSecondary</code> variables.
<code>ClusterCertificateCommonNames</code>	Recommended for a production environment. This certificate is required to secure the communication between the nodes on a cluster. You can use one or two cluster certificate common names. The <code>CertificateIssuerThumbprint</code> corresponds to the thumbprint of the issuer of this certificate. If more than one certificate with the same common name is used, you can specify multiple issuer thumbprints.
<code>ServerCertificate</code>	Recommended for a test environment. This certificate is presented to the client when it tries to connect to this cluster. For convenience, you can choose to use the same certificate for <code>ClusterCertificate</code> and <code>ServerCertificate</code> . You can use two different server certificates, a primary and a secondary, for upgrade. Set the thumbprint of the primary certificate in the <code>Thumbprint</code> section and that of the secondary in the <code>ThumbprintSecondary</code> variables.
<code>ServerCertificateCommonNames</code>	Recommended for a production environment. This certificate is presented to the client when it tries to connect to this cluster. The <code>CertificateIssuerThumbprint</code> corresponds to the thumbprint of the issuer of this certificate. If more than one certificate with the same common name is used, you can specify multiple issuer thumbprints. For convenience, you can choose to use the same certificate for <code>ClusterCertificateCommonNames</code> and <code>ServerCertificateCommonNames</code> . You can use one or two server certificate common names.

CERTIFICATE INFORMATION SETTING	DESCRIPTION
ClientCertificateThumbprints	Install this set of certificates on the authenticated clients. You can have a number of different client certificates installed on the machines that you want to allow access to the cluster. Set the thumbprint of each certificate in the CertificateThumbprint variable. If you set IsAdmin to <i>true</i> , the client with this certificate installed on it can do administrator management activities on the cluster. If IsAdmin is <i>false</i> , the client with this certificate can perform the actions only allowed for user-access rights, typically read-only. For more information on roles, see <a href="#">Role-Based Access Control (RBAC)</a> .
ClientCertificateCommonNames	Set the common name of the first client certificate for the CertificateCommonName. The CertificateIssuerThumbprint is the thumbprint for the issuer of this certificate. To learn more about common names and the issuer, see <a href="#">Work with certificates</a> .
ReverseProxyCertificate	Recommended for a test environment. This optional certificate can be specified if you want to secure your <a href="#">reverse proxy</a> . Make sure that reverseProxyEndpointPort is set in nodeTypes if you use this certificate.
ReverseProxyCertificateCommonNames	Recommended for a production environment. This optional certificate can be specified if you want to secure your <a href="#">reverse proxy</a> . Make sure that reverseProxyEndpointPort is set in nodeTypes if you use this certificate.

Here is an example cluster configuration where the cluster, server, and client certificates have been provided. For cluster/server/reverseProxy certificates, the thumbprint and the common name can't be configured together for the same certificate type.

```
{
  "name": "SampleCluster",
  "clusterConfigurationVersion": "1.0.0",
  "apiVersion": "2016-09-26",
  "nodes": [
    {
      "nodeId": "n1",
      "nodeTypeRef": "NodeType0",
      "ipAddress": "10.7.0.5",
      "faultDomain": "fd:/dc1/r0",
      "upgradeDomain": "UD0"
    },
    {
      "nodeId": "n2",
      "nodeTypeRef": "NodeType0",
      "ipAddress": "10.7.0.4",
      "faultDomain": "fd:/dc1/r1",
      "upgradeDomain": "UD1"
    },
    {
      "nodeId": "n3",
      "nodeTypeRef": "NodeType0",
      "ipAddress": "10.7.0.6",
      "faultDomain": "fd:/dc1/r2",
      "upgradeDomain": "UD2"
    }
  ],
  "properties": {
    "diagnosticsStore": {
      "metadata": "Please replace the diagnostics store with an actual file share accessible from all cluster machines."
    }
  }
}
```

```

        "dataDeletionAgeInDays": "7",
        "storeType": "FileShare",
        "IsEncrypted": "false",
        "connectionstring": "c:\\ProgramData\\SF\\DiagnosticsStore"
    }
    "security": {
        "metadata": "The Credential type X509 indicates this cluster is secured by using X509 certificates. The thumbprint format is d5 ec 42 3b 79 cb e5 07 fd 83 59 3c 56 b9 d5 31 24 25 42 64.",
        "ClusterCredentialType": "X509",
        "ServerCredentialType": "X509",
        "CertificateInformation": {
            "ClusterCertificateCommonNames": [
                "CommonNames": [
                    {
                        "CertificateCommonName": "myClusterCertCommonName",
                        "CertificateIssuerThumbprint": "7c fc 91 97 13 66 8d 9f a8 ee 71 2b a2 f4 37 62 00 03 49
0d"
                    }
                ],
                "X509StoreName": "My"
            },
            "ServerCertificateCommonNames": [
                "CommonNames": [
                    {
                        "CertificateCommonName": "myServerCertCommonName",
                        "CertificateIssuerThumbprint": "7c fc 91 97 13 16 8d ff a8 ee 71 2b a2 f4 62 62 00 03 49
0d"
                    }
                ],
                "X509StoreName": "My"
            },
            "ClientCertificateThumbprints": [
                {
                    "CertificateThumbprint": "c4 c18 8e aa a8 58 77 98 65 f8 61 4a 0d da 4c 13 c5 a1 37 6e",
                    "IsAdmin": false
                },
                {
                    "CertificateThumbprint": "71 de 04 46 7c 9e d0 54 4d 02 10 98 bc d4 4c 71 e1 83 41 4e",
                    "IsAdmin": true
                }
            ]
        }
    },
    "reliabilityLevel": "Bronze",
    "nodeTypes": [
        {
            "name": "NodeType0",
            "clientConnectionEndpointPort": "19000",
            "clusterConnectionEndpointPort": "19001",
            "leaseDriverEndpointPort": "19002",
            "serviceConnectionEndpointPort": "19003",
            "httpGatewayEndpointPort": "19080",
            "applicationPorts": {
                "startPort": "20001",
                "endPort": "20031"
            },
            "ephemeralPorts": {
                "startPort": "20032",
                "endPort": "20062"
            },
            "isPrimary": true
        }
    ],
    "fabricSettings": [
        {
            "name": "Setup",
            "parameters": [
                {
                    "name": "FabricDataRoot",
                    "value": "C:\\ProgramData\\SF"
                },
                {
                    "name": "FabricLogRoot",
                    "value": "C:\\ProgramData\\SF\\Log"
                }
            ]
        }
    ]
}

```

```
}
```

## Certificate rollover

When you use a certificate common name instead of a thumbprint, certificate rollover doesn't require a cluster configuration upgrade. For issuer thumbprint upgrades, make sure that the new thumbprint list intersects with the old list. You first have to do a config upgrade with the new issuer thumbprints, and then install the new certificates (both cluster/server certificate and issuer certificates) in the store. Keep the old issuer certificate in the certificate store for at least two hours after you install the new issuer certificate.

## Acquire the X.509 certificates

To secure communication within the cluster, you first need to obtain X.509 certificates for your cluster nodes. Additionally, to limit connection to this cluster to authorized machines/users, you need to obtain and install certificates for the client machines.

For clusters that are running production workloads, use a [certificate authority \(CA\)](#)-signed X.509 certificate to secure the cluster. For more information on how to obtain these certificates, see [How to obtain a certificate](#).

For clusters that you use for test purposes, you can choose to use a self-signed certificate.

## Optional: Create a self-signed certificate

One way to create a self-signed certificate that can be secured correctly is to use the CertSetup.ps1 script in the Service Fabric SDK folder in the directory C:\Program Files\Microsoft SDKs\Service Fabric\ClusterSetup\Secure. Edit this file to change the default name of the certificate. (Look for the value CN=ServiceFabricDevClusterCert.) Run this script as `.\CertSetup.ps1 -Install`.

Now export the certificate to a .pfx file with a protected password. First, get the thumbprint of the certificate.

1. From the **Start** menu, run **Manage computer certificates**.
2. Go to the **Local Computer\Personal** folder, and find the certificate you created.
3. Double-click the certificate to open it, select the **Details** tab, and scroll down to the **Thumbprint** field.
4. Remove the spaces, and copy the thumbprint value into the following PowerShell command.
5. Change the `string` value to a suitable secure password to protect it, and run the following in PowerShell:

```
$pswd = ConvertTo-SecureString -String "1234" -Force -AsPlainText  
Get-ChildItem -Path cert:\localMachine\my\<Thumbprint> | Export-PfxCertificate -FilePath C:\mypfx.pfx -  
Password $pswd
```

6. To see the details of a certificate installed on the machine, run the following PowerShell command:

```
$cert = Get-Item Cert:\LocalMachine\My\<Thumbprint>  
Write-Host $cert.ToString($true)
```

Alternatively, if you have an Azure subscription, follow the steps in the section [Add certificates to your key vault](#).

## Install the certificates

After you have certificates, you can install them on the cluster nodes. Your nodes need to have the latest Windows

PowerShell 3.x installed on them. Repeat these steps on each node for both cluster and server certificates and any secondary certificates.

1. Copy the .pfx file or files to the node.
2. Open a PowerShell window as an administrator, and enter the following commands. Replace \$pswd with the password that you used to create this certificate. Replace \$PfxFilePath with the full path of the .pfx copied to this node.

```
$pswd = "1234"
$PfxFilePath = "C:\mypfx.pfx"
Import-PfxCertificate -Exportable -CertStoreLocation Cert:\LocalMachine\My -FilePath $PfxFilePath -
Password (ConvertTo-SecureString -String $pswd -AsPlainText -Force)
```

3. Now set the access control on this certificate so that the Service Fabric process, which runs under the Network Service account, can use it by running the following script. Provide the thumbprint of the certificate and **NETWORK SERVICE** for the service account. You can check that the ACLs on the certificate are correct by opening the certificate in **Start > Manage computer certificates** and looking at **All Tasks > Manage Private Keys**.

```
param
(
[Parameter(Position=1, Mandatory=$true)]
[ValidateNotNullOrEmpty()]
[string]$pfxThumbPrint,

[Parameter(Position=2, Mandatory=$true)]
[ValidateNotNullOrEmpty()]
[string]$serviceAccount
)

$cert = Get-ChildItem -Path cert:\LocalMachine\My | Where-Object -FilterScript { $PSItem.ThumbPrint -eq
$pfxThumbPrint; }

# Specify the user, the permissions, and the permission type
$permission = "$($serviceAccount)", "FullControl", "Allow"
$accessRule = New-Object -TypeName System.Security.AccessControl.FileSystemAccessRule -ArgumentList
$permission

# Location of the machine-related keys
$keyPath = Join-Path -Path $env:ProgramData -ChildPath "\Microsoft\Crypto\RSA\MachineKeys"
$keyName = $cert.PrivateKey.CspKeyContainerInfo.UniqueKeyContainerName
$keyFullPath = Join-Path -Path $keyPath -ChildPath $keyName

# Get the current ACL of the private key
$acl = (Get-Item $keyFullPath).GetAccessControl('Access')

# Add the new ACE to the ACL of the private key
$acl.SetAccessRule($accessRule)

# Write back the new ACL
Set-Acl -Path $keyFullPath -AclObject $acl -ErrorAction Stop

# Observe the access rights currently assigned to this certificate
get-acl $keyFullPath| fl
```

4. Repeat the previous steps for each server certificate. You also can use these steps to install the client certificates on the machines that you want to allow access to the cluster.

## Create the secure cluster

After you configure the security section of the ClusterConfig.X509.MultiMachine.json file, you can proceed to the [Create the cluster](#) section to configure the nodes and create the standalone cluster. Remember to use the ClusterConfig.X509.MultiMachine.json file while you create the cluster. For example, your command might look like the following:

```
.\CreateServiceFabricCluster.ps1 -ClusterConfigFilePath .\ClusterConfig.X509.MultiMachine.json
```

After you have the secure standalone Windows cluster successfully running and have set up the authenticated clients to connect to it, follow the steps in the section [Connect to a cluster using PowerShell](#) to connect to it. For example:

```
$ConnectArgs = @{ ConnectionEndpoint = '10.7.0.5:19000'; X509Credential = $True; StoreLocation = 'LocalMachine'; StoreName = "MY"; ServerCertThumbprint = "057b9544a6f2733e0c8d3a60013a58948213f551"; FindType = 'FindByThumbprint'; FindValue = "057b9544a6f2733e0c8d3a60013a58948213f551" }  
Connect-ServiceFabricCluster $ConnectArgs
```

You can then run other PowerShell commands to work with this cluster. For example, you can run [Get-ServiceFabricNode](#) to show a list of nodes on this secure cluster.

To remove the cluster, connect to the node on the cluster where you downloaded the Service Fabric package, open a command line, and go to the package folder. Now run the following command:

```
.\RemoveServiceFabricCluster.ps1 -ClusterConfigFilePath .\ClusterConfig.X509.MultiMachine.json
```

#### NOTE

Incorrect certificate configuration might prevent the cluster from coming up during deployment. To self-diagnose security issues, look in the Event Viewer group **Applications and Services Logs > Microsoft-Service Fabric**.

# Secure a standalone cluster on Windows by using Windows security

8/24/2017 • 5 min to read • [Edit Online](#)

To prevent unauthorized access to a Service Fabric cluster, you must secure the cluster. Security is especially important when the cluster runs production workloads. This article describes how to configure node-to-node and client-to-node security by using Windows security in the *ClusterConfig.JSON* file. The process corresponds to the configure security step of [Create a standalone cluster running on Windows](#). For more information about how Service Fabric uses Windows security, see [Cluster security scenarios](#).

## NOTE

You should consider the selection of node-to-node security carefully because there is no cluster upgrade from one security choice to another. To change the security selection, you have to rebuild the full cluster.

## Configure Windows security using gMSA

The sample *ClusterConfig.gMSA.Windows.MultiMachine.JSON* configuration file downloaded with the [Microsoft.Azure.ServiceFabric.WindowsServer.zip](#) standalone cluster package contains a template for configuring Windows security using [Group Managed Service Account \(gMSA\)](#):

```
"security": {  
    "WindowsIdentities": {  
        "ClustergMSAIdentity": "accountname@fqdn"  
        "ClusterSPN": "fqdn"  
        "ClientIdentities": [  
            {  
                "Identity": "domain\\username",  
                "IsAdmin": true  
            }  
        ]  
    }  
}
```

CONFIGURATION SETTING	DESCRIPTION
WindowsIdentities	Contains the cluster and client identities.
ClustergMSAIdentity	Configures node-to-node security. A group managed service account.
ClusterSPN	Fully qualified domain SPN for gMSA account
ClientIdentities	Configures client-to-node security. An array of client user accounts.
Identity	The client identity, a domain user.
isAdmin	True specifies that the domain user has administrator client access, false for user client access.

[Node to node security](#) is configured by setting **ClustergMSAIdentity** when service fabric needs to run under gMSA. In order to build trust relationships between nodes, they must be made aware of each other. This can be accomplished in two different ways: Specify the Group Managed Service Account that includes all nodes in the cluster or Specify the domain machine group that includes all nodes in the cluster. We strongly recommend using the [Group Managed Service Account \(gMSA\)](#) approach, particularly for larger clusters (more than 10 nodes) or for clusters that are likely to grow or shrink.

This approach does not require the creation of a domain group for which cluster administrators have been granted access rights to add and remove members. These accounts are also useful for automatic password management. For more information, see [Getting Started with Group Managed Service Accounts](#).

[Client to node security](#) is configured using **ClientIdentities**. In order to establish trust between a client and the cluster, you must configure the cluster to know which client identities that it can trust. This can be done in two different ways: Specify the domain group users that can connect or specify the domain node users that can connect. Service Fabric supports two different access control types for clients that are connected to a Service Fabric cluster: administrator and user. Access control provides the ability for the cluster administrator to limit access to certain types of cluster operations for different groups of users, making the cluster more secure. Administrators have full access to management capabilities (including read/write capabilities). Users, by default, have only read access to management capabilities (for example, query capabilities), and the ability to resolve applications and services. For more information on access controls, see [Role based access control for Service Fabric clients](#).

The following example **security** section configures Windows security using gMSA and specifies that the machines in *ServiceFabric.clusterA.contoso.com* gMSA are part of the cluster and that *CONTOSO\usera* has admin client access:

```
"security": {  
    "WindowsIdentities": {  
        "ClustergMSAIdentity" : "ServiceFabric.clusterA.contoso.com",  
        "ClusterSPN" : "clusterA.contoso.com",  
        "ClientIdentities": [{  
            "Identity": "CONTOSO\\usera",  
            "IsAdmin": true  
        }]  
    }  
}
```

## Configure Windows security using a machine group

The sample *ClusterConfig.Windows.MultiMachine.JSON* configuration file downloaded with the [Microsoft.Azure.ServiceFabric.WindowsServer.zip](#) standalone cluster package contains a template for configuring Windows security. Windows security is configured in the **Properties** section:

```
"security": {  
    "ClusterCredentialType": "Windows",  
    "ServerCredentialType": "Windows",  
    "WindowsIdentities": {  
        "ClusterIdentity" : "[domain\\machinegroup]",  
        "ClientIdentities": [{  
            "Identity": "[domain\\username]",  
            "IsAdmin": true  
        }]  
    }  
}
```

CONFIGURATION SETTING	DESCRIPTION
ClusterCredentialType	<b>ClusterCredentialType</b> is set to <i>Windows</i> if ClusterIdentity specifies an Active Directory Machine Group Name.
ServerCredentialType	Set to <i>Windows</i> to enable Windows security for clients.  This indicates that the clients of the cluster and the cluster itself are running within an Active Directory domain.
WindowsIdentities	Contains the cluster and client identities.
ClusterIdentity	Use a machine group name, domain\machinegroup, to configure node-to-node security.
ClientIdentities	Configures client-to-node security. An array of client user accounts.
Identity	Add the domain user, domain\username, for the client identity.
IsAdmin	Set to true to specify that the domain user has administrator client access or false for user client access.

[Node to node security](#) is configured by setting using **ClusterIdentity** if you want to use a machine group within an Active Directory Domain. For more information, see [Create a Machine Group in Active Directory](#).

[Client-to-node security](#) is configured by using **ClientIdentities**. To establish trust between a client and the cluster, you must configure the cluster to know the client identities that the cluster can trust. You can establish trust in two different ways:

- Specify the domain group users that can connect.
- Specify the domain node users that can connect.

Service Fabric supports two different access control types for clients that are connected to a Service Fabric cluster: administrator and user. Access control enables the cluster administrator to limit access to certain types of cluster operations for different groups of users, which makes the cluster more secure. Administrators have full access to management capabilities (including read/write capabilities). Users, by default, have only read access to management capabilities (for example, query capabilities), and the ability to resolve applications and services.

The following example **security** section configures Windows security, specifies that the machines in *ServiceFabric/clusterA.contoso.com* are part of the cluster, and specifies that *CONTOSO\usera* has admin client access:

```

"security": {
    "ClusterCredentialType": "Windows",
    "ServerCredentialType": "Windows",
    "WindowsIdentities": {
        "ClusterIdentity" : "ServiceFabric/clusterA.contoso.com",
        "ClientIdentities": [
            {
                "Identity": "CONTOSO\\usera",
                "IsAdmin": true
            }
        ]
    },
},

```

**NOTE**

Service Fabric should not be deployed on a domain controller. Make sure that `ClusterConfig.json` does not include the IP address of the domain controller when using a machine group or group Managed Service Account (gMSA).

## Next steps

After configuring Windows security in the `ClusterConfig.JSON` file, resume the cluster creation process in [Create a standalone cluster running on Windows](#).

For more information about how node-to-node security, client-to-node security, and role-based access control, see [Cluster security scenarios](#).

See [Connect to a secure cluster](#) for examples of connecting by using PowerShell or FabricClient.

# Contents of Service Fabric Standalone package for Windows Server

8/11/2017 • 2 min to read • [Edit Online](#)

In the [downloaded](#) Service Fabric Standalone package, you will find the following files:

FILE NAME	SHORT DESCRIPTION
CreateServiceFabricCluster.ps1	A PowerShell script that creates the cluster using the settings in ClusterConfig.json.
RemoveServiceFabricCluster.ps1	A PowerShell script that removes a cluster using the settings in ClusterConfig.json.
AddNode.ps1	A PowerShell script for adding a node to an existing deployed cluster on the current machine.
RemoveNode.ps1	A PowerShell script for removing a node from an existing deployed cluster from the current machine.
CleanFabric.ps1	A PowerShell script for cleaning a standalone Service Fabric installation off the current machine. Previous MSI installations should be removed using their own associated uninstallers.
TestConfiguration.ps1	A PowerShell script for analyzing the infrastructure as specified in the Cluster.json.
DownloadServiceFabricRuntimePackage.ps1	A PowerShell script used for downloading the latest runtime package out of band, for scenarios where the deploying machine is not connected to the internet.
DeploymentComponentsAutoextractor.exe	Self-extracting archive containing Deployment Components used by the Standalone package scripts.
EULA_ENU.txt	The license terms for the use of Microsoft Azure Service Fabric standalone Windows Server package. You can <a href="#">download a copy of the EULA</a> now.
Readme.txt	A link to the release notes and basic installation instructions. It is a subset of the instructions in this document.
ThirdPartyNotice.rtf	Notice of third-party software that is in the package.
Tools\Microsoft.Azure.ServiceFabric.WindowsServer.SupportPackage.zip	StandaloneLogCollector.exe which is run on demand to collect and upload trace logs to Microsoft for support purpose.
Tools\ServiceFabricUpdateService.zip	A tool used to enable auto code upgrade for clusters which don't have internet access. More details can be found <a href="#">here</a>

## Templates

FILE NAME	SHORT DESCRIPTION
ClusterConfig.Unsecure.DevCluster.json	A cluster configuration sample file that contains the settings for an unsecured, three-node, single-machine (or virtual machine) development cluster, including the information for each node in the cluster.
ClusterConfig.Unsecure.MultiMachine.json	A cluster configuration sample file that contains the settings for an unsecured, multi-machine (or virtual machine) cluster, including the information for each machine in the cluster.
ClusterConfig.Windows.DevCluster.json	A cluster configuration sample file that contains all the settings for a secure, three-node, single-machine (or virtual machine) development cluster, including the information for each node that is in the cluster. The cluster is secured by using <a href="#">Windows identities</a> .
ClusterConfig.Windows.MultiMachine.json	A cluster configuration sample file that contains all the settings for a secure, multi-machine (or virtual machine) cluster using Windows security, including the information for each machine that is in the secure cluster. The cluster is secured by using <a href="#">Windows identities</a> .
ClusterConfig.x509.DevCluster.json	A cluster configuration sample file that contains all the settings for a secure, three-node, single-machine (or virtual machine) development cluster, including the information for each node in the cluster. The cluster is secured using x509 certificates.
ClusterConfig.x509.MultiMachine.json	A cluster configuration sample file that contains all the settings for the secure, multi-machine (or virtual machine) cluster, including the information for each node in the secure cluster. The cluster is secured using x509 certificates.
ClusterConfig.gMSA.Windows.MultiMachine.json	A cluster configuration sample file that contains all the settings for the secure, multi-machine (or virtual machine) cluster, including the information for each node in the secure cluster. The cluster is secured using <a href="#">Group Managed Service Accounts</a> .

## Cluster Configuration Samples

Latest versions of cluster configuration templates can be found at the GitHub page: [Standalone Cluster Configuration Samples](#).

## Independent Runtime Package

The latest runtime package is downloaded automatically during cluster deployment from [Download Link - Service Fabric Runtime - Windows Server](#).

## Related

- [Create a standalone Azure Service Fabric cluster](#)
- [Service Fabric cluster security scenarios](#)

# Add or remove nodes to a standalone Service Fabric cluster running on Windows Server

8/22/2017 • 3 min to read • [Edit Online](#)

After you have [created your standalone Service Fabric cluster on Windows Server machines](#), your business needs may change and you might need to add or remove nodes to your cluster. This article provides detailed steps to achieve this. Please note that add/remove node functionality is not supported in local development clusters.

## Add nodes to your cluster

1. Prepare the VM/machine you want to add to your cluster by following the steps mentioned in the [Prepare the machines to meet the prerequisites for cluster deployment](#) section
2. Identify which fault domain and upgrade domain you are going to add this VM/machine to
3. Remote desktop (RDP) into the VM/machine that you want to add to the cluster
4. Copy or [download the standalone package for Service Fabric for Windows Server](#) to the VM/machine and unzip the package
5. Run Powershell with elevated privileges, and navigate to the location of the unzipped package
6. Run the *AddNode.ps1* script with the parameters describing the new node to add. The example below adds a new node called VM5, with type NodeType0 and IP address 182.17.34.52, into UD1 and fd:/dc1/r0. The *ExistingClusterConnectionEndPoint* is a connection endpoint for a node already in the existing cluster, which can be the IP address of *any* node in the cluster.

```
.\\AddNode.ps1 -NodeName VM5 -NodeType NodeType0 -NodeIPAddrressorFQDN 182.17.34.52 -  
ExistingClientConnectionEndpoint 182.17.34.50:19000 -UpgradeDomain UD1 -FaultDomain fd:/dc1/r0 -  
AcceptEULA
```

Once the script finishes running, you can check if the new node has been added by running the [Get-ServiceFabricNode](#) cmdlet.

7. To ensure consistency across different nodes in the cluster, you must initiate a configuration upgrade. Run [Get-ServiceFabricClusterConfiguration](#) to get the latest configuration file and add the newly added node to "Nodes" section. It is also recommended to always have the latest cluster configuration available in the case that you need to redeploy a cluster with the same configuration.

```
{  
    "nodeName": "vm5",  
    "iPAddress": "182.17.34.52",  
    "nodeTypeRef": "NodeType0",  
    "faultDomain": "fd:/dc1/r0",  
    "upgradeDomain": "UD1"  
}
```

8. Run [Start-ServiceFabricClusterConfigurationUpgrade](#) to begin the upgrade.

```
Start-ServiceFabricClusterConfigurationUpgrade -ClusterConfigPath <Path to Configuration File>
```

You can monitor the progress of the upgrade on Service Fabric Explorer. Alternatively, you can run [Get-ServiceFabricClusterUpgrade](#)

## Add nodes to clusters configured with Windows Security using gMSA

For clusters configured with Group Managed Service Account(gMSA)

(<https://technet.microsoft.com/library/hh831782.aspx>), a new node can be added using a configuration upgrade:

1. Run [Get-ServiceFabricClusterConfiguration](#) on any of the existing nodes to get the latest configuration file and add details about the new node you want to add in the "Nodes" section. Make sure the new node is part of the same group managed account. This account should be an Administrator on all machines.

```
{  
    "nodeName": "vm5",  
    "iPAddress": "182.17.34.52",  
    "nodeTypeRef": "NodeType0",  
    "faultDomain": "fd:/dc1/r0",  
    "upgradeDomain": "UD1"  
}
```

2. Run [Start-ServiceFabricClusterConfigurationUpgrade](#) to begin the upgrade.

```
Start-ServiceFabricClusterConfigurationUpgrade -ClusterConfigPath <Path to Configuration File>
```

You can monitor the progress of the upgrade on Service Fabric Explorer. Alternatively, you can run [Get-ServiceFabricClusterUpgrade](#)

## Add node types to your cluster

In order to add a new node type, modify your configuration to include the new node type in "NodeTypes" section under "Properties" and begin a configuration upgrade using [Start-ServiceFabricClusterConfigurationUpgrade](#). Once the upgrade completes, you can add new nodes to your cluster with this node type.

## Remove nodes from your cluster

A node can be removed from a cluster using a configuration upgrade, in the following manner:

1. Run [Get-ServiceFabricClusterConfiguration](#) to get the latest configuration file and *remove* the node from "Nodes" section. Add the "NodesToBeRemoved" parameter to "Setup" section inside "FabricSettings" section. The "value" should be a comma separated list of node names of nodes that need to be removed.

```
"fabricSettings": [  
    {  
        "name": "Setup",  
        "parameters": [  
            {  
                "name": "FabricDataRoot",  
                "value": "C:\\\\ProgramData\\\\SF"  
            },  
            {  
                "name": "FabricLogRoot",  
                "value": "C:\\\\ProgramData\\\\SF\\\\Log"  
            },  
            {  
                "name": "NodesToBeRemoved",  
                "value": "vm0, vm1"  
            }  
        ]  
    }  
]
```

2. Run [Start-ServiceFabricClusterConfigurationUpgrade](#) to begin the upgrade.

```
Start-ServiceFabricClusterConfigurationUpgrade -ClusterConfigPath <Path to Configuration File>
```

You can monitor the progress of the upgrade on Service Fabric Explorer. Alternatively, you can run [Get-ServiceFabricClusterUpgrade](#)

#### **NOTE**

Removal of nodes may initiate multiple upgrades. Some nodes are marked with `IsSeedNode="true"` tag and can be identified by querying the cluster manifest using [Get-ServiceFabricClusterManifest](#). Removal of such nodes may take longer than others since the seed nodes will have to be moved around in such scenarios. The cluster must maintain a minimum of 3 primary node type nodes.

### **Remove node types from your cluster**

Before removing a node type, please double check if there are any nodes referencing the node type. Remove these nodes before removing the corresponding node type. Once all corresponding nodes are removed, you can remove the NodeType from the cluster configuration and begin a configuration upgrade using [Start-ServiceFabricClusterConfigurationUpgrade](#).

### **Replace primary nodes of your cluster**

The replacement of primary nodes should be performed one node after another, instead of removing and then adding in batches.

## **Next steps**

- [Configuration settings for standalone Windows cluster](#)
- [Secure a standalone cluster on Windows using X509 certificates](#)
- [Create a standalone Service Fabric cluster with Azure VMs running Windows](#)

# Role-based access control for Service Fabric clients

8/15/2017 • 2 min to read • [Edit Online](#)

Azure Service Fabric supports two different access control types for clients that are connected to a Service Fabric cluster: administrator and user. Access control allows the cluster administrator to limit access to certain cluster operations for different groups of users, making the cluster more secure.

**Administrators** have full access to management capabilities (including read/write capabilities). By default, **users** only have read access to management capabilities (for example, query capabilities), and the ability to resolve applications and services.

You specify the two client roles (administrator and client) at the time of cluster creation by providing separate certificates for each. See [Service Fabric cluster security](#) for details on setting up a secure Service Fabric cluster.

## Default access control settings

The administrator access control type has full access to all the FabricClient APIs. It can perform any reads and writes on the Service Fabric cluster, including the following operations:

### Application and service operations

- **CreateService**: service creation
- **CreateServiceFromTemplate**: service creation from template
- **UpdateService**: service updates
- **DeleteService**: service deletion
- **ProvisionApplicationType**: application type provisioning
- **CreateApplication**: application creation
- **DeleteApplication**: application deletion
- **UpgradeApplication**: starting or interrupting application upgrades
- **UnprovisionApplicationType**: application type unprovisioning
- **MoveNextUpgradeDomain**: resuming application upgrades with an explicit update domain
- **ReportUpgradeHealth**: resuming application upgrades with the current upgrade progress
- **ReportHealth**: reporting health
- **PredeployPackageToNode**: predeployment API
- **CodePackageControl**: restarting code packages
- **RecoverPartition**: recovering a partition
- **RecoverPartitions**: recovering partitions
- **RecoverServicePartitions**: recovering service partitions
- **RecoverSystemPartitions**: recovering system service partitions

### Cluster operations

- **ProvisionFabric**: MSI and/or cluster manifest provisioning
- **UpgradeFabric**: starting cluster upgrades
- **UnprovisionFabric**: MSI and/or cluster manifest unprovisioning
- **MoveNextFabricUpgradeDomain**: resuming cluster upgrades with an explicit update domain
- **ReportFabricUpgradeHealth**: resuming cluster upgrades with the current upgrade progress
- **StartInfrastructureTask**: starting infrastructure tasks
- **FinishInfrastructureTask**: finishing infrastructure tasks

- **InvokeInfrastructureCommand**: infrastructure task management commands
- **ActivateNode**: activating a node
- **DeactivateNode**: deactivating a node
- **DeactivateNodesBatch**: deactivating multiple nodes
- **RemoveNodeDeactivations**: reverting deactivation on multiple nodes
- **GetNodeDeactivationStatus**: checking deactivation status
- **NodeStateRemoved**: reporting node state removed
- **ReportFault**: reporting fault
- **FileContent**: image store client file transfer (external to cluster)
- **FileDownload**: image store client file download initiation (external to cluster)
- **InternalList**: image store client file list operation (internal)
- **Delete**: image store client delete operation
- **Upload**: image store client upload operation
- **NodeControl**: starting, stopping, and restarting nodes
- **MoveReplicaControl**: moving replicas from one node to another

### Miscellaneous operations

- **Ping**: client pings
- **Query**: all queries allowed
- **NameExists**: naming URI existence checks

The user access control type is, by default, limited to the following operations:

- **EnumerateSubnames**: naming URI enumeration
- **EnumerateProperties**: naming property enumeration
- **PropertyReadBatch**: naming property read operations
- **GetServiceDescription**: long-poll service notifications and reading service descriptions
- **ResolveService**: complaint-based service resolution
- **ResolveNameOwner**: resolving naming URI owner
- **ResolvePartition**: resolving system services
- **ServiceNotifications**: event-based service notifications
- **GetUpgradeStatus**: polling application upgrade status
- **GetFabricUpgradeStatus**: polling cluster upgrade status
- **InvokeInfrastructureQuery**: querying infrastructure tasks
- **List**: image store client file list operation
- **ResetPartitionLoad**: resetting load for a failover unit
- **ToggleVerboseServicePlacementHealthReporting**: toggling verbose service placement health reporting

The admin access control also has access to the preceding operations.

## Changing default settings for client roles

In the cluster manifest file, you can provide admin capabilities to the client if needed. You can change the defaults by going to the **Fabric Settings** option during [cluster creation](#), and providing the preceding settings in the **name**, **admin**, **user**, and **value** fields.

## Next steps

[Service Fabric cluster security](#)

[Service Fabric cluster creation](#)

# Configuration settings for a standalone Windows cluster

10/30/2017 • 8 min to read • [Edit Online](#)

This article describes how to configure a standalone Azure Service Fabric cluster by using the ClusterConfig.JSON file. You can use this file to specify information such as the Service Fabric nodes and their IP addresses and the different types of nodes on the cluster. You can also specify security configurations as well as the network topology in terms of fault/upgrade domains for your standalone cluster.

When you [download the standalone Service Fabric package](#), a few samples of the ClusterConfig.JSON file are downloaded to your work machine. The samples that have DevCluster in their names help you create a cluster with all three nodes on the same machine, like logical nodes. Out of these nodes, at least one must be marked as a primary node. This cluster is useful for a development or test environment. It is not supported as a production cluster. The samples that have MultiMachine in their names help you create a production-quality cluster, with each node on a separate machine. The number of primary nodes for these clusters is based on the [reliability level](#). In release 5.7 API Version 05-2017, we removed the reliability-level property. Instead, our code calculates the most optimized reliability level for your cluster. Do not use this property in 5.7 and later code versions.

- ClusterConfig.Unsecure.DevCluster.JSON and ClusterConfig.Unsecure.MultiMachine.JSON show how to create an unsecured test or production cluster, respectively.
- ClusterConfig.Windows.DevCluster.JSON and ClusterConfig.Windows.MultiMachine.JSON show how to create test or production clusters that are secured by using [Windows security](#).
- ClusterConfig.X509.DevCluster.JSON and ClusterConfig.X509.MultiMachine.JSON show how to create test or production clusters that are secured by using [X509 certificate-based security](#).

Now let's examine the various sections of a ClusterConfig.JSON file.

## General cluster configurations

General cluster configurations cover the broad cluster-specific configurations, as shown in the following JSON snippet:

```
"name": "SampleCluster",
"clusterConfigurationVersion": "1.0.0",
"apiVersion": "01-2017",
```

You can give any friendly name to your Service Fabric cluster by assigning it to the name variable. The clusterConfigurationVersion is the version number of your cluster. Increase it every time you upgrade your Service Fabric cluster. Leave apiVersion set to the default value.

```
<a id="clusternodes"></a>
```

## Nodes on the cluster

You can configure the nodes on your Service Fabric cluster by using the nodes section, as the following snippet shows:

```

"nodes": [
    {
        "nodeName": "vm0",
        "iPAddress": "localhost",
        "nodeTypeRef": "NodeType0",
        "faultDomain": "fd:/dc1/r0",
        "upgradeDomain": "UD0"
    },
    {
        "nodeName": "vm1",
        "iPAddress": "localhost",
        "nodeTypeRef": "NodeType1",
        "faultDomain": "fd:/dc1/r1",
        "upgradeDomain": "UD1"
    },
    {
        "nodeName": "vm2",
        "iPAddress": "localhost",
        "nodeTypeRef": "NodeType2",
        "faultDomain": "fd:/dc1/r2",
        "upgradeDomain": "UD2"
    }
],

```

A Service Fabric cluster must contain at least three nodes. You can add more nodes to this section according to your setup. The following table explains the configuration settings for each node:

NODE CONFIGURATION	DESCRIPTION
nodeName	You can give any friendly name to the node.
iPAddress	Find out the IP address of your node by opening a command window and typing <code>ipconfig</code> . Note the IPV4 address, and assign it to the ipAddress variable.
nodeTypeRef	Each node can be assigned a different node type. The <a href="#">node types</a> are defined in the following section.
faultDomain	Fault domains enable cluster administrators to define the physical nodes that might fail at the same time due to shared physical dependencies.
upgradeDomain	Upgrade domains describe sets of nodes that are shut down for Service Fabric upgrades at about the same time. You can choose which nodes to assign to which upgrade domains, because they aren't limited by any physical requirements.

## Cluster properties

The properties section in the ClusterConfig.JSON is used to configure the cluster as shown:

```

<a id="reliability"></a>

```

### Reliability

The concept of reliabilityLevel defines the number of replicas or instances of the Service Fabric system services that can run on the primary nodes of the cluster. It determines the reliability of these services and hence the cluster. The value is calculated by the system at cluster creation and upgrade time.

### Diagnostics

In the diagnosticsStore section, you can configure parameters to enable diagnostics and troubleshooting node or cluster failures, as shown in the following snippet:

```
"diagnosticsStore": {
    "metadata": "Please replace the diagnostics store with an actual file share accessible from all cluster machines.",
    "dataDeletionAgeInDays": "7",
    "storeType": "FileShare",
    "IsEncrypted": "false",
    "connectionstring": "c:\\ProgramData\\SF\\DiagnosticsStore"
}
```

The metadata is a description of your cluster diagnostics and can be set according to your setup. These variables help in collecting ETW trace logs and crash dumps as well as performance counters. For more information on ETW trace logs, see [Tracelog](#) and [ETW tracing](#). All logs, including [crash dumps](#) and [performance counters](#), can be directed to the connectionString folder on your machine. You also can use AzureStorage to store diagnostics. See the following sample snippet:

```
"diagnosticsStore": {
    "metadata": "Please replace the diagnostics store with an actual file share accessible from all cluster machines.",
    "dataDeletionAgeInDays": "7",
    "storeType": "AzureStorage",
    "IsEncrypted": "false",
    "connectionstring": "xstore:DefaultEndpointsProtocol=https;AccountName=[AzureAccountName];AccountKey=[AzureAccountKey]"
}
```

## Security

The security section is necessary for a secure standalone Service Fabric cluster. The following snippet shows a part of this section:

```
"security": {
    "metadata": "This cluster is secured using X509 certificates.",
    "ClusterCredentialType": "X509",
    "ServerCredentialType": "X509",
    . . .
}
```

The metadata is a description of your secure cluster and can be set according to your setup. The ClusterCredentialType and ServerCredentialType determine the type of security that the cluster and the nodes implement. They can be set to either *X509* for a certificate-based security or *Windows* for Azure Active Directory-based security. The rest of the security section is based on the type of security. For information on how to fill out the rest of the security section, see [Certificates-based security in a standalone cluster](#) or [Windows security in a standalone cluster](#).

```
<a id="nodetypes"></a>
```

## Node types

The nodeTypes section describes the type of nodes that your cluster has. At least one node type must be specified for a cluster, as shown in the following snippet:

```

"nodeTypes": [
    {
        "name": "NodeType0",
        "clientConnectionEndpointPort": "19000",
        "clusterConnectionEndpointPort": "19001",
        "leaseDriverEndpointPort": "19002",
        "serviceConnectionEndpointPort": "19003",
        "httpGatewayEndpointPort": "19080",
        "reverseProxyEndpointPort": "19081",
        "applicationPorts": {
            "startPort": "20575",
            "endPort": "20605"
        },
        "ephemeralPorts": {
            "startPort": "20606",
            "endPort": "20861"
        },
        "isPrimary": true
    }
]

```

The name is the friendly name for this particular node type. To create a node of this node type, assign its friendly name to the `nodeTypeRef` variable for that node, as [previously mentioned](#). For each node type, define the connection endpoints that are used. You can choose any port number for these connection endpoints, as long as they don't conflict with any other endpoints in this cluster. In a multinode cluster, there are one or more primary nodes (that is, `isPrimary` is set to `true`), depending on the `reliabilityLevel`. To learn more about primary and nonprimary node types, see [Service Fabric cluster capacity planning considerations](#) for information on `nodeTypes` and `reliabilityLevel`.

#### **Endpoints used to configure the node types**

- `clientConnectionEndpointPort` is the port used by the client to connect to the cluster when client APIs are used.
- `clusterConnectionEndpointPort` is the port where the nodes communicate with each other.
- `leaseDriverEndpointPort` is the port used by the cluster lease driver to find out if the nodes are still active.
- `serviceConnectionEndpointPort` is the port used by the applications and services deployed on a node to communicate with the Service Fabric client on that particular node.
- `httpGatewayEndpointPort` is the port used by Service Fabric Explorer to connect to the cluster.
- `ephemeralPorts` override the [dynamic ports used by the OS](#). Service Fabric uses a part of these ports as application ports, and the remaining are available for the OS. It also maps this range to the existing range present in the OS, so for all purposes, you can use the ranges given in the sample JSON files. Make sure that the difference between the start and the end ports is at least 255. You might run into conflicts if this difference is too low, because this range is shared with the OS. To see the configured dynamic port range, run  
`netsh int ipv4 show dynamicport tcp`.
- `applicationPorts` are the ports that are used by the Service Fabric applications. The application port range should be large enough to cover the endpoint requirement of your applications. This range should be exclusive from the dynamic port range on the machine, that is, the `ephemeralPorts` range as set in the configuration. Service Fabric uses these ports whenever new ports are required and takes care of opening the firewall for these ports.
- `reverseProxyEndpointPort` is an optional reverse proxy endpoint. For more information, see [Service Fabric reverse proxy](#).

#### **Log settings**

In the `fabricSettings` section, you can set the root directories for the Service Fabric data and logs. You can customize these directories only during the initial cluster creation. See the following sample snippet of this section:

```
"fabricSettings": [{"name": "Setup", "parameters": [{"name": "FabricDataRoot", "value": "C:\\ProgramData\\SF"}, {"name": "FabricLogRoot", "value": "C:\\ProgramData\\SF\\Log"}]}]
```

We recommend that you use a non-OS drive as the FabricDataRoot and FabricLogRoot. It provides more reliability in avoiding situations when the OS stops responding. If you customize only the data root, the log root is placed one level below the data root.

### Stateful Reliable Services settings

In the KtlLogger section, you can set the global configuration settings for Reliable Services. For more information on these settings, see [Configure Stateful Reliable Services](#). The following example shows how to change the shared transaction log that gets created to back any reliable collections for stateful services:

```
"fabricSettings": [{"name": "KtlLogger", "parameters": [{"name": "SharedLogSizeInMB", "value": "4096"}]}]
```

### Add-on features

To configure add-on features, configure the apiVersion as 04-2017 or higher, and configure the addonFeatures as shown here:

```
"apiVersion": "04-2017",  
"properties": {  
    "addOnFeatures": [  
        "DnsService",  
        "RepairManager"  
    ]  
}
```

### Container support

To enable container support for both Windows Server containers and Hyper-V containers for standalone clusters, the DnsService add-on feature must be enabled.

## Next steps

After you have a complete ClusterConfig.JSON file configured according to your standalone cluster setup, you can deploy your cluster. Follow the steps in [Create a standalone Service Fabric cluster](#). Then proceed to [Visualize your cluster with Service Fabric Explorer](#) and follow the steps.

# Upgrade your standalone Azure Service Fabric cluster on Windows Server

10/30/2017 • 6 min to read • [Edit Online](#)

For any modern system, the ability to upgrade is key to the long-term success of your product. An Azure Service Fabric cluster is a resource that you own. This article describes how you can make sure that the cluster always runs supported versions of Service Fabric code and configurations.

## Control the Service Fabric version that runs on your cluster

To set your cluster to download updates of Service Fabric when Microsoft releases a new version, set the `fabricClusterAutoupgradeEnabled` cluster configuration to *true*. To select a supported version of Service Fabric that you want your cluster to be on, set the `fabricClusterAutoupgradeEnabled` cluster configuration to *false*.

### NOTE

Make sure that your cluster always runs a supported Service Fabric version. When Microsoft announces the release of a new version of Service Fabric, the previous version is marked for end of support after a minimum of 60 days from the date of the announcement. New releases are announced [on the Service Fabric team blog](#). The new release is available to choose at that point.

You can upgrade your cluster to the new version only if you're using a production-style node configuration, where each Service Fabric node is allocated on a separate physical or virtual machine. If you have a development cluster, where more than one Service Fabric node is on a single physical or virtual machine, you must re-create the cluster with the new version.

Two distinct workflows can upgrade your cluster to the latest version or a supported Service Fabric version. One workflow is for clusters that have connectivity to download the latest version automatically. The other workflow is for clusters that don't have connectivity to download the latest Service Fabric version.

### Upgrade clusters that have connectivity to download the latest code and configuration

Use these steps to upgrade your cluster to a supported version if your cluster nodes have internet connectivity to the [Microsoft Download Center](#).

For clusters that have connectivity to the [Microsoft Download Center](#), Microsoft periodically checks for the availability of new Service Fabric versions.

When a new Service Fabric version is available, the package is downloaded locally to the cluster and provisioned for upgrade. Additionally, to inform the customer of this new version, the system shows an explicit cluster health warning that's similar to the following:

"The current cluster version [version #] support ends [date]."

After the cluster is running the latest version, the warning goes away.

### Cluster upgrade workflow

When you see the cluster health warning, do the following:

1. Connect to the cluster from any machine that has administrator access to all the machines that are listed as nodes in the cluster. The machine that this script is run on doesn't have to be part of the cluster.

```
##### connect to the secure cluster using certs
$ClusterName= "mysecurecluster.something.com:19000"
$CertThumbprint= "70EF5E22ADB649799DA3C8B6A6BF7FG2D630F8F3"
Connect-serviceFabricCluster -ConnectionEndpoint $ClusterName -KeepAliveIntervalInSec 10 ` 
    -X509Credential ` 
    -ServerCertThumbprint $CertThumbprint ` 
    -FindType FindByThumbprint ` 
    -FindValue $CertThumbprint ` 
    -StoreLocation CurrentUser ` 
    -StoreName My
```

- Get the list of Service Fabric versions that you can upgrade to.

```
##### Get the list of available Service Fabric versions
Get-ServiceFabricRegisteredClusterCodeVersion
```

You should get an output similar to this:

```
PS C:\> Get-ServiceFabricRegisteredClusterCodeVersion
CodeVersion
-----
5.3.204.9494
5.3.301.9590
```

- Start a cluster upgrade to an available version by using the [Start-ServiceFabricClusterUpgrade](#) Windows PowerShell command.

```
Start-ServiceFabricClusterUpgrade -Code -CodePackageVersion <codeversion#> -Monitored -FailureAction Rollback

##### Here is a filled-out example

Start-ServiceFabricClusterUpgrade -Code -CodePackageVersion 5.3.301.9590 -Monitored -FailureAction Rollback
```

To monitor the progress of the upgrade, you can use Service Fabric Explorer or run the following PowerShell command:

```
Get-ServiceFabricClusterUpgrade
```

If the cluster health policies aren't met, the upgrade is rolled back. To specify custom health policies for the `Start-ServiceFabricClusterUpgrade` command, see documentation for [Start-ServiceFabricClusterUpgrade](#).

After you fix the issues that resulted in the rollback, initiate the upgrade again by following the same steps as previously described.

### Upgrade clusters that have **no connectivity** to download the latest code and configuration

Use these steps to upgrade your cluster to a supported version if your cluster nodes don't have internet connectivity to the [Microsoft Download Center](#).

#### NOTE

If you're running a cluster that isn't connected to the internet, you have to monitor the Service Fabric team blog to learn about new releases. The system doesn't show a cluster health warning to alert you of new releases.

### Auto provisioning vs. manual provisioning

To enable automatic downloading and registration for the latest code version, set up the Service Fabric Update Service. For instructions, see Tools\ServiceFabricUpdateService.zip\Readme\_InstructionsAndHowTos.txt inside the [standalone package](#). For the manual process, follow these instructions.

Modify your cluster configuration to set the following property to *false* before you start a configuration upgrade:

```
"fabricClusterAutoupgradeEnabled": false,
```

For usage details, see the [Start-ServiceFabricClusterConfigurationUpgrade PowerShell command](#). Make sure to update 'clusterConfigurationVersion' in your JSON before you start the configuration upgrade.

```
Start-ServiceFabricClusterConfigurationUpgrade -ClusterConfigPath <Path to Configuration File>
```

### Cluster upgrade workflow

1. Run Get-ServiceFabricClusterUpgrade from one of the nodes in the cluster, and note the TargetCodeVersion.
2. Run the following from an internet-connected machine to list all upgrade-compatible versions with the current version and download the corresponding package from the associated download links:

```
##### Get list of all upgrade compatible packages  
Get-ServiceFabricRuntimeUpgradeVersion -BaseVersion <TargetCodeVersion as noted in Step 1>
```

3. Connect to the cluster from any machine that has administrator access to all the machines that are listed as nodes in the cluster. The machine that this script is run on doesn't have to be part of the cluster.

```
##### Get the list of available Service Fabric versions  
Copy-ServiceFabricClusterPackage -Code -CodePackagePath <name of the .cab file including the path to it> -ImageStoreConnectionString "fabric:ImageStore"  
  
##### Here is a filled-out example  
Copy-ServiceFabricClusterPackage -Code -CodePackagePath .\MicrosoftAzureServiceFabric.5.3.301.9590.cab  
-ImageStoreConnectionString "fabric:ImageStore"
```

4. Copy the downloaded package into the cluster image store.
5. Register the copied package.

```
##### Get the list of available Service Fabric versions  
Register-ServiceFabricClusterPackage -Code -CodePackagePath <name of the .cab file>  
  
##### Here is a filled-out example  
Register-ServiceFabricClusterPackage -Code -CodePackagePath  
MicrosoftAzureServiceFabric.5.3.301.9590.cab
```

6. Start a cluster upgrade to an available version.

```
Start-ServiceFabricClusterUpgrade -Code -CodePackageVersion <codeversion#> -Monitored -FailureAction Rollback

##### Here is a filled-out example
Start-ServiceFabricClusterUpgrade -Code -CodePackageVersion 5.3.301.9590 -Monitored -FailureAction Rollback
```

You can monitor the progress of the upgrade on Service Fabric Explorer, or you can run the following PowerShell command:

```
Get-ServiceFabricClusterUpgrade
```

If the cluster health policies aren't met, the upgrade is rolled back. To specify custom health policies for the Start-ServiceFabricClusterUpgrade command, see the documentation for [Start-ServiceFabricClusterUpgrade](#).

After you fix the issues that resulted in the rollback, initiate the upgrade again by following the same steps as previously described.

## Upgrade the cluster configuration

Before you initiate the configuration upgrade, you can test your new cluster configuration JSON by running the following PowerShell script in the standalone package:

```
TestConfiguration.ps1 -ClusterConfigFilePath <Path to the new Configuration File> -
OldClusterConfigFilePath <Path to the old Configuration File>
```

Or use this script:

```
TestConfiguration.ps1 -ClusterConfigFilePath <Path to the new Configuration File> -
OldClusterConfigFilePath <Path to the old Configuration File> -FabricRuntimePackagePath <Path to the .cab file which you want to test the configuration against>
```

Some configurations can't be upgraded, such as endpoints, cluster name, node IP, etc. The new cluster configuration JSON is tested against the old one and throws errors in the PowerShell window if there's an issue.

To upgrade the cluster configuration upgrade, run Start-ServiceFabricClusterConfigurationUpgrade. The configuration upgrade is processed upgrade domain by upgrade domain.

```
Start-ServiceFabricClusterConfigurationUpgrade -ClusterConfigPath <Path to Configuration File>
```

### Cluster certificate config upgrade

A cluster certificate is used for authentication between cluster nodes. The certificate rollover should be performed with extra caution because failure blocks the communication among cluster nodes.

Technically, four options are supported:

- Single certificate upgrade: The upgrade path is Certificate A (Primary) -> Certificate B (Primary) -> Certificate C (Primary) -> ...

- Double certificate upgrade: The upgrade path is Certificate A (Primary) -> Certificate A (Primary) and B (Secondary) -> Certificate B (Primary) -> Certificate B (Primary) and C (Secondary) -> Certificate C (Primary) ->....
- Certificate type upgrade: Thumbprint-based certificate configuration <-> CommonName-based certificate configuration. For example, Certificate Thumbprint A (Primary) and Thumbprint B (Secondary) -> Certificate CommonName C.
- Certificate issuer thumbprint upgrade: The upgrade path is Certificate CN=A,IssuerThumbprint=IT1 (Primary) -> Certificate CN=A,IssuerThumbprint=IT1,IT2 (Primary) -> Certificate CN=A,IssuerThumbprint=IT2 (Primary).

## Next steps

- Learn how to customize some [Service Fabric cluster settings](#).
- Learn how to [scale your cluster in and out](#).
- Learn about [application upgrades](#).

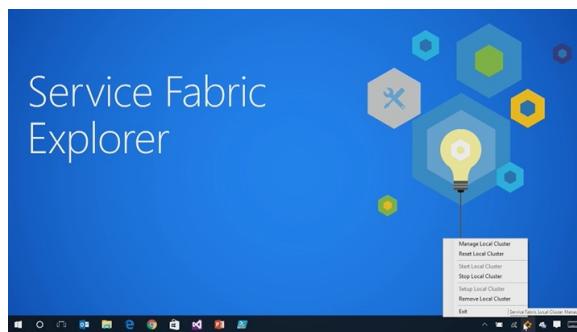
# Visualize your cluster with Service Fabric Explorer

9/28/2017 • 4 min to read • [Edit Online](#)

Service Fabric Explorer is a web-based tool for inspecting and managing applications and nodes in an Azure Service Fabric cluster. Service Fabric Explorer is hosted directly within the cluster, so it is always available, regardless of where your cluster is running.

## Video tutorial

To learn how to use Service Fabric Explorer, watch the following Microsoft Virtual Academy video:



## Connect to Service Fabric Explorer

If you have followed the instructions to [prepare your development environment](#), you can launch Service Fabric Explorer on your local cluster by navigating to <http://localhost:19080/Explorer>.

## Understand the Service Fabric Explorer layout

You can navigate through Service Fabric Explorer by using the tree on the left. At the root of the tree, the cluster dashboard provides an overview of your cluster, including a summary of application and node health.

## View the cluster's layout

Nodes in a Service Fabric cluster are placed across a two-dimensional grid of fault domains and upgrade domains. This placement ensures that your applications remain available in the presence of hardware failures and application upgrades. You can view how the current cluster is laid out by using the cluster map.

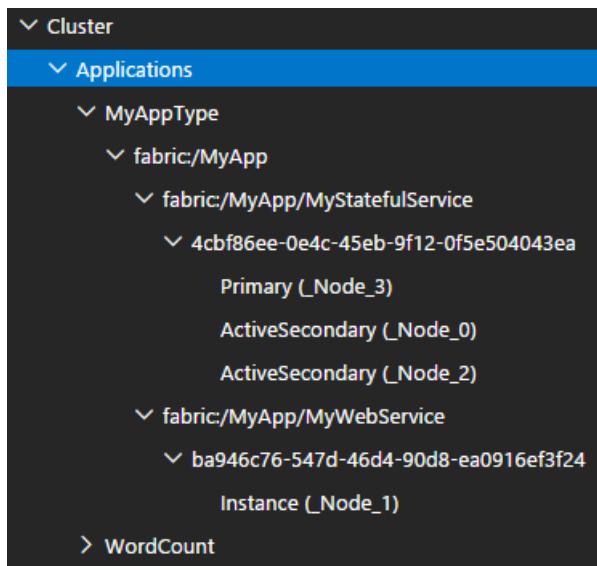
## View applications and services

The cluster contains two subtrees: one for applications and another for nodes.

You can use the application view to navigate through Service Fabric's logical hierarchy: applications, services,

partitions, and replicas.

In the example below, the application **MyApp** consists of two services, **MyStatefulService** and **WebService**. Since **MyStatefulService** is stateful, it includes a partition with one primary and two secondary replicas. By contrast, WebSvcService is stateless and contains a single instance.



At each level of the tree, the main pane shows pertinent information about the item. For example, you can see the health status and version for a particular service.

This screenshot shows the 'ESSENTIALS' tab of the Service fabric:/MyApp/MyStatefulService details page. The page header is 'Service fabric:/MyApp/MyStatefulService' and there is a 'DETAILS' and 'MANIFEST' tab. The 'ESSENTIALS' section contains the following data:

Name	Service Type
fabric:/MyApp/MyStatefulService	MyStatefulServiceType

Below this, there are sections for Health State (OK), Status (Active), Service Kind (Stateful), Service Type Version (1.0.0), Minimum Replica Size (3), and Target Replica Size (3). A blue 'ACTIONS' button is located in the top right corner.

### View the cluster's nodes

The node view shows the physical layout of the cluster. For a given node, you can inspect which applications have code deployed on that node. More specifically, you can see which replicas are currently running there.

## Actions

Service Fabric Explorer offers a quick way to invoke actions on nodes, applications, and services within your cluster.

For example, to delete an application instance, choose the application from the tree on the left, and then choose **Actions > Delete Application**.

# Application fabric:/MyApp

ACTIONS ▾

Delete Application

ESSENTIALS

DETAILS

UPGRADE

MANIFEST

**Id**  
MyApp

**Application Type**  
MyAppType

**Health State**  
■ OK

**Application Version**  
1.0.0

**Status**  
■ Ready

## TIP

You can perform the same actions by clicking the ellipsis next to each element.

The following table lists the actions available for each entity:

ENTITY	ACTION	DESCRIPTION
Application type	Unprovision type	Removes the application package from the cluster's image store. Requires all applications of that type to be removed first.
Application	Delete Application	Delete the application, including all its services and their state (if any).
Service	Delete Service	Delete the service and its state (if any).
Node	Activate	Activate the node.
Node	Deactivate (pause)	Pause the node in its current state. Services continue to run but Service Fabric does not proactively move anything onto or off it unless it is required to prevent an outage or data inconsistency. This action is typically used to enable debugging services on a specific node to ensure that they do not move during inspection.
Node	Deactivate (restart)	Safely move all in-memory services off a node and close persistent services. Typically used when the host processes or machine need to be restarted.

ENTITY	ACTION	DESCRIPTION
Node	Deactivate (remove data)	Safely close all services running on the node after building sufficient spare replicas. Typically used when a node (or at least its storage) is being permanently taken out of commission.
Node	Remove node state	Remove knowledge of a node's replicas from the cluster. Typically used when an already failed node is deemed unrecoverable.
Node	Restart	Simulate a node failure by restarting the node. More information <a href="#">here</a>

Since many actions are destructive, you may be asked to confirm your intent before the action is completed.

#### TIP

Every action that can be performed through Service Fabric Explorer can also be performed through PowerShell or a REST API, to enable automation.

You can also use Service Fabric Explorer to create application instances for a given application type and version. Choose the application type in the tree view, then click the **Create app instance** link next to the version you'd like in the right pane.

Application Type	Version	Actions
MyApp	1.0.0	Unprovision <b>Create app instance</b>

APPLICATIONS				
Name	Application Type	Version	Health State	Status
fabric:/MyApp	MyApp	1.0.0	<span style="color: green;">OK</span>	Ready

#### NOTE

Application instances created through Service Fabric Explorer cannot currently be parameterized. They are created using default parameter values.

## Connect to a remote Service Fabric cluster

If you know the cluster's endpoint and have sufficient permissions you can access Service Fabric Explorer from any browser. This is because Service Fabric Explorer is just another service that runs in the cluster.

### Discover the Service Fabric Explorer endpoint for a remote cluster

To reach Service Fabric Explorer for a given cluster, point your browser to:

<http://<your-cluster-endpoint>:19080/Explorer>

For Azure clusters, the full URL is also available in the cluster essentials pane of the Azure portal.

### Connect to a secure cluster

You can control client access to your Service Fabric cluster either with certificates or using Azure Active Directory (AAD).

If you attempt to connect to Service Fabric Explorer on a secure cluster, then depending on the cluster's configuration you'll be required to present a client certificate or log in using AAD.

## Next steps

- [Testability overview](#)
- [Managing your Service Fabric applications in Visual Studio](#)
- [Service Fabric application deployment using PowerShell](#)

# Connect to a secure cluster

9/29/2017 • 7 min to read • [Edit Online](#)

When a client connects to a Service Fabric cluster node, the client can be authenticated and secure communication established using certificate security or Azure Active Directory (AAD). This authentication ensures that only authorized users can access the cluster and deployed applications and perform management tasks. Certificate or AAD security must have been previously enabled on the cluster when the cluster was created. For more information on cluster security scenarios, see [Cluster security](#). If you are connecting to a cluster secured with certificates, [set up the client certificate](#) on the computer that connects to the cluster.

## Connect to a secure cluster using Azure Service Fabric CLI (sfctl)

There are a few different ways to connect to a secure cluster using the Service Fabric CLI (sfctl). When using a client certificate for authentication, the certificate details must match a certificate deployed to the cluster nodes. If your certificate has Certificate Authorities (CAs), you need to additionally specify the trusted CAs.

You can connect to a cluster using the `sfctl cluster select` command.

Client certificates can be specified in two different fashions, either as a cert and key pair, or as a single pem file. For password protected `pem` files, you will be prompted automatically to enter the password.

To specify the client certificate as a pem file, specify the file path in the `--pem` argument. For example:

```
sfctl cluster select --endpoint https://testsecurecluster.com:19080 --pem ./client.pem
```

Password protected pem files will prompt for password prior to running any command.

To specify a cert, key pair use the `--cert` and `--key` arguments to specify the file paths to each respective file.

```
sfctl cluster select --endpoint https://testsecurecluster.com:19080 --cert ./client.crt --key ./keyfile.key
```

Sometimes certificates used to secure test or dev clusters fail certificate validation. To bypass certificate verification, specify the `--no-verify` option. For example:

### WARNING

Do not use the `no-verify` option when connecting to production Service Fabric clusters.

```
sfctl cluster select --endpoint https://testsecurecluster.com:19080 --pem ./client.pem --no-verify
```

In addition, you can specify paths to directories of trusted CA certs, or individual certs. To specify these paths, use the `--ca` argument. For example:

```
sfctl cluster select --endpoint https://testsecurecluster.com:19080 --pem ./client.pem --ca ./trusted_ca
```

After you connect, you should be able to [run other sfctl commands](#) to interact with the cluster.

# Connect to a cluster using PowerShell

Before you perform operations on a cluster through PowerShell, first establish a connection to the cluster. The cluster connection is used for all subsequent commands in the given PowerShell session.

## Connect to an unsecure cluster

To connect to an unsecure cluster, provide the cluster endpoint address to the **Connect-ServiceFabricCluster** command:

```
Connect-ServiceFabricCluster -ConnectionEndpoint <Cluster FQDN>:19000
```

## Connect to a secure cluster using Azure Active Directory

To connect to a secure cluster that uses Azure Active Directory to authorize cluster administrator access, provide the cluster certificate thumbprint and use the *AzureActiveDirectory* flag.

```
Connect-ServiceFabricCluster -ConnectionEndpoint <Cluster FQDN>:19000 `  
-ServerCertThumbprint <Server Certificate Thumbprint> `  
-AzureActiveDirectory
```

## Connect to a secure cluster using a client certificate

Run the following PowerShell command to connect to a secure cluster that uses client certificates to authorize administrator access. Provide the cluster certificate thumbprint and the thumbprint of the client certificate that has been granted permissions for cluster management. The certificate details must match a certificate on the cluster nodes.

```
Connect-ServiceFabricCluster -ConnectionEndpoint <Cluster FQDN>:19000 `  
-KeepAliveIntervalInSec 10 `  
-X509Credential -ServerCertThumbprint <Certificate Thumbprint> `  
-FindType FindByThumbprint -FindValue <Certificate Thumbprint> `  
-StoreLocation CurrentUser -StoreName My
```

*ServerCertThumbprint* is the thumbprint of the server certificate installed on the cluster nodes. *FindValue* is the thumbprint of the admin client certificate. When the parameters are filled in, the command looks like the following example:

```
Connect-ServiceFabricCluster -ConnectionEndpoint clustername.westus.cloudapp.azure.com:19000 `  
-KeepAliveIntervalInSec 10 `  
-X509Credential -ServerCertThumbprint A8136758F4AB8962AF2BF3F27921BE1DF67F4326 `  
-FindType FindByThumbprint -FindValue 71DE04467C9ED0544D021098BCD44C71E183414E `  
-StoreLocation CurrentUser -StoreName My
```

## Connect to a secure cluster using Windows Active Directory

If your standalone cluster is deployed using AD security, connect to the cluster by appending the switch "WindowsCredential".

```
Connect-ServiceFabricCluster -ConnectionEndpoint <Cluster FQDN>:19000 `  
-WindowsCredential
```

# Connect to a cluster using the FabricClient APIs

The Service Fabric SDK provides the [FabricClient](#) class for cluster management. To use the FabricClient APIs, get the Microsoft.ServiceFabric NuGet package.

## Connect to an unsecure cluster

To connect to a remote unsecured cluster, create a FabricClient instance and provide the cluster address:

```
FabricClient fabricClient = new FabricClient("clustername.westus.cloudapp.azure.com:19000");
```

For code that is running from within a cluster, for example, in a Reliable Service, create a FabricClient *without* specifying the cluster address. FabricClient connects to the local management gateway on the node the code is currently running on, avoiding an extra network hop.

```
FabricClient fabricClient = new FabricClient();
```

## Connect to a secure cluster using a client certificate

The nodes in the cluster must have valid certificates whose common name or DNS name in SAN appears in the [RemoteCommonNames property](#) set on [FabricClient](#). Following this process enables mutual authentication between the client and the cluster nodes.

```
using System.Fabric;
using System.Security.Cryptography.X509Certificates;

string clientCertThumb = "71DE04467C9ED0544D021098BCD44C71E183414E";
string serverCertThumb = "A8136758F4AB8962AF2BF3F27921BE1DF67F4326";
string CommonName = "www.clustername.westus.azure.com";
string connection = "clustername.westus.cloudapp.azure.com:19000";

var xc = GetCredentials(clientCertThumb, serverCertThumb, CommonName);
var fc = new FabricClient(xc, connection);

try
{
    var ret = fc.ClusterManager.GetClusterManifestAsync().Result;
    Console.WriteLine(ret.ToString());
}
catch (Exception e)
{
    Console.WriteLine("Connect failed: {0}", e.Message);
}

static X509Credentials GetCredentials(string clientCertThumb, string serverCertThumb, string name)
{
    X509Credentials xc = new X509Credentials();
    xc.StoreLocation = StoreLocation.CurrentUser;
    xc.StoreName = "My";
    xc.FindType = X509FindType.FindByThumbprint;
    xc.FindValue = clientCertThumb;
    xc.RemoteCommonNames.Add(name);
    xc.RemoteCertThumbprints.Add(serverCertThumb);
    xc.ProtectionLevel = ProtectionLevel.EncryptAndSign;
    return xc;
}
```

## Connect to a secure cluster interactively using Azure Active Directory

The following example uses Azure Active Directory for client identity and server certificate for server identity.

A dialog window automatically pops up for interactive sign-in upon connecting to the cluster.

```
string serverCertThumb = "A8136758F4AB8962AF2BF3F27921BE1DF67F4326";
string connection = "clustername.westus.cloudapp.azure.com:19000";

var claimsCredentials = new ClaimsCredentials();
claimsCredentials.ServerThumbprints.Add(serverCertThumb);

var fc = new FabricClient(claimsCredentials, connection);

try
{
    var ret = fc.ClusterManager.GetClusterManifestAsync().Result;
    Console.WriteLine(ret.ToString());
}
catch (Exception e)
{
    Console.WriteLine("Connect failed: {0}", e.Message);
}
```

### Connect to a secure cluster non-interactively using Azure Active Directory

The following example relies on Microsoft.IdentityModel.Clients.ActiveDirectory, Version: 2.19.208020213.

For more information on AAD token acquisition, see [Microsoft.IdentityModel.Clients.ActiveDirectory](#).

```

string tenantId = "C15CFCEA-02C1-40DC-8466-FBD0EE0B05D2";
string clientApplicationId = "118473C2-7619-46E3-A8E4-6DA8D5F56E12";
string webApplicationId = "53E6948C-0897-4DA6-B26A-EE2A38A690B4";

string token = GetAccessToken(
    tenantId,
    webApplicationId,
    clientApplicationId,
    "urn:ietf:wg:oauth:2.0:oob");

string serverCertThumb = "A8136758F4AB8962AF2BF3F27921BE1DF67F4326";
string connection = "clustername.westus.cloudapp.azure.com:19000";

var claimsCredentials = new ClaimsCredentials();
claimsCredentials.ServerThumbprints.Add(serverCertThumb);
claimsCredentials.LocalClaims = token;

var fc = new FabricClient(claimsCredentials, connection);

try
{
    var ret = fc.ClusterManager.GetClusterManifestAsync().Result;
    Console.WriteLine(ret.ToString());
}
catch (Exception e)
{
    Console.WriteLine("Connect failed: {0}", e.Message);
}

...

static string GetAccessToken(
    string tenantId,
    string resource,
    string clientId,
    string redirectUri)
{
    string authorityFormat = @"https://login.microsoftonline.com/{0}";
    string authority = string.Format(CultureInfo.InvariantCulture, authorityFormat, tenantId);
    var authContext = new AuthenticationContext(authority);

    var authResult = authContext.AcquireToken(
        resource,
        clientId,
        new UserCredential("TestAdmin@clustenametenant.onmicrosoft.com", "TestPassword"));
    return authResult.AccessToken;
}

```

### Connect to a secure cluster without prior metadata knowledge using Azure Active Directory

The following example uses non-interactive token acquisition, but the same approach can be used to build a custom interactive token acquisition experience. The Azure Active Directory metadata needed for token acquisition is read from cluster configuration.

```

string serverCertThumb = "A8136758F4AB8962AF2BF3F27921BE1DF67F4326";
string connection = "clustername.westus.cloudapp.azure.com:19000";

var claimsCredentials = new ClaimsCredentials();
claimsCredentials.ServerThumbprints.Add(serverCertThumb);

var fc = new FabricClient(claimsCredentials, connection);

fc.ClaimsRetrieval += (o, e) =>
{
    return GetAccessToken(e.AzureActiveDirectoryMetadata);
};

try
{
    var ret = fc.ClusterManager.GetClusterManifestAsync().Result;
    Console.WriteLine(ret.ToString());
}
catch (Exception e)
{
    Console.WriteLine("Connect failed: {0}", e.Message);
}

...

static string GetAccessToken(AzureActiveDirectoryMetadata aad)
{
    var authContext = new AuthenticationContext(aad.Authority);

    var authResult = authContext.AcquireToken(
        aad.ClusterApplication,
        aad.ClientApplication,
        new UserCredential("TestAdmin@clustenametenant.onmicrosoft.com", "TestPassword"));
    return authResult.AccessToken;
}

```

## Connect to a secure cluster using Service Fabric Explorer

To reach [Service Fabric Explorer](#) for a given cluster, point your browser to:

`http://<your-cluster-endpoint>:19080/Explorer`

The full URL is also available in the cluster essentials pane of the Azure portal.

For connecting to a secure cluster on Windows or OS X using a browser, you can import the client certificate, and the browser will prompt you for the certificate to use for connecting to the cluster. On Linux machines, the certificate will have to be imported using advanced browser settings (each browser has different mechanisms) and point it to the certificate location on disk.

### Connect to a secure cluster using Azure Active Directory

To connect to a cluster that is secured with AAD, point your browser to:

`https://<your-cluster-endpoint>:19080/Explorer`

You are automatically prompted to log in with AAD.

### Connect to a secure cluster using a client certificate

To connect to a cluster that is secured with certificates, point your browser to:

`https://<your-cluster-endpoint>:19080/Explorer`

You are automatically prompted to select a client certificate.

## Set up a client certificate on the remote computer

At least two certificates should be used for securing the cluster, one for the cluster and server certificate and another for client access. We recommend that you also use additional secondary certificates and client access certificates. To secure the communication between a client and a cluster node using certificate security, you first need to obtain and install the client certificate. The certificate can be installed into the Personal (My) store of the local computer or the current user. You also need the thumbprint of the server certificate so that the client can authenticate the cluster.

Run the following PowerShell cmdlet to set up the client certificate on the computer from which you access the cluster.

```
Import-PfxCertificate -Exportable -CertStoreLocation Cert:\CurrentUser\My `  
-FilePath C:\docDemo\certs\DocDemoClusterCert.pfx `  
-Password (ConvertTo-SecureString -String test -AsPlainText -Force)
```

If it is a self-signed certificate, you need to import it to your machine's "trusted people" store before you can use this certificate to connect to a secure cluster.

```
Import-PfxCertificate -Exportable -CertStoreLocation Cert:\CurrentUser\TrustedPeople `  
-FilePath C:\docDemo\certs\DocDemoClusterCert.pfx `  
-Password (ConvertTo-SecureString -String test -AsPlainText -Force)
```

## Next steps

- [Service Fabric Cluster upgrade process and expectations from you](#)
- [Managing your Service Fabric applications in Visual Studio](#)
- [Service Fabric Health model introduction](#)
- [Application Security and RunAs](#)
- [Getting started with Service Fabric CLI](#)

# Patch the Windows operating system in your Service Fabric cluster

10/18/2017 • 15 min to read • [Edit Online](#)

The patch orchestration application is an Azure Service Fabric application that automates operating system patching on a Service Fabric cluster without downtime.

The patch orchestration app provides the following:

- **Automatic operating system update installation.** Operating system updates are automatically downloaded and installed. Cluster nodes are rebooted as needed without cluster downtime.
- **Cluster-aware patching and health integration.** While applying updates, the patch orchestration app monitors the health of the cluster nodes. Cluster nodes are upgraded one node at a time or one upgrade domain at a time. If the health of the cluster goes down due to the patching process, patching is stopped to prevent aggravating the problem.

## Internal details of the app

The patch orchestration app is composed of the following subcomponents:

- **Coordinator Service:** This stateful service is responsible for:
  - Coordinating the Windows Update job on the entire cluster.
  - Storing the result of completed Windows Update operations.
- **Node Agent Service:** This stateless service runs on all Service Fabric cluster nodes. The service is responsible for:
  - Bootstrapping the Node Agent NTService.
  - Monitoring the Node Agent NTService.
- **Node Agent NTService:** This Windows NT service runs at a higher-level privilege (SYSTEM). In contrast, the Node Agent Service and the Coordinator Service run at a lower-level privilege (NETWORK SERVICE). The service is responsible for performing the following Windows Update jobs on all the cluster nodes:
  - Disabling automatic Windows Update on the node.
  - Downloading and installing Windows Update according to the policy the user has provided.
  - Restarting the machine post Windows Update installation.
  - Uploading the results of Windows updates to the Coordinator Service.
  - Reporting health reports in case an operation has failed after exhausting all retries.

### NOTE

The patch orchestration app uses the Service Fabric repair manager system service to disable or enable the node and perform health checks. The repair task created by the patch orchestration app tracks the Windows Update progress for each node.

## Prerequisites

### Minimum supported Service Fabric runtime version

#### Azure clusters

The patch orchestration app must be run on Azure clusters that have Service Fabric runtime version v5.5 or later.

## Standalone on-premises clusters

The patch orchestration app must be run on Standalone clusters that have Service Fabric runtime version v5.6 or later.

### Enable the repair manager service (if it's not running already)

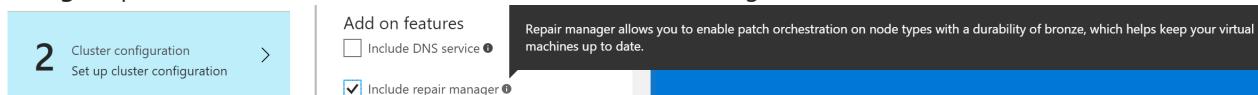
The patch orchestration app requires the repair manager system service to be enabled on the cluster.

#### Azure clusters

Azure clusters in the silver durability tier have the repair manager service enabled by default. Azure clusters in the gold durability tier might or might not have the repair manager service enabled, depending on when those clusters were created. Azure clusters in the bronze durability tier, by default, do not have the repair manager service enabled. If the service is already enabled, you can see it running in the system services section in the Service Fabric Explorer.

#### Azure portal

You can enable repair manager from Azure portal at the time of setting up of cluster. Select **Include Repair Manager** option under **Add on features** at the time of cluster configuration.



#### Azure Resource Manager template

Alternatively you can use the [Azure Resource Manager template](#) to enable the repair manager service on new and existing Service Fabric clusters. Get the template for the cluster that you want to deploy. You can either use the sample templates or create a custom Resource Manager template.

To enable the repair manager service using [Azure Resource Manager template](#):

1. First check that the `apiVersion` is set to `2017-07-01-preview` for the `Microsoft.ServiceFabric/clusters` resource, as shown in the following snippet. If it is different, then you need to update the `apiVersion` to the value `2017-07-01-preview`:

```
{  
    "apiVersion": "2017-07-01-preview",  
    "type": "Microsoft.ServiceFabric/clusters",  
    "name": "[parameters('clusterName')]",  
    "location": "[parameters('clusterLocation')]",  
    ...  
}
```

2. Now enable the repair manager service by adding the following `addonFeatures` section after the `fabricSettings` section:

```
"fabricSettings": [  
    ...  
,  
    "addonFeatures": [  
        "RepairManager"  
    ],
```

3. After you have updated your cluster template with these changes, apply them and let the upgrade finish.

You can now see the repair manager system service running in your cluster. It is called `fabric:/System/RepairManagerService` in the system services section in the Service Fabric Explorer.

## Standalone on-premises clusters

You can use the [Configuration settings for standalone Windows cluster](#) to enable the repair manager service on new and existing Service Fabric cluster.

To enable the repair manager service:

1. First check that the `apiVersion` in [General cluster configurations](#) is set to `04-2017` or higher:

```
{  
    "name": "SampleCluster",  
    "clusterConfigurationVersion": "1.0.0",  
    "apiVersion": "04-2017",  
    ...  
}
```

2. Now enable repair manager service by adding the following `addonFeatures` section after the `fabricSettings` section as shown below:

```
"fabricSettings": [  
    ...  
,  
    "addonFeatures": [  
        "RepairManager"  
    ],
```

3. Update your cluster manifest with these changes, using the updated cluster manifest [create a new cluster](#) or [upgrade the cluster configuration](#). Once the cluster is running with updated cluster manifest, you can now see the repair manager system service running in your cluster, which is called `fabric:/System/RepairManagerService`, under system services section in the Service Fabric explorer.

### Disable automatic Windows Update on all nodes

Automatic Windows updates might lead to availability loss because multiple cluster nodes can restart at the same time. The patch orchestration app, by default, tries to disable the automatic Windows Update on each cluster node. However, if the settings are managed by an administrator or group policy, we recommend setting the Windows Update policy to "Notify before Download" explicitly.

### Optional: Enable Azure Diagnostics

Clusters running Service Fabric runtime version `5.6.220.9494` and above collect patch orchestration app logs as part of Service Fabric logs. You can skip this step if your cluster is running on Service Fabric runtime version `5.6.220.9494` and above.

For clusters running Service Fabric runtime version less than `5.6.220.9494`, logs for the patch orchestration app are collected locally on each of the cluster nodes. We recommend that you configure Azure Diagnostics to upload logs from all nodes to a central location.

For information on enabling Azure Diagnostics, see [Collect logs by using Azure Diagnostics](#).

Logs for the patch orchestration app are generated on the following fixed provider IDs:

- e39b723c-590c-4090-abb0-11e3e6616346
- fc0028ff-bfdc-499f-80dc-ed922c52c5e9
- 24afa313-0d3b-4c7c-b485-1047fd964b60
- 05dc046c-60e9-4ef7-965e-91660adffa68

In Resource Manager template goto `EtwEventSourceProviderConfiguration` section under `WadCfg` and add the following entries:

```
{
  "provider": "e39b723c-590c-4090-abb0-11e3e6616346",
  "scheduledTransferPeriod": "PT5M",
  "DefaultEvents": {
    "eventDestination": "PatchOrchestrationApplicationTable"
  }
},
{
  "provider": "fc0028ff-bfdc-499f-80dc-ed922c52c5e9",
  "scheduledTransferPeriod": "PT5M",
  "DefaultEvents": {
    "eventDestination": "PatchOrchestrationApplicationTable"
  }
},
{
  "provider": "24afa313-0d3b-4c7c-b485-1047fd964b60",
  "scheduledTransferPeriod": "PT5M",
  "DefaultEvents": {
    "eventDestination": "PatchOrchestrationApplicationTable"
  }
},
{
  "provider": "05dc046c-60e9-4ef7-965e-91660adffa68",
  "scheduledTransferPeriod": "PT5M",
  "DefaultEvents": {
    "eventDestination": "PatchOrchestrationApplicationTable"
  }
}
```

#### NOTE

If your Service Fabric cluster has multiple node types, then the previous section must be added for all the `WadCfg` sections.

## Download the app package

Download the application from the [download link](#).

## Configure the app

The behavior of the patch orchestration app can be configured to meet your needs. Override the default values by passing in the application parameter during application creation or update. Application parameters can be provided by specifying `ApplicationParameter` to the `Start-ServiceFabricApplicationUpgrade` or `New-ServiceFabricApplication` cmdlets.

PARAMETER	TYPE	DETAILS
MaxResultsToCache	Long	Maximum number of Windows Update results, which should be cached. Default value is 3000 assuming the: - Number of nodes is 20. - Number of updates happening on a node per month is five. - Number of results per operation can be 10. - Results for the past three months should be stored.

PARAMETER	TYPE	DETAILS
TaskApprovalPolicy	Enum { NodeWise, UpgradeDomainWise }	TaskApprovalPolicy indicates the policy that is to be used by the Coordinator Service to install Windows updates across the Service Fabric cluster nodes. Allowed values are: <b>NodeWise</b> . Windows Update is installed one node at a time. <b>UpgradeDomainWise</b> . Windows Update is installed one upgrade domain at a time. (At the maximum, all the nodes belonging to an upgrade domain can go for Windows Update.)
LogsDiskQuotaInMB	Long (Default: 1024)	Maximum size of patch orchestration app logs in MB, which can be persisted locally on nodes.
WUQuery	string (Default: "IsInstalled=0")	Query to get Windows updates. For more information, see <a href="#">WuQuery</a> .
InstallWindowsOSOnlyUpdates	Bool (default: True)	This flag allows Windows operating system updates to be installed.
WUOperationTimeOutInMinutes	Int (Default: 90)	Specifies the timeout for any Windows Update operation (search or download or install). If the operation is not completed within the specified timeout, it is aborted.
WURescheduleCount	Int (Default: 5)	The maximum number of times the service reschedules the Windows update in case an operation fails persistently.
WURescheduleTimeInMinutes	Int (Default: 30)	The interval at which the service reschedules the Windows update in case failure persists.
WUFrequency	Comma-separated string (Default: "Weekly, Wednesday, 7:00:00")	The frequency for installing Windows Update. The format and possible values are: <ul style="list-style-type: none"> <li>- Monthly, DD,HH:MM:SS, for example, Monthly, 5,12:22:32.</li> <li>- Weekly, DAY,HH:MM:SS, for example, Weekly, Tuesday, 12:22:32.</li> <li>- Daily, HH:MM:SS, for example, Daily, 12:22:32.</li> <li>- None indicates that Windows Update shouldn't be done.</li> </ul> <p>Note that all the times are in UTC.</p>
AcceptWindowsUpdateEula	Bool (Default: true)	By setting this flag, the application accepts the End-User License Agreement for Windows Update on behalf of the owner of the machine.

#### TIP

If you want Windows Update to happen immediately, set `WUFrequency` relative to the application deployment time. For example, suppose that you have a five-node test cluster and plan to deploy the app at around 5:00 PM UTC. If you assume that the application upgrade or deployment takes 30 minutes at the maximum, set the `WUFrequency` as "Daily, 17:30:00."

## Deploy the app

1. Finish all the prerequisite steps to prepare the cluster.
2. Deploy the patch orchestration app like any other Service Fabric app. You can deploy the app by using PowerShell. Follow the steps in [Deploy and remove applications using PowerShell](#).
3. To configure the application at the time of deployment, pass the `ApplicationParameter` to the `New-ServiceFabricApplication` cmdlet. For your convenience, we've provided the script `Deploy.ps1` along with the application. To use the script:
  - Connect to a Service Fabric cluster by using `Connect-ServiceFabricCluster`.
  - Execute the PowerShell script `Deploy.ps1` with the appropriate `ApplicationParameter` value.

#### NOTE

Keep the script and the application folder `PatchOrchestrationApplication` in the same directory.

## Upgrade the app

To upgrade an existing patch orchestration app by using PowerShell, follow the steps in [Service Fabric application upgrade using PowerShell](#).

## Remove the app

To remove the application, follow the steps in [Deploy and remove applications using PowerShell](#).

For your convenience, we've provided the script `Undeploy.ps1` along with the application. To use the script:

- Connect to a Service Fabric cluster by using `Connect-ServiceFabricCluster`.
- Execute the PowerShell script `Undeploy.ps1`.

#### NOTE

Keep the script and the application folder `PatchOrchestrationApplication` in the same directory.

## View the Windows Update results

The patch orchestration app exposes REST APIs to display the historical results to the user. An example of the result JSON:

```

[
  {
    "NodeName": "_stg1vm_1",
    "WindowsUpdateOperationResults": [
      {
        "OperationResult": 0,
        "NodeName": "_stg1vm_1",
        "OperationTime": "2017-05-21T11:46:52.1953713Z",
        "UpdateDetails": [
          {
            "UpdateId": "7392acaf-6a85-427c-8a8d-058c25beb0d6",
            "Title": "Cumulative Security Update for Internet Explorer 11 for Windows Server 2012 R2 (KB3185319)",
            "Description": "A security issue has been identified in a Microsoft software product that could affect your system. You can help protect your system by installing this update from Microsoft. For a complete listing of the issues that are included in this update, see the associated Microsoft Knowledge Base article. After you install this update, you may have to restart your system.",
            "ResultCode": 0
          }
        ],
        "OperationType": 1,
        "WindowsUpdateQuery": "IsInstalled=0",
        "WindowsUpdateFrequency": "Daily,10:00:00",
        "RebootRequired": false
      }
    ],
    ...
  ]
]

```

Fields of the JSON are described below.

FIELD	VALUES	DETAILS
OperationResult	0 - Succeeded 1 - Succeeded With Errors 2 - Failed 3 - Aborted 4 - Aborted With Timeout	Indicates the result of overall operation (typically involving installation of one or more updates).
ResultCode	Same as OperationResult	This field indicates result of installation operation for an individual update.
OperationType	1 - Installation 0 - Search and Download.	Installation is the only OperationType which would be shown in the results by default.
WindowsUpdateQuery	Default is "IsInstalled=0"	Windows update query which was used to search for updates. For more information, see <a href="#">WuQuery</a> .
RebootRequired	true - reboot was required false - reboot was not required	Indicates if reboot was required to complete installation of updates.

If no update is scheduled yet, the result JSON is empty.

Log in to the cluster to query Windows Update results. Then find out the replica address for the primary of the Coordinator Service, and hit the URL from the browser: `http://<REPLICA-IP>:<ApplicationPort>/PatchOrchestrationApplication/v1/GetWindowsUpdateResults`.

The REST endpoint for the Coordinator Service has a dynamic port. To check the exact URL, refer to the Service

Fabric Explorer. For example, the results are available at

<http://10.0.0.7:20000/PatchOrchestrationApplication/v1/GetWindowsUpdateResults>.

94b20b39-3997-43e7-843b-6e6b3af306...	
ADDRESS	
Endpoints	
Rpc Primary Endpoint	10.0.0.7:49670+94b20b39-3997-43e7-843b-6e6b3af306d-131387073778817356
REST Endpoint	<a href="http://10.0.0.7:20000/PatchOrchestrationApplication">http://10.0.0.7:20000/PatchOrchestrationApplication</a>

If the reverse proxy is enabled on the cluster, you can access the URL from outside of the cluster as well. The endpoint that needs to be hit is `http://<SERVERURL>:`

`<REVERSEPROXYPORT>/PatchOrchestrationApplication/CoordinatorService/v1/GetWindowsUpdateResults`.

To enable the reverse proxy on the cluster, follow the steps in [Reverse proxy in Azure Service Fabric](#).

#### WARNING

After the reverse proxy is configured, all micro services in the cluster that expose an HTTP endpoint are addressable from outside the cluster.

## Diagnostics/health events

### Collect patch orchestration app logs

Patch orchestration app logs are collected as part of Service Fabric logs from runtime version [5.6.220.9494](#) and above. For clusters running Service Fabric runtime version less than [5.6.220.9494](#), logs can be collected by using one of the following methods.

#### Locally on each node

Logs are collected locally on each Service Fabric cluster node if Service Fabric runtime version is less than [5.6.220.9494](#). The location to access the logs is [Service Fabric\_Installation\_Drive]:\PatchOrchestrationApplication\logs.

For example, if Service Fabric is installed on drive D, the path is D:\PatchOrchestrationApplication\logs.

#### Central location

If Azure Diagnostics is configured as part of prerequisite steps, logs for the patch orchestration app are available in Azure Storage.

### Health reports

The patch orchestration app also publishes health reports against the Coordinator Service or the Node Agent Service in the following cases:

#### A Windows Update operation failed

If a Windows Update operation fails on a node, a health report is generated against the Node Agent Service. Details of the health report contain the problematic node name.

After patching is successfully completed on the problematic node, the report is automatically cleared.

#### The Node Agent NTService is down

If the Node Agent NTService is down on a node, a warning-level health report is generated against the Node Agent Service.

#### The repair manager service is not enabled

If the repair manager service is not found on the cluster, a warning-level health report is generated for the Coordinator Service.

## Frequently asked questions

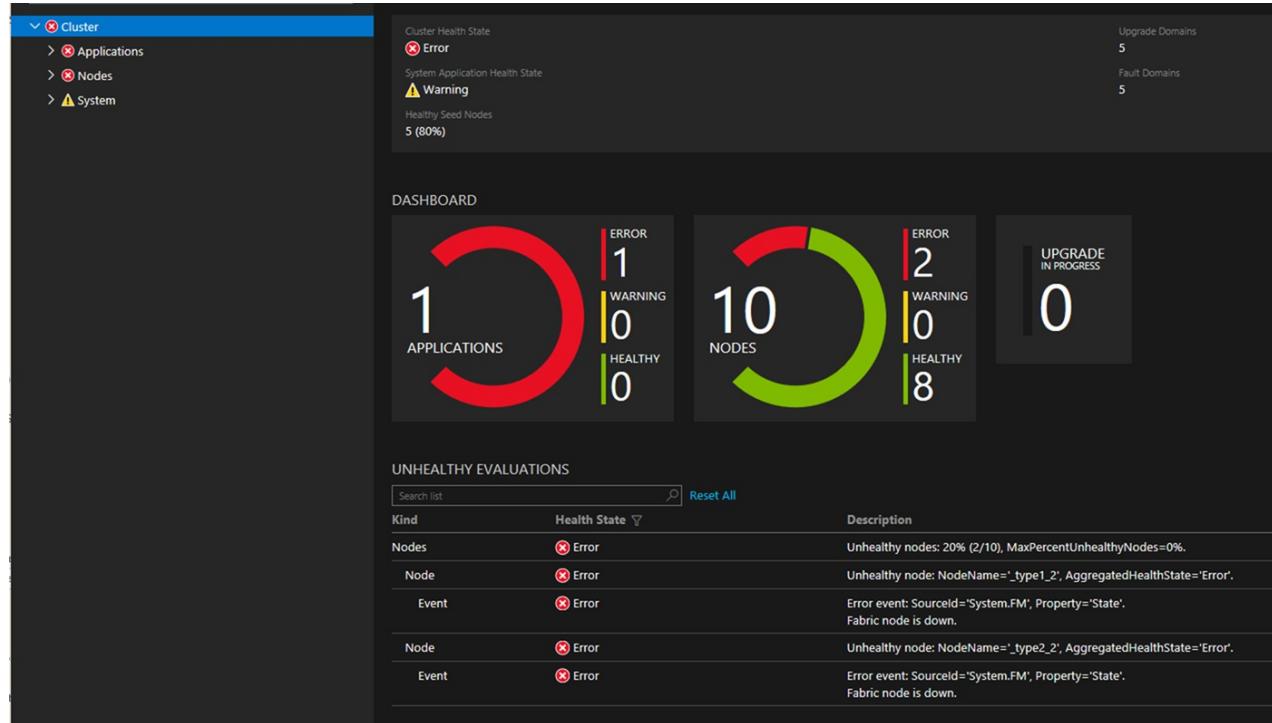
## Q. Why do I see my cluster in an error state when the patch orchestration app is running?

A. During the installation process, the patch orchestration app disables or restarts nodes, which can temporarily result in the health of the cluster going down.

Based on the policy for the application, either one node can go down during a patching operation *or* an entire upgrade domain can go down simultaneously.

By the end of the Windows Update installation, the nodes are reenabled post restart.

In the following example, the cluster went to an error state temporarily because two nodes were down and the MaxPercentageUnhealthNodes policy was violated. The error is temporary until the patching operation is ongoing.



If the issue persists, refer to the Troubleshooting section.

## Q. Patch orchestration app is in warning state

A. Check to see if a health report posted against the application is the root cause. Usually, the warning contains details of the problem. If the issue is transient, the application is expected to auto-recover from this state.

## Q. What can I do if my cluster is unhealthy and I need to do an urgent operating system update?

A. The patch orchestration app does not install updates while the cluster is unhealthy. Try to bring your cluster to a healthy state to unblock the patch orchestration app workflow.

## Q. Why does patching across clusters take so long to run?

A. The time needed by the patch orchestration app is mostly dependent on the following factors:

- The policy of the Coordinator Service.
  - The default policy, `NodeWise`, results in patching only one node at a time. Especially in the case of bigger clusters, we recommend that you use the `UpgradeDomainWise` policy to achieve faster patching across clusters.
- The number of updates available for download and installation.
- The average time needed to download and install an update, which should not exceed a couple of hours.
- The performance of the VM and network bandwidth.

## Q. Why do I see some updates in Windows Update results obtained via REST API's, but not under the

## **Windows Update history on the machine?**

A. Some product updates need to be checked in their respective update/patch history. For example, Windows Defender updates do not show up in Windows Update history on Windows Server 2016.

## **Disclaimers**

- The patch orchestration app accepts the End-User License Agreement of Windows Update on behalf of the user. Optionally, the setting can be turned off in the configuration of the application.
- The patch orchestration app collects telemetry to track usage and performance. The application's telemetry follows the setting of the Service Fabric runtime's telemetry setting (which is on by default).

## **Troubleshooting**

### **A node is not coming back to up state**

#### **The node might be stuck in a disabling state because:**

A safety check is pending. To remedy this situation, ensure that enough nodes are available in a healthy state.

#### **The node might be stuck in a disabled state because:**

- The node was disabled manually.
- The node was disabled due to an ongoing Azure infrastructure job.
- The node was disabled temporarily by the patch orchestration app to patch the node.

#### **The node might be stuck in a down state because:**

- The node was put in a down state manually.
- The node is undergoing a restart (which might be triggered by the patch orchestration app).
- The node is down due to a faulty VM or machine or network connectivity issues.

### **Updates were skipped on some nodes**

The patch orchestration app tries to install a Windows update according to the rescheduling policy. The service tries to recover the node and skip the update according to the application policy.

In such a case, a warning-level health report is generated against the Node Agent Service. The result for Windows Update also contains the possible reason for the failure.

### **The health of the cluster goes to error while the update installs**

A faulty Windows update can bring down the health of an application or cluster on a particular node or upgrade domain. The patch orchestration app discontinues any subsequent Windows Update operation until the cluster is healthy again.

An administrator must intervene and determine why the application or cluster became unhealthy due to Windows Update.

## **Release Notes**

### **Version 1.1.0**

- Public release

### **Version 1.1.1**

- Fixed a bug in SetupEntryPoint of NodeAgentService that prevented installation of NodeAgentNTService.

### **Version 1.2.0 (Latest)**

- Bug fixes around system restart workflow.
- Bug fix in creation of RM tasks due to which health check during preparing repair tasks wasn't happening as expected.
- Changed the startup mode for windows service POANodeSvc from auto to delayed-auto.

# Introducing the Service Fabric cluster resource manager

8/18/2017 • 6 min to read • [Edit Online](#)

Traditionally managing IT systems or online services meant dedicating specific physical or virtual machines to those specific services or systems. Services were architected as tiers. There would be a "web" tier and a "data" or "storage" tier. Applications would have a messaging tier where requests flowed in and out, as well as a set of machines dedicated to caching. Each tier or type of workload had specific machines dedicated to it: the database got a couple machines dedicated to it, the web servers a few. If a particular type of workload caused the machines it was on to run too hot, then you added more machines with that same configuration to that tier. However, not all workloads could be scaled out so easily - particularly with the data tier you would typically replace machines with larger machines. Easy. If a machine failed, that part of the overall application ran at lower capacity until the machine could be restored. Still fairly easy (if not necessarily fun).

Now however the world of service and software architecture has changed. It's more common that applications have adopted a scale-out design. Building applications with containers or microservices (or both) is common. Now, while you may still have only a few machines, they're not running just a single instance of a workload. They may even be running multiple different workloads at the same time. You now have dozens of different types of services (none consuming a full machine's worth of resources), perhaps hundreds of different instances of those services. Each named instance has one or more instances or replicas for High Availability (HA). Depending on the sizes of those workloads, and how busy they are, you may find yourself with hundreds or thousands of machines.

Suddenly managing your environment is not so simple as managing a few machines dedicated to single types of workloads. Your servers are virtual and no longer have names (you have switched mindsets from [pets to cattle](#) after all). Configuration is less about the machines and more about the services themselves. Hardware that is dedicated to a single instance of a workload is largely a thing of the past. Services themselves have become small distributed systems that span multiple smaller pieces of commodity hardware.

Because your app is no longer a series of monoliths spread across several tiers, you now have many more combinations to deal with. Who decides what types of workloads can run on which hardware, or how many? Which workloads work well on the same hardware, and which conflict? When a machine goes down how do you know what was running there on that machine? Who is in charge of making sure that workload starts running again? Do you wait for the (virtual?) machine to come back or do your workloads automatically fail over to other machines and keep running? Is human intervention required? What about upgrades in this environment?

As developers and operators dealing in this environment, we're going to want help managing this complexity. A hiring binge and trying to hide the complexity with people is probably not the right answer, so what do we do?

## Introducing orchestrators

An "Orchestrator" is the general term for a piece of software that helps administrators manage these types of environments. Orchestrators are the components that take in requests like "I would like five copies of this service running in my environment." They try to make the environment match the desired state, no matter what happens.

Orchestrators (not humans) are what take action when a machine fails or a workload terminates for some unexpected reason. Most orchestrators do more than just deal with failure. Other features they have are managing new deployments, handling upgrades, and dealing with resource consumption and governance. All

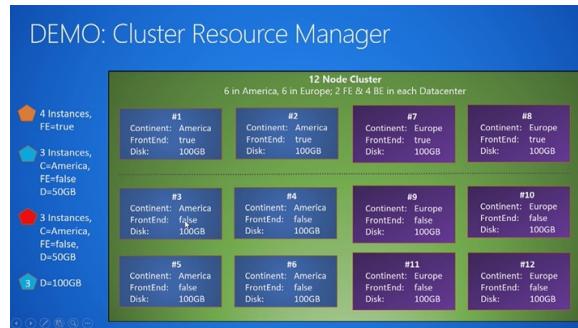
orchestrators are fundamentally about maintaining some desired state of configuration in the environment. You want to be able to tell an orchestrator what you want and have it do the heavy lifting. Aurora on top of Mesos, Docker Datacenter/Docker Swarm, Kubernetes, and Service Fabric are all examples of orchestrators. These orchestrators are being actively developed to meet the needs of real workloads in production environments.

## Orchestration as a service

The Cluster Resource Manager is the system component that handles orchestration in Service Fabric. The Cluster Resource Manager's job is broken down into three parts:

1. Enforcing Rules
2. Optimizing Your Environment
3. Helping with Other Processes

To see how the Cluster Resource Manager works, watch the following Microsoft Virtual Academy video:



### What it isn't

In traditional N tier applications, there's always a [Load Balancer](#). Usually this was a Network Load Balancer (NLB) or an Application Load Balancer (ALB) depending on where it sat in the networking stack. Some load balancers are Hardware-based like F5's BigIP offering, others are software-based such as Microsoft's NLB. In other environments, you might see something like HAProxy, nginx, Istio, or Envoy in this role. In these architectures, the job of load balancing is to ensure stateless workloads receive (roughly) the same amount of work. Strategies for balancing load varied. Some balancers would send each different call to a different server. Others provided session pinning/stickiness. More advanced balancers use actual load estimation or reporting to route a call based on its expected cost and current machine load.

Network balancers or message routers tried to ensure that the web/worker tier remained roughly balanced. Strategies for balancing the data tier were different and depended on the data storage mechanism. Balancing the data tier relied on data sharding, caching, managed views, stored procedures, and other store-specific mechanisms.

While some of these strategies are interesting, the Service Fabric Cluster Resource Manager is not anything like a network load balancer or a cache. A Network Load Balancer balances frontends by spreading traffic across frontends. The Service Fabric Cluster Resource Manager has a different strategy. Fundamentally, Service Fabric moves *services* to where they make the most sense, expecting traffic or load to follow. For example, it might move services to nodes that are currently cold because the services that are there are not doing much work. The nodes may be cold since the services that were present were deleted or moved elsewhere. As another example, the Cluster Resource Manager could also move a service away from a machine. Perhaps the machine is about to be upgraded, or is overloaded due to a spike in consumption by the services running on it. Alternatively, the service's resource requirements may have increased. As a result there aren't sufficient resources on this machine to continue running it.

Because the Cluster Resource Manager is responsible for moving services around, it contains a different feature set compared to what you would find in a network load balancer. This is because network load balancers deliver network traffic to where services already are, even if that location is not ideal for running the service itself. The

Service Fabric Cluster Resource Manager employs fundamentally different strategies for ensuring that the resources in the cluster are efficiently utilized.

## Next steps

- For information on the architecture and information flow within the Cluster Resource Manager, check out [this article](#)
- The Cluster Resource Manager has many options for describing the cluster. To find out more about metrics, check out this article on [describing a Service Fabric cluster](#)
- For more information on configuring services, [Learn about configuring Services](#)(service-fabric-cluster-resource-manager-configure-services.md)
- Metrics are how the Service Fabric Cluster Resource Manager manages consumption and capacity in the cluster. To learn more about metrics and how to configure them check out [this article](#)
- The Cluster Resource Manager works with Service Fabric's management capabilities. To find out more about that integration, read [this article](#)
- To find out about how the Cluster Resource Manager manages and balances load in the cluster, check out the article on [balancing load](#)

# Cluster resource manager architecture overview

8/18/2017 • 3 min to read • [Edit Online](#)

The Service Fabric Cluster Resource Manager is a central service that runs in the cluster. It manages the desired state of the services in the cluster, particularly with respect to resource consumption and any placement rules.

To manage the resources in your cluster, the Service Fabric Cluster Resource Manager must have several pieces of information:

- Which services currently exist
- Each service's current (or default) resource consumption
- The remaining cluster capacity
- The capacity of the nodes in the cluster
- The amount of resources consumed on each node

The resource consumption of a given service can change over time, and services usually care about more than one type of resource. Across different services, there may be both real physical and physical resources being measured. Services may track physical metrics like memory and disk consumption. More commonly, services may care about logical metrics - things like "WorkQueueDepth" or "TotalRequests". Both logical and physical metrics can be used in the same cluster. Metrics can be shared across many services or be specific to a particular service.

## Other considerations

The owners and operators of the cluster can be different from the service and application authors, or at a minimum are the same people wearing different hats. When you develop your application you know a few things about what it requires. You have an estimate of the resources it will consume and how different services should be deployed. For example, the web tier needs to run on nodes exposed to the Internet, while the database services should not. As another example, the web services are probably constrained by CPU and network, while the data tier services care more about memory and disk consumption. However, the person handling a live-site incident for that service in production, or who is managing an upgrade to the service has a different job to do, and requires different tools.

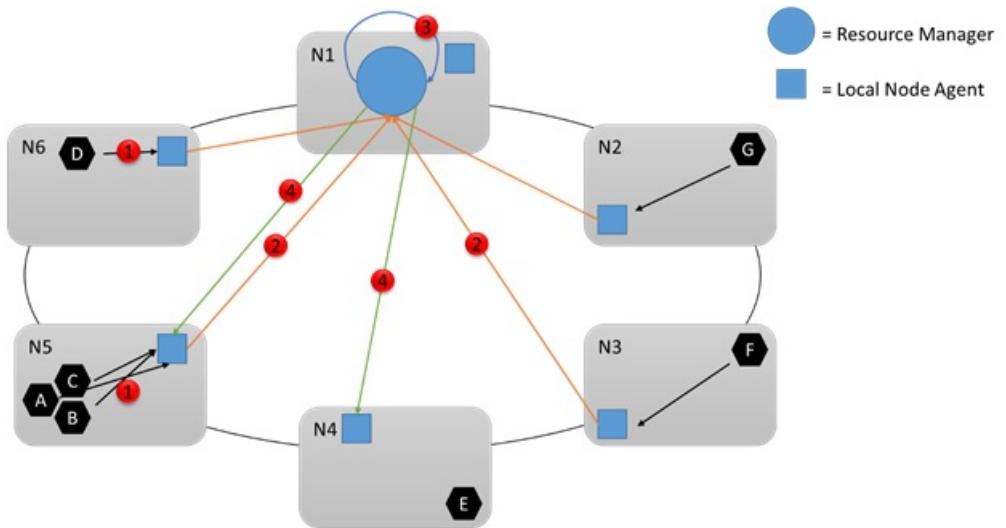
Both the cluster and services are dynamic:

- The number of nodes in the cluster can grow and shrink
- Nodes of different sizes and types can come and go
- Services can be created, removed, and change their desired resource allocations and placement rules
- Upgrades or other management operations can roll through the cluster at the application or infrastructure levels
- Failures can happen at any time.

## Cluster resource manager components and data flow

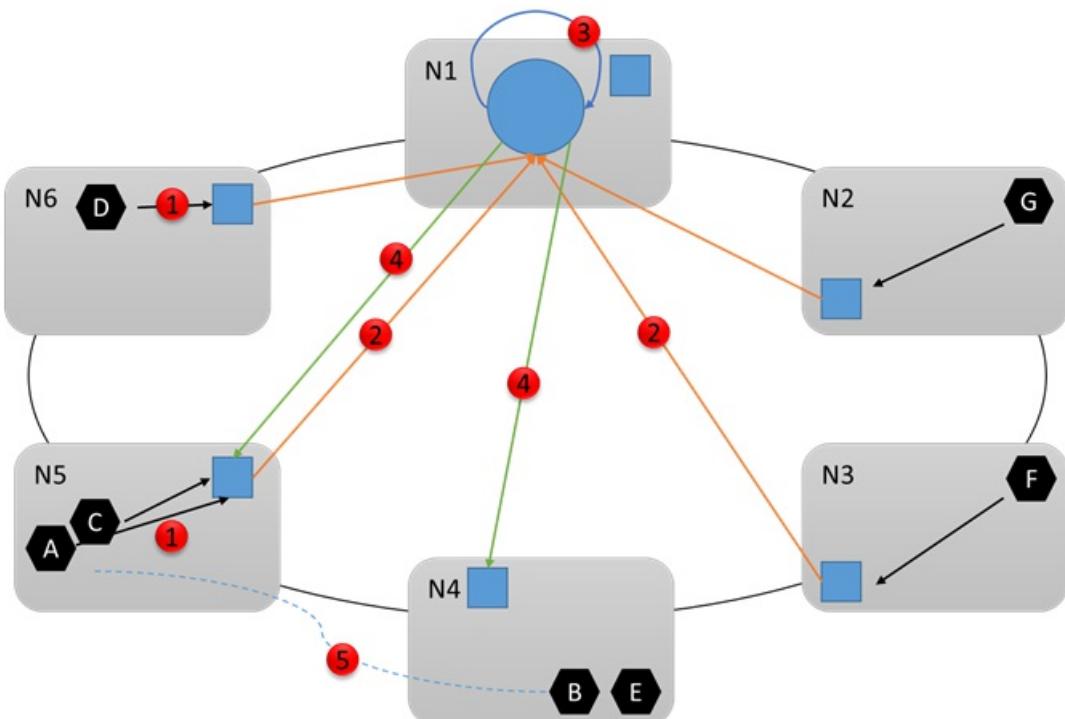
The Cluster Resource Manager has to track the requirements of each service and the consumption of resources by each service object within those services. The Cluster Resource Manager has two conceptual parts: agents that run on each node and a fault-tolerant service. The agents on each node track load reports from services, aggregate them, and periodically report them. The Cluster Resource Manager service aggregates all the information from the local agents and reacts based on its current configuration.

Let's look at the following diagram:



During runtime, there are many changes that could happen. For example, let's say the amount of resources some services consume changes, some services fail, and some nodes join and leave the cluster. All the changes on a node are aggregated and periodically sent to the Cluster Resource Manager service (1,2) where they are aggregated again, analyzed, and stored. Every few seconds that service looks at the changes and determines if any actions are necessary (3). For example, it could notice that some empty nodes have been added to the cluster. As a result, it decides to move some services to those nodes. The Cluster Resource Manager could also notice that a particular node is overloaded, or that certain services have failed or been deleted, freeing up resources elsewhere.

Let's look at the following diagram and see what happens next. Let's say that the Cluster Resource Manager determines that changes are necessary. It coordinates with other system services (in particular the Failover Manager) to make the necessary changes. Then the necessary commands are sent to the appropriate nodes (4). For example, let's say the Resource Manager noticed that Node5 was overloaded, and so decided to move service B from Node5 to Node4. At the end of the reconfiguration (5), the cluster looks like this:



## Next steps

- The Cluster Resource Manager has many options for describing the cluster. To find out more about them, check out this article on [describing a Service Fabric cluster](#)
- The Cluster Resource Manager's primary duties are rebalancing the cluster and enforcing placement rules. For

more information on configuring these behaviors, see [balancing your Service Fabric cluster](#)

# Describing a service fabric cluster

10/26/2017 • 23 min to read • [Edit Online](#)

The Service Fabric Cluster Resource Manager provides several mechanisms for describing a cluster. During runtime, the Cluster Resource Manager uses this information to ensure high availability of the services running in the cluster. While enforcing these important rules, it also attempts to optimize the resource consumption within the cluster.

## Key concepts

The Cluster Resource Manager supports several features that describe a cluster:

- Fault Domains
- Upgrade Domains
- Node Properties
- Node Capacities

## Fault domains

A Fault Domain is any area of coordinated failure. A single machine is a Fault Domain (since it can fail on its own for various reasons, from power supply failures to drive failures to bad NIC firmware). Machines connected to the same Ethernet switch are in the same Fault Domain, as are machines sharing a single source of power or in a single location. Since it's natural for hardware faults to overlap, Fault Domains are inherently hierachal and are represented as URIs in Service Fabric.

It is important that Fault Domains are set up correctly since Service Fabric uses this information to safely place services. Service Fabric doesn't want to place services such that the loss of a Fault Domain (caused by the failure of some component) causes a service to go down. In the Azure environment Service Fabric uses the Fault Domain information provided by the environment to correctly configure the nodes in the cluster on your behalf. For Service Fabric Standalone, Fault Domains are defined at the time that the cluster is set up

### WARNING

It is important that the Fault Domain information provided to Service Fabric is accurate. For example, let's say that your Service Fabric cluster's nodes are running inside 10 virtual machines, running on five physical hosts. In this case, even though there are 10 virtual machines, there are only 5 different (top level) fault domains. Sharing the same physical host causes VMs to share the same root fault domain, since the VMs experience coordinated failure if their physical host fails.

Service Fabric expects the Fault Domain of a node not to change. Other mechanisms of ensuring high availability of the VMs such as [HA-VMs](#) may cause conflicts with Service Fabric, as they use transparent migration of VMs from one host to another. These mechanisms do not reconfigure or notify the running code inside the VM. As such, they are **not supported** as environments for running Service Fabric clusters. Service Fabric should be the only high-availability technology employed. Mechanisms like live VM migration, SANs, or others are not necessary. If used in conjunction with Service Fabric, these mechanisms *reduce* application availability and reliability because they introduce additional complexity, add centralized sources of failure, and utilize reliability and availability strategies that conflict with those in Service Fabric.

In the graphic below we color all the entities that contribute to Fault Domains and list all the different Fault Domains that result. In this example, we have datacenters ("DC"), racks ("R"), and blades ("B"). Conceivably, if each blade holds more than one virtual machine, there could be another layer in the Fault Domain hierarchy.

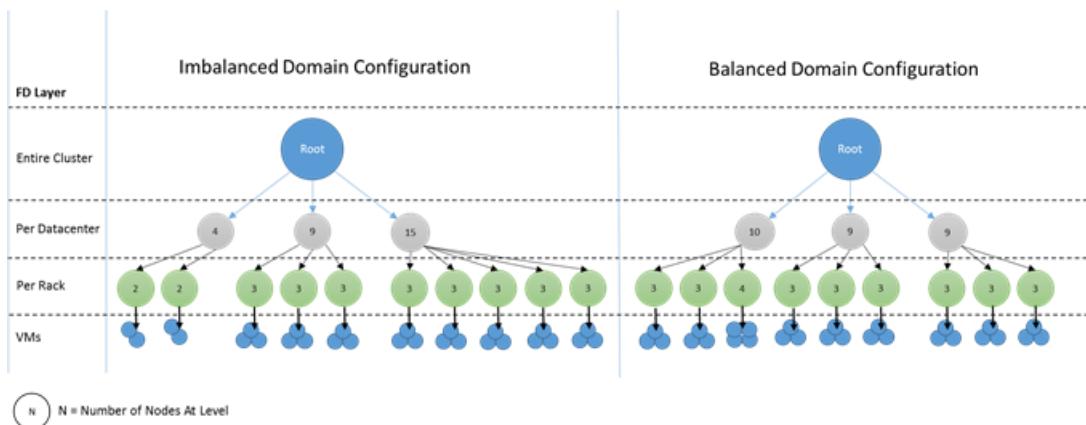


During runtime, the Service Fabric Cluster Resource Manager considers the Fault Domains in the cluster and plans layouts. The stateful replicas or stateless instances for a given service are distributed so they are in separate Fault Domains. Distributing the service across fault domains ensures the availability of the service is not compromised when a Fault Domain fails at any level of the hierarchy.

Service Fabric's Cluster Resource Manager doesn't care how many layers there are in the Fault Domain hierarchy. However, it tries to ensure that the loss of any one portion of the hierarchy doesn't impact services running in it.

It is best if there are the same number of nodes at each level of depth in the Fault Domain hierarchy. If the "tree" of Fault Domains is unbalanced in your cluster, it makes it harder for the Cluster Resource Manager to figure out the best allocation of services. Imbalanced Fault Domains layouts mean that the loss of some domains impact the availability of services more than other domains. As a result, the Cluster Resource Manager is torn between two goals: It wants to use the machines in that "heavy" domain by placing services on them, and it wants to place services in other domains so that the loss of a domain doesn't cause problems.

What do imbalanced domains look like? In the diagram below, we show two different cluster layouts. In the first example, the nodes are distributed evenly across the Fault Domains. In the second example, one Fault Domain has many more nodes than the other Fault Domains.



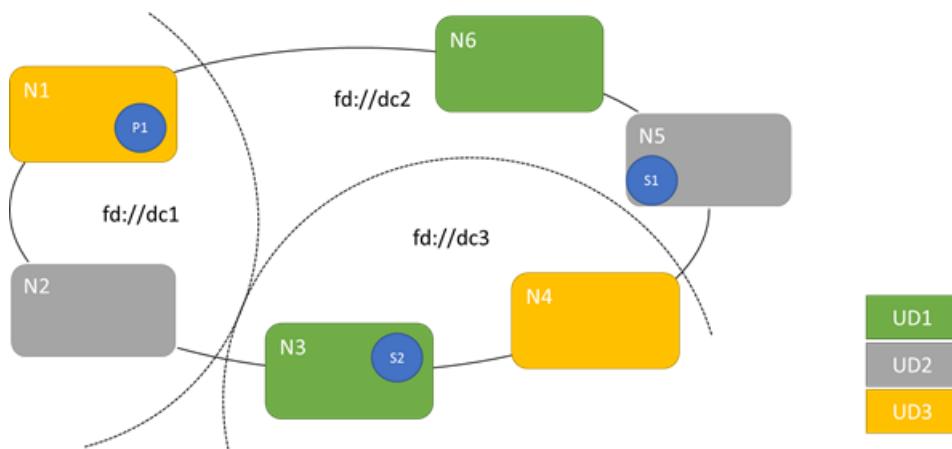
In Azure, the choice of which Fault Domain contains a node is managed for you. However, depending on the number of nodes that you provision you can still end up with Fault Domains with more nodes in them than others. For example, say you have five Fault Domains in the cluster but provision seven nodes for a given NodeType. In this case, the first two Fault Domains end up with more nodes. If you continue to deploy more NodeTypes with only a couple instances, the problem gets worse. For this reason it's recommended that the number of nodes in each node type is a multiple of the number of Fault Domains.

## Upgrade domains

Upgrade Domains are another feature that helps the Service Fabric Cluster Resource Manager understand the layout of the cluster. Upgrade Domains define sets of nodes that are upgraded at the same time. Upgrade Domains help the Cluster Resource Manager understand and orchestrate management operations like upgrades.

Upgrade Domains are a lot like Fault Domains, but with a couple key differences. First, areas of coordinated hardware failures define Fault Domains. Upgrade Domains, on the other hand, are defined by policy. You get to decide how many you want, rather than it being dictated by the environment. You could have as many Upgrade Domains as you do nodes. Another difference between Fault Domains and Upgrade Domains is that Upgrade Domains are not hierarchical. Instead, they are more like a simple tag.

The following diagram shows three Upgrade Domains are striped across three Fault Domains. It also shows one possible placement for three different replicas of a stateful service, where each ends up in different Fault and Upgrade Domains. This placement allows the loss of a Fault Domain while in the middle of a service upgrade and still have one copy of the code and data.



There are pros and cons to having large numbers of Upgrade Domains. More Upgrade Domains means each step of the upgrade is more granular and therefore affects a smaller number of nodes or services. As a result, fewer services have to move at a time, introducing less churn into the system. This tends to improve reliability, since less of the service is impacted by any issue introduced during the upgrade. More Upgrade Domains also means that you need less available buffer on other nodes to handle the impact of the upgrade. For example, if you have five Upgrade Domains, the nodes in each are handling roughly 20% of your traffic. If you need to take down that Upgrade Domain for an upgrade, that load usually needs to go somewhere. Since you have four remaining Upgrade Domains, each must have room for about 5% of the total traffic. More Upgrade Domains means you need less buffer on the nodes in the cluster. For example, consider if you had 10 Upgrade Domains instead. In that case, each UD would only be handling about 10% of the total traffic. When an upgrade steps through the cluster, each domain would only need to have room for about 1.1% of the total traffic. More Upgrade Domains generally allow you to run your nodes at higher utilization, since you need less reserved capacity. The same is true for Fault Domains.

The downside of having many Upgrade Domains is that upgrades tend to take longer. Service Fabric waits a short period of time after an Upgrade Domain is completed and performs checks before starting to upgrade the next one. These delays enable detecting issues introduced by the upgrade before the upgrade proceeds. The tradeoff is acceptable because it prevents bad changes from affecting too much of the service at a time.

Too few Upgrade Domains has many negative side effects – while each individual Upgrade Domain is down and being upgraded a large portion of your overall capacity is unavailable. For example, if you only have three Upgrade Domains you are taking down about 1/3 of your overall service or cluster capacity at a time. Having so much of your service down at once isn't desirable since you have to have enough capacity in the rest of your cluster to handle the workload. Maintaining that buffer means that during normal operation those nodes are less loaded than they would be otherwise. This increases the cost of running your service.

There's no real limit to the total number of fault or Upgrade Domains in an environment, or constraints on how they overlap. That said, there are several common patterns:

- Fault Domains and Upgrade Domains mapped 1:1
- One Upgrade Domain per Node (physical or virtual OS instance)
- A "striped" or "matrix" model where the Fault Domains and Upgrade Domains form a matrix with machines usually running down the diagonals

UD Per Node			
	FD1	FD2	FD3
UD1	Node1		
UD2		Node2	
UD3			Node3
UD4	Node4		
UD5		Node5	
UD6			Node6
UD7	Node7		
UD8		Node8	
UD9			Node9

FD/UD Matrix (Sparse/Diagonal)					
	FD1	FD2	FD3	FD4	FD5
UD1	Node1				
UD2		Node2			
UD3			Node3		
UD4				Node4	
UD5					Node5

#### 1:1 FD & UD

FD1/UD1	FD2/UD2	FD3/UD3
Node1	Node2	Node3
Node4	Node5	Node6
Node7	Node8	Node9

#### FD/UD Matrix

	FD1	FD2	FD3
UD1	Node1	Node2	Node3
UD2	Node4	Node5	Node6
UD3	Node7	Node8	Node9

There's no best answer which layout to choose, each has some pros and cons. For example, the 1FD:1UD model is simple to set up. The 1 Upgrade Domain per Node model is most like what people are used to. During upgrades each node is updated independently. This is similar to how small sets of machines were upgraded manually in the past.

The most common model is the FD/UD matrix, where the FDs and UDs form a table and nodes are placed starting along the diagonal. This is the model used by default in Service Fabric clusters in Azure. For clusters with many nodes everything ends up looking like the dense matrix pattern above.

## Fault and Upgrade Domain constraints and resulting behavior

The Cluster Resource Manager treats the desire to keep a service balanced across fault and Upgrade Domains as a constraint. You can find out more about constraints in [this article](#). The Fault and Upgrade Domain constraints state: "For a given service partition there should never be a difference *greater than one* in the number of service objects (stateless service instances or stateful service replicas) between two domains." This prevents certain moves or arrangements that violate this constraint.

Let's look at one example. Let's say that we have a cluster with six nodes, configured with five Fault Domains and five Upgrade Domains.

	FD0	FD1	FD2	FD3	FD4
UD0	N1				
UD1	N6	N2			
UD2			N3		
UD3				N4	

	<b>FD0</b>	<b>FD1</b>	<b>FD2</b>	<b>FD3</b>	<b>FD4</b>	
<b>UD4</b>						N5

Now let's say that we create a service with a TargetReplicaSetSize (or, for a stateless service an InstanceCount) of five. The replicas land on N1-N5. In fact, N6 is never used no matter how many services like this you create. But why? Let's look at the difference between the current layout and what would happen if N6 is chosen.

Here's the layout we got and the total number of replicas per Fault and Upgrade Domain:

	<b>FD0</b>	<b>FD1</b>	<b>FD2</b>	<b>FD3</b>	<b>FD4</b>	<b>UDTOTAL</b>
<b>UD0</b>	R1					1
<b>UD1</b>		R2				1
<b>UD2</b>			R3			1
<b>UD3</b>				R4		1
<b>UD4</b>					R5	1
<b>FTotal</b>	1	1	1	1	1	-

This layout is balanced in terms of nodes per Fault Domain and Upgrade Domain. It is also balanced in terms of the number of replicas per Fault and Upgrade Domain. Each domain has the same number of nodes and the same number of replicas.

Now, let's look at what would happen if we'd used N6 instead of N2. How would the replicas be distributed then?

	<b>FD0</b>	<b>FD1</b>	<b>FD2</b>	<b>FD3</b>	<b>FD4</b>	<b>UDTOTAL</b>
<b>UD0</b>	R1					1
<b>UD1</b>		R5				1
<b>UD2</b>			R2			1
<b>UD3</b>				R3		1
<b>UD4</b>					R4	1
<b>FTotal</b>	2	0	1	1	1	-

This layout violates our definition for the Fault Domain constraint. FD0 has two replicas, while FD1 has zero, making the difference between FD0 and FD1 a total of two. The Cluster Resource Manager does not allow this arrangement. Similarly if we picked N2 and N6 (instead of N1 and N2) we'd get:

	<b>FD0</b>	<b>FD1</b>	<b>FD2</b>	<b>FD3</b>	<b>FD4</b>	<b>UDTOTAL</b>
<b>UD0</b>						0

	<b>FD0</b>	<b>FD1</b>	<b>FD2</b>	<b>FD3</b>	<b>FD4</b>	<b>UDTOTAL</b>
<b>UD1</b>	R5	R1				2
<b>UD2</b>			R2			1
<b>UD3</b>				R3		1
<b>UD4</b>					R4	1
<b>FDTotal</b>	1	1	1	1	1	-

This layout is balanced in terms of Fault Domains. However, now it's violating the Upgrade Domain constraint. This is because UD0 has zero replicas while UD1 has two. Therefore, this layout is also invalid and won't be picked by the Cluster Resource Manager.

## Configuring fault and Upgrade Domains

Defining Fault Domains and Upgrade Domains is done automatically in Azure hosted Service Fabric deployments. Service Fabric picks up and uses the environment information from Azure.

If you're creating your own cluster (or want to run a particular topology in development), you can provide the Fault Domain and Upgrade Domain information yourself. In this example, we define a nine node local development cluster that spans three "datacenters" (each with three racks). This cluster also has three Upgrade Domains striped across those three datacenters. An example of the configuration is below:

ClusterManifest.xml

```

<Infrastructure>
    <!-- IsScaleMin indicates that this cluster runs on one-box /one single server -->
    <WindowsServer IsScaleMin="true">
        <NodeList>
            <Node NodeName="Node01" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType01"
FaultDomain="fd:/DC01/Rack01" UpgradeDomain="UpgradeDomain1" IsSeedNode="true" />
            <Node NodeName="Node02" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType02"
FaultDomain="fd:/DC01/Rack02" UpgradeDomain="UpgradeDomain2" IsSeedNode="true" />
            <Node NodeName="Node03" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType03"
FaultDomain="fd:/DC01/Rack03" UpgradeDomain="UpgradeDomain3" IsSeedNode="true" />
            <Node NodeName="Node04" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType04"
FaultDomain="fd:/DC02/Rack01" UpgradeDomain="UpgradeDomain1" IsSeedNode="true" />
            <Node NodeName="Node05" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType05"
FaultDomain="fd:/DC02/Rack02" UpgradeDomain="UpgradeDomain2" IsSeedNode="true" />
            <Node NodeName="Node06" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType06"
FaultDomain="fd:/DC02/Rack03" UpgradeDomain="UpgradeDomain3" IsSeedNode="true" />
            <Node NodeName="Node07" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType07"
FaultDomain="fd:/DC03/Rack01" UpgradeDomain="UpgradeDomain1" IsSeedNode="true" />
            <Node NodeName="Node08" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType08"
FaultDomain="fd:/DC03/Rack02" UpgradeDomain="UpgradeDomain2" IsSeedNode="true" />
            <Node NodeName="Node09" IPAddressOrFQDN="localhost" NodeTypeRef="NodeType09"
FaultDomain="fd:/DC03/Rack03" UpgradeDomain="UpgradeDomain3" IsSeedNode="true" />
        </NodeList>
    </WindowsServer>
</Infrastructure>
```

via ClusterConfig.json for Standalone deployments

```
"nodes": [
  {
    "nodeName": "vm1",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc1/r0",
    "upgradeDomain": "UD1"
  },
  {
    "nodeName": "vm2",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc1/r0",
    "upgradeDomain": "UD2"
  },
  {
    "nodeName": "vm3",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc1/r0",
    "upgradeDomain": "UD3"
  },
  {
    "nodeName": "vm4",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc2/r0",
    "upgradeDomain": "UD1"
  },
  {
    "nodeName": "vm5",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc2/r0",
    "upgradeDomain": "UD2"
  },
  {
    "nodeName": "vm6",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc2/r0",
    "upgradeDomain": "UD3"
  },
  {
    "nodeName": "vm7",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc3/r0",
    "upgradeDomain": "UD1"
  },
  {
    "nodeName": "vm8",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc3/r0",
    "upgradeDomain": "UD2"
  },
  {
    "nodeName": "vm9",
    "iPAddress": "localhost",
    "nodeTypeRef": "NodeType0",
    "faultDomain": "fd:/dc3/r0",
    "upgradeDomain": "UD3"
  }
],
```

#### NOTE

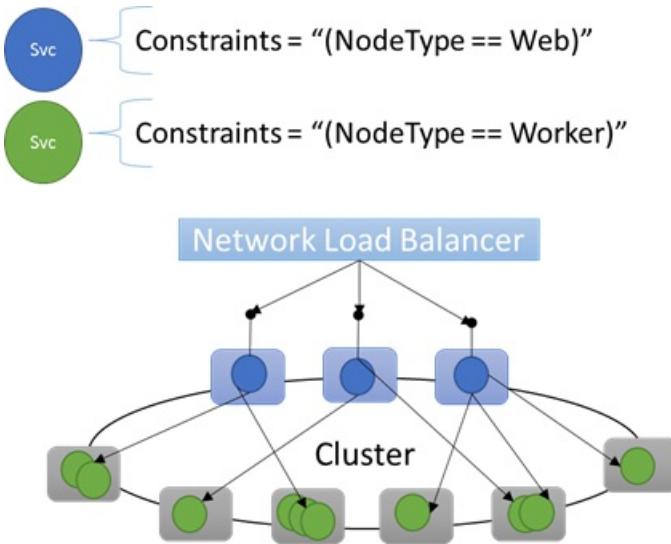
When defining clusters via Azure Resource Manager, Fault Domains and Upgrade Domains are assigned by Azure. Therefore, the definition of your Node Types and Virtual Machine Scale Sets in your Azure Resource Manager template does not include Fault Domain or Upgrade Domain information.

## Node properties and placement constraints

Sometimes (in fact, most of the time) you're going to want to ensure that certain workloads run only on certain types of nodes in the cluster. For example, some workload may require GPUs or SSDs while others may not. A great example of targeting hardware to particular workloads is almost every n-tier architecture out there. Certain machines serve as the front end or API serving side of the application and are exposed to the clients or the internet. Different machines, often with different hardware resources, handle the work of the compute or storage layers. These are usually *not* directly exposed to clients or the internet. Service Fabric expects that there are cases where particular workloads need to run on particular hardware configurations. For example:

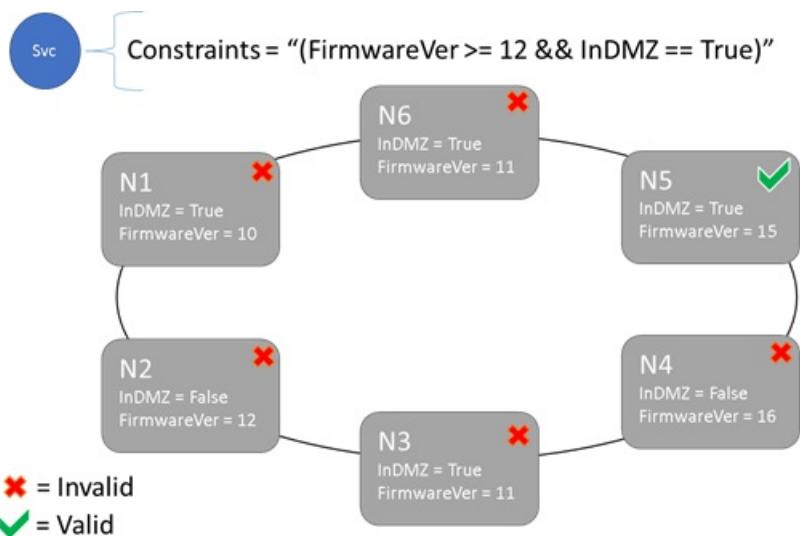
- an existing n-tier application has been "lifted and shifted" into a Service Fabric environment
- a workload wants to run on specific hardware for performance, scale, or security isolation reasons
- A workload should be isolated from other workloads for policy or resource consumption reasons

To support these sorts of configurations, Service Fabric has a first class notion of tags that can be applied to nodes. These tags are called **node properties**. **Placement constraints** are the statements attached to individual services that select for one or more node properties. Placement constraints define where services should run. The set of constraints is extensible - any key/value pair can work.



### Built in node properties

Service Fabric defines some default node properties that can be used automatically without the user having to define them. The default properties defined at each node are the **NodeType** and the **NodeName**. So for example you could write a placement constraint as `"(NodeType == NodeType03)"`. Generally we have found NodeType to be one of the most commonly used properties. It is useful since it corresponds 1:1 with a type of a machine. Each type of machine corresponds to a type of workload in a traditional n-tier application.



## Placement Constraint and Node Property Syntax

The value specified in the node property can be a string, bool, or signed long. The statement at the service is called a placement *constraint* since it constrains where the service can run in the cluster. The constraint can be any Boolean statement that operates on the different node properties in the cluster. The valid selectors in these boolean statements are:

- 1) conditional checks for creating particular statements

STATEMENT	SYNTAX
"equal to"	"=="
"not equal to"	"!="
"greater than"	
"greater than or equal to"	>= "
"less than"	<"
"less than or equal to"	<= "

- 2) boolean statements for grouping and logical operations

STATEMENT	SYNTAX
"and"	"&&"
"or"	"  "
"not"	"!"
"group as single statement"	"0"

Here are some examples of basic constraint statements.

- "Value >= 5"

- "NodeColor != green"
- "((OneProperty < 100) || ((AnotherProperty == false) && (OneProperty >= 100)))"

Only nodes where the overall placement constraint statement evaluates to "True" can have the service placed on it. Nodes that do not have a property defined do not match any placement constraint containing that property.

Let's say that the following node properties were defined for a given node type:

ClusterManifest.xml

```
<NodeType Name="NodeType01">
  <PlacementProperties>
    <Property Name="HasSSD" Value="true"/>
    <Property Name="NodeColor" Value="green"/>
    <Property Name="SomeProperty" Value="5"/>
  </PlacementProperties>
</NodeType>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters.

#### NOTE

In your Azure Resource Manager template the node type is usually parameterized. It would look like "[parameters('vmNodeType1Name')]" rather than "NodeType01".

```
"nodeTypes": [
  {
    "name": "NodeType01",
    "placementProperties": {
      "HasSSD": "true",
      "NodeColor": "green",
      "SomeProperty": "5"
    }
  }
],
```

You can create service placement *constraints* for a service like as follows:

C#

```
FabricClient fabricClient = new FabricClient();
StatefulServiceDescription serviceDescription = new StatefulServiceDescription();
serviceDescription.PlacementConstraints = "(HasSSD == true && SomeProperty >= 4)";
// add other required servicedescription fields
//...
await fabricClient.ServiceManager.CreateServiceAsync(serviceDescription);
```

Powershell:

```
New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName
$serviceType -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -
PlacementConstraint "HasSSD == true && SomeProperty >= 4"
```

If all nodes of NodeType01 are valid, you can also select that node type with the constraint "(NodeType == NodeType01)".

One of the cool things about a service's placement constraints is that they can be updated dynamically during runtime. So if you need to, you can move a service around in the cluster, add and remove requirements, etc. Service Fabric takes care of ensuring that the service stays up and available even when these types of changes are made.

C#:

```
StatefulServiceUpdateDescription updateDescription = new StatefulServiceUpdateDescription();
updateDescription.PlacementConstraints = "NodeType == NodeType01";
await fabricClient.ServiceManager.UpdateServiceAsync(new Uri("fabric:/app/service"), updateDescription);
```

Powershell:

```
Update-ServiceFabricService -Stateful -ServiceName $serviceName -PlacementConstraints "NodeType == NodeType01"
```

Placement constraints are specified for every different named service instance. Updates always take the place of (overwrite) what was previously specified.

The cluster definition defines the properties on a node. Changing a node's properties requires a cluster configuration upgrade. Upgrading a node's properties requires each affected node to restart to report its new properties. These rolling upgrades are managed by Service Fabric.

## Describing and Managing Cluster Resources

One of the most important jobs of any orchestrator is to help manage resource consumption in the cluster. Managing cluster resources can mean a couple of different things. First, there's ensuring that machines are not overloaded. This means making sure that machines aren't running more services than they can handle. Second, there's balancing and optimization which is critical to running services efficiently. Cost effective or performance sensitive service offerings can't allow some nodes to be hot while others are cold. Hot nodes lead to resource contention and poor performance, and cold nodes represent wasted resources and increased costs.

Service Fabric represents resources as [Metrics](#). Metrics are any logical or physical resource that you want to describe to Service Fabric. Examples of metrics are things like "WorkQueueDepth" or "MemoryInMb". For information about the physical resources that Service Fabric can govern on nodes, see [resource governance](#). For information on configuring custom metrics and their uses, see [this article](#)

Metrics are different from placements constraints and node properties. Node properties are static descriptors of the nodes themselves. Metrics describe resources that nodes have and that services consume when they are run on a node. A node property could be "HasSSD" and could be set to true or false. The amount of space available on that SSD and how much is consumed by services would be a metric like "DriveSpaceInMb".

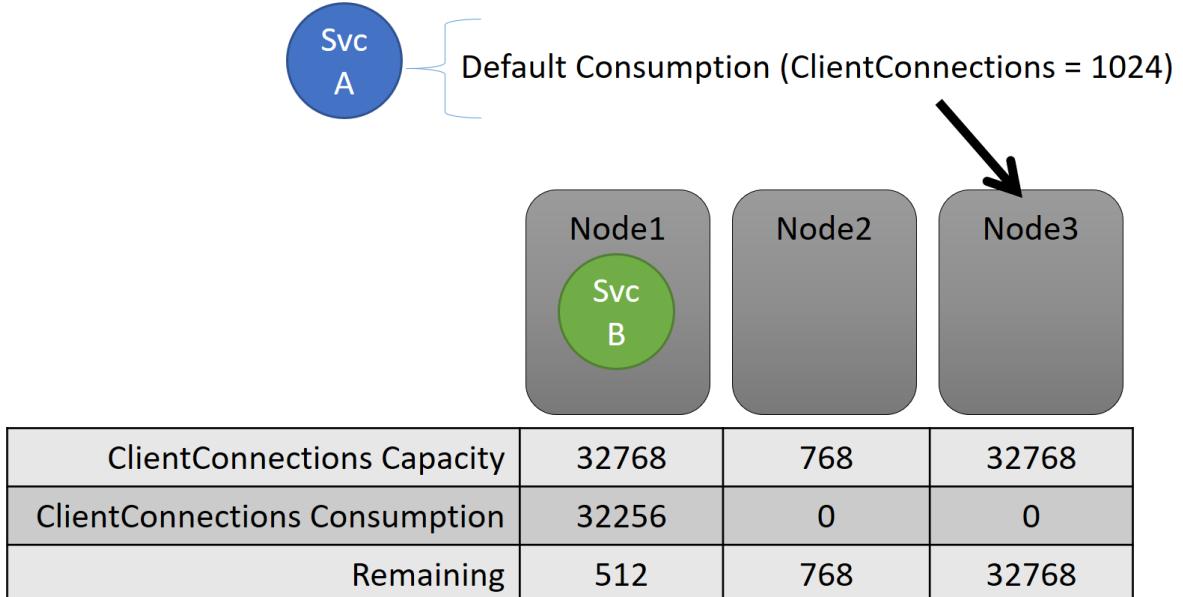
It is important to note that just like for placement constraints and node properties, the Service Fabric Cluster Resource Manager doesn't understand what the names of the metrics mean. Metric names are just strings. It is a good practice to declare units as a part of the metric names that you create when it could be ambiguous.

## Capacity

If you turned off all resource *balancing*, Service Fabric's Cluster Resource Manager would still ensure that no node ended up over its capacity. Managing capacity overruns is possible unless the cluster is too full or the workload is larger than any node. Capacity is another *constraint* that the Cluster Resource Manager uses to understand how much of a resource a node has. Remaining capacity is also tracked for the cluster as a whole. Both the capacity and the consumption at the service level are expressed in terms of metrics. So for example,

the metric might be "ClientConnections" and a given Node may have a capacity for "ClientConnections" of 32768. Other nodes can have other limits Some service running on that node can say it is currently consuming 32256 of the metric "ClientConnections".

During runtime, the Cluster Resource Manager tracks remaining capacity in the cluster and on nodes. In order to track capacity the Cluster Resource Manager subtracts each service's usage from node's capacity where the service runs. With this information, the Service Fabric Cluster Resource Manager can figure out where to place or move replicas so that nodes don't go over capacity.



C#:

```
StatefulServiceDescription serviceDescription = new StatefulServiceDescription();
ServiceLoadMetricDescription metric = new ServiceLoadMetricDescription();
metric.Name = "ClientConnections";
metric.PrimaryDefaultLoad = 1024;
metric.SecondaryDefaultLoad = 0;
metric.Weight = ServiceLoadMetricWeight.High;
serviceDescription.Metrics.Add(metric);
await fabricClient.ServiceManager.CreateServiceAsync(serviceDescription);
```

Powershell:

```
New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName
$serviceName -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -Metric
@("ClientConnections,High,1024,0)
```

You can see capacities defined in the cluster manifest:

ClusterManifest.xml

```
<NodeType Name="NodeType03">
  <Capacities>
    <Capacity Name="ClientConnections" Value="65536"/>
  </Capacities>
</NodeType>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters.

```

"nodeTypes": [
    {
        "name": "NodeType03",
        "capacities": {
            "ClientConnections": "65536",
        }
    }
],

```

Commonly a service's load changes dynamically. Say that a replica's load of "ClientConnections" changed from 1024 to 2048, but the node it was running on then only had 512 capacity remaining for that metric. Now that replica or instance's placement is invalid, since there's not enough room on that node. The Cluster Resource Manager has to kick in and get the node back below capacity. It reduces load on the node that is over capacity by moving one or more of the replicas or instances from that node to other nodes. When moving replicas, the Cluster Resource Manager tries to minimize the cost of those movements. Movement cost is discussed in [this article](#) and more about the Cluster Resource Manager's rebalancing strategies and rules is described [here](#).

## Cluster capacity

So how does the Service Fabric Cluster Resource Manager keep the overall cluster from being too full? Well, with dynamic load there's not a lot it can do. Services can have their load spike independently of actions taken by the Cluster Resource Manager. As a result, your cluster with plenty of headroom today may be underpowered when you become famous tomorrow. That said, there are some controls that are baked in to prevent problems. The first thing we can do is prevent the creation of new workloads that would cause the cluster to become full.

Say that you create a stateless service and it has some load associated with it. Let's say that the service cares about the "DiskSpaceInMb" metric. Let's also say that it is going to consume five units of "DiskSpaceInMb" for every instance of the service. You want to create three instances of the service. Great! So that means that we need 15 units of "DiskSpaceInMb" to be present in the cluster in order for us to even be able to create these service instances. The Cluster Resource Manager continually calculates the capacity and consumption of each metric so it can determine the remaining capacity in the cluster. If there isn't enough space, the Cluster Resource Manager rejects the create service call.

Since the requirement is only that there be 15 units available, this space could be allocated many different ways. For example, there could be one remaining unit of capacity on 15 different nodes, or three remaining units of capacity on five different nodes. If the Cluster Resource Manager can rearrange things so there's five units available on three nodes, it places the service. Rearranging the cluster is usually possible unless the cluster is almost full or the existing services can't be consolidated for some reason.

## Buffered Capacity

Buffered capacity is another feature of the Cluster Resource Manager. It allows reservation of some portion of the overall node capacity. This capacity buffer is only used to place services during upgrades and node failures. Buffered Capacity is specified globally per metric for all nodes. The value you pick for the reserved capacity is a function of the number of Fault and Upgrade Domains you have in the cluster. More Fault and Upgrade Domains means that you can pick a lower number for your buffered capacity. If you have more domains, you can expect smaller amounts of your cluster to be unavailable during upgrades and failures. Specifying Buffered Capacity only makes sense if you have also specified the node capacity for a metric.

Here's an example of how to specify buffered capacity:

ClusterManifest.xml

```

<Section Name="NodeBufferPercentage">
  <Parameter Name="SomeMetric" Value="0.15" />
  <Parameter Name="SomeOtherMetric" Value="0.20" />
</Section>

```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters:

```

"fabricSettings": [
  {
    "name": "NodeBufferPercentage",
    "parameters": [
      {
        "name": "SomeMetric",
        "value": "0.15"
      },
      {
        "name": "SomeOtherMetric",
        "value": "0.20"
      }
    ]
  }
]

```

The creation of new services fails when the cluster is out of buffered capacity for a metric. Preventing the creation of new services to preserve the buffer ensures that upgrades and failures don't cause nodes to go over capacity. Buffered capacity is optional but is recommended in any cluster that defines a capacity for a metric.

The Cluster Resource Manager exposes this load information. For each metric, this information includes:

- the buffered capacity settings
- the total capacity
- the current consumption
- whether each metric is considered balanced or not
- statistics about the standard deviation
- the nodes which have the most and least load

Below we see an example of that output:

```
PS C:\Users\user> Get-ServiceFabricClusterLoadInformation
LastBalancingStartTimeUtc : 9/1/2016 12:54:59 AM
LastBalancingEndTimeUtc  : 9/1/2016 12:54:59 AM
LoadMetricInformation   :
    LoadMetricName      : Metric1
    IsBalancedBefore    : False
    IsBalancedAfter     : False
    DeviationBefore     : 0.192450089729875
    DeviationAfter      : 0.192450089729875
    BalancingThreshold  : 1
    Action              : NoActionNeeded
    ActivityThreshold   : 0
    ClusterCapacity     : 189
    ClusterLoad         : 45
    ClusterRemainingCapacity : 144
    NodeBufferPercentage : 10
    ClusterBufferedCapacity : 170
    ClusterRemainingBufferedCapacity : 125
    ClusterCapacityViolation : False
    MinNodeLoadValue    : 0
    MinNodeLoadNodeId   : 3ea71e8e01f4b0999b121abcbf27d74d
    MaxNodeLoadValue    : 15
    MaxNodeLoadNodeId   : 2cc648b6770be1bc9824fa995d5b68b1
```

## Next steps

- For information on the architecture and information flow within the Cluster Resource Manager, check out [this article](#)
- Defining Defragmentation Metrics is one way to consolidate load on nodes instead of spreading it out. To learn how to configure defragmentation, refer to [this article](#)
- Start from the beginning and [get an Introduction to the Service Fabric Cluster Resource Manager](#)
- To find out about how the Cluster Resource Manager manages and balances load in the cluster, check out the article on [balancing load](#)

# Introduction to Application Groups

8/18/2017 • 7 min to read • [Edit Online](#)

Service Fabric's Cluster Resource Manager typically manages cluster resources by spreading the load (represented via [Metrics](#)) evenly throughout the cluster. Service Fabric manages the capacity of the nodes in the cluster and the cluster as a whole via [capacity](#). Metrics and capacity work great for many workloads, but patterns that make heavy use of different Service Fabric Application Instances sometimes bring in additional requirements. For example you may want to:

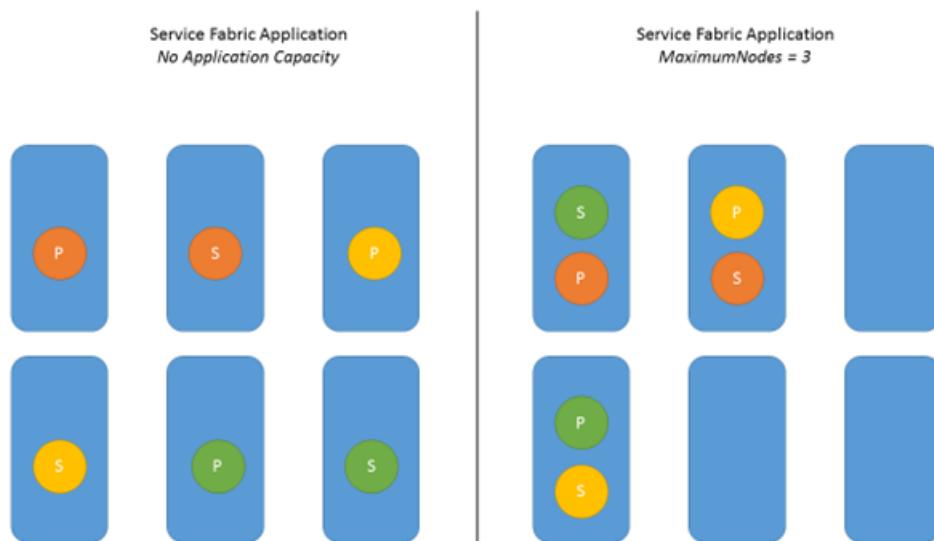
- Reserve some capacity on the nodes in the cluster for the services within some named application instance
- Limit the total number of nodes that the services within a named application instance run on (instead of spreading them out over the entire cluster)
- Define capacities on the named application instance itself to limit the number of services or total resource consumption of the services inside it

To meet these requirements, the Service Fabric Cluster Resource Manager supports a feature called Application Groups.

## Limits the maximum number of nodes

The simplest use case for Application capacity is when an application instance needs to be limited to a certain maximum number of nodes. This consolidates all services within that application instance onto a set number of machines. Consolidation is useful when you're trying to predict or cap physical resource use by the services within that named application instance.

The following image shows an application instance with and without a maximum number of nodes defined:



In the left example, the application doesn't have a maximum number of nodes defined, and it has three services. The Cluster Resource Manager has spread out all replicas across six available nodes to achieve the best balance in the cluster (the default behavior). In the right example, we see the same application limited to three nodes.

The parameter that controls this behavior is called `MaximumNodes`. This parameter can be set during application creation, or updated for an application instance that was already running.

Powershell

```
New-ServiceFabricApplication -ApplicationName fabric:/AppName -ApplicationTypeName AppType1 -  
ApplicationTypeVersion 1.0.0.0 -MaximumNodes 3  
Update-ServiceFabricApplication -Name fabric:/AppName -MaximumNodes 5
```

C#

```
ApplicationDescription ad = new ApplicationDescription();  
ad.ApplicationName = new Uri("fabric:/AppName");  
ad.ApplicationTypeName = "AppType1";  
ad.ApplicationTypeVersion = "1.0.0.0";  
ad.MaximumNodes = 3;  
await fc.ApplicationManager.CreateApplicationAsync(ad);  
  
ApplicationUpdateDescription adUpdate = new ApplicationUpdateDescription(new Uri("fabric:/AppName"));  
adUpdate.MaximumNodes = 5;  
await fc.ApplicationManager.UpdateApplicationAsync(adUpdate);
```

Within the set of nodes, the Cluster Resource Manager doesn't guarantee which service objects get placed together or which nodes get used.

## Application Metrics, Load, and Capacity

Application Groups also allow you to define metrics associated with a given named application instance, and that application instance's capacity for those metrics. Application metrics allow you to track, reserve, and limit the resource consumption of the services inside that application instance.

For each application metric, there are two values that can be set:

- **Total Application Capacity** – This setting represents the total capacity of the application for a particular metric. The Cluster Resource Manager disallows the creation of any new services within this application instance that would cause total load to exceed this value. For example, let's say the application instance had a capacity of 10 and already had load of five. The creation of a service with a total default load of 10 would be disallowed.
- **Maximum Node Capacity** – This setting specifies the maximum total load for the application on a single node. If load goes over this capacity, the Cluster Resource Manager moves replicas to other nodes so that the load decreases.

Powershell:

```
New-ServiceFabricApplication -ApplicationName fabric:/AppName -ApplicationTypeName AppType1 -  
ApplicationTypeVersion 1.0.0.0 -Metrics  
@("MetricName:Metric1,MaximumNodeCapacity:100,MaximumApplicationCapacity:1000")
```

C#:

```
ApplicationDescription ad = new ApplicationDescription();  
ad.ApplicationName = new Uri("fabric:/AppName");  
ad.ApplicationTypeName = "AppType1";  
ad.ApplicationTypeVersion = "1.0.0.0";  
  
var appMetric = new ApplicationMetricDescription();  
appMetric.Name = "Metric1";  
appMetric.TotalApplicationCapacity = 1000;  
appMetric.MaximumNodeCapacity = 100;  
ad.Metrics.Add(appMetric);  
await fc.ApplicationManager.CreateApplicationAsync(ad);
```

# Reserving Capacity

Another common use for application groups is to ensure that resources within the cluster are reserved for a given application instance. The space is always reserved when the application instance is created.

Reserving space in the cluster for the application happens immediately even when:

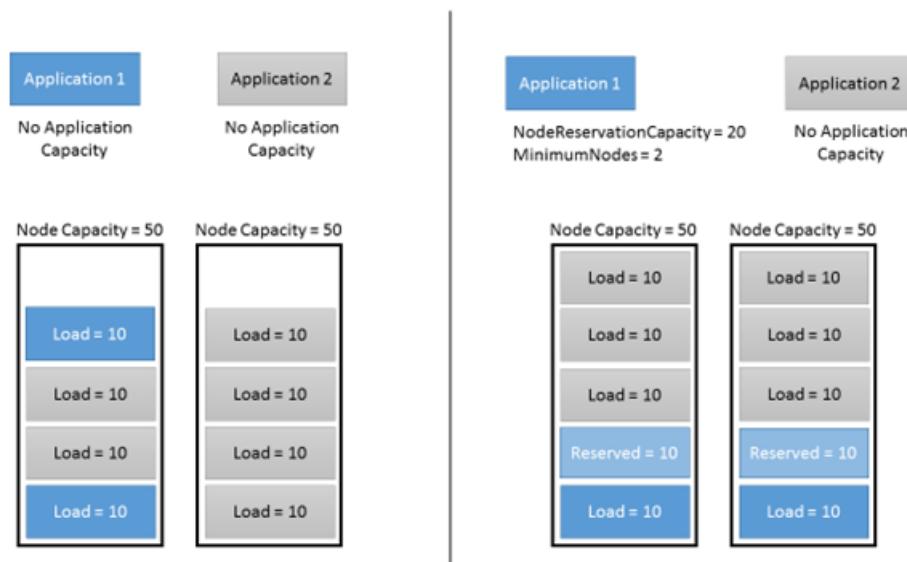
- the application instance is created but doesn't have any services within it yet
- the number of services within the application instance changes every time
- the services exist but aren't consuming the resources

Reserving resources for an application instance requires specifying two additional parameters: *MinimumNodes* and *NodeReservationCapacity*

- **MinimumNodes** - Defines the minimum number of nodes that the application instance should run on.
- **NodeReservationCapacity** - This setting is per metric for the application. The value is the amount of that metric reserved for the application on any node where that the services in that application run.

Combining **MinimumNodes** and **NodeReservationCapacity** guarantees a minimum load reservation for the application within the cluster. If there's less remaining capacity in the cluster than the total reservation required, creation of the application fails.

Let's look at an example of capacity reservation:



In the left example, applications do not have any Application Capacity defined. The Cluster Resource Manager balances everything according to normal rules.

In the example on the right, let's say that Application1 was created with the following settings:

- MinimumNodes set to two
- An application Metric defined with
  - NodeReservationCapacity of 20

Powershell

```
New-ServiceFabricApplication -ApplicationName fabric:/AppName -ApplicationTypeName AppType1 -  
ApplicationTypeVersion 1.0.0.0 -MinimumNodes 2 -Metrics @("MetricName:Metric1,NodeReservationCapacity:20")
```

C#

```

ApplicationDescription ad = new ApplicationDescription();
ad.ApplicationName = new Uri("fabric:/AppName");
ad.ApplicationTypeName = "AppType1";
ad.ApplicationTypeVersion = "1.0.0.0";
ad.MinimumNodes = 2;

var appMetric = new ApplicationMetricDescription();
appMetric.Name = "Metric1";
appMetric.NodeReservationCapacity = 20;

ad.Metrics.Add(appMetric);

await fc.ApplicationManager.CreateApplicationAsync(ad);

```

Service Fabric reserves capacity on two nodes for Application1, and doesn't allow services from Application2 to consume that capacity even if there are no load is being consumed by the services inside Application1 at the time. This reserved application capacity is considered consumed and counts against the remaining capacity on that node and within the cluster. The reservation is deducted from the remaining cluster capacity immediately, however the reserved consumption is deducted from the capacity of a specific node only when at least one service object is placed on it. This later reservation allows for flexibility and better resource utilization since resources are only reserved on nodes when needed.

## Obtaining the application load information

For each application that has an Application Capacity defined for one or more metrics you can obtain the information about the aggregate load reported by replicas of its services.

Powershell:

```
Get-ServiceFabricApplicationLoad -ApplicationName fabric:/MyApplication1
```

C#

```

var v = await fc.QueryManager.GetApplicationLoadInformationAsync("fabric:/MyApplication1");
var metrics = v.ApplicationLoadMetricInformation;
foreach (ApplicationLoadMetricInformation metric in metrics)
{
    Console.WriteLine(metric.ApplicationCapacity); //total capacity for this metric in this application
    instance
    Console.WriteLine(metric.ReservationCapacity); //reserved capacity for this metric in this application
    instance
    Console.WriteLine(metric.ApplicationLoad); //current load for this metric in this application instance
}

```

The ApplicationLoad query returns the basic information about Application Capacity that was specified for the application. This information includes the Minimum Nodes and Maximum Nodes info, and the number that the application is currently occupying. It also includes information about each application load metric, including:

- Metric Name: Name of the metric.
- Reservation Capacity: Cluster Capacity that is reserved in the cluster for this Application.
- Application Load: Total Load of this Application's child replicas.
- Application Capacity: Maximum permitted value of Application Load.

## Removing Application Capacity

Once the Application Capacity parameters are set for an application, they can be removed using Update

Application APIs or PowerShell cmdlets. For example:

```
Update-ServiceFabricApplication -Name fabric:/MyApplication1 -RemoveApplicationCapacity
```

This command removes all Application capacity management parameters from the application instance. This includes MinimumNodes, MaximumNodes, and the Application's metrics, if any. The effect of the command is immediate. After this command completes, the Cluster Resource Manager uses the default behavior for managing applications. Application Capacity parameters can be specified again via `Update-ServiceFabricApplication / System.Fabric.FabricClient.ApplicationManagementClient.UpdateApplicationAsync()`.

### Restrictions on Application Capacity

There are several restrictions on Application Capacity parameters that must be respected. If there are validation errors no changes take place.

- All integer parameters must be non-negative numbers.
- MinimumNodes must never be greater than MaximumNodes.
- If capacities for a load metric are defined, then they must follow these rules:
  - Node Reservation Capacity must not be greater than Maximum Node Capacity. For example, you cannot limit the capacity for the metric "CPU" on the node to two units and try to reserve three units on each node.
  - If MaximumNodes is specified, then the product of MaximumNodes and Maximum Node Capacity must not be greater than Total Application Capacity. For example, let's say the Maximum Node Capacity for load metric "CPU" is set to eight. Let's also say you set the Maximum Nodes to 10. In this case, Total Application Capacity must be greater than 80 for this load metric.

The restrictions are enforced both during application creation and updates.

## How not to use Application Capacity

- Do not try to use the Application Group features to constrain the application to a *specific* subset of nodes. In other words, you can specify that the application runs on at most five nodes, but not which specific five nodes in the cluster. Constraining an application to specific nodes can be achieved using placement constraints for services.
- Do not try to use the Application Capacity to ensure that two services from the same application are placed on the same nodes. Instead use [affinity](#) or [placement constraints](#).

## Next steps

- For more information on configuring services, [Learn about configuring Services](#)
- To find out about how the Cluster Resource Manager manages and balances load in the cluster, check out the article on [balancing load](#)
- Start from the beginning and [get an Introduction to the Service Fabric Cluster Resource Manager](#)
- For more information on how metrics work generally, read up on [Service Fabric Load Metrics](#)
- The Cluster Resource Manager has many options for describing the cluster. To find out more about them, check out this article on [describing a Service Fabric cluster](#)

# Configuring cluster resource manager settings for Service Fabric services

8/18/2017 • 2 min to read • [Edit Online](#)

The Service Fabric Cluster Resource Manager allows fine-grained control over the rules that govern every individual named service. Each named service can specify rules for how it should be allocated in the cluster. Each named service can also define the set of metrics that it wants to report, including how important they are to that service. Configuring services breaks down into three different tasks:

1. Configuring placement constraints
2. Configuring metrics
3. Configuring advanced placement policies and other rules (less common)

## Placement constraints

Placement constraints are used to control which nodes in the cluster a service can actually run on. Typically a particular named service instance or all services of a given type constrained to run on a particular type of node. Placement constraints are extensible. You can define any set of properties per node type, and then select for them with constraints when creating services. You can also change a service's placement constraints while it is running. This allows you to respond to changes in the cluster or the requirements of the service. The properties of a given node can also be updated dynamically in the cluster. More information on placement constraints and how to configure them can be found in [this article](#)

## Metrics

Metrics are the set of resources that a given named service needs. A service's metric configuration includes how much of that resource each stateful replica or stateless instance of that service consumes by default. Metrics also include a weight that indicates how important balancing that metric is to that service, in case tradeoffs are necessary.

## Advanced placement rules

There are other types of placement rules that are useful in less common scenarios. Some examples are:

- Constraints that help with geographically distributed clusters
- Certain application architectures

Other placement rules are configured via either Correlations or Policies.

## Next steps

- Metrics are how the Service Fabric Cluster Resource Manager manages consumption and capacity in the cluster. To learn more about metrics and how to configure them, check out [this article](#)
- Affinity is one mode you can configure for your services. It is not common, but if you need it you can learn about it [here](#)
- There are many different placement rules that can be configured on your service to handle additional scenarios. You can find out about those different placement policies [here](#)
- Start from the beginning and [get an Introduction to the Service Fabric Cluster Resource Manager](#)
- To find out about how the Cluster Resource Manager manages and balances load in the cluster, check out the

article on [balancing load](#)

- The Cluster Resource Manager has many options for describing the cluster. To find out more about them, check out this article on [describing a Service Fabric cluster](#)

# Managing resource consumption and load in Service Fabric with metrics

8/18/2017 • 17 min to read • [Edit Online](#)

*Metrics* are the resources that your services care about and which are provided by the nodes in the cluster. A metric is anything that you want to manage in order to improve or monitor the performance of your services. For example, you might watch memory consumption to know if your service is overloaded. Another use is to figure out whether the service could move elsewhere where memory is less constrained in order to get better performance.

Things like Memory, Disk, and CPU usage are examples of metrics. These metrics are physical metrics, resources that correspond to physical resources on the node that need to be managed. Metrics can also be (and commonly are) logical metrics. Logical metrics are things like "MyWorkQueueDepth" or "MessagesToProcess" or "TotalRecords". Logical metrics are application-defined and indirectly correspond to some physical resource consumption. Logical metrics are common because it can be hard to measure and report consumption of physical resources on a per-service basis. The complexity of measuring and reporting your own physical metrics is also why Service Fabric provides some default metrics.

## Default metrics

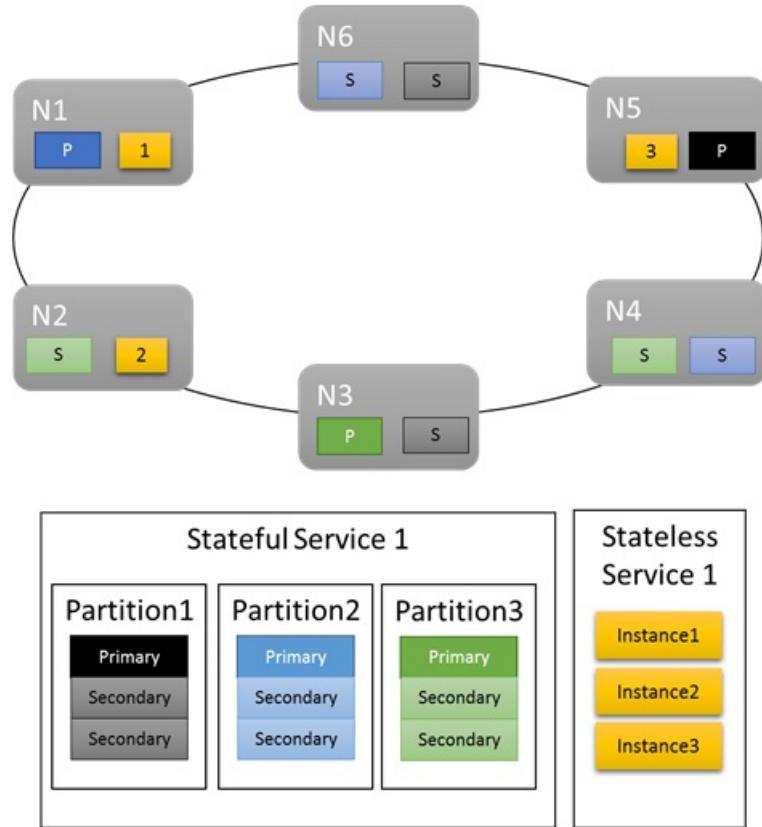
Let's say that you want to get started writing and deploying your service. At this point you don't know what physical or logical resources it consumes. That's fine! The Service Fabric Cluster Resource Manager uses some default metrics when no other metrics are specified. They are:

- PrimaryCount - count of Primary replicas on the node
- ReplicaCount - count of total stateful replicas on the node
- Count - count of all service objects (stateless and stateful) on the node

METRIC	STATELESS INSTANCE LOAD	STATEFUL SECONDARY LOAD	STATEFUL PRIMARY LOAD
PrimaryCount	0	0	1
ReplicaCount	0	1	1
Count	1	1	1

For basic workloads, the default metrics provide a decent distribution of work in the cluster. In the following example, let's see what happens when we create two services and rely on the default metrics for balancing. The first service is a stateful service with three partitions and a target replica set size of three. The second service is a stateless service with one partition and an instance count of three.

Here's what you get:



Some things to note:

- Primary replicas for the stateful service are distributed across several nodes
- Replicas for the same partition are on different nodes
- The total number of primaries and secondaries is distributed in the cluster
- The total number of service objects are evenly allocated on each node

Good!

The default metrics work great as a start. However, the default metrics will only carry you so far. For example: What's the likelihood that the partitioning scheme you picked results in perfectly even utilization by all partitions? What's the chance that the load for a given service is constant over time, or even just the same across multiple partitions right now?

You could run with just the default metrics. However, doing so usually means that your cluster utilization is lower and more uneven than you'd like. This is because the default metrics aren't adaptive and presume everything is equivalent. For example, a Primary that is busy and one that is not both contribute "1" to the PrimaryCount metric. In the worst case, using only the default metrics can also result in overscheduled nodes resulting in performance issues. If you're interested in getting the most out of your cluster and avoiding performance issues, you need to use custom metrics and dynamic load reporting.

## Custom metrics

Metrics are configured on a per-named-service-instance basis when you're creating the service.

Any metric has some properties that describe it: a name, a weight, and a default load.

- **Metric Name:** The name of the metric. The metric name is a unique identifier for the metric within the cluster from the Resource Manager's perspective.
- **Weight:** Metric weight defines how important this metric is relative to the other metrics for this service.
- **Default Load:** The default load is represented differently depending on whether the service is stateless or stateful.

- For stateless services, each metric has a single property named DefaultLoad
- For stateful services you define:
  - PrimaryDefaultLoad: The default amount of this metric this service consumes when it is a Primary
  - SecondaryDefaultLoad: The default amount of this metric this service consumes when it is a Secondary

**NOTE**

If you define custom metrics and you want to *also* use the default metrics, you need to *explicitly* add the default metrics back and define weights and values for them. This is because you must define the relationship between the default metrics and your custom metrics. For example, maybe you care about ConnectionCount or WorkQueueDepth more than Primary distribution. By default the weight of the PrimaryCount metric is High, so you want to reduce it to Medium when you add your other metrics to ensure they take precedence.

### Defining metrics for your service - an example

Let's say you want the following configuration:

- Your service reports a metric named "ConnectionCount"
- You also want to use the default metrics
- You've done some measurements and know that normally a Primary replica of that service takes up 20 units of "ConnectionCount"
- Secondaries use 5 units of "ConnectionCount"
- You know that "ConnectionCount" is the most important metric in terms of managing the performance of this particular service
- You still want Primary replicas balanced. Balancing primary replicas is generally a good idea no matter what. This helps prevent the loss of some node or fault domain from impacting a majority of primary replicas along with it.
- Otherwise, the default metrics are fine

Here's the code that you would write to create a service with that metric configuration:

Code:

```

StatefulServiceDescription serviceDescription = new StatefulServiceDescription();
StatefulServiceLoadMetricDescription connectionMetric = new StatefulServiceLoadMetricDescription();
connectionMetric.Name = "ConnectionCount";
connectionMetric.PrimaryDefaultLoad = 20;
connectionMetric.SecondaryDefaultLoad = 5;
connectionMetric.Weight = ServiceLoadMetricWeight.High;

StatefulServiceLoadMetricDescription primaryCountMetric = new StatefulServiceLoadMetricDescription();
primaryCountMetric.Name = "PrimaryCount";
primaryCountMetric.PrimaryDefaultLoad = 1;
primaryCountMetric.SecondaryDefaultLoad = 0;
primaryCountMetric.Weight = ServiceLoadMetricWeight.Medium;

StatefulServiceLoadMetricDescription replicaCountMetric = new StatefulServiceLoadMetricDescription();
replicaCountMetric.Name = "ReplicaCount";
replicaCountMetric.PrimaryDefaultLoad = 1;
replicaCountMetric.SecondaryDefaultLoad = 1;
replicaCountMetric.Weight = ServiceLoadMetricWeight.Low;

StatefulServiceLoadMetricDescription totalCountMetric = new StatefulServiceLoadMetricDescription();
totalCountMetric.Name = "Count";
totalCountMetric.PrimaryDefaultLoad = 1;
totalCountMetric.SecondaryDefaultLoad = 1;
totalCountMetric.Weight = ServiceLoadMetricWeight.Low;

serviceDescription.Metrics.Add(connectionMetric);
serviceDescription.Metrics.Add(primaryCountMetric);
serviceDescription.Metrics.Add(replicaCountMetric);
serviceDescription.Metrics.Add(totalCountMetric);

await fabricClient.ServiceManager.CreateServiceAsync(serviceDescription);

```

Powershell:

```

New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName
$serviceName -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -
Metric @("ConnectionCount,High,20,5","PrimaryCount,Medium,1,0","ReplicaCount,Low,1,1","Count,Low,1,1")

```

#### **NOTE**

The above examples and the rest of this document describe managing metrics on a per-named-service basis. It is also possible to define metrics for your services at the service *type* level. This is accomplished by specifying them in your service manifests. Defining type level metrics is not recommended for several reasons. The first reason is that metric names are frequently environment-specific. Unless there is a firm contract in place, you cannot be sure that the metric "Cores" in one environment isn't "MiliCores" or "CoReS" in others. If your metrics are defined in your manifest you need to create new manifests per environment. This usually leads to a proliferation of different manifests with only minor differences, which can lead to management difficulties.

Metric loads are commonly assigned on a per-named-service-instance basis. For example, let's say you create one instance of the service for CustomerA who plans to use it only lightly. Let's also say you create another for CustomerB who has a larger workload. In this case, you'd probably want to tweak the default loads for those services. If you have metrics and loads defined via manifests and you want to support this scenario, it requires different application and service types for each customer. The values defined at service creation time override those defined in the manifest, so you could use that to set the specific defaults. However, doing that causes the values declared in the manifests to not match those the service actually runs with. This can lead to confusion.

As a reminder: if you just want to use the default metrics, you don't need to touch the metrics collection at all or do anything special when creating your service. The default metrics get used automatically when no others are defined.

Now, let's go through each of these settings in more detail and talk about the behavior that it influences.

## Load

The whole point of defining metrics is to represent some load. *Load* is how much of a given metric is consumed by some service instance or replica on a given node. Load can be configured at almost any point. For example:

- Load can be defined when a service is created. This is called *default load*.
- The metric information, including default loads, for a service can be updated after the service is created. This is called *updating a service*.
- The loads for a given partition can be reset to the default values for that service. This is called *resetting partition load*.
- Load can be reported on a per service object basis dynamically during runtime. This is called *reporting load*.

All of these strategies can be used within the same service over its lifetime.

## Default load

*Default load* is how much of the metric each service object (stateless instance or stateful replica) of this service consumes. The Cluster Resource Manager uses this number for the load of the service object until it receives other information, such as a dynamic load report. For simpler services, the default load is a static definition. The default load is never updated and is used for the lifetime of the service. Default loads work great for simple capacity planning scenarios where certain amounts of resources are dedicated to different workloads and do not change.

### NOTE

For more information on capacity management and defining capacities for the nodes in your cluster, please see [this article](#).

The Cluster Resource Manager allows stateful services to specify a different default load for their Primaries and Secondaries. Stateless services can only specify one value that applies to all instances. For stateful services, the default load for Primary and Secondary replicas are typically different since replicas do different kinds of work in each role. For example, Primaries usually serve both reads and writes, and handle most of the computational burden, while secondaries do not. Usually the default load for a primary replica is higher than the default load for secondary replicas. The real numbers should depend on your own measurements.

## Dynamic load

Let's say that you've been running your service for a while. With some monitoring, you've noticed that:

1. Some partitions or instances of a given service consume more resources than others
2. Some services have load that varies over time.

There's lots of things that could cause these types of load fluctuations. For example, different services or partitions are associated with different customers with different requirements. Load could also change because the amount of work the service does varies over the course of the day. Regardless of the reason, there's usually no single number that you can use for default. This is especially true if you want to get the most utilization out of the cluster. Any value you pick for default load is wrong some of the time. Incorrect default loads result in the Cluster Resource Manager either over or under allocating resources. As a result, you have nodes that are over or under utilized even though the Cluster Resource Manager thinks the cluster

is balanced. Default loads are still good since they provide some information for initial placement, but they're not a complete story for real workloads. To accurately capture changing resource requirements, the Cluster Resource Manager allows each service object to update its own load during runtime. This is called dynamic load reporting.

Dynamic load reports allow replicas or instances to adjust their allocation/reported load of metrics over their lifetime. A service replica or instance that was cold and not doing any work would usually report that it was using low amounts of a given metric. A busy replica or instance would report that they are using more.

Reporting load per replica or instance allows the Cluster Resource Manager to reorganize the individual service objects in the cluster. Reorganizing the services helps ensure that they get the resources they require. Busy services effectively get to "reclaim" resources from other replicas or instances that are currently cold or doing less work.

Within Reliable Services, the code to report load dynamically looks like this:

Code:

```
this.Partition.ReportLoad(new List<LoadMetric> { new LoadMetric("CurrentConnectionCount", 1234), new LoadMetric("metric1", 42) });
```

A service can report on any of the metrics defined for it at creation time. If a service reports load for a metric that it is not configured to use, Service Fabric ignores that report. If there are other metrics reported at the same time that are valid, those reports are accepted. Service code can measure and report all the metrics it knows how to, and operators can specify the metric configuration to use without having to change the service code.

### Updating a service's metric configuration

The list of metrics associated with the service, and the properties of those metrics can be updated dynamically while the service is live. This allows for experimentation and flexibility. Some examples of when this is useful are:

- disabling a metric with a buggy report for a particular service
- reconfiguring the weights of metrics based on desired behavior
- enabling a new metric only after the code has already been deployed and validated via other mechanisms
- changing the default load for a service based on observed behavior and consumption

The main APIs for changing metric configuration are

`FabricClient.ServiceManagementClient.UpdateServiceAsync` in C# and `Update-ServiceFabricService` in PowerShell. Whatever information you specify with these APIs replaces the existing metric information for the service immediately.

## Mixing default load values and dynamic load reports

Default load and dynamic loads can be used for the same service. When a service utilizes both default load and dynamic load reports, default load serves as an estimate until dynamic reports show up. Default load is good because it gives the Cluster Resource Manager something to work with. The default load allows the Cluster Resource Manager to place the service objects in good locations when they are created. If no default load information is provided, placement of services is effectively random. When load reports arrive later the initial random placement is often wrong and the Cluster Resource Manager has to move services.

Let's take our previous example and see what happens when we add some custom metrics and dynamic load reporting. In this example, we use "MemoryInMb" as an example metric.

### NOTE

Memory is one of the system metrics that Service Fabric can [resource govern](#), and reporting it yourself is typically difficult. We don't actually expect you to report on Memory consumption; Memory is used here as an aid to learning about the capabilities of the Cluster Resource Manager.

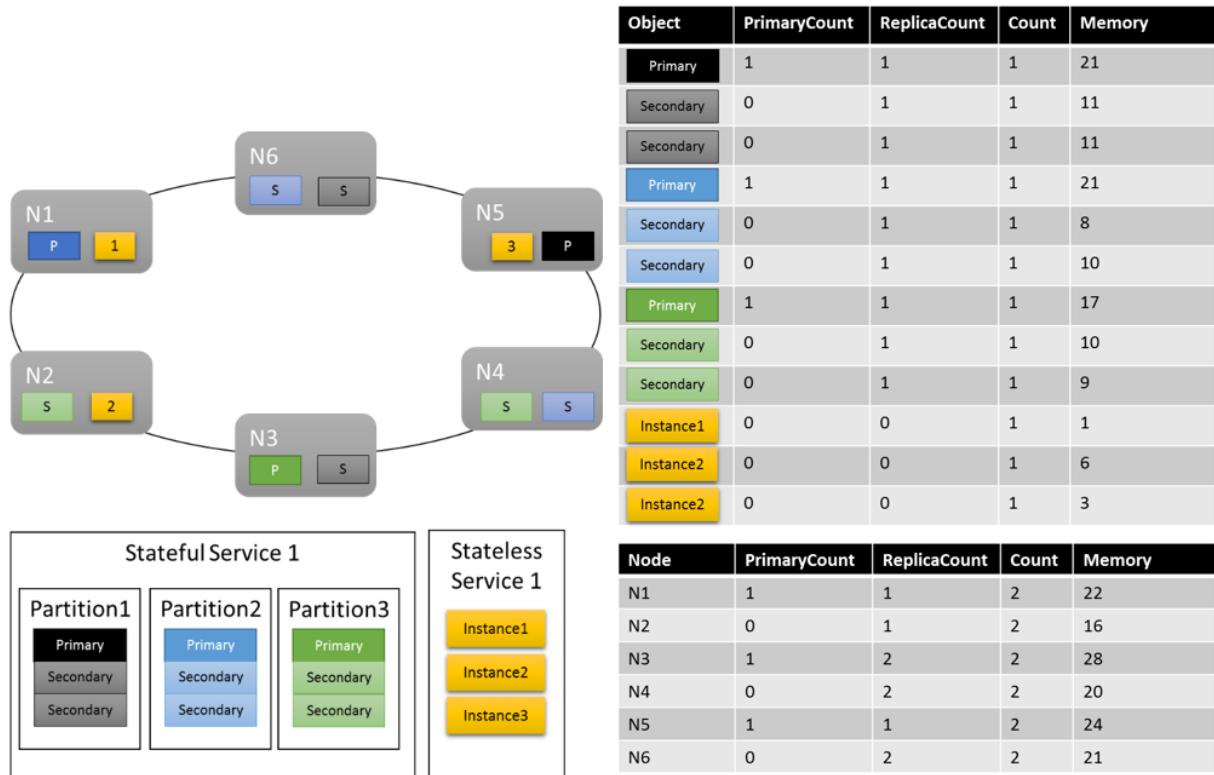
Let's presume that we initially created the stateful service with the following command:

Powershell:

```
New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName  
$serviceName -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -  
Metric @("MemoryInMb,High,21,11","PrimaryCount,Medium,1,0","ReplicaCount,Low,1,1","Count,Low,1,1")
```

As a reminder, this syntax is ("MetricName, MetricWeight, PrimaryDefaultLoad, SecondaryDefaultLoad").

Let's see what one possible cluster layout could look like:



Some things that are worth noting:

- Secondary replicas within a partition can each have their own load
- Overall the metrics look balanced. For Memory, the ratio between the maximum and minimum load is 1.75 (the node with the most load is N3, the least is N2, and  $28/16 = 1.75$ ).

There are some things that we still need to explain:

- What determined whether a ratio of 1.75 was reasonable or not? How does the Cluster Resource Manager know if that's good enough or if there is more work to do?
- When does balancing happen?
- What does it mean that Memory was weighted "High"?

## Metric weights

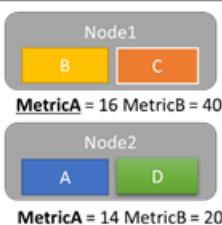
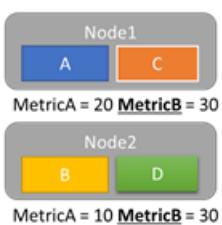
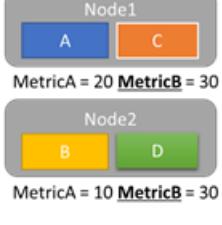
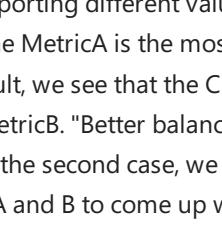
Tracking the same metrics across different services is important. That global view is what allows the Cluster

Resource Manager to track consumption in the cluster, balance consumption across nodes, and ensure that nodes don't go over capacity. However, services may have different views as to the importance of the same metric. Also, in a cluster with many metrics and lots of services, perfectly balanced solutions may not exist for all metrics. How should the Cluster Resource Manager handle these situations?

Metric weights allow the Cluster Resource Manager to decide how to balance the cluster when there's no perfect answer. Metric weights also allow the Cluster Resource Manager to balance specific services differently. Metrics can have four different weight levels: Zero, Low, Medium, and High. A metric with a weight of Zero contributes nothing when considering whether things are balanced or not. However, its load does still contribute to capacity management. Metrics with Zero weight are still useful and are frequently used as a part of service behavior and performance monitoring. [This article](#) provides more information on the use of metrics for monitoring and diagnostics of your services.

The real impact of different metric weights in the cluster is that the Cluster Resource Manager generates different solutions. Metric weights tell the Cluster Resource Manager that certain metrics are more important than others. When there's no perfect solution the Cluster Resource Manager can prefer solutions which balance the higher weighted metrics better. If a service thinks a particular metric is unimportant, it may find their use of that metric imbalanced. This allows another service to get an even distribution of some metric that is important to it.

Let's look at an example of some load reports and how different metric weights results in different allocations in the cluster. In this example, we see that switching the relative weight of the metrics causes the Cluster Resource Manager to create different arrangements of services.

Services & Load	Metric Weight Selection	Placement & Total Load	Results
<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;"> <b>ServiceA</b>    <math>M_A = 5</math>  <math>M_B = 10</math> </div> <div style="text-align: center;">   <b>ServiceB</b>  <math>M_A = 1</math>  <math>M_B = 20</math> </div> </div>	<p>MetricA Weight = "HIGH"  MetricB Weight = "LOW"</p>	 <b>MetricA = 16 MetricB = 40</b>   <b>MetricA = 14 MetricB = 20</b>	<p><u>MetricA more balanced</u>  RatioA = 16/14 = 1.067  RatioB = 40/20 = 2</p>
<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;">   <b>ServiceC</b>  <math>M_A = 15</math>  <math>M_B = 20</math> </div> <div style="text-align: center;">   <b>ServiceD</b>  <math>M_A = 9</math>  <math>M_B = 10</math> </div> </div>	<p>MetricA Weight = "LOW"  MetricB Weight = "HIGH"</p>	 <b>MetricA = 20 MetricB = 30</b>   <b>MetricA = 10 MetricB = 30</b>	<p><u>MetricB more balanced</u>  RatioA = 20/10 = 2  RatioB = 30/30 = 1</p>

In this example, there are four different services, all reporting different values for two different metrics, MetricA and MetricB. In one case, all the services define MetricA as the most important one (Weight = High) and MetricB as unimportant (Weight = Low). As a result, we see that the Cluster Resource Manager places the services so that MetricA is better balanced than MetricB. "Better balanced" means that MetricA has a lower standard deviation than MetricB. In the second case, we reverse the metric weights. As a result, the Cluster Resource Manager swaps services A and B to come up with an allocation where MetricB is better balanced than MetricA.

#### NOTE

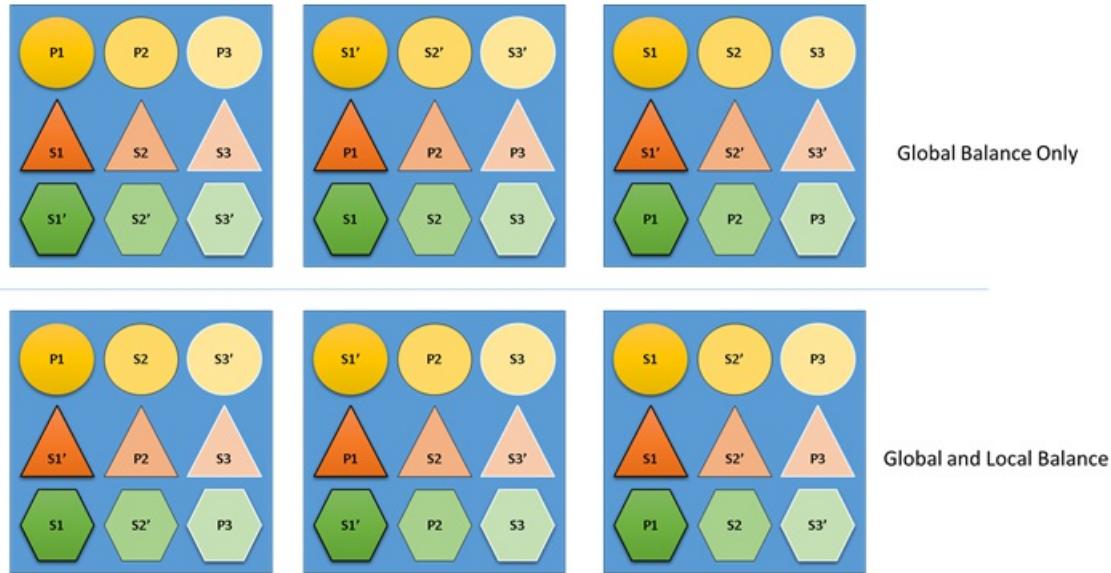
Metric weights determine how the Cluster Resource Manager should balance, but not when balancing should happen. For more information on balancing, check out [this article](#)

#### Global metric weights

Let's say ServiceA defines MetricA as weight High, and ServiceB sets the weight for MetricA to Low or Zero. What's the actual weight that ends up getting used?

There are multiple weights that are tracked for every metric. The first weight is the one defined for the metric when the service is created. The other weight is a global weight, which is computed automatically. The Cluster Resource Manager uses both these weights when scoring solutions. Taking both weights into account is important. This allows the Cluster Resource Manager to balance each service according to its own priorities, and also ensure that the cluster as a whole is allocated correctly.

What would happen if the Cluster Resource Manager didn't care about both global and local balance? Well, it's easy to construct solutions that are globally balanced, but which result in poor resource balance for individual services. In the following example, let's look at a service configured with just the default metrics, and see what happens when only global balance is considered:



In the top example based only on global balance, the cluster as a whole is indeed balanced. All nodes have the same count of primaries and the same number total replicas. However, if you look at the actual impact of this allocation it's not so good: the loss of any node impacts a particular workload disproportionately, because it takes out all of its primaries. For example, if the first node fails the three primaries for the three different partitions of the Circle service would all be lost. Conversely, the Triangle and Hexagon services have their partitions lose a replica. This causes no disruption, other than having to recover the down replica.

In the bottom example, the Cluster Resource Manager has distributed the replicas based on both the global and per-service balance. When calculating the score of the solution it gives most of the weight to the global solution, and a (configurable) portion to individual services. Global balance for a metric is calculated based on the average of the metric weights from each service. Each service is balanced according to its own defined metric weights. This ensures that the services are balanced within themselves according to their own needs. As a result, if the same first node fails the failure is distributed across all partitions of all services. The impact to each is the same.

## Next steps

- For more information on configuring services, [Learn about configuring Services](#)(service-fabric-cluster-resource-manager-configure-services.md)
- Defining Defragmentation Metrics is one way to consolidate load on nodes instead of spreading it out. To learn how to configure defragmentation, refer to [this article](#)
- To find out about how the Cluster Resource Manager manages and balances load in the cluster, check out the article on [balancing load](#)
- Start from the beginning and [get an Introduction to the Service Fabric Cluster Resource Manager](#)
- Movement Cost is one way of signaling to the Cluster Resource Manager that certain services are more expensive to move than others. To learn more about movement cost, refer to [this article](#)

# Configuring and using service affinity in Service Fabric

8/18/2017 • 5 min to read • [Edit Online](#)

Affinity is a control that is provided mainly to help ease the transition of larger monolithic applications into the cloud and microservices world. It is also used as an optimization for improving the performance of services, although doing so can have side effects.

Let's say you're bringing a larger app, or one that just wasn't designed with microservices in mind, to Service Fabric (or any distributed environment). This type of transition is common. You start by lifting the entire app into the environment, packaging it, and making sure it is running smoothly. Then you start breaking it down into different smaller services that all talk to each other.

Eventually you may find that the application is experiencing some issues. The issues usually fall into one of these categories:

1. Some component X in the monolithic app had an undocumented dependency on component Y, and you just turned those components into separate services. Since these services are now running on different nodes in the cluster, they're broken.
2. These components communicate via (local named pipes | shared memory | files on disk) and they really need to be able to write to a shared local resource for performance reasons right now. That hard dependency gets removed later, maybe.
3. Everything is fine, but it turns out that these two components are actually chatty/performance sensitive. When they moved them into separate services overall application performance tanked or latency increased. As a result, the overall application is not meeting expectations.

In these cases, we don't want to lose our refactoring work, and don't want to go back to the monolith. The last condition may even be desirable as a plain optimization. However, until we can redesign the components to work naturally as services (or until we can solve the performance expectations some other way) we're going to need some sense of locality.

What to do? Well, you could try turning on affinity.

## How to configure affinity

To set up affinity, you define an affinity relationship between two different services. You can think of affinity as "pointing" one service at another and saying "This service can only run where that service is running." Sometimes we refer to affinity as a parent/child relationship (where you point the child at the parent). Affinity ensures that the replicas or instances of one service are placed on the same nodes as those of another service.

```
ServiceCorrelationDescription affinityDescription = new ServiceCorrelationDescription();
affinityDescription.Scheme = ServiceCorrelationScheme.Affinity;
affinityDescription.ServiceName = new Uri("fabric:/otherApplication/parentService");
serviceDescription.Correlations.Add(affinityDescription);
await fabricClient.ServiceManager.CreateServiceAsync(serviceDescription);
```

#### NOTE

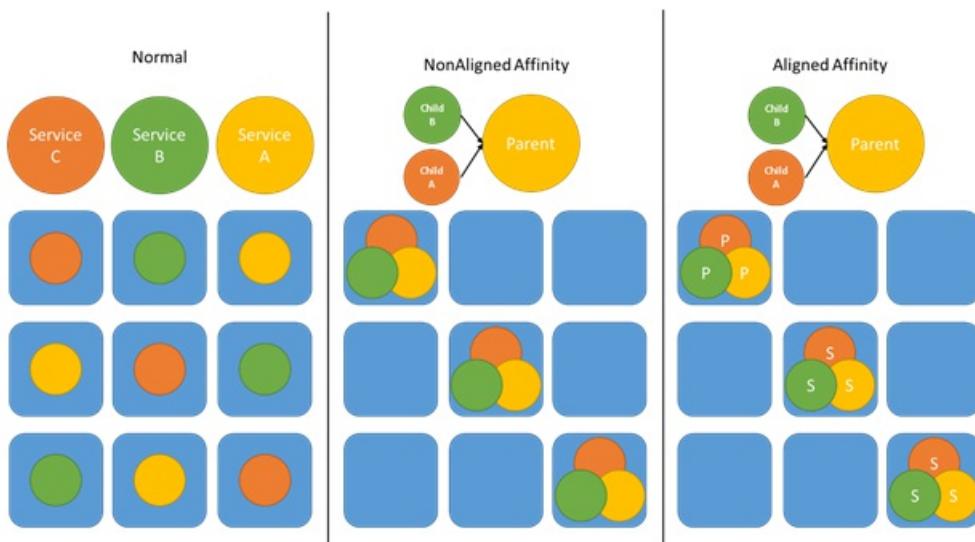
A child service can only participate in a single affinity relationship. If you wanted the child to be affinitized to two parent services at once you have a couple options:

- Reverse the relationships (have parentService1 and parentService2 point at the current child service), or
- Designate one of the parents as a hub by convention and have all services point at that service.

The resulting placement behavior in the cluster should be the same.

## Different affinity options

Affinity is represented via one of several correlation schemes, and has two different modes. The most common mode of affinity is what we call NonAlignedAffinity. In NonAlignedAffinity, the replicas or instances of the different services are placed on the same nodes. The other mode is AlignedAffinity. Aligned Affinity is useful only with stateful services. Configuring two stateful services to have aligned affinity ensures that the primaries of those services are placed on the same nodes as each other. It also causes each pair of secondaries for those services to be placed on the same nodes. It is also possible (though less common) to configure NonAlignedAffinity for stateful services. For NonAlignedAffinity, the different replicas of the two stateful services would run on the same nodes, but their primaries could end up on different nodes.

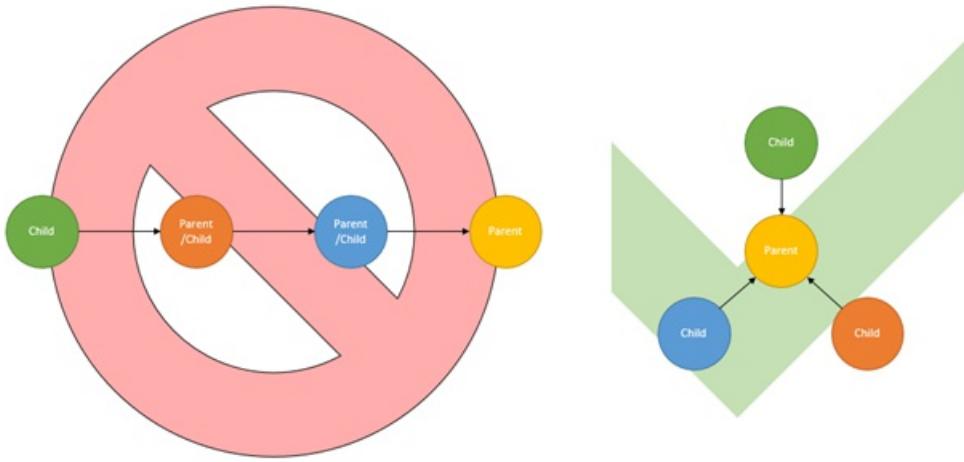


### Best effort desired state

An affinity relationship is best effort. It does not provide the same guarantees of collocation or reliability that running in the same executable process does. The services in an affinity relationship are fundamentally different entities that can fail and be moved independently. An affinity relationship could also break, though these breaks are temporary. For example, capacity limitations may mean that only some of the service objects in the affinity relationship can fit on a given node. In these cases even though there's an affinity relationship in place, it can't be enforced due to the other constraints. If it is possible to do so, the violation is automatically corrected later.

### Chains vs. stars

Today the Cluster Resource Manager isn't able to model chains of affinity relationships. What this means is that a service that is a child in one affinity relationship can't be a parent in another affinity relationship. If you want to model this type of relationship, you effectively have to model it as a star, rather than a chain. To move from a chain to a star, the bottommost child would be parented to the first child's parent instead. Depending on the arrangement of your services, you may have to do this multiple times. If there's no natural parent service, you may have to create one that serves as a placeholder. Depending on your requirements, you may also want to look into [Application Groups](#).



Another thing to note about affinity relationships today is that they are directional. This means that the affinity rule only enforces that the child placed with the parent. It does not ensure that the parent is located with the child. It is also important to note that the affinity relationship can't be perfect or instantly enforced since different services have with different lifecycles and can fail and move independently. For example, let's say the parent suddenly fails over to another node because it crashed. The Cluster Resource Manager and Failover Manager handle the failover first, since keeping the services up, consistent, and available is the priority. Once the failover completes, the affinity relationship is broken, but the Cluster Resource Manager thinks everything is fine until it notices that the child is not located with the parent. These sorts of checks are performed periodically. More information on how the Cluster Resource Manager evaluates constraints is available in [this article](#), and [this one](#) talks more about how to configure the cadence on which these constraints are evaluated.

### **Partitioning support**

The final thing to notice about affinity is that affinity relationships aren't supported where the parent is partitioned. Partitioned parent services may be supported eventually, but today it is not allowed.

## Next steps

- For more information on configuring services, [Learn about configuring Services](#)
- To limit services to a small set of machines or aggregating the load of services, use [Application Groups](#)

# Placement policies for service fabric services

8/18/2017 • 5 min to read • [Edit Online](#)

Placement policies are additional rules that can be used to govern service placement in some specific, less-common scenarios. Some examples of those scenarios are:

- Your Service Fabric cluster spans geographic distances, such as multiple on-premises datacenters or across Azure regions
- Your environment spans multiple areas of geopolitical or legal control, or some other case where you have policy boundaries you need to enforce
- There are communication performance or latency considerations due to large distances or use of slower or less reliable network links
- You need to keep certain workloads collocated as a best effort, either with other workloads or in proximity to customers

Most of these requirements align with the physical layout of the cluster, represented as the fault domains of the cluster.

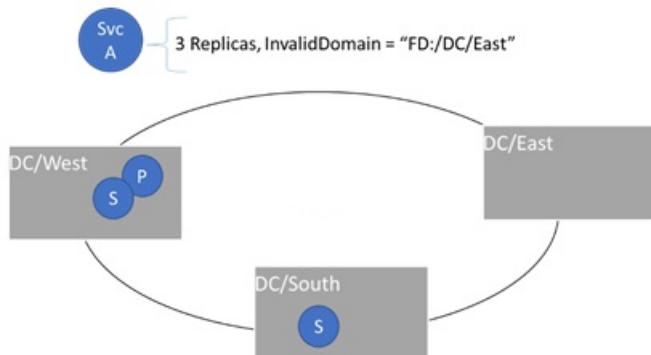
The advanced placement policies that help address these scenarios are:

1. Invalid domains
2. Required domains
3. Preferred domains
4. Disallowing replica packing

Most of the following controls could be configured via node properties and placement constraints, but some are more complicated. To make things simpler, the Service Fabric Cluster Resource Manager provides these additional placement policies. Placement policies are configured on a per-named service instance basis. They can also be updated dynamically.

## Specifying invalid domains

The **InvalidDomain** placement policy allows you to specify that a particular Fault Domain is invalid for a specific service. This policy ensures that a particular service never runs in a particular area, for example for geopolitical or corporate policy reasons. Multiple invalid domains may be specified via separate policies.



Code:

```

ServicePlacementInvalidDomainPolicyDescription invalidDomain = new
ServicePlacementInvalidDomainPolicyDescription();
invalidDomain.DomainName = "fd:/DCEast"; //regulations prohibit this workload here
serviceDescription.PlacementPolicies.Add(invalidDomain);

```

Powershell:

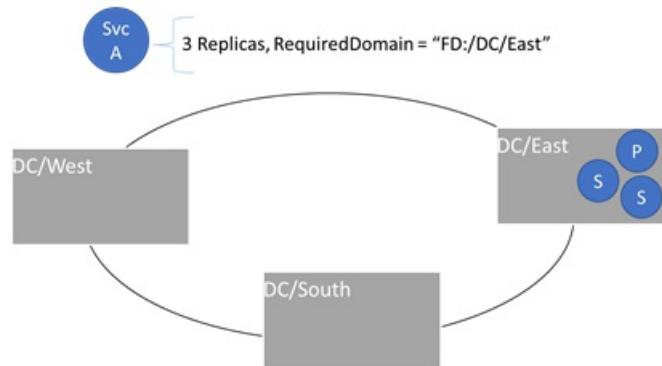
```

New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName
$serviceTypeName -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -
PlacementPolicy @("InvalidDomain,fd:/DCEast")

```

## Specifying required domains

The required domain placement policy requires that the service is present only in the specified domain. Multiple required domains can be specified via separate policies.



Code:

```

ServicePlacementRequiredDomainPolicyDescription requiredDomain = new
ServicePlacementRequiredDomainPolicyDescription();
requiredDomain.DomainName = "fd:/DC01/RK03/BL2";
serviceDescription.PlacementPolicies.Add(requiredDomain);

```

Powershell:

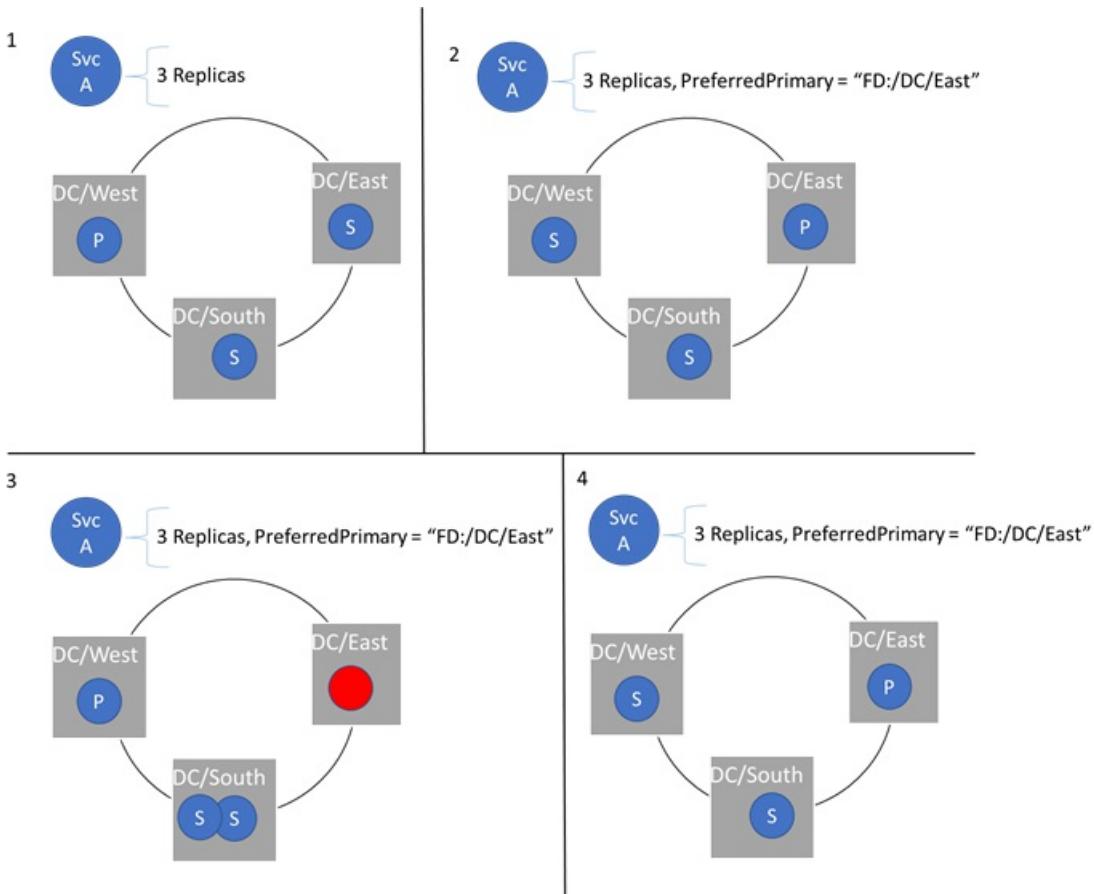
```

New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName
$serviceTypeName -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -
PlacementPolicy @("RequiredDomain,fd:/DC01/RK03/BL2")

```

## Specifying a preferred domain for the primary replicas of a stateful service

The Preferred Primary Domain specifies the fault domain to place the Primary in. The Primary ends up in this domain when everything is healthy. If the domain or the Primary replica fails or shuts down, the Primary moves to some other location, ideally in the same domain. If this new location isn't in the preferred domain, the Cluster Resource Manager moves it back to the preferred domain as soon as possible. Naturally this setting only makes sense for stateful services. This policy is most useful in clusters that are spanned across Azure regions or multiple datacenters but have services that prefer placement in a certain location. Keeping Primaries close to their users or other services helps provide lower latency, especially for reads, which are handled by Primaries by default.



```
ServicePlacementPreferPrimaryDomainPolicyDescription primaryDomain = new
ServicePlacementPreferPrimaryDomainPolicyDescription();
primaryDomain.DomainName = "fd:/EastUS/";
serviceDescription.PlacementPolicies.Add(invalidDomain);
```

Powershell:

```
New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName
$serviceName -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -
PlacementPolicy @("PreferredPrimaryDomain,fd:/EastUS")
```

## Requiring replica distribution and disallowing packing

Replicas are *normally* distributed across fault and upgrade domains when the cluster is healthy. However, there are cases where more than one replica for a given partition may end up temporarily packed into a single domain. For example, let's say that the cluster has nine nodes in three fault domains, fd:/0, fd:/1, and fd:/2. Let's also say that your service has three replicas. Let's say that the nodes that were being used for those replicas in fd:/1 and fd:/2 went down. Normally the Cluster Resource Manager would prefer other nodes in those same fault domains. In this case, let's say due to capacity issues none of the other nodes in those domains were valid. If the Cluster Resource Manager builds replacements for those replicas, it would have to choose nodes in fd:/0. However, doing *that* creates a situation where the Fault Domain constraint is violated. Packing replicas increases the chance that the whole replica set could go down or be lost.

### NOTE

For more information on constraints and constraint priorities generally, check out [this topic](#).

If you've ever seen a health message such as "

```
The Load Balancer has detected a Constraint Violation for this Replica:fabric:<some service name> Secondary Partition <some partition ID> is violating the Constraint: FaultDomain
```

", then you've hit this condition or something like it. Usually only one or two replicas are packed together temporarily. So long as there are fewer than a quorum of replicas in a given domain, you're safe. Packing is rare, but it can happen, and usually these situations are transient since the nodes come back. If the nodes do stay down and the Cluster Resource Manager needs to build replacements, usually there are other nodes available in the ideal fault domains.

Some workloads would prefer always having the target number of replicas, even if they are packed into fewer domains. These workloads are betting against total simultaneous permanent domain failures and can usually recover local state. Other workloads would rather take the downtime earlier than risk correctness or loss of data. Most production workloads run with more than three replicas, more than three fault domains, and many valid nodes per fault domain. Because of this, the default behavior allows domain packing by default. The default behavior allows normal balancing and failover to handle these extreme cases, even if that means temporary domain packing.

If you want to disable such packing for a given workload, you can specify the `RequireDomainDistribution` policy on the service. When this policy is set, the Cluster Resource Manager ensures no two replicas from the same partition run in the same fault or upgrade domain.

Code:

```
ServicePlacementRequireDomainDistributionPolicyDescription distributeDomain = new ServicePlacementRequireDomainDistributionPolicyDescription();
serviceDescription.PlacementPolicies.Add(distributeDomain);
```

Powershell:

```
New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName $serviceTypeName -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -PlacementPolicy @("RequiredDomainDistribution")
```

Now, would it be possible to use these configurations for services in a cluster that was not geographically spanned? You could, but there's not a great reason too. The required, invalid, and preferred domain configurations should be avoided unless the scenarios require them. It doesn't make any sense to try to force a given workload to run in a single rack, or to prefer some segment of your local cluster over another. Different hardware configurations should be spread across fault domains and handled via normal placement constraints and node properties.

## Next steps

- For more information on configuring services, [Learn about configuring Services](#)

# Cluster resource manager integration with Service Fabric cluster management

8/18/2017 • 11 min to read • [Edit Online](#)

The Service Fabric Cluster Resource Manager doesn't drive upgrades in Service Fabric, but it is involved. The first way that the Cluster Resource Manager helps with management is by tracking the desired state of the cluster and the services inside it. The Cluster Resource Manager sends out health reports when it cannot put the cluster into the desired configuration. For example, if there is insufficient capacity the Cluster Resource Manager sends out health warnings and errors indicating the problem. Another piece of integration has to do with how upgrades work. The Cluster Resource Manager alters its behavior slightly during upgrades.

## Health integration

The Cluster Resource Manager constantly tracks the rules you have defined for placing your services. It also tracks the remaining capacity for each metric on the nodes and in the cluster and in the cluster as a whole. If it can't satisfy those rules or if there is insufficient capacity, health warnings and errors are emitted. For example, if a node is over capacity and the Cluster Resource Manager will try to fix the situation by moving services. If it can't correct the situation it emits a health warning indicating which node is over capacity, and for which metrics.

Another example of the Resource Manager's health warnings is violations of placement constraints. For example, if you have defined a placement constraint (such as `“NodeColor == Blue”`) and the Resource Manager detects a violation of that constraint, it emits a health warning. This is true for custom constraints and the default constraints (like the Fault Domain and Upgrade Domain constraints).

Here's an example of one such health report. In this case, the health report is for one of the system service's partitions. The health message indicates the replicas of that partition are temporarily packed into too few Upgrade Domains.

```

PS C:\Users\User > Get-WindowsFabricPartitionHealth -PartitionId '00000000-0000-0000-0000-000000000001'

PartitionId          : 00000000-0000-0000-0000-000000000001
AggregatedHealthState : Warning
UnhealthyEvaluations :
    Unhealthy event: SourceId='System.PLB',
Property='ReplicaConstraintViolation_UpgradeDomain', HealthState='Warning', ConsiderWarningAsError=false.

ReplicaHealthStates   :
    ReplicaId          : 130766528804733380
    AggregatedHealthState : Ok

    ReplicaId          : 130766528804577821
    AggregatedHealthState : Ok

    ReplicaId          : 130766528854889931
    AggregatedHealthState : Ok

    ReplicaId          : 130766528804577822
    AggregatedHealthState : Ok

    ReplicaId          : 130837073190680024
    AggregatedHealthState : Ok

HealthEvents          :
    SourceId           : System.PLB
    Property            : ReplicaConstraintViolation_UpgradeDomain
    HealthState         : Warning
    SequenceNumber      : 130837100116930204
    SentAt              : 8/10/2015 7:53:31 PM
    ReceivedAt          : 8/10/2015 7:53:33 PM
    TTL                 : 00:01:05
    Description          : The Load Balancer has detected a Constraint Violation for this
Replica: fabric:/System/FailoverManagerService Secondary Partition 00000000-0000-0000-0000-000000000001 is
          violating the Constraint: UpgradeDomain Details: UpgradeDomain ID -- 4, Replica on
NodeName -- Node.8 Currently Upgrading -- false Distribution Policy -- Packing
          RemoveWhenExpired   : True
          IsExpired           : False
          Transitions          : Ok->Warning = 8/10/2015 7:13:02 PM, LastError = 1/1/0001
12:00:00 AM

```

Here's what this health message is telling us is:

1. All the replicas themselves are healthy: Each has AggregatedHealthState : Ok
2. The Upgrade Domain distribution constraint is currently being violated. This means a particular Upgrade Domain has more replicas from this partition than it should.
3. Which node contains the replica causing the violation. In this case it's the node with the name "Node.8"
4. Whether an upgrade is currently happening for this partition ("Currently Upgrading -- false")
5. The distribution policy for this service: "Distribution Policy -- Packing". This is governed by the [RequireDomainDistribution placement policy](#). "Packing" indicates that in this case DomainDistribution was *not* required, so we know that placement policy was not specified for this service.
6. When the report happened - 8/10/2015 7:13:02 PM

Information like this powers alerts that fire in production to let you know something has gone wrong and is also used to detect and halt bad upgrades. In this case, we'd want to see if we can figure out why the Resource Manager had to pack the replicas into the Upgrade Domain. Usually packing is transient because the nodes in the other Upgrade Domains were down, for example.

Let's say the Cluster Resource Manager is trying to place some services, but there aren't any solutions that work. When services can't be placed, it is usually for one of the following reasons:

1. Some transient condition has made it impossible to place this service instance or replica correctly
2. The service's placement requirements are unsatisfiable.

In these cases, health reports from the Cluster Resource Manager help you determine why the service can't be placed. We call this process the constraint elimination sequence. During it, the system walks through the configured constraints affecting the service and records what they eliminate. This way when services aren't able to be placed, you can see which nodes were eliminated and why.

## Constraint types

Let's talk about each of the different constraints in these health reports. You will see health messages related to these constraints when replicas can't be placed.

- **ReplicaExclusionStatic** and **ReplicaExclusionDynamic**: These constraints indicates that a solution was rejected because two service objects from the same partition would have to be placed on the same node. This isn't allowed because then failure of that node would overly impact that partition. ReplicaExclusionStatic and ReplicaExclusionDynamic are almost the same rule and the differences don't really matter. If you are seeing a constraint elimination sequence containing either the ReplicaExclusionStatic or ReplicaExclusionDynamic constraint, the Cluster Resource Manager thinks that there aren't enough nodes. This requires remaining solutions to use these invalid placements which are disallowed. The other constraints in the sequence will usually tell us why nodes are being eliminated in the first place.
- **PlacementConstraint**: If you see this message, it means that we eliminated some nodes because they didn't match the service's placement constraints. We trace out the currently configured placement constraints as a part of this message. This is normal if you have a placement constraint defined. However, if placement constraint is incorrectly causing too many nodes to be eliminated this is how you would notice.
- **NodeCapacity**: This constraint means that the Cluster Resource Manager couldn't place the replicas on the indicated nodes because that would put them over capacity.
- **Affinity**: This constraint indicates that we couldn't place the replica on the affected nodes since it would cause a violation of the affinity constraint. More information on affinity is in [this article](#)
- **FaultDomain** and **UpgradeDomain**: This constraint eliminates nodes if placing the replica on the indicated nodes would cause packing in a particular fault or upgrade domain. Several examples discussing this constraint are presented in the topic on [fault and upgrade domain constraints and resulting behavior](#)
- **PreferredLocation**: You shouldn't normally see this constraint removing nodes from the solution since it runs as an optimization by default. The preferred location constraint is also present during upgrades. During upgrade it is used to move services back to where they were when the upgrade started.

## Blocklisting Nodes

Another health message the Cluster Resource Manager reports is when nodes are blocklisted. You can think of blocklisting as a temporary constraint that is automatically applied for you. Nodes get blocklisted when they experience repeated failures when launching instances of that service type. Nodes are blocklisted on a per-service-type basis. A node may be blocklisted for one service type but not another.

You'll see blocklisting kick in often during development: some bug causes your service host to crash on startup. Service Fabric tries to create the service host a few times, and the failure keeps occurring. After a few attempts, the node gets blocklisted, and the Cluster Resource Manager will try to create the service elsewhere. If that failure keeps happening on multiple nodes, it's possible that all of the valid nodes in the cluster end up blocked.

Blocklisting can also remove so many nodes that not enough can successfully launch the service to meet the desired scale. You'll typically see additional errors or warnings from the Cluster Resource Manager indicating that the service is below the desired replica or instance count, as well as health messages indicating what the failure is that's leading to the blocklisting in the first place.

Blocklisting is not a permanent condition. After a few minutes, the node is removed from the blocklist and Service

Fabric may activate the services on that node again. If services continue to fail, the node is blacklisted for that service type again.

## Constraint priorities

### WARNING

Changing constraint priorities is not recommended and may have significant adverse effects on your cluster. The below information is provided for reference of the default constraint priorities and their behavior.

With all of these constraints, you may have been thinking "Hey – I think that fault domain constraints are the most important thing in my system. In order to ensure the fault domain constraint isn't violated, I'm willing to violate other constraints."

Constraints can be configured with different priority levels. These are:

- "hard" (0)
- "soft" (1)
- "optimization" (2)
- "off" (-1).

Most of the constraints are configured as hard constraints by default.

Changing the priority of constraints is uncommon. There have been times where constraint priorities needed to change, usually to work around some other bug or behavior that was impacting the environment. Generally the flexibility of the constraint priority infrastructure has worked very well, but it isn't needed often. Most of the time everything sits at their default priorities.

The priority levels don't mean that a given constraint *will* be violated, nor that it will always be met. Constraint priorities define an order in which constraints are enforced. Priorities define the tradeoffs when it is impossible to satisfy all constraints. Usually all the constraints can be satisfied unless there's something else going on in the environment. Some examples of scenarios that will lead to constraint violations are conflicting constraints, or large numbers of concurrent failures.

In advanced situations, you can change the constraint priorities. For example, say you wanted to ensure that affinity would always be violated when necessary to solve node capacity issues. To achieve this, you could set the priority of the affinity constraint to "soft" (1) and leave the capacity constraint set to "hard" (0).

The default priority values for the different constraints are specified in the following config:

ClusterManifest.xml

```
<Section Name="PlacementAndLoadBalancing">
    <Parameter Name="PlacementConstraintPriority" Value="0" />
    <Parameter Name="CapacityConstraintPriority" Value="0" />
    <Parameter Name="AffinityConstraintPriority" Value="0" />
    <Parameter Name="FaultDomainConstraintPriority" Value="0" />
    <Parameter Name="UpgradeDomainConstraintPriority" Value="1" />
    <Parameter Name="PreferredLocationConstraintPriority" Value="2" />
</Section>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters:

```

"fabricSettings": [
  {
    "name": "PlacementAndLoadBalancing",
    "parameters": [
      {
        "name": "PlacementConstraintPriority",
        "value": "0"
      },
      {
        "name": "CapacityConstraintPriority",
        "value": "0"
      },
      {
        "name": "AffinityConstraintPriority",
        "value": "0"
      },
      {
        "name": "FaultDomainConstraintPriority",
        "value": "0"
      },
      {
        "name": "UpgradeDomainConstraintPriority",
        "value": "1"
      },
      {
        "name": "PreferredLocationConstraintPriority",
        "value": "2"
      }
    ]
  }
]

```

## Fault domain and upgrade domain constraints

The Cluster Resource Manager wants to keep services spread out among fault and upgrade domains. It models this as a constraint inside the Cluster Resource Manager's engine. For more information on how they are used and their specific behavior, check out the article on [cluster configuration](#).

The Cluster Resource Manager may need to pack a couple replicas into an upgrade domain in order to deal with upgrades, failures, or other constraint violations. Packing into fault or upgrade domains normally happens only when there are several failures or other churn in the system preventing correct placement. If you wish to prevent packing even during these situations, you can utilize the [RequireDomainDistribution](#) [placement policy](#). Note that this may affect service availability and reliability as a side effect, so consider it carefully.

If the environment is configured correctly, all constraints are fully respected, even during upgrades. The key thing is that the Cluster Resource Manager is watching out for your constraints. When it detects a violation it immediately reports it and tries to correct the issue.

## The preferred location constraint

The PreferredLocation constraint is a little different, as it has two different uses. One use of this constraint is during application upgrades. The Cluster Resource Manager automatically manages this constraint during upgrades. It is used to ensure that when upgrades are complete that replicas return to their initial locations. The other use of the PreferredLocation constraint is for the [PreferredPrimaryDomain](#) [placement policy](#). Both of these are optimizations, and hence the PreferredLocation constraint is the only constraint set to "Optimization" by default.

## Upgrades

The Cluster Resource Manager also helps during application and cluster upgrades, during which it has two jobs:

- ensure that the rules of the cluster are not compromised
- try to help the upgrade go smoothly

### Keep enforcing the rules

The main thing to be aware of is that the rules – the strict constraints like placement constraints and capacities – are still enforced during upgrades. Placement constraints ensure that your workloads only run where they are allowed to, even during upgrades. When services are highly constrained, upgrades can take longer. When the service or the node it is running on is brought down for an update there may be few options for where it can go.

### Smart replacements

When an upgrade starts, the Resource Manager takes a snapshot of the current arrangement of the cluster. As each Upgrade Domain completes, it attempts to return the services that were in that Upgrade Domain to their original arrangement. This way there are at most two transitions for a service during the upgrade. There is one move out of the affected node and one move back in. Returning the cluster or service to how it was before the upgrade also ensures the upgrade doesn't impact the layout of the cluster.

### Reduced churn

Another thing that happens during upgrades is that the Cluster Resource Manager turns off balancing. Preventing balancing prevents unnecessary reactions to the upgrade itself, like moving services into nodes that were emptied for the upgrade. If the upgrade in question is a Cluster upgrade, then the entire cluster is not balanced during the upgrade. Constraint checks stay active, only movement based on the proactive balancing of metrics is disabled.

### Buffered Capacity & Upgrade

Generally you want the upgrade to complete even if the cluster is constrained or close to full. Managing the capacity of the cluster is even more important during upgrades than usual. Depending on the number of upgrade domains, between 5 and 20 percent of capacity must be migrated as the upgrade rolls through the cluster. That work has to go somewhere. This is where the notion of [buffered capacities](#) is useful. Buffered capacity is respected during normal operation. The Cluster Resource Manager may fill nodes up to their total capacity (consuming the buffer) during upgrades if necessary.

## Next steps

- Start from the beginning and [get an Introduction to the Service Fabric Cluster Resource Manager](#)

# Defragmentation of metrics and load in Service Fabric

8/18/2017 • 4 min to read • [Edit Online](#)

The Service Fabric Cluster Resource Manager's default strategy for managing load metrics in the cluster is to distribute the load. Ensuring that nodes are evenly utilized avoids hot and cold spots that lead to both contention and wasted resources. Distributing workloads in the cluster is also the safest in terms of surviving failures since it ensures that a failure doesn't take out a large percentage of a given workload.

The Service Fabric Cluster Resource Manager does support a different strategy for managing load, which is defragmentation. Defragmentation means that instead of trying to distribute the utilization of a metric across the cluster, it is consolidated. Consolidation is just an inversion of the default balancing strategy – instead of minimizing the average standard deviation of metric load, the Cluster Resource Manager tries to increase it.

## When to use defragmentation

Distributing load in the cluster consumes some of the resources on each node. Some workloads create services that are exceptionally large and consume most or all of a node. In these cases, it's possible that when there are large workloads getting created that there isn't enough space on any node to run them. Large workloads aren't a problem in Service Fabric; in these cases the Cluster Resource Manager determines that it needs to reorganize the cluster to make room for this large workload. However, in the meantime that workload has to wait to be scheduled in the cluster.

If there are many services and state to move around, then it could take a long time for the large workload to be placed in the cluster. This is more likely if other workloads in the cluster are also large and so take longer to reorganize. The Service Fabric team measured creation times in simulations of this scenario. We found that creating large services took much longer as soon as cluster utilization got above between 30% and 50%. To handle this scenario, we introduced defragmentation as a balancing strategy. We found that for large workloads, especially ones where creation time was important, defragmentation really helped those new workloads get scheduled in the cluster.

You can configure defragmentation metrics to have the Cluster Resource Manager to proactively try to condense the load of the services into fewer nodes. This helps ensure that there is almost always room for large services without reorganizing the cluster. Not having to reorganize the cluster allows creating large workloads quickly.

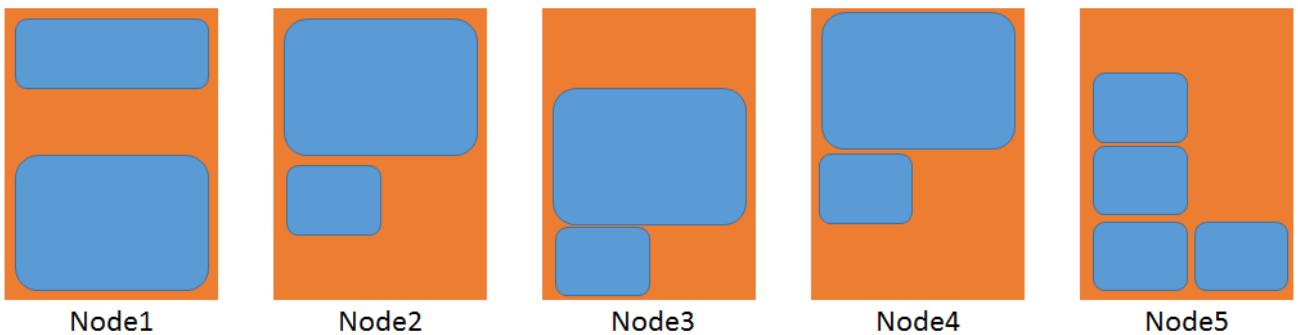
Most people don't need defragmentation. Services are usually be small, so it's not hard to find room for them in the cluster. When reorganization is possible, it goes quickly, again because most services are small and can be moved quickly and in parallel. However, if you have large services and need them created quickly then the defragmentation strategy is for you. We'll discuss the tradeoffs of using defragmentation next.

## Defragmentation tradeoffs

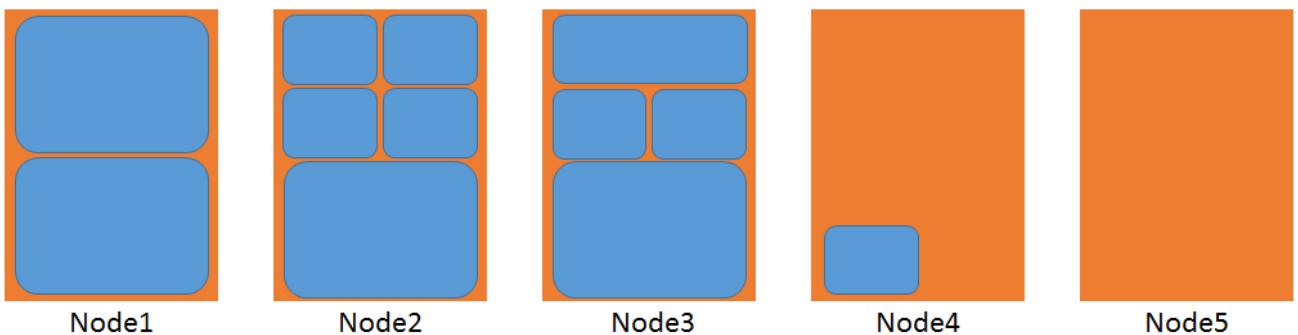
Defragmentation can increase impactfulness of failures, since more services are running on nodes that fail. Defragmentation can also increase costs, since resources in the cluster must be held in reserve, waiting for the creation of large workloads.

The following diagram gives a visual representation of two clusters, one that is defragmented and one that is not.

# Balanced Cluster



# Defragmented Cluster



In the balanced case, consider the number of movements that would be necessary to place one of the largest service objects. In the defragmented cluster, the large workload could be placed on nodes four or five without having to wait for any other services to move.

## Defragmentation pros and cons

So what are those other conceptual tradeoffs? Here's a quick table of things to think about:

DEFRAAGMENTATION PROS	DEFRAAGMENTATION CONS
Allows faster creation of large services	Concentrates load onto fewer nodes, increasing contention
Enables lower data movement during creation	Failures can impact more services and cause more churn
Allows rich description of requirements and reclamation of space	More complex overall Resource Management configuration

You can mix defragmented and normal metrics in the same cluster. The Cluster Resource Manager tries to consolidate the defragmentation metrics as much as possible while spreading out the others. The results of mixing defragmentation and balancing strategies depends on several factors, including:

- the number of balancing metrics vs. the number of defragmentation metrics
- Whether any service uses both types of metrics
- the metric weights
- current metric loads

Experimentation is required to determine the exact configuration necessary. We recommend thorough measurement of your workloads before you enable defragmentation metrics in production. This is especially true when mixing defragmentation and balanced metrics within the same service.

## Configuring defragmentation metrics

Configuring defragmentation metrics is a global decision in the cluster, and individual metrics can be selected for defragmentation. The following config snippets show how to configure metrics for defragmentation. In this case, "Metric1" is configured as a defragmentation metric, while "Metric2" will continue to be balanced normally.

ClusterManifest.xml:

```
<Section Name="DefragmentationMetrics">
    <Parameter Name="Metric1" Value="true" />
    <Parameter Name="Metric2" Value="false" />
</Section>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters:

```
"fabricSettings": [
{
    "name": "DefragmentationMetrics",
    "parameters": [
        {
            "name": "Metric1",
            "value": "true"
        },
        {
            "name": "Metric2",
            "value": "false"
        }
    ]
}]
```

## Next steps

- The Cluster Resource Manager has man options for describing the cluster. To find out more about them, check out this article on [describing a Service Fabric cluster](#)
- Metrics are how the Service Fabric Cluster Resource Manger manages consumption and capacity in the cluster. To learn more about metrics and how to configure them, check out [this article](#)

# Balancing your service fabric cluster

9/5/2017 • 8 min to read • [Edit Online](#)

The Service Fabric Cluster Resource Manager supports dynamic load changes, reacting to additions or removals of nodes or services. It also automatically corrects constraint violations, and proactively rebalances the cluster. But how often are these actions taken, and what triggers them?

There are three different categories of work that the Cluster Resource Manager performs. They are:

1. Placement – this stage deals with placing any stateful replicas or stateless instances that are missing. Placement includes both new services and handling stateful replicas or stateless instances that have failed. Deleting and dropping replicas or instances are handled here.
2. Constraint Checks – this stage checks for and corrects violations of the different placement constraints (rules) within the system. Examples of rules are things like ensuring that nodes are not over capacity and that a service's placement constraints are met.
3. Balancing – this stage checks to see if rebalancing is necessary based on the configured desired level of balance for different metrics. If so it attempts to find an arrangement in the cluster that is more balanced.

## Configuring Cluster Resource Manager Timers

The first set of controls around balancing are a set of timers. These timers govern how often the Cluster Resource Manager examines the cluster and takes corrective actions.

Each of these different types of corrections the Cluster Resource Manager can make is controlled by a different timer that governs its frequency. When each timer fires, the task is scheduled. By default the Resource Manager:

- scans its state and applies updates (like recording that a node is down) every 1/10th of a second
- sets the placement check flag
- sets the constraint check flag every second
- sets the balancing flag every five seconds.

Examples of the configuration governing these timers are below:

ClusterManifest.xml:

```
<Section Name="PlacementAndLoadBalancing">
    <Parameter Name="PLBRefreshGap" Value="0.1" />
    <Parameter Name="MinPlacementInterval" Value="1.0" />
    <Parameter Name="MinConstraintCheckInterval" Value="1.0" />
    <Parameter Name="MinLoadBalancingInterval" Value="5.0" />
</Section>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters:

```

"fabricSettings": [
    {
        "name": "PlacementAndLoadBalancing",
        "parameters": [
            {
                "name": "PLBRefreshGap",
                "value": "0.10"
            },
            {
                "name": "MinPlacementInterval",
                "value": "1.0"
            },
            {
                "name": "MinConstraintCheckInterval",
                "value": "1.0"
            },
            {
                "name": "MinLoadBalancingInterval",
                "value": "5.0"
            }
        ]
    }
]

```

Today the Cluster Resource Manager only performs one of these actions at a time, sequentially. This is why we refer to these timers as "minimum intervals" and the actions that get taken when the timers go off as "setting flags". For example, the Cluster Resource Manager takes care of pending requests to create services before balancing the cluster. As you can see by the default time intervals specified, the Cluster Resource Manager scans for anything it needs to do frequently. Normally this means that the set of changes made during each step is small. Making small changes frequently allows the Cluster Resource Manager to be responsive when things happen in the cluster. The default timers provide some batching since many of the same types of events tend to occur simultaneously.

For example, when nodes fail they can do so entire fault domains at a time. All these failures are captured during the next state update after the *PLBRefreshGap*. The corrections are determined during the following placement, constraint check, and balancing runs. By default the Cluster Resource Manager is not scanning through hours of changes in the cluster and trying to address all changes at once. Doing so would lead to bursts of churn.

The Cluster Resource Manager also needs some additional information to determine if the cluster imbalanced. For that we have two other pieces of configuration: *BalancingThresholds* and *ActivityThresholds*.

## Balancing thresholds

A Balancing Threshold is the main control for triggering rebalancing. The Balancing Threshold for a metric is a *ratio*. If the load for a metric on the most loaded node divided by the amount of load on the least loaded node exceeds that metric's *BalancingThreshold*, then the cluster is imbalanced. As a result balancing is triggered the next time the Cluster Resource Manager checks. The *MinLoadBalancingInterval* timer defines how often the Cluster Resource Manager should check if rebalancing is necessary. Checking doesn't mean that anything happens.

Balancing Thresholds are defined on a per-metric basis as a part of the cluster definition. For more information on metrics, check out [this article](#).

`ClusterManifest.xml`

```

<Section Name="MetricBalancingThresholds">
    <Parameter Name="MetricName1" Value="2"/>
    <Parameter Name="MetricName2" Value="3.5"/>
</Section>

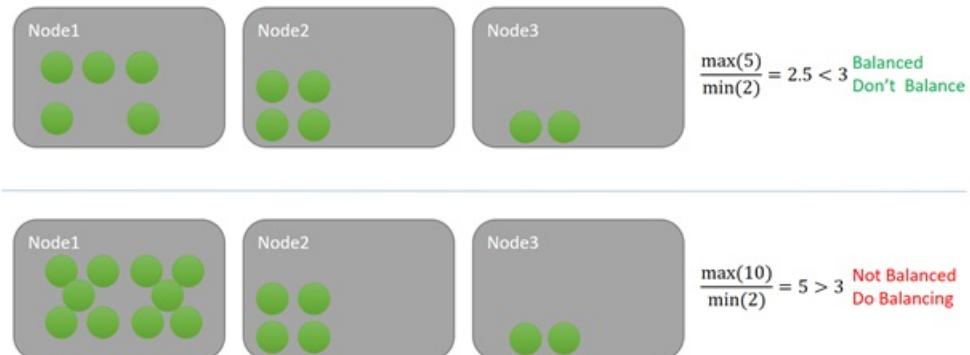
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters:

```

"fabricSettings": [
    {
        "name": "MetricBalancingThresholds",
        "parameters": [
            {
                "name": "MetricName1",
                "value": "2"
            },
            {
                "name": "MetricName2",
                "value": "3.5"
            }
        ]
    }
]

```



In this example, each service is consuming one unit of some metric. In the top example, the maximum load on a node is five and the minimum is two. Let's say that the balancing threshold for this metric is three. Since the ratio in the cluster is  $5/2 = 2.5$  and that is less than the specified balancing threshold of three, the cluster is balanced. No rebalancing is triggered when the Cluster Resource Manager checks.

In the bottom example, the maximum load on a node is ten, while the minimum is two, resulting in a ratio of five. Five is greater than the designated balancing threshold of three for that metric. As a result, a rebalancing run will be scheduled next time the balancing timer fires. In a situation like this some load is usually distributed to Node3. Because the Service Fabric Cluster Resource Manager doesn't use a greedy approach, some load could also be distributed to Node2.



#### NOTE

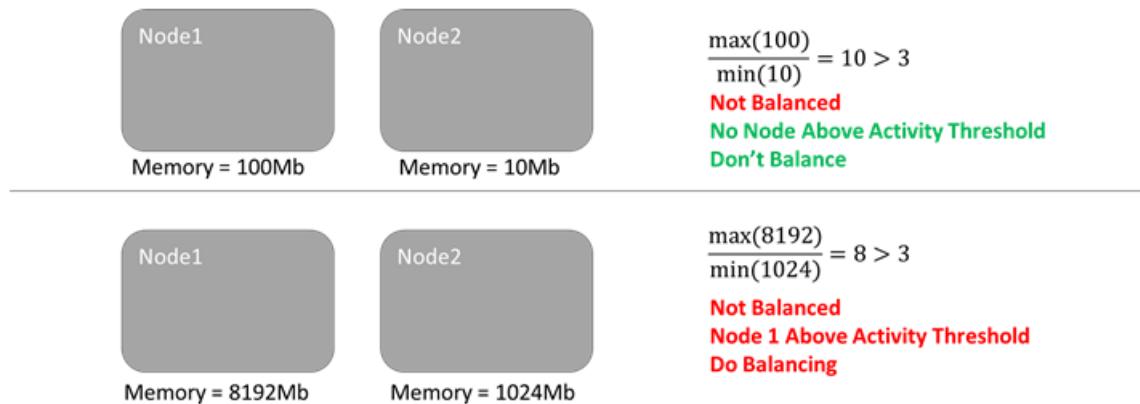
"Balancing" handles two different strategies for managing load in your cluster. The default strategy that the Cluster Resource Manager uses is to distribute load across the nodes in the cluster. The other strategy is [defragmentation](#). Defragmentation is performed during the same balancing run. The balancing and defragmentation strategies can be used for different metrics within the same cluster. A service can have both balancing and defragmentation metrics. For defragmentation metrics, the ratio of the loads in the cluster triggers rebalancing when it is *below* the balancing threshold.

Getting below the balancing threshold is not an explicit goal. Balancing Thresholds are just a *trigger*. When balancing runs, the Cluster Resource Manager determines what improvements it can make, if any. Just because a balancing search is kicked off doesn't mean anything moves. Sometimes the cluster is imbalanced but too constrained to correct. Alternatively, the improvements require movements that are too [costly](#).

## Activity thresholds

Sometimes, although nodes are relatively imbalanced, the *total* amount of load in the cluster is low. The lack of load could be a transient dip, or because the cluster is new and just getting bootstrapped. In either case, you may not want to spend time balancing the cluster because there's little to be gained. If the cluster underwent balancing, you'd spend network and compute resources to move things around without making any large *absolute* difference. To avoid unnecessary moves, there's another control known as Activity Thresholds. Activity Thresholds allows you to specify some absolute lower bound for activity. If no node is over this threshold, balancing isn't triggered even if the Balancing Threshold is met.

Let's say that we retain our Balancing Threshold of three for this metric. Let's also say we have an Activity Threshold of 1536. In the first case, while the cluster is imbalanced per the Balancing Threshold there's no node meets that Activity Threshold, so nothing happens. In the bottom example, Node1 is over the Activity Threshold. Since both the Balancing Threshold and the Activity Threshold for the metric are exceeded, balancing is scheduled. As an example, let's look at the following diagram:



Just like Balancing Thresholds, Activity Thresholds are defined per-metric via the cluster definition:

ClusterManifest.xml

```
<Section Name="MetricActivityThresholds">
  <Parameter Name="Memory" Value="1536"/>
</Section>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters:

```
"fabricSettings": [
  {
    "name": "MetricActivityThresholds",
    "parameters": [
      {
        "name": "Memory",
        "value": "1536"
      }
    ]
  }
]
```

Balancing and activity thresholds are both tied to a specific metric - balancing is triggered only if both the Balancing Threshold and Activity Threshold is exceeded for the same metric.

**NOTE**

When not specified, the Balancing Threshold for a metric is 1, and the Activity Threshold is 0. This means that the Cluster Resource Manager will try to keep that metric perfectly balanced for any given load. If you are using custom metrics it is recommended that you explicitly define your own balancing and activity thresholds for your metrics.

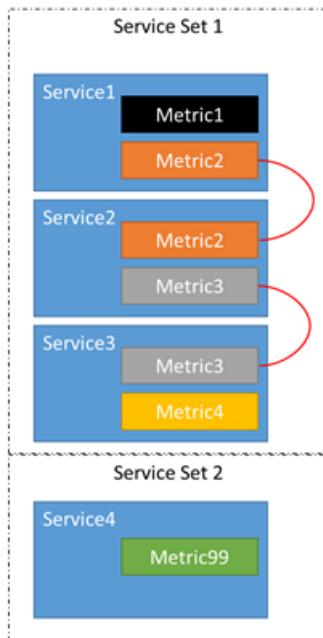
## Balancing services together

Whether the cluster is imbalanced or not is a cluster-wide decision. However, the way we go about fixing it is moving individual service replicas and instances around. This makes sense, right? If memory is stacked up on one node, multiple replicas or instances could be contributing to it. Fixing the imbalance could require moving any of the stateful replicas or stateless instances that use the imbalanced metric.

Occasionally though, a service that wasn't itself imbalanced gets moved (remember the discussion of local and global weights earlier). Why would a service get moved when all that service's metrics were balanced? Let's see an example:

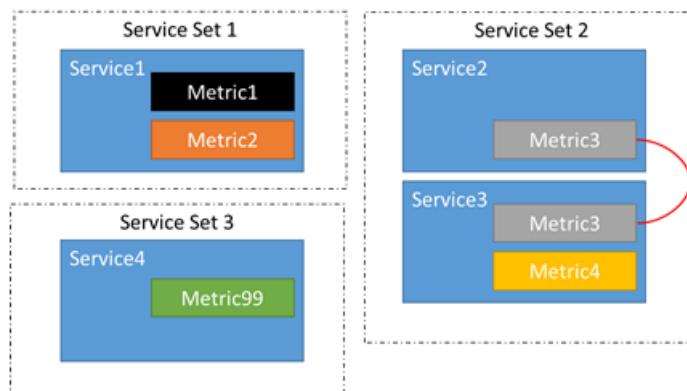
- Let's say there are four services, Service1, Service2, Service3, and Service4.
- Service1 reports metrics Metric1 and Metric2.
- Service2 reports metrics Metric2 and Metric3.
- Service3 reports metrics Metric3 and Metric4.
- Service4 reports metric Metric99.

Surely you can see where we're going here: There's a chain! We don't really have four independent services, we have three services that are related and one that is off on its own.



Because of this chain, it's possible that an imbalance in metrics 1-4 can cause replicas or instances belonging to services 1-3 to move around. We also know that an imbalance in Metrics 1, 2, or 3 can't cause movements in Service4. There would be no point since moving the replicas or instances belonging to Service4 around can do absolutely nothing to impact the balance of Metrics 1-3.

The Cluster Resource Manager automatically figures out what services are related. Adding, removing, or changing the metrics for services can impact their relationships. For example, between two runs of balancing Service2 may have been updated to remove Metric2. This breaks the chain between Service1 and Service2. Now instead of two groups of related services, there are three:



## Next steps

- Metrics are how the Service Fabric Cluster Resource Manager manages consumption and capacity in the cluster. To learn more about metrics and how to configure them, check out [this article](#)
- Movement Cost is one way of signaling to the Cluster Resource Manager that certain services are more expensive to move than others. For more about movement cost, refer to [this article](#)
- The Cluster Resource Manager has several throttles that you can configure to slow down churn in the cluster. They're not normally necessary, but if you need them you can learn about them [here](#)

# Throttling the Service Fabric Cluster Resource Manager

8/18/2017 • 4 min to read • [Edit Online](#)

Even if you've configured the Cluster Resource Manager correctly, the cluster can get disrupted. For example, there could be simultaneous node and fault domain failures - what would happen if that occurred during an upgrade? The Cluster Resource Manager always tries to fix everything, consuming the cluster's resources trying to reorganize and fix the cluster. Throttles help provide a backstop so that the cluster can use resources to stabilize - the nodes come back, the network partitions heal, corrected bits get deployed.

To help with these sorts of situations, the Service Fabric Cluster Resource Manager includes several throttles. These throttles are all fairly large hammers. Generally they shouldn't be changed without careful planning and testing.

If you change the Cluster Resource Manager's throttles, you should tune them to your expected actual load. You may determine you need to have some throttles in place, even if it means the cluster takes longer to stabilize in some situations. Testing is required to determine the correct values for throttles. Throttles need to be high enough to allow the cluster to respond to changes in a reasonable amount of time, and low enough to actually prevent too much resource consumption.

Most of the time we've seen customers use throttles it has been because they were already in a resource constrained environment. Some examples would be limited network bandwidth for individual nodes, or disks that aren't able to build many stateful replicas in parallel due to throughput limitations. Without throttles, operations could overwhelm these resources, causing operations to fail or be slow. In these situations customers used throttles and knew they were extending the amount of time it would take the cluster to reach a stable state. Customers also understood they could end up running at lower overall reliability while they were throttled.

## Configuring the throttles

Service Fabric has two mechanisms for throttling the number of replica movements. The default mechanism that existed before Service Fabric 5.7 represents throttling as an absolute number of moves allowed. This does not work for clusters of all sizes. In particular, for large clusters the default value can be too small, significantly slowing down balancing even when it is necessary, while having no effect in smaller clusters. This prior mechanism has been superseded by percentage-based throttling, which scales better with dynamic clusters in which the number of services and nodes change regularly.

The throttles are based on a percentage of the number of replicas in the clusters. Percentage based throttles enable expressing the rule: "do not move more than 10% of replicas in a 10 minute interval", for example.

The configuration settings for percentage-based throttling are:

- `GlobalMovementThrottleThresholdPercentage` - Maximum number of movements allowed in cluster at any time, expressed as percentage of total number of replicas in the cluster. 0 indicates no limit. The default value is 0. If both this setting and `GlobalMovementThrottleThreshold` are specified, then the more conservative limit is used.
- `GlobalMovementThrottleThresholdPercentageForPlacement` - Maximum number of movements allowed during the placement phase, expressed as percentage of total number of replicas in the cluster. 0 indicates no limit. The default value is 0. If both this setting and `GlobalMovementThrottleThresholdForPlacement` are specified, then the more conservative limit is used.
- `GlobalMovementThrottleThresholdPercentageForBalancing` - Maximum number of movements allowed during the balancing phase, expressed as percentage of total number of replicas in the cluster. 0 indicates no limit. The

default value is 0. If both this setting and GlobalMovementThrottleThresholdForBalancing are specified, then the more conservative limit is used.

When specifying the throttle percentage, you'd specify 5% as 0.05. The interval on which these throttles are governed is the GlobalMovementThrottleCountingInterval, which is specified in seconds.

```
<Section Name="PlacementAndLoadBalancing">
  <Parameter Name="GlobalMovementThrottleThresholdPercentage" Value="0" />
  <Parameter Name="GlobalMovementThrottleThresholdPercentageForPlacement" Value="0" />
  <Parameter Name="GlobalMovementThrottleThresholdPercentageForBalancing" Value="0" />
  <Parameter Name="GlobalMovementThrottleCountingInterval" Value="600" />
</Section>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters:

```
"fabricSettings": [
  {
    "name": "PlacementAndLoadBalancing",
    "parameters": [
      {
        "name": "GlobalMovementThrottleThresholdPercentage",
        "value": "0.0"
      },
      {
        "name": "GlobalMovementThrottleThresholdPercentageForPlacement",
        "value": "0.0"
      },
      {
        "name": "GlobalMovementThrottleThresholdPercentageForBalancing",
        "value": "0.0"
      },
      {
        "name": "GlobalMovementThrottleCountingInterval",
        "value": "600"
      }
    ]
  }
]
```

### Default count based throttles

This information is provided in case you have older clusters or still retain these configurations in clusters that have since been upgraded. In general, it is recommended that these are replaced with the percentage-based throttles above. Since percentage-based throttling is disabled by default, these throttles remain the default throttles for a cluster until they are disabled and replaced with the percentage-based throttles.

- GlobalMovementThrottleThreshold – this setting controls the total number of movements in the cluster over some time. The amount of time is specified in seconds as the GlobalMovementThrottleCountingInterval. The default value for the GlobalMovementThrottleThreshold is 1000 and the default value for the GlobalMovementThrottleCountingInterval is 600.
- MovementPerPartitionThrottleThreshold – this setting controls the total number of movements for any service partition over some time. The amount of time is specified in seconds as the MovementPerPartitionThrottleCountingInterval. The default value for the MovementPerPartitionThrottleThreshold is 50 and the default value for the MovementPerPartitionThrottleCountingInterval is 600.

The configuration for these throttles follows the same pattern as the percentage-based throttling.

## Next steps

- To find out about how the Cluster Resource Manager manages and balances load in the cluster, check out the article on [balancing load](#)
- The Cluster Resource Manager has many options for describing the cluster. To find out more about them, check out this article on [describing a Service Fabric cluster](#)

# Service movement cost

8/18/2017 • 3 min to read • [Edit Online](#)

A factor that the Service Fabric Cluster Resource Manager considers when trying to determine what changes to make to a cluster is the cost of those changes. The notion of "cost" is traded off against how much the cluster can be improved. Cost is factored in when moving services for balancing, defragmentation, and other requirements. The goal is to meet the requirements in the least disruptive or expensive way.

Moving services costs CPU time and network bandwidth at a minimum. For stateful services, it requires copying the state of those services, consuming additional memory and disk. Minimizing the cost of solutions that the Azure Service Fabric Cluster Resource Manager comes up with helps ensure that the cluster's resources aren't spent unnecessarily. However, you also don't want to ignore solutions that would significantly improve the allocation of resources in the cluster.

The Cluster Resource Manager has two ways of computing costs and limiting them while it tries to manage the cluster. The first mechanism is simply counting every move that it would make. If two solutions are generated with about the same balance (score), then the Cluster Resource Manager prefers the one with the lowest cost (total number of moves).

This strategy works well. But as with default or static loads, it's unlikely in any complex system that all moves are equal. Some are likely to be much more expensive.

## Setting Move Costs

You can specify the default move cost for a service when it is created:

PowerShell:

```
New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName  
$serviceTypeName -Stateful -MinReplicaSetSize 3 -TargetReplicaSetSize 3 -PartitionSchemeSingleton -  
DefaultMoveCost Medium
```

C#:

```
FabricClient fabricClient = new FabricClient();  
StatefulServiceDescription serviceDescription = new StatefulServiceDescription();  
//set up the rest of the ServiceDescription  
serviceDescription.DefaultMoveCost = MoveCost.Medium;  
await fabricClient.ServiceManager.CreateServiceAsync(serviceDescription);
```

You can also specify or update MoveCost dynamically for a service after the service has been created:

PowerShell:

```
Update-ServiceFabricService -Stateful -ServiceName "fabric:/AppName/ServiceName" -DefaultMoveCost High
```

C#:

```

StatefulServiceUpdateDescription updateDescription = new StatefulServiceUpdateDescription();
updateDescription.DefaultMoveCost = MoveCost.High;
await fabricClient.ServiceManager.UpdateServiceAsync(new Uri("fabric:/AppName/ServiceName"),
updateDescription);

```

## Dynamically specifying move cost on a per-replica basis

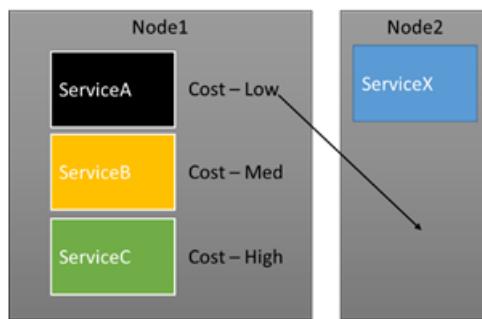
The preceding snippets are all for specifying MoveCost for a whole service at once from outside the service itself. However, move cost is most useful is when the move cost of a specific service object changes over its lifespan. Since the services themselves probably have the best idea of how costly they are to move a given time, there's an API for services to report their own individual move cost during runtime.

C#:

```
this.Partition.ReportMoveCost(MoveCost.Medium);
```

## Impact of move cost

MoveCost has four levels: Zero, Low, Medium, and High. MoveCosts are relative to each other, except for Zero. Zero move cost means that movement is free and should not count against the score of the solution. Setting your move cost to High does *not* guarantee that the replica stays in one place.



MoveCost helps you find the solutions that cause the least disruption overall and are easiest to achieve while still arriving at equivalent balance. A service's notion of cost can be relative to many things. The most common factors in calculating your move cost are:

- The amount of state or data that the service has to move.
- The cost of disconnection of clients. Moving a primary replica is usually more costly than the cost of moving a secondary replica.
- The cost of interrupting an in-flight operation. Some operations at the data store level or operations performed in response to a client call are costly. After a certain point, you don't want to stop them if you don't have to. So while the operation is going on, you increase the move cost of this service object to reduce the likelihood that it moves. When the operation is done, you set the cost back to normal.

## Enabling move cost in your cluster

In order for the more granular MoveCosts to be taken into account, MoveCost must be enabled in your cluster. Without this setting, the default mode of counting moves is used for calculating MoveCost, and MoveCost reports are ignored.

ClusterManifest.xml:

```
<Section Name="PlacementAndLoadBalancing">
    <Parameter Name="UseMoveCostReports" Value="true" />
</Section>
```

via ClusterConfig.json for Standalone deployments or Template.json for Azure hosted clusters:

```
"fabricSettings": [
{
    "name": "PlacementAndLoadBalancing",
    "parameters": [
        {
            "name": "UseMoveCostReports",
            "value": "true"
        }
    ]
}]
```

## Next steps

- Service Fabric Cluster Resource Manager uses metrics to manage consumption and capacity in the cluster. To learn more about metrics and how to configure them, check out [Managing resource consumption and load in Service Fabric with metrics](#).
- To learn about how the Cluster Resource Manager manages and balances load in the cluster, check out [Balancing your Service Fabric cluster](#).

# Monitoring and diagnostics for Azure Service Fabric

7/20/2017 • 6 min to read • [Edit Online](#)

Monitoring and diagnostics are critical to developing, testing, and deploying applications and services in any environment. Service Fabric solutions work best when you plan and implement monitoring and diagnostics that help ensure applications and services are working as expected in a local development environment or in production.

The main goals of monitoring and diagnostics are to:

- Detect and diagnose hardware and infrastructure issues
- Detect software and app issues, reduce service downtime
- Understand resource consumption and help drive operations decisions
- Optimize application, service, and infrastructure performance
- Generate business insights and identify areas of improvement

The overall workflow of monitoring and diagnostics consists of three steps:

1. **Event generation:** this includes events (logs, traces, custom events) at the infrastructure (cluster), platform, and application / service level
2. **Event aggregation:** generated events need to be collected and aggregated before they can be displayed
3. **Analysis:** events need to be visualized and accessible in some format, to allow for analysis and display as needed

Multiple products are available that cover these three areas, and you are free to choose different technologies for each. It is important to make sure that the various pieces work together to deliver an end-to-end monitoring solution for your cluster.

## Event generation

The first step in the monitoring and diagnostics workflow is the creation and generation of events and logs. These events, logs, and traces are generated at two levels: the platform layer (including the cluster, the machines, or Service Fabric actions) or the application layer (any instrumentation added to apps and services deployed to the cluster). Events at each of these levels are customizable, though Service Fabric does provide some instrumentation by default.

Read more about [platform level events](#) and [application level events](#) to understand what is provided and how to add further instrumentation.

After making a decision on the logging provider you would like to use, you need to make sure your logs are being aggregated and stored correctly.

## Event aggregation

For collecting the logs and events being generated by your applications and your cluster, we typically recommend using [Azure Diagnostics](#) (more similar to agent-based log collection) or [EventFlow](#) (in-process log collection).

Collecting application logs using Azure Diagnostics extension is a good option for Service Fabric services if the set of log sources and destinations does not change often and there is a straightforward mapping between the sources and their destinations. The reason for this is configuring Azure Diagnostics happens at the Resource Manager layer, so making significant changes to the configuration requires updating/redeploying the cluster.

Additionally, it is best utilized in making sure your logs are being stored somewhere a little more permanent, from where they can be accessed by various analysis platforms. This means that it ends up being less efficient of a pipeline than going with an option like EventFlow.

Using [EventFlow](#) allows you to have services send their logs directly to an analysis and visualization platform, and/or to storage. Other libraries (ILogger, Serilog, etc.) might be used for the same purpose, but EventFlow has the benefit of having been designed specifically for in-process log collection and to support Service Fabric services. This tends to have several potential advantages:

- Easy configuration and deployment
  - The configuration of diagnostic data collection is just part of the service configuration. It is easy to always keep it "in sync" with the rest of the application
  - Per-application or per-service configuration is easily achievable
  - Configuring data destinations through EventFlow is just a matter of adding the appropriate NuGet package and changing the *eventFlowConfig.json* file
- Flexibility
  - The application can send the data wherever it needs to go, as long as there is a client library that supports the targeted data storage system. New destinations can be added as desired
  - Complex capture, filtering, and data-aggregation rules can be implemented
- Access to internal application data and context
  - The diagnostic subsystem running inside the application/service process can easily augment the traces with contextual information

One thing to note is that these two options are not mutually exclusive and while it is possible to get a similar job done with using one or the other, it could also make sense for you to set up both. In most situations, combining an agent with in-process collection could lead to a more reliable monitoring workflow. The Azure Diagnostics extension (agent) could be your chosen path for platform level logs while you could use EventFlow (in-process collection) for your application level logs. Once you have figured out what works best for you, it is time to think about how you want your data to be displayed and analyzed.

## Event analysis

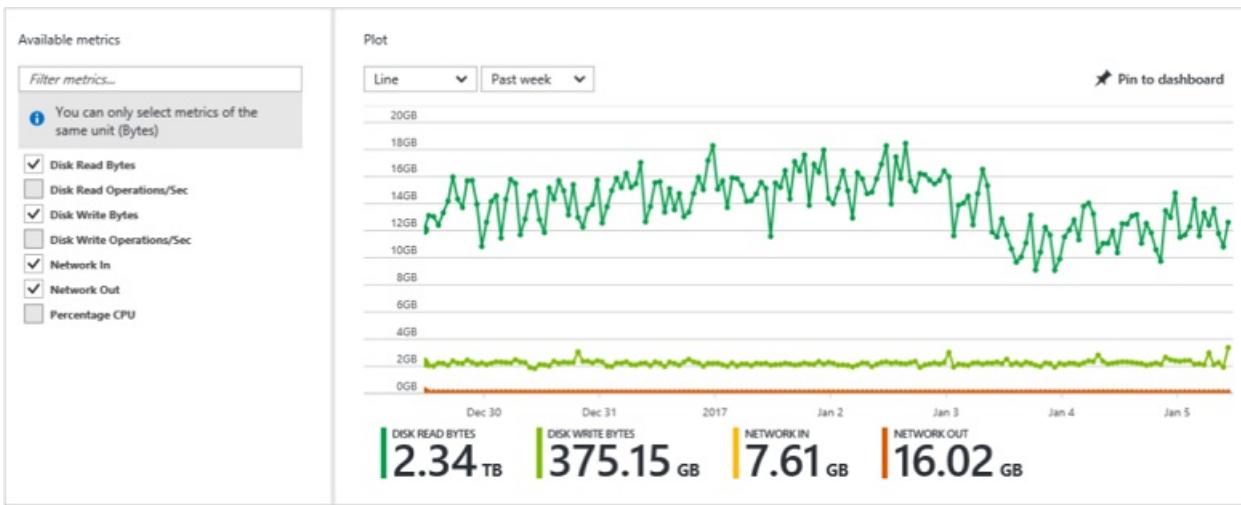
There are several great platforms that exist in the market when it comes to the analysis and visualization of monitoring and diagnostics data. The two that we recommend are [OMS](#) and [Application Insights](#) due to their better integration with Service Fabric, but you should also look into the [Elastic Stack](#) (especially if you are considering running a cluster in an offline environment), [Splunk](#), or any other platform of your preference.

The key points for any platform you choose should include how comfortable you are with the user interface and querying options, the ability to visualize data and create easily readable dashboards, and the additional tools they provide to enhance your monitoring, such as automated alerting.

In addition to the platform you choose, when you set up a Service Fabric cluster as an Azure resource, you also get access to Azure's out-of-the-box monitoring for machines, which can be useful for specific performance and metric monitoring.

### Azure Monitor

You can use [Azure Monitor](#) to monitor many of the Azure resources on which a Service Fabric cluster is built. A set of metrics for the [virtual machine scale set](#) and individual [virtual machines](#) is automatically collected and displayed in the Azure portal. To view the collected information, in the Azure portal, select the resource group that contains the Service Fabric cluster. Then, select the virtual machine scale set that you want to view. In the **Monitoring** section, select **Metrics** to view a graph of the values.



To customize the charts, follow the instructions in [Metrics in Microsoft Azure](#). You also can create alerts based on these metrics, as described in [Create alerts in Azure Monitor for Azure services](#). You can send alerts to a notification service by using web hooks, as described in [Configure a web hook on an Azure metric alert](#). Azure Monitor supports only one subscription. If you need to monitor multiple subscriptions, or if you need additional features, [Log Analytics](#), part of Microsoft Operations Management Suite, provides a holistic IT management solution both for on-premises and cloud-based infrastructure. You can route data from Azure Monitor directly to Log Analytics, so you can see metrics and logs for your entire environment in a single place.

## Next steps

### Watchdogs

A watchdog is a separate service that can watch health and load across services, and report health for anything in the health model hierarchy. This can help prevent errors that would not be detected based on the view of a single service. Watchdogs are also a good place to host code that performs remedial actions without user interaction (for example, cleaning up log files in storage at certain time intervals). You can find a sample watchdog service implementation [here](#).

Get started with understanding how events and logs get generated at the [platform level](#) and the [application level](#).

# Platform level event and log generation

8/25/2017 • 7 min to read • [Edit Online](#)

## Monitoring the cluster

It is important to monitor at the platform level to determine whether or not your hardware and cluster are behaving as expected. Though Service Fabric can keep applications running during a hardware failure, but you still need to diagnose whether an error is occurring in an application or in the underlying infrastructure. You also should monitor your clusters to better plan for capacity, helping in decisions about adding or removing hardware.

Service Fabric provides five different log channels out-of-the-box that generate the following events:

- Operational channel: high-level operations performed by Service Fabric and the cluster, including events for a node coming up, a new application being deployed, or a SF upgrade rollback, etc.
- Operational channel - detailed: health reports and load balancing decisions
- Data & Messaging channel: critical logs and events generated in our messaging (currently only the ReverseProxy) and data path (reliable services models)
- Data & Messaging channel - detailed: verbose channel that contains all the non-critical logs from data and messaging in the cluster (this channel has a very high volume of events)
- [Reliable Services events](#): programming model specific events
- [Reliable Actors events](#): programming model specific events and performance counters
- Support logs: system logs generated by Service Fabric only to be used by us when providing support

These various channels cover most of the platform level logging that is recommended. To improve platform level logging, consider investing in better understanding the health model and adding custom health reports, and adding custom **Performance Counters** to build a real-time understanding of the impact of your services and applications on the cluster.

### Azure Service Fabric health and load reporting

Service Fabric has its own health model, which is described in detail in these articles:

- [Introduction to Service Fabric health monitoring](#)
- [Report and check service health](#)
- [Add custom Service Fabric health reports](#)
- [View Service Fabric health reports](#)

Health monitoring is critical to multiple aspects of operating a service. Health monitoring is especially important when Service Fabric performs a named application upgrade. After each upgrade domain of the service is upgraded and is available to your customers, the upgrade domain must pass health checks before the deployment moves to the next upgrade domain. If good health status cannot be achieved, the deployment is rolled back, so that the application is in a known, good state. Although some customers might be affected before the services are rolled back, most customers won't experience an issue. Also, a resolution occurs relatively quickly, and without having to wait for action from a human operator. The more health checks that are incorporated into your code, the more resilient your service is to deployment issues.

Another aspect of service health is reporting metrics from the service. Metrics are important in Service Fabric because they are used to balance resource usage. Metrics also can be an indicator of system health. For example, you might have an application that has many services, and each instance reports a requests per second (RPS) metric. If one service is using more resources than another service, Service Fabric moves service instances around the cluster, to try to maintain even resource utilization. For a more detailed explanation of how resource

utilization works, see [Manage resource consumption and load in Service Fabric with metrics](#).

Metrics also can help give you insight into how your service is performing. Over time, you can use metrics to check that the service is operating within expected parameters. For example, if trends show that at 9 AM on Monday morning the average RPS is 1,000, then you might set up a health report that alerts you if the RPS is below 500 or above 1,500. Everything might be perfectly fine, but it might be worth a look to be sure that your customers are having a great experience. Your service can define a set of metrics that can be reported for health check purposes, but that don't affect the resource balancing of the cluster. To do this, set the metric weight to zero. We recommend that you start all metrics with a weight of zero, and not increase the weight until you are sure that you understand how weighting the metrics affects resource balancing for your cluster.

**TIP**

Don't use too many weighted metrics. It can be difficult to understand why service instances are being moved around for balancing. A few metrics can go a long way!

Any information that can indicate the health and performance of your application is a candidate for metrics and health reports. A CPU performance counter can tell you how your node is utilized, but it doesn't tell you whether a particular service is healthy, because multiple services might be running on a single node. But, metrics like RPS, items processed, and request latency all can indicate the health of a specific service.

To report health, use code similar to this:

```
if (!result.HasValue)
{
    HealthInformation healthInformation = new HealthInformation("ServiceCode", "StateDictionary",
    HealthState.Error);
    this.Partition.ReportInstanceHealth(healthInformation);
}
```

To report a metric, use code similar to this:

```
this.Partition.ReportLoad(new List<LoadMetric> { new LoadMetric("MemoryInMb", 1234), new
LoadMetric("metric1", 42) });
```

### Service Fabric support logs

If you need to contact Microsoft support for help with your Azure Service Fabric cluster, support logs are almost always required. If your cluster is hosted in Azure, support logs are automatically configured and collected as part of creating a cluster. The logs are stored in a dedicated storage account in your cluster's resource group. The storage account doesn't have a fixed name, but in the account, you see blob containers and tables with names that start with *fabric*. For information about setting up log collections for a standalone cluster, see [Create and manage a standalone Azure Service Fabric cluster](#) and [Configuration settings for a standalone Windows cluster](#). For standalone Service Fabric instances, the logs should be sent to a local file share. You are **required** to have these logs for support, but they are not intended to be usable by anyone outside of the Microsoft customer support team.

## Enabling Diagnostics for a cluster

In order to take advantage of these logs, it is highly recommended that during cluster creation, "Diagnostics" is enabled. By turning on diagnostics, when the cluster is deployed, Windows Azure Diagnostics is able to acknowledge the Operational, Reliable Services, and Reliable actors channels, and store the data as explained further in [Aggregate events with Azure Diagnostics](#).

As seen above, there is also an optional field to add an Application Insights (AI) instrumentation key. If you choose

to use AI for any event analysis (read more about this in [Event Analysis with Application Insights](#)), include the AppInsights resource instrumentationKey (GUID) here.

If you are going to deploy containers to your cluster, enable WAD to pick up docker stats by adding this to your "WadCfg > DiagnosticMonitorConfiguration":

```
"DockerSources": {  
    "Stats": {  
        "enabled": true,  
        "sampleRate": "PT1M"  
    }  
},
```

## Measuring performance

Measure performance of your cluster will help you understand how it is able to handle load and drive decisions around scaling your cluster (see more about scaling a cluster [on Azure](#), or [on-premises](#)). Performance data is also useful when compared to actions you or your applications and services may have taken, when analyzing logs in the future.

For a list of performance counters to collect when using Service Fabric, see [Performance Counters in Service Fabric](#)

Here are two common ways in which you can set up collecting performance data for your cluster:

- Using an agent: this is the preferred way of collecting performance from a machine, since agents usually have a list of possible performance metrics that can be collected, and it is a relatively easy process to choose the metrics you want to collect or change them. Read about [how to configure the OMS for Service Fabric](#) and [Setting up the OMS Windows Agent](#) articles to learn more about the OMS agent, which is one such monitoring agent that is able to pick up performance data for cluster VMs and deployed containers.
- Configuring diagnostics to write performance counters to a table: for clusters on Azure, this means changing the Azure Diagnostics configuration to pick up the appropriate performance counters from the VMs in your cluster, and enabling it to pick up docker stats if you will be deploying any containers. Read about configuring [Performance Counters in WAD](#) in Service Fabric to set up performance counter collection.

## Next steps

Your logs and events need to be aggregated before they can be sent to any analysis platform. Read about [EventFlow](#) and [WAD](#) to better understand some of the recommended options.

# Operational channel

7/20/2017 • 1 min to read • [Edit Online](#)

The operational channel consists of logs from high-level actions performed by Service Fabric on your nodes and your cluster. When "Diagnostics" is enabled for a cluster, the Azure Diagnostics agent is deployed on your cluster, and is by default configured to read in logs from the Operational channel. Read more about configuring the [Azure Diagnostics agent](#) to modify the diagnostics configuration of your cluster to pick up more logs or performance counters.

## Operational channel logs

Here is a comprehensive list of logs provided by Service Fabric in the operational channel.

EventID	Name	Source (Task)	Level
25620	NodeOpening	FabricNode	Informational
25621	NodeOpenedSuccess	FabricNode	Informational
25622	NodeOpenedFailed	FabricNode	Informational
25623	NodeClosing	FabricNode	Informational
25624	NodeClosed	FabricNode	Informational
25625	NodeAborting	FabricNode	Informational
25626	NodeAborted	FabricNode	Informational
29627	ClusterUpgradeStart	CM	Informational
29628	ClusterUpgradeComplete	CM	Informational
29629	ClusterUpgradeRollback	CM	Informational
29630	ClusterUpgradeRollbackComplete	CM	Informational
29631	ClusterUpgradeDomainComplete	CM	Informational
23074	ContainerActivated	Hosting	Informational
23075	ContainerDeactivated	Hosting	Informational
29620	ApplicationCreated	CM	Informational
29621	ApplicationUpgradeStart	CM	Informational

EventID	Name	Source (Task)	Level
29622	ApplicationUpgradeComplete	CM	Informational
29623	ApplicationUpgradeRollback	CM	Informational
29624	ApplicationUpgradeRollback Complete	CM	Informational
29625	ApplicationDeleted	CM	Informational
29626	ApplicationUpgradeDomainComplete	CM	Informational
18566	ServiceCreated	FM	Informational
18567	ServiceDeleted	FM	Informational

## Next steps

- Learn more about overall [event generation at the platform level](#) in Service Fabric
- Modifying your [Azure Diagnostics](#) configuration to collect more logs
- [Setting up Application Insights](#) to see your Operational channel logs

# Diagnostic functionality for Stateful Reliable Services

10/30/2017 • 1 min to read • [Edit Online](#)

The Azure Service Fabric Stateful Reliable Services StatefulServiceBase class emits [EventSource](#) events that can be used to debug the service, provide insights into how the runtime is operating, and help with troubleshooting.

## EventSource events

The EventSource name for the Stateful Reliable Services StatefulServiceBase class is "Microsoft-ServiceFabric-Services." Events from this event source appear in the [Diagnostics Events](#) window when the service is being debugged in Visual Studio.

Examples of tools and technologies that help in collecting and/or viewing EventSource events are [PerfView](#), [Azure Diagnostics](#), and the [Microsoft TraceEvent Library](#).

## Events

EVENT NAME	EVENT ID	LEVEL	EVENT DESCRIPTION
StatefulRunAsyncInvocation	1	Informational	Emitted when the service RunAsync task is started
StatefulRunAsyncCancellation	2	Informational	Emitted when the service RunAsync task is canceled
StatefulRunAsyncCompletion	3	Informational	Emitted when the service RunAsync task is finished
StatefulRunAsyncSlowCancellation	4	Warning	Emitted when the service RunAsync task takes too long to complete cancellation
StatefulRunAsyncFailure	5	Error	Emitted when the service RunAsync task throws an exception

## Interpret events

StatefulRunAsyncInvocation, StatefulRunAsyncCompletion, and StatefulRunAsyncCancellation events are useful to the service writer to understand the lifecycle of a service, as well as the timing for when a service starts, cancels, or finishes. This information can be useful when debugging service issues or understanding the service lifecycle.

Service writers should pay close attention to StatefulRunAsyncSlowCancellation and StatefulRunAsyncFailure events because they indicate issues with the service.

StatefulRunAsyncFailure is emitted whenever the service RunAsync() task throws an exception. Typically, an exception thrown indicates an error or bug in the service. Additionally, the exception causes the service to fail, so it is moved to a different node. This operation can be expensive and can delay incoming requests while the service is moved. Service writers should determine the cause of the exception and, if possible, mitigate it.

StatefulRunAsyncSlowCancellation is emitted whenever a cancellation request for the RunAsync task takes longer

than four seconds. When a service takes too long to complete cancellation, it affects the ability of the service to be quickly restarted on another node. This scenario might affect the overall availability of the service.

## Next steps

[EventSource providers in PerfView](#)

# Diagnostics and performance monitoring for Reliable Actors

10/27/2017 • 8 min to read • [Edit Online](#)

The Reliable Actors runtime emits [EventSource](#) events and [performance counters](#). These provide insights into how the runtime is operating and help with troubleshooting and performance monitoring.

## EventSource events

The EventSource provider name for the Reliable Actors runtime is "Microsoft-ServiceFabric-Actors". Events from this event source appear in the [Diagnostics Events](#) window when the actor application is being [debugged in Visual Studio](#).

Examples of tools and technologies that help in collecting and/or viewing EventSource events are [PerfView](#), [Azure Diagnostics](#), [Semantic Logging](#), and the [Microsoft TraceEvent Library](#).

### Keywords

All events that belong to the Reliable Actors EventSource are associated with one or more keywords. This enables filtering of events that are collected. The following keyword bits are defined.

BIT	DESCRIPTION
0x1	Set of important events that summarize the operation of the Fabric Actors runtime.
0x2	Set of events that describe actor method calls. For more information, see the <a href="#">introductory topic on actors</a> .
0x4	Set of events related to actor state. For more information, see the topic on <a href="#">actor state management</a> .
0x8	Set of events related to turn-based concurrency in the actor. For more information, see the topic on <a href="#">concurrency</a> .

## Performance counters

The Reliable Actors runtime defines the following performance counter categories.

CATEGORY	DESCRIPTION
Service Fabric Actor	Counters specific to Azure Service Fabric actors, e.g. time taken to save actor state
Service Fabric Actor Method	Counters specific to methods implemented by Service Fabric actors, e.g. how often an actor method is invoked

Each of the above categories has one or more counters.

The [Windows Performance Monitor](#) application that is available by default in the Windows operating system can be used to collect and view performance counter data. [Azure Diagnostics](#) is another option for collecting

performance counter data and uploading it to Azure tables.

## Performance counter instance names

A cluster that has a large number of actor services or actor service partitions will have a large number of actor performance counter instances. The performance counter instance names can help in identifying the specific [partition](#) and actor method (if applicable) that the performance counter instance is associated with.

### Service Fabric Actor category

For the category `Service Fabric Actor`, the counter instance names are in the following format:

`ServiceFabricPartitionID_ActorsRuntimeInternalID`

*ServiceFabricPartitionID* is the string representation of the Service Fabric partition ID that the performance counter instance is associated with. The partition ID is a GUID, and its string representation is generated through the `Guid.ToString` method with format specifier "D".

*ActorRuntimeInternalID* is the string representation of a 64-bit integer that is generated by the Fabric Actors runtime for its internal use. This is included in the performance counter instance name to ensure its uniqueness and avoid conflict with other performance counter instance names. Users should not try to interpret this portion of the performance counter instance name.

The following is an example of a counter instance name for a counter that belongs to the `Service Fabric Actor` category:

`2740af29-78aa-44bc-a20b-7e60fb783264_635650083799324046`

In the example above, `2740af29-78aa-44bc-a20b-7e60fb783264` is the string representation of the Service Fabric partition ID, and `635650083799324046` is the 64-bit ID that is generated for the runtime's internal use.

### Service Fabric Actor Method category

For the category `Service Fabric Actor Method`, the counter instance names are in the following format:

`MethodName_ActorsRuntimeMethodId_ServiceFabricPartitionID_ActorsRuntimeInternalID`

*MethodName* is the name of the actor method that the performance counter instance is associated with. The format of the method name is determined based on some logic in the Fabric Actors runtime that balances the readability of the name with constraints on the maximum length of the performance counter instance names on Windows.

*ActorsRuntimeMethodId* is the string representation of a 32-bit integer that is generated by the Fabric Actors runtime for its internal use. This is included in the performance counter instance name to ensure its uniqueness and avoid conflict with other performance counter instance names. Users should not try to interpret this portion of the performance counter instance name.

*ServiceFabricPartitionID* is the string representation of the Service Fabric partition ID that the performance counter instance is associated with. The partition ID is a GUID, and its string representation is generated through the `Guid.ToString` method with format specifier "D".

*ActorRuntimeInternalID* is the string representation of a 64-bit integer that is generated by the Fabric Actors runtime for its internal use. This is included in the performance counter instance name to ensure its uniqueness and avoid conflict with other performance counter instance names. Users should not try to interpret this portion of the performance counter instance name.

The following is an example of a counter instance name for a counter that belongs to the

`Service Fabric Actor Method` category:

`ivoicemailboxactor.leavemessageasync_2_89383d32-e57e-4a9b-a6ad-57c6792aa521_635650083804480486`

In the example above, `ivoicemailboxactor.leavemessageasync` is the method name, `2` is the 32-bit ID generated

for the runtime's internal use, `89383d32-e57e-4a9b-a6ad-57c6792aa521` is the string representation of the Service Fabric partition ID, and `635650083804480486` is the 64-bit ID generated for the runtime's internal use.

## List of events and performance counters

### Actor method events and performance counters

The Reliable Actors runtime emits the following events related to [actor methods](#).

Event Name	Event ID	Level	Keyword	Description
ActorMethodStart	7	Verbose	0x2	Actors runtime is about to invoke an actor method.
ActorMethodStop	8	Verbose	0x2	An actor method has finished executing. That is, the runtime's asynchronous call to the actor method has returned, and the task returned by the actor method has finished.
ActorMethodThrewException	9	Warning	0x3	An exception was thrown during the execution of an actor method, either during the runtime's asynchronous call to the actor method or during the execution of the task returned by the actor method. This event indicates some sort of failure in the actor code that needs investigation.

The Reliable Actors runtime publishes the following performance counters related to the execution of actor methods.

Category Name	Counter Name	Description
Service Fabric Actor Method	Invocations/Sec	Number of times that the actor service method is invoked per second
Service Fabric Actor Method	Average milliseconds per invocation	Time taken to execute the actor service method in milliseconds
Service Fabric Actor Method	Exceptions thrown/Sec	Number of times that the actor service method threw an exception per second

### Concurrency events and performance counters

The Reliable Actors runtime emits the following events related to [concurrency](#).

Event Name	Event ID	Level	Keyword	Description
ActorMethodCallsWaitingForLock	12	Verbose	0x8	This event is written at the start of each new turn in an actor. It contains the number of pending actor calls that are waiting to acquire the per-actor lock that enforces turn-based concurrency.

The Reliable Actors runtime publishes the following performance counters related to concurrency.

Category Name	Counter Name	Description
Service Fabric Actor	# of actor calls waiting for actor lock	Number of pending actor calls waiting to acquire the per-actor lock that enforces turn-based concurrency
Service Fabric Actor	Average milliseconds per lock wait	Time taken (in milliseconds) to acquire the per-actor lock that enforces turn-based concurrency
Service Fabric Actor	Average milliseconds actor lock held	Time (in milliseconds) for which the per-actor lock is held

### Actor state management events and performance counters

The Reliable Actors runtime emits the following events related to [actor state management](#).

Event Name	Event ID	Level	Keyword	Description
ActorSaveStateStart	10	Verbose	0x4	Actors runtime is about to save the actor state.
ActorSaveStateStop	11	Verbose	0x4	Actors runtime has finished saving the actor state.

The Reliable Actors runtime publishes the following performance counters related to actor state management.

Category Name	Counter Name	Description
Service Fabric Actor	Average milliseconds per save state operation	Time taken to save actor state in milliseconds
Service Fabric Actor	Average milliseconds per load state operation	Time taken to load actor state in milliseconds

### Events related to actor replicas

The Reliable Actors runtime emits the following events related to [actor replicas](#).

Event Name	Event ID	Level	Keyword	Description
ReplicaChangeRoleToPrimary	1	Informational	0x1	Actor replica changed role to Primary. This implies that the actors for this partition will be created inside this replica.
ReplicaChangeRoleFromPrimary	2	Informational	0x1	Actor replica changed role to non-Primary. This implies that the actors for this partition will no longer be created inside this replica. No new requests will be delivered to actors already created within this replica. The actors will be destroyed after any in-progress requests are completed.

### Actor activation and deactivation events and performance counters

The Reliable Actors runtime emits the following events related to [actor activation and deactivation](#).

Event Name	Event ID	Level	Keyword	Description
ActorActivated	5	Informational	0x1	An actor has been activated.
ActorDeactivated	6	Informational	0x1	An actor has been deactivated.

The Reliable Actors runtime publishes the following performance counters related to actor activation and deactivation.

Category Name	Counter Name	Description
Service Fabric Actor	Average OnActivateAsync milliseconds	Time taken to execute OnActivateAsync method in milliseconds

### Actor request processing performance counters

When a client invokes a method via an actor proxy object, it results in a request message being sent over the network to the actor service. The service processes the request message and sends a response back to the client. The Reliable Actors runtime publishes the following performance counters related to actor request processing.

Category Name	Counter Name	Description
Service Fabric Actor	# of outstanding requests	Number of requests being processed in the service

CATEGORY NAME	COUNTER NAME	DESCRIPTION
Service Fabric Actor	Average milliseconds per request	Time taken (in milliseconds) by the service to process a request
Service Fabric Actor	Average milliseconds for request deserialization	Time taken (in milliseconds) to deserialize actor request message when it is received at the service
Service Fabric Actor	Average milliseconds for response serialization	Time taken (in milliseconds) to serialize the actor response message at the service before the response is sent to the client

## Next steps

- [How Reliable Actors use the Service Fabric platform](#)
- [Actor API reference documentation](#)
- [Sample code](#)
- [EventSource providers in PerfView](#)

# Performance metrics

10/16/2017 • 2 min to read • [Edit Online](#)

Metrics should be collected to understand the performance of your cluster as well as the applications running in it. For Service Fabric clusters, we recommend collecting the following performance counters.

## Nodes

For the machines in your cluster, consider collecting the following performance counters to better understand the load on each machine and make appropriate cluster scaling decisions.

COUNTER CATEGORY	COUNTER NAME
PhysicalDisk(per Disk)	Avg. Disk Read Queue Length
PhysicalDisk(per Disk)	Avg. Disk Write Queue Length
PhysicalDisk(per Disk)	Avg. Disk sec/Read
PhysicalDisk(per Disk)	Avg. Disk sec/Write
PhysicalDisk(per Disk)	Disk Reads/sec
PhysicalDisk(per Disk)	Disk Read Bytes/sec
PhysicalDisk(per Disk)	Disk Writes/sec
PhysicalDisk(per Disk)	Disk Write Bytes/sec
Memory	Available MBytes
PagingFile	% Usage
Processor(Total)	% Processor Time
Process (per service)	% Processor Time
Process (per service)	ID Process
Process (per service)	Private Bytes
Process (per service)	Thread Count
Process (per service)	Virtual Bytes
Process (per service)	Working Set
Process (per service)	Working Set - Private

COUNTER CATEGORY	COUNTER NAME
Network Interface(all-instances)	Output Queue Length
Network Interface(all-instances)	Packets Outbound Discarded
Network Interface(all-instances)	Packets Received Discarded
Network Interface(all-instances)	Packets Outbound Errors
Network Interface(all-instances)	Packets Received Errors

## .NET applications and services

Collect the following counters if you are deploying .NET services to your cluster.

COUNTER CATEGORY	COUNTER NAME
.NET CLR Memory (per service)	Process ID
.NET CLR Memory (per service)	# Total committed Bytes
.NET CLR Memory (per service)	# Total reserved Bytes
.NET CLR Memory (per service)	# Bytes in all Heaps
.NET CLR Memory (per service)	# Gen 0 Collections
.NET CLR Memory (per service)	# Gen 1 Collections
.NET CLR Memory (per service)	# Gen 2 Collections
.NET CLR Memory (per service)	% Time in GC

### Service Fabric's custom performance counters

Service Fabric generates a substantial amount of custom performance counters. If you have the SDK installed, you can see the comprehensive list on your Windows machine in your Performance Monitor application (Start > Performance Monitor).

In the applications you are deploying to your cluster, if you are using Reliable Actors, add counts from `Service Fabric Actor` and `Service Fabric Actor Method` categories (see [Service Fabric Reliable Actors Diagnostics](#)).

If you use Reliable Services, we similarly have `Service Fabric Service` and `Service Fabric Service Method` counter categories that you should collect counters from.

If you use Reliable Collections, we recommend adding the `Avg. Transaction ms/Commit` from the `Service Fabric Transactional Replicator` to collect the average commit latency per transaction metric.

## Next steps

- Learn more about [event generation at the platform level](#) in Service Fabric
- Collect performance metrics through [Azure Diagnostics](#)

# Application and service level event and log generation

10/16/2017 • 6 min to read • [Edit Online](#)

## Instrumenting the code with custom events

Instrumenting the code is the basis for most other aspects of monitoring your services. Instrumentation is the only way you can know that something is wrong, and to diagnose what needs to be fixed. Although technically it's possible to connect a debugger to a production service, it's not a common practice. So, having detailed instrumentation data is important.

Some products automatically instrument your code. Although these solutions can work well, manual instrumentation is almost always required. In the end, you must have enough information to forensically debug the application. This document describes different approaches to instrumenting your code, and when to choose one approach over another.

## EventSource

When you create a Service Fabric solution from a template in Visual Studio, an **EventSource**-derived class (**ServiceEventSource** or **ActorEventSource**) is generated. A template is created, in which you can add events for your application or service. The **EventSource** name **must** be unique, and should be renamed from the default template string MyCompany-<solution>-<project>. Having multiple **EventSource** definitions that use the same name causes an issue at run time. Each defined event must have a unique identifier. If an identifier is not unique, a runtime failure occurs. Some organizations preassign ranges of values for identifiers to avoid conflicts between separate development teams. For more information, see [Vance's blog](#) or the [MSDN documentation](#).

### Using structured EventSource events

Each of the events in the code examples in this section are defined for a specific case, for example, when a service type is registered. When you define messages by use case, data can be packaged with the text of the error, and you can more easily search and filter based on the names or values of the specified properties. Structuring the instrumentation output makes it easier to consume, but requires more thought and time to define a new event for each use case. Some event definitions can be shared across the entire application. For example, a method start or stop event would be reused across many services within an application. A domain-specific service, like an order system, might have a **CreateOrder** event, which has its own unique event. This approach might generate many events, and potentially require coordination of identifiers across project teams.

```

[EventSource(Name = "MyCompany-VotingState-VotingStateService")]
internal sealed class ServiceEventSource : EventSource
{
    public static readonly ServiceEventSource Current = new ServiceEventSource();

    // The instance constructor is private to enforce singleton semantics.
    private ServiceEventSource() : base() { }

    ...

    // The ServiceTypeRegistered event contains a unique identifier, an event attribute that defined the
    event, and the code implementation of the event.
    private const int ServiceTypeRegisteredEventId = 3;
    [Event(ServiceTypeRegisteredEventId, Level = EventLevel.Informational, Message = "Service host process
{0} registered service type {1}", Keywords = Keywords.ServiceInitialization)]
    public void ServiceTypeRegistered(int hostProcessId, string serviceType)
    {
        WriteEvent(ServiceTypeRegisteredEventId, hostProcessId, serviceType);
    }

    // The ServiceHostInitializationFailed event contains a unique identifier, an event attribute that
    defined the event, and the code implementation of the event.
    private const int ServiceHostInitializationFailedEventId = 4;
    [Event(ServiceHostInitializationFailedEventId, Level = EventLevel.Error, Message = "Service host
initialization failed", Keywords = Keywords.ServiceInitialization)]
    public void ServiceHostInitializationFailed(string exception)
    {
        WriteEvent(ServiceHostInitializationFailedEventId, exception);
    }
}

```

## Using EventSource generically

Because defining specific events can be difficult, many people define a few events with a common set of parameters that generally output their information as a string. Much of the structured aspect is lost, and it's more difficult to search and filter the results. In this approach, a few events that usually correspond to the logging levels are defined. The following snippet defines a debug and error message:

```

[EventSource(Name = "MyCompany-VotingState-VotingStateService")]
internal sealed class ServiceEventSource : EventSource
{
    public static readonly ServiceEventSource Current = new ServiceEventSource();

    // The Instance constructor is private, to enforce singleton semantics.
    private ServiceEventSource() : base() { }

    ...

    private const int DebugEventId = 10;
    [Event(DebugEventId, Level = EventLevel.Verbose, Message = "{0}")]
    public void Debug(string msg)
    {
        WriteEvent(DebugEventId, msg);
    }

    private const int ErrorEventId = 11;
    [Event(ErrorEventId, Level = EventLevel.Error, Message = "Error: {0} - {1}")]
    public void Error(string error, string msg)
    {
        WriteEvent(ErrorEventId, error, msg);
    }
}

```

Using a hybrid of structured and generic instrumentation also can work well. Structured instrumentation is used for reporting errors and metrics. Generic events can be used for the detailed logging that is consumed by

engineers for troubleshooting.

## ASP.NET Core logging

It's important to carefully plan how you will instrument your code. The right instrumentation plan can help you avoid potentially destabilizing your code base, and then needing to reinstrument the code. To reduce risk, you can choose an instrumentation library like [Microsoft.Extensions.Logging](#), which is part of Microsoft ASP.NET Core. ASP.NET Core has an [ILogger](#) interface that you can use with the provider of your choice, while minimizing the effect on existing code. You can use the code in ASP.NET Core on Windows and Linux, and in the full .NET Framework, so your instrumentation code is standardized. This is further explored below:

### Using Microsoft.Extensions.Logging in Service Fabric

1. Add the Microsoft.Extensions.Logging NuGet package to the project you want to instrument. Also, add any provider packages (for a third-party package, see the following example). For more information, see [Logging in ASP.NET Core](#).
2. Add a **using** directive for Microsoft.Extensions.Logging to your service file.
3. Define a private variable within your service class.

```
private ILogger _logger = null;
```

4. In the constructor of your service class, add this code:

```
_logger = new LoggerFactory().CreateLogger<Stateless>();
```

5. Start instrumenting your code in your methods. Here are a few samples:

```
_logger.LogDebug("Debug-level event from Microsoft.Logging");
_logger.LogInformation("Informational-level event from Microsoft.Logging");

// In this variant, we're adding structured properties RequestName and Duration, which have values
MyRequest and the duration of the request.
// Later in the article, we discuss why this step is useful.
_logger.LogInformation("{RequestName} {Duration}", "MyRequest", requestDuration);
```

## Using other logging providers

Some third-party providers use the approach described in the preceding section, including [Serilog](#), [NLog](#), and [Loggr](#). You can plug each of these into ASP.NET Core logging, or you can use them separately. Serilog has a feature that enriches all messages sent from a logger. This feature can be useful to output the service name, type, and partition information. To use this capability in the ASP.NET Core infrastructure, do these steps:

1. Add the Serilog, Serilog.Extensions.Logging, and Serilog.Sinks.Observable NuGet packages to the project. For the next example, also add Serilog.Sinks.Literate. A better approach is shown later in this article.
2. In Serilog, create a LoggerConfiguration and the logger instance.

```
Log.Logger = new LoggerConfiguration().WriteTo.LiterateConsole().CreateLogger();
```

3. Add a Serilog.ILogger argument to the service constructor, and pass the newly created logger.

```
ServiceRuntime.RegisterServiceAsync("StatelessType", context => new Stateless(context,
Log.Logger)).GetAwaiter().GetResult();
```

4. In the service constructor, add the following code, which creates the property enrichers for the **ServiceTypeName**, **ServiceName**, **PartitionId**, and **InstanceId** properties of the service. It also adds a property enricher to the ASP.NET Core logging factory, so you can use `Microsoft.Extensions.Logging.ILogger` in your code.

```
public Stateless(StatelessServiceContext context, Serilog.ILogger serilog)
    : base(context)
{
    PropertyEnricher[] properties = new PropertyEnricher[]
    {
        new PropertyEnricher("ServiceTypeName", context.ServiceTypeName),
        new PropertyEnricher("ServiceName", context.ServiceName),
        new PropertyEnricher("PartitionId", context.PartitionId),
        new PropertyEnricher("InstanceId", context.ReplicaOrInstanceId),
    };

    serilog.ForContext(properties);

    _logger = new LoggerFactory().AddSerilog(serilog.ForContext(properties)).CreateLogger<Stateless>();
}
```

5. Instrument the code the same as if you were using ASP.NET Core without Serilog.

#### NOTE

We recommend that you don't use the static `Log.Logger` with the preceding example. Service Fabric can host multiple instances of the same service type within a single process. If you use the static `Log.Logger`, the last writer of the property enrichers will show values for all instances that are running. This is one reason why the `_logger` variable is a private member variable of the service class. Also, you must make the `_logger` available to common code, which might be used across services.

## Choosing a logging provider

If your application relies on high performance, **EventSource** is usually a good approach. **EventSource** generally uses fewer resources and performs better than ASP.NET Core logging or any of the available third-party solutions. This isn't an issue for many services, but if your service is performance-oriented, using **EventSource** might be a better choice. However, to get these benefits of structured logging, **EventSource** requires a larger investment from your engineering team. If possible, do a quick prototype of a few logging options, and then choose the one that best meets your needs.

## Next steps

Once you have chosen your logging provider to instrument your applications and services, your logs and events need to be aggregated before they can be sent to any analysis platform. Read about [EventFlow](#) and [WAD](#) to better understand some of the recommended options.

# Introduction to Service Fabric health monitoring

9/19/2017 • 20 min to read • [Edit Online](#)

Azure Service Fabric introduces a health model that provides rich, flexible, and extensible health evaluation and reporting. The model allows near-real-time monitoring of the state of the cluster and the services running in it. You can easily obtain health information and correct potential issues before they cascade and cause massive outages. In the typical model, services send reports based on their local views, and that information is aggregated to provide an overall cluster-level view.

Service Fabric components use this rich health model to report their current state. You can use the same mechanism to report health from your applications. If you invest in high-quality health reporting that captures your custom conditions, you can detect and fix issues for your running application much more easily.

The following Microsoft Virtual Academy video also describes the Service Fabric health model and how it's used:



## NOTE

We started the health subsystem to address a need for monitored upgrades. Service Fabric provides monitored application and cluster upgrades that ensure full availability, no downtime and minimal to no user intervention. To achieve these goals, the upgrade checks health based on configured upgrade policies. An upgrade can proceed only when health respects desired thresholds. Otherwise, the upgrade is either automatically rolled back or paused to give administrators a chance to fix the issues. To learn more about application upgrades, see [this article](#).

## Health store

The health store keeps health-related information about entities in the cluster for easy retrieval and evaluation. It is implemented as a Service Fabric persisted stateful service to ensure high availability and scalability. The health store is part of the **fabric:/System** application, and it is available when the cluster is up and running.

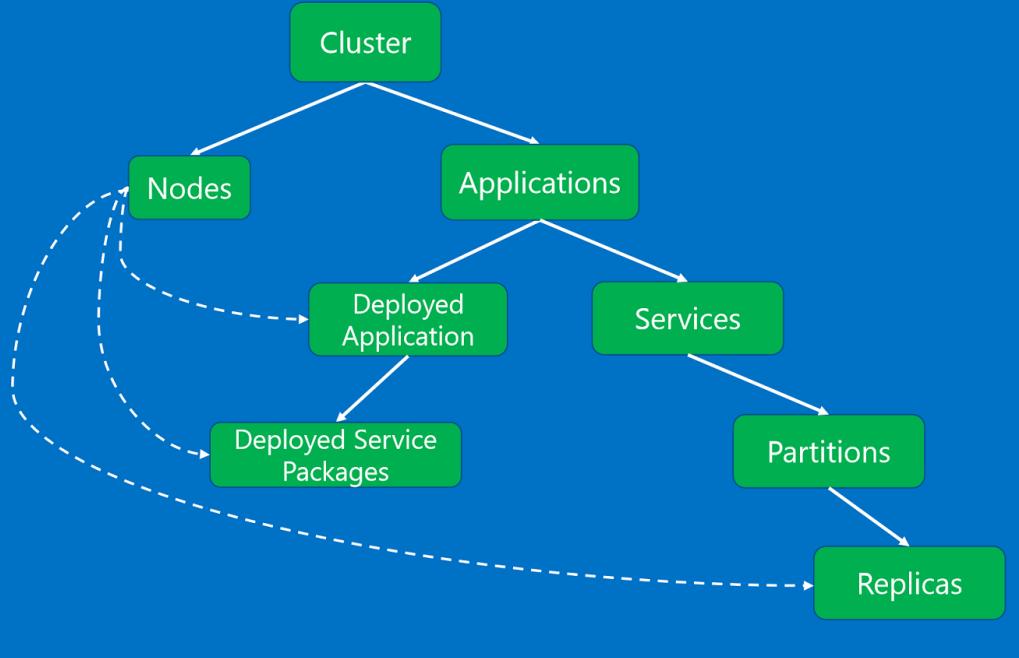
## Health entities and hierarchy

The health entities are organized in a logical hierarchy that captures interactions and dependencies among different entities. The health store automatically builds health entities and hierarchy based on reports received from Service Fabric components.

The health entities mirror the Service Fabric entities. (For example, **health application entity** matches an application instance deployed in the cluster, while **health node entity** matches a Service Fabric cluster node.) The health hierarchy captures the interactions of the system entities, and it is the basis for advanced health evaluation. You can learn about key Service Fabric concepts in [Service Fabric technical overview](#). For more on application, see [Service Fabric application model](#).

The health entities and hierarchy allow the cluster and applications to be effectively reported, debugged, and monitored. The health model provides an accurate, *granular* representation of the health of the many moving pieces in the cluster.

## Health Hierarchy



The health entities, organized in a hierarchy based on parent-child relationships.

The health entities are:

- **Cluster.** Represents the health of a Service Fabric cluster. Cluster health reports describe conditions that affect the entire cluster. These conditions affect multiple entities in the cluster or the cluster itself. Based on the condition, the reporter can't narrow the issue down to one or more unhealthy children. Examples include the brain of the cluster splitting due to network partitioning or communication issues.
- **Node.** Represents the health of a Service Fabric node. Node health reports describe conditions that affect the node functionality. They typically affect all the deployed entities running on it. Examples include node out of disk space (or other machine-wide properties, such as memory, connections) and when a node is down. The node entity is identified by the node name (string).
- **Application.** Represents the health of an application instance running in the cluster. Application health reports describe conditions that affect the overall health of the application. They can't be narrowed down to individual children (services or deployed applications). Examples include the end-to-end interaction among different services in the application. The application entity is identified by the application name (URI).
- **Service.** Represents the health of a service running in the cluster. Service health reports describe conditions that affect the overall health of the service. The reporter can't narrow down the issue to an unhealthy partition or replica. Examples include a service configuration (such as port or external file share) that is causing issues for all partitions. The service entity is identified by the service name (URI).
- **Partition.** Represents the health of a service partition. Partition health reports describe conditions that affect the entire replica set. Examples include when the number of replicas is below target count and when a partition is in quorum loss. The partition entity is identified by the partition ID (GUID).
- **Replica.** Represents the health of a stateful service replica or a stateless service instance. The replica is the smallest unit that watchdogs and system components can report on for an application. For stateful services, examples include a primary replica that can't replicate operations to secondaries and slow replication. Also, a stateless instance can report when it is running out of resources or has connectivity issues. The replica entity is identified by the partition ID (GUID) and the replica or instance ID (long).

- **DeployedApplication**. Represents the health of an *application running on a node*. Deployed application health reports describe conditions specific to the application on the node that can't be narrowed down to service packages deployed on the same node. Examples include errors when application package can't be downloaded on that node and issues setting up application security principals on the node. The deployed application is identified by application name (URI) and node name (string).
- **DeployedServicePackage**. Represents the health of a service package running on a node in the cluster. It describes conditions specific to a service package that do not affect the other service packages on the same node for the same application. Examples include a code package in the service package that cannot be started and a configuration package that cannot be read. The deployed service package is identified by application name (URI), node name (string), service manifest name (string), and service package activation ID (string).

The granularity of the health model makes it easy to detect and correct issues. For example, if a service is not responding, it is feasible to report that the application instance is unhealthy. Reporting at that level is not ideal, however, because the issue might not be affecting all the services within that application. The report should be applied to the unhealthy service or to a specific child partition, if more information points to that partition. The data automatically surfaces through the hierarchy, and an unhealthy partition is made visible at service and application levels. This aggregation helps to pinpoint and resolve the root cause of the issue more quickly.

The health hierarchy is composed of parent-child relationships. A cluster is composed of nodes and applications. Applications have services and deployed applications. Deployed applications have deployed service packages. Services have partitions, and each partition has one or more replicas. There is a special relationship between nodes and deployed entities. An unhealthy node as reported by its authority system component, the Failover Manager service, affects the deployed applications, service packages, and replicas deployed on it.

The health hierarchy represents the latest state of the system based on the latest health reports, which is almost real-time information. Internal and external watchdogs can report on the same entities based on application-specific logic or custom monitored conditions. User reports coexist with the system reports.

Plan to invest in how to report and respond to health during the design of a large cloud service. This up-front investment makes the service easier to debug, monitor, and operate.

## Health states

Service Fabric uses three health states to describe whether an entity is healthy or not: OK, warning, and error. Any report sent to the health store must specify one of these states. The health evaluation result is one of these states.

The possible [health states](#) are:

- **OK**. The entity is healthy. There are no known issues reported on it or its children (when applicable).
- **Warning**. The entity has some issues, but it can still function correctly. For example, there are delays, but they do not cause any functional issues yet. In some cases, the warning condition may fix itself without external intervention. In these cases, health reports raise awareness and provide visibility into what is going on. In other cases, the warning condition may degrade into a severe problem without user intervention.
- **Error**. The entity is unhealthy. Action should be taken to fix the state of the entity, because it can't function properly.
- **Unknown**. The entity doesn't exist in the health store. This result can be obtained from the distributed queries that merge results from multiple components. For example, the get node list query goes to **FailoverManager**, **ClusterManager**, and **HealthManager**; get application list query goes to **ClusterManager** and **HealthManager**. These queries merge results from multiple system components. If another system component returns an entity that is not present in health store, the merged result has unknown health state. An entity is not in store because health reports have not yet been processed or the

entity has been cleaned up after deletion.

## Health policies

The health store applies health policies to determine whether an entity is healthy based on its reports and its children.

### NOTE

Health policies can be specified in the cluster manifest (for cluster and node health evaluation) or in the application manifest (for application evaluation and any of its children). Health evaluation requests can also pass in custom health evaluation policies, which are used only for that evaluation.

By default, Service Fabric applies strict rules (everything must be healthy) for the parent-child hierarchical relationship. If even one of the children has one unhealthy event, the parent is considered unhealthy.

### Cluster health policy

The [cluster health policy](#) is used to evaluate the cluster health state and node health states. The policy can be defined in the cluster manifest. If it is not present, the default policy (zero tolerated failures) is used. The cluster health policy contains:

- [ConsiderWarningAsError](#). Specifies whether to treat warning health reports as errors during health evaluation. Default: false.
- [MaxPercentUnhealthyApplications](#). Specifies the maximum tolerated percentage of applications that can be unhealthy before the cluster is considered in error.
- [MaxPercentUnhealthyNodes](#). Specifies the maximum tolerated percentage of nodes that can be unhealthy before the cluster is considered in error. In large clusters, some nodes are always down or out for repairs, so this percentage should be configured to tolerate that.
- [ApplicationTypeHealthPolicyMap](#). The application type health policy map can be used during cluster health evaluation to describe special application types. By default, all applications are put into a pool and evaluated with MaxPercentUnhealthyApplications. If some application types should be treated differently, they can be taken out of the global pool. Instead, they are evaluated against the percentages associated with their application type name in the map. For example, in a cluster there are thousands of applications of different types, and a few control application instances of a special application type. The control applications should never be in error. You can specify global MaxPercentUnhealthyApplications to 20% to tolerate some failures, but for the application type "ControlApplicationType" set the MaxPercentUnhealthyApplications to 0. This way, if some of the many applications are unhealthy, but below the global unhealthy percentage, the cluster would be evaluated to Warning. A warning health state does not impact cluster upgrade or other monitoring triggered by Error health state. But even one control application in error would make cluster unhealthy, which triggers roll back or pauses the cluster upgrade, depending on the upgrade configuration. For the application types defined in the map, all application instances are taken out of the global pool of applications. They are evaluated based on the total number of applications of the application type, using the specific MaxPercentUnhealthyApplications from the map. All the rest of the applications remain in the global pool and are evaluated with MaxPercentUnhealthyApplications.

The following example is an excerpt from a cluster manifest. To define entries in the application type map, prefix the parameter name with "ApplicationTypeMaxPercentUnhealthyApplications-", followed by the application type name.

```

<FabricSettings>
  <Section Name="HealthManager/ClusterHealthPolicy">
    <Parameter Name="ConsiderWarningAsError" Value="False" />
    <Parameter Name="MaxPercentUnhealthyApplications" Value="20" />
    <Parameter Name="MaxPercentUnhealthyNodes" Value="20" />
    <Parameter Name="ApplicationTypeMaxPercentUnhealthyApplications-ControlApplicationType" Value="0" />
  </Section>
</FabricSettings>

```

## Application health policy

The [application health policy](#) describes how the evaluation of events and child-states aggregation is done for applications and their children. It can be defined in the application manifest, **ApplicationManifest.xml**, in the application package. If no policies are specified, Service Fabric assumes that the entity is unhealthy if it has a health report or a child at the warning or error health state. The configurable policies are:

- [ConsiderWarningAsError](#). Specifies whether to treat warning health reports as errors during health evaluation. Default: false.
- [MaxPercentUnhealthyDeployedApplications](#). Specifies the maximum tolerated percentage of deployed applications that can be unhealthy before the application is considered in error. This percentage is calculated by dividing the number of unhealthy deployed applications over the number of nodes that the applications are currently deployed on in the cluster. The computation rounds up to tolerate one failure on small numbers of nodes. Default percentage: zero.
- [DefaultServiceTypeHealthPolicy](#). Specifies the default service type health policy, which replaces the default health policy for all service types in the application.
- [ServiceTypeHealthPolicyMap](#). Provides a map of service health policies per service type. These policies replace the default service type health policies for each specified service type. For example, if an application has a stateless gateway service type and a stateful engine service type, you can configure the health policies for their evaluation differently. When you specify policy per service type, you can gain more granular control of the health of the service.

## Service type health policy

The [service type health policy](#) specifies how to evaluate and aggregate the services and the children of services. The policy contains:

- [MaxPercentUnhealthyPartitionsPerService](#). Specifies the maximum tolerated percentage of unhealthy partitions before a service is considered unhealthy. Default percentage: zero.
- [MaxPercentUnhealthyReplicasPerPartition](#). Specifies the maximum tolerated percentage of unhealthy replicas before a partition is considered unhealthy. Default percentage: zero.
- [MaxPercentUnhealthyServices](#). Specifies the maximum tolerated percentage of unhealthy services before the application is considered unhealthy. Default percentage: zero.

The following example is an excerpt from an application manifest:

```

<Policies>
  <HealthPolicy ConsiderWarningAsError="true" MaxPercentUnhealthyDeployedApplications="20">
    <DefaultServiceTypeHealthPolicy
      MaxPercentUnhealthyServices="0"
      MaxPercentUnhealthyPartitionsPerService="10"
      MaxPercentUnhealthyReplicasPerPartition="0"/>
    <ServiceTypeHealthPolicy ServiceTypeName="FrontEndServiceType"
      MaxPercentUnhealthyServices="0"
      MaxPercentUnhealthyPartitionsPerService="20"
      MaxPercentUnhealthyReplicasPerPartition="0"/>
    <ServiceTypeHealthPolicy ServiceTypeName="BackEndServiceType"
      MaxPercentUnhealthyServices="20"
      MaxPercentUnhealthyPartitionsPerService="0"
      MaxPercentUnhealthyReplicasPerPartition="0"/>
  </ServiceTypeHealthPolicy>
</HealthPolicy>
</Policies>

```

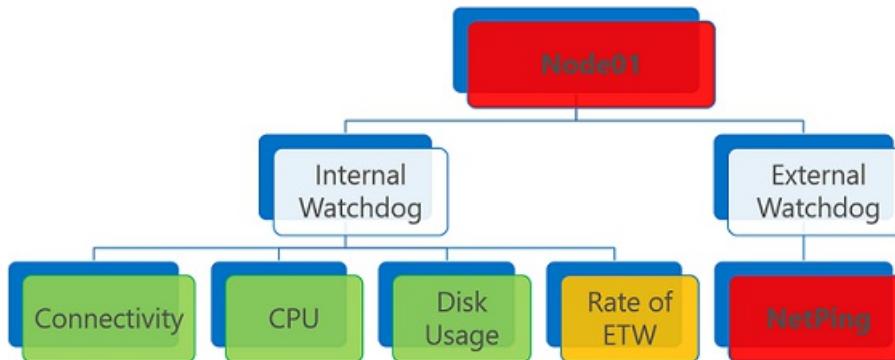
## Health evaluation

Users and automated services can evaluate health for any entity at any time. To evaluate an entity's health, the health store aggregates all health reports on the entity and evaluates all its children (when applicable). The health aggregation algorithm uses health policies that specify how to evaluate health reports and how to aggregate child health states (when applicable).

### Health report aggregation

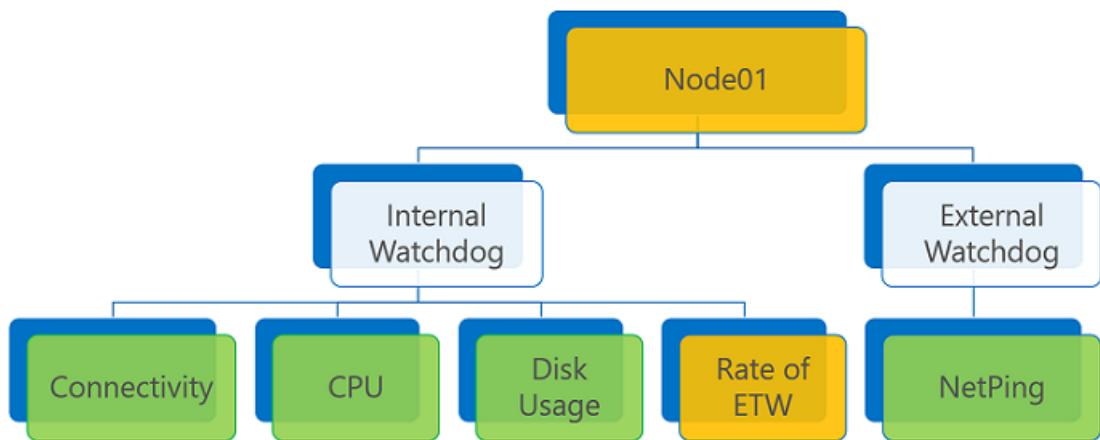
One entity can have multiple health reports sent by different reporters (system components or watchdogs) on different properties. The aggregation uses the associated health policies, in particular the ConsiderWarningAsError member of application or cluster health policy. ConsiderWarningAsError specifies how to evaluate warnings.

The aggregated health state is triggered by the *worst* health reports on the entity. If there is at least one error health report, the aggregated health state is an error.



A health entity that has one or more error health reports is evaluated as Error. The same is true for an expired health report, regardless of its health state.

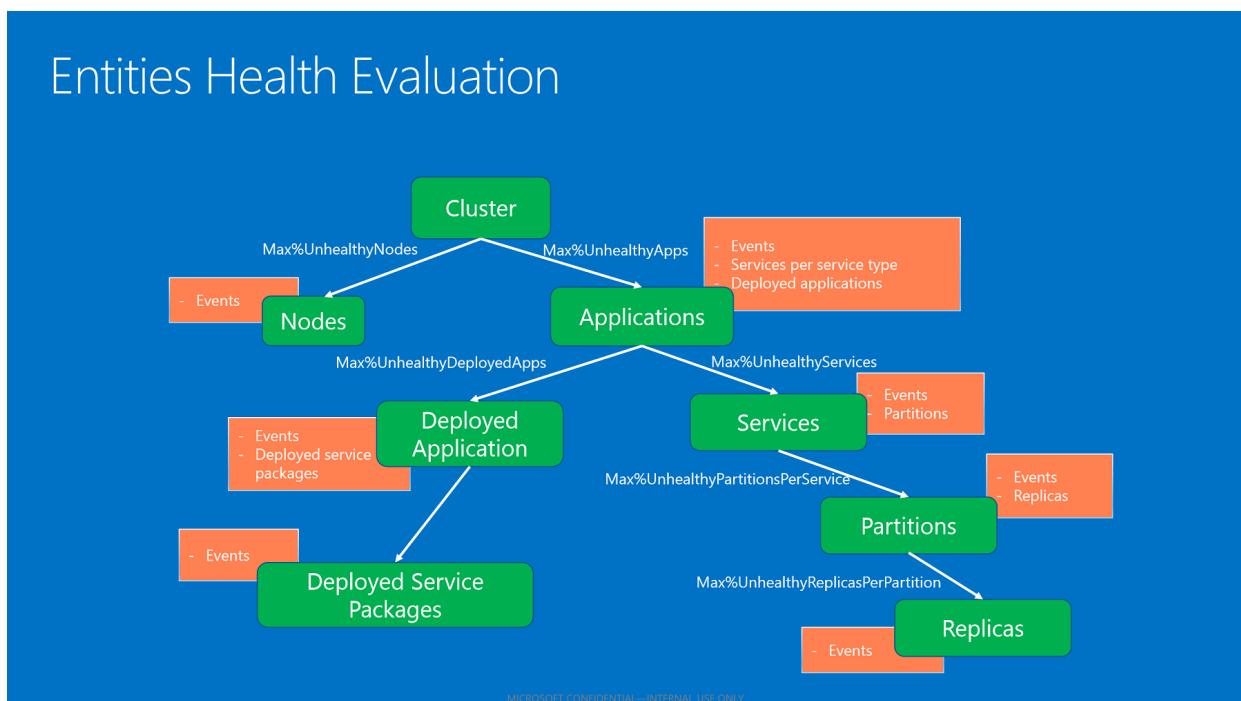
If there are no error reports and one or more warnings, the aggregated health state is either warning or error, depending on the ConsiderWarningAsError policy flag.



Health report aggregation with warning report and ConsiderWarningAsError set to false (default).

### Child health aggregation

The aggregated health state of an entity reflects the child health states (when applicable). The algorithm for aggregating child health states uses the health policies applicable based on the entity type.



Child aggregation based on health policies.

After the health store has evaluated all the children, it aggregates their health states based on the configured maximum percentage of unhealthy children. This percentage is taken from the policy based on the entity and child type.

- If all children have OK states, the child aggregated health state is OK.
- If children have both OK and warning states, the child aggregated health state is warning.
- If there are children with error states that do not respect the maximum allowed percentage of unhealthy children, the aggregated health state is an error.
- If the children with error states respect the maximum allowed percentage of unhealthy children, the aggregated health state is warning.

## Health reporting

System components, System Fabric applications, and internal/external watchdogs can report against Service Fabric entities. The reporters make *local* determinations of the health of the monitored entities, based on the conditions they are monitoring. They don't need to look at any global state or aggregate data. The desired

behavior is to have simple reporters, and not complex organisms that need to look at many things to infer what information to send.

To send health data to the health store, a reporter needs to identify the affected entity and create a health report. To send the report, use the [FabricClient.HealthClient.ReportHealth](#) API, report health APIs exposed on the `Partition` or `CodePackageActivationContext` objects, PowerShell cmdlets, or REST.

## Health reports

The [health reports](#) for each of the entities in the cluster contain the following information:

- **Sourceld**. A string that uniquely identifies the reporter of the health event.
- **Entity identifier**. Identifies the entity where the report is applied. It differs based on the [entity type](#):
  - Cluster. None.
  - Node. Node name (string).
  - Application. Application name (URI). Represents the name of the application instance deployed in the cluster.
  - Service. Service name (URI). Represents the name of the service instance deployed in the cluster.
  - Partition. Partition ID (GUID). Represents the partition unique identifier.
  - Replica. The stateful service replica ID or the stateless service instance ID (INT64).
  - DeployedApplication. Application name (URI) and node name (string).
  - DeployedServicePackage. Application name (URI), node name (string), and service manifest name (string).
- **Property**. A *string* (not a fixed enumeration) that allows the reporter to categorize the health event for a specific property of the entity. For example, reporter A can report the health of the Node01 "Storage" property and reporter B can report the health of the Node01 "Connectivity" property. In the health store, these reports are treated as separate health events for the Node01 entity.
- **Description**. A string that allows a reporter to provide detailed information about the health event. **Sourceld**, **Property**, and **HealthState** should fully describe the report. The description adds human-readable information about the report. The text makes it easier for administrators and users to understand the health report.
- **HealthState**. An [enumeration](#) that describes the health state of the report. The accepted values are OK, Warning, and Error.
- **TimeToLive**. A timespan that indicates how long the health report is valid. Coupled with **RemoveWhenExpired**, it lets the health store know how to evaluate expired events. By default, the value is infinite, and the report is valid forever.
- **RemoveWhenExpired**. A Boolean. If set to true, the expired health report is automatically removed from the health store, and the report doesn't impact entity health evaluation. Used when the report is valid for a specified period of time only, and the reporter doesn't need to explicitly clear it out. It's also used to delete reports from the health store (for example, a watchdog is changed and stops sending reports with previous source and property). It can send a report with a brief TimeToLive along with RemoveWhenExpired to clear up any previous state from the health store. If the value is set to false, the expired report is treated as an error on the health evaluation. The false value signals to the health store that the source should report periodically on this property. If it doesn't, then there must be something wrong with the watchdog. The watchdog's health is captured by considering the event as an error.
- **SequenceNumber**. A positive integer that needs to be ever-increasing, it represents the order of the reports. It is used by the health store to detect stale reports that are received late because of network delays or other issues. A report is rejected if the sequence number is less than or equal to the most recently applied number for the same entity, source, and property. If it is not specified, the sequence number is generated automatically. It is necessary to put in the sequence number only when reporting on state transitions. In this situation, the source needs to remember which reports it sent and keep the information for recovery on failover.

These four pieces of information--Sourceld, entity identifier, Property, and HealthState--are required for every health report. The Sourceld string is not allowed to start with the prefix "**System.**", which is reserved for system reports. For the same entity, there is only one report for the same source and property. Multiple reports for the same source and property override each other, either on the health client side (if they are batched) or on the health store side. The replacement is based on sequence numbers; newer reports (with higher sequence numbers) replace older reports.

## Health events

Internally, the health store keeps [health events](#), which contain all the information from the reports, and additional metadata. The metadata includes the time the report was given to the health client and the time it was modified on the server side. The health events are returned by [health queries](#).

The added metadata contains:

- **SourceUtcTimestamp**. The time the report was given to the health client (Coordinated Universal Time).
- **LastModifiedUtcTimestamp**. The time the report was last modified on the server side (Coordinated Universal Time).
- **IsExpired**. A flag to indicate whether the report was expired when the query was executed by the health store. An event can be expired only if RemoveWhenExpired is false. Otherwise, the event is not returned by query and is removed from the store.
- **LastOkTransitionAt**, **LastWarningTransitionAt**, **LastErrorTransitionAt**. The last time for OK/warning/error transitions. These fields give the history of the health state transitions for the event.

The state transition fields can be used for smarter alerts or "historical" health event information. They enable scenarios such as:

- Alert when a property has been at warning/error for more than X minutes. Checking the condition for a period of time avoids alerts on temporary conditions. For example, an alert if the health state has been warning for more than five minutes can be translated into (HealthState == Warning and Now - LastWarningTransitionTime > 5 minutes).
- Alert only on conditions that have changed in the last X minutes. If a report was already at error before the specified time, it can be ignored because it was already signaled previously.
- If a property is toggling between warning and error, determine how long it has been unhealthy (that is, not OK). For example, an alert if the property hasn't been healthy for more than five minutes can be translated into (HealthState != Ok and Now - LastOkTransitionTime > 5 minutes).

## Example: Report and evaluate application health

The following example sends a health report through PowerShell on the application **fabric:/WordCount** from the source **MyWatchdog**. The health report contains information about the health property "availability" in an error health state, with infinite TimeToLive. Then it queries the application health, which returns aggregated health state errors and the reported health events in the list of health events.

```
PS C:\> Send-ServiceFabricApplicationHealthReport -ApplicationName fabric:/WordCount -SourceId "MyWatchdog"
-HealthProperty "Availability" -HealthState Error
```

```
PS C:\> Get-ServiceFabricApplicationHealth fabric:/WordCount -ExcludeHealthStatistics
```

```
ApplicationName          : fabric:/WordCount
AggregatedHealthState   : Error
UnhealthyEvaluations    :
                           Error event: SourceId='MyWatchdog', Property='Availability'.

ServiceHealthStates     :
                           ServiceName      : fabric:/WordCount/WordCountService
                           AggregatedHealthState : Error

                           ServiceName      : fabric:/WordCount/WordCountWebService
                           AggregatedHealthState : Ok

DeployedApplicationHealthStates :
                           ApplicationName   : fabric:/WordCount
                           NodeName         : _Node_0
                           AggregatedHealthState : Ok

                           ApplicationName   : fabric:/WordCount
                           NodeName         : _Node_2
                           AggregatedHealthState : Ok

                           ApplicationName   : fabric:/WordCount
                           NodeName         : _Node_3
                           AggregatedHealthState : Ok

                           ApplicationName   : fabric:/WordCount
                           NodeName         : _Node_4
                           AggregatedHealthState : Ok

                           ApplicationName   : fabric:/WordCount
                           NodeName         : _Node_1
                           AggregatedHealthState : Ok

HealthEvents             :
                           SourceId        : System.CM
                           Property        : State
                           HealthState    : Ok
                           SequenceNumber : 360
                           SentAt         : 3/22/2016 7:56:53 PM
                           ReceivedAt     : 3/22/2016 7:56:53 PM
                           TTL            : Infinite
                           Description    : Application has been created.
                           RemoveWhenExpired : False
                           IsExpired      : False
                           Transitions    : Error->Ok = 3/22/2016 7:56:53 PM, LastWarning =
1/1/0001 12:00:00 AM

                           SourceId        : MyWatchdog
                           Property        : Availability
                           HealthState    : Error
                           SequenceNumber : 131032204762818013
                           SentAt         : 3/23/2016 3:27:56 PM
                           ReceivedAt     : 3/23/2016 3:27:56 PM
                           TTL            : Infinite
                           Description    :
                           RemoveWhenExpired : False
                           IsExpired      : False
                           Transitions    : Ok->Error = 3/23/2016 3:27:56 PM, LastWarning =
1/1/0001 12:00:00 AM
```

# Health model usage

The health model allows cloud services and the underlying Service Fabric platform to scale, because monitoring and health determinations are distributed among the different monitors within the cluster. Other systems have a single, centralized service at the cluster level that parses all the *potentially* useful information emitted by services. This approach hinders their scalability. It also doesn't allow them to collect specific information to help identify issues and potential issues as close to the root cause as possible.

The health model is used heavily for monitoring and diagnosis, for evaluating cluster and application health, and for monitored upgrades. Other services use health data to perform automatic repairs, build cluster health history, and issue alerts on certain conditions.

## Next steps

[View Service Fabric health reports](#)

[Use system health reports for troubleshooting](#)

[How to report and check service health](#)

[Add custom Service Fabric health reports](#)

[Monitor and diagnose services locally](#)

[Service Fabric application upgrade](#)

# Report and check service health

8/8/2017 • 5 min to read • [Edit Online](#)

When your services encounter problems, your ability to respond to and fix incidents and outages depends on your ability to detect the issues quickly. If you report problems and failures to the Azure Service Fabric health manager from your service code, you can use standard health monitoring tools that Service Fabric provides to check the health status.

There are three ways that you can report health from the service:

- Use [Partition](#) or [CodePackageActivationContext](#) objects.

You can use the `Partition` and `CodePackageActivationContext` objects to report the health of elements that are part of the current context. For example, code that runs as part of a replica can report health only on that replica, the partition that it belongs to, and the application that it is a part of.

- Use `FabricClient`.

You can use `FabricClient` to report health from the service code if the cluster is not [secure](#) or if the service is running with admin privileges. Most real-world scenarios do not use unsecured clusters, or provide admin privileges. With `FabricClient`, you can report health on any entity that is a part of the cluster. Ideally, however, service code should only send reports that are related to its own health.

- Use the REST APIs at the cluster, application, deployed application, service, service package, partition, replica, or node levels. This can be used to report health from within a container.

This article walks you through an example that reports health from the service code. The example also shows how the tools provided by Service Fabric can be used to check the health status. This article is intended to be a quick introduction to the health monitoring capabilities of Service Fabric. For more detailed information, you can read the series of in-depth articles about health that start with the link at the end of this article.

## Prerequisites

You must have the following installed:

- Visual Studio 2015 or Visual Studio 2017
- Service Fabric SDK

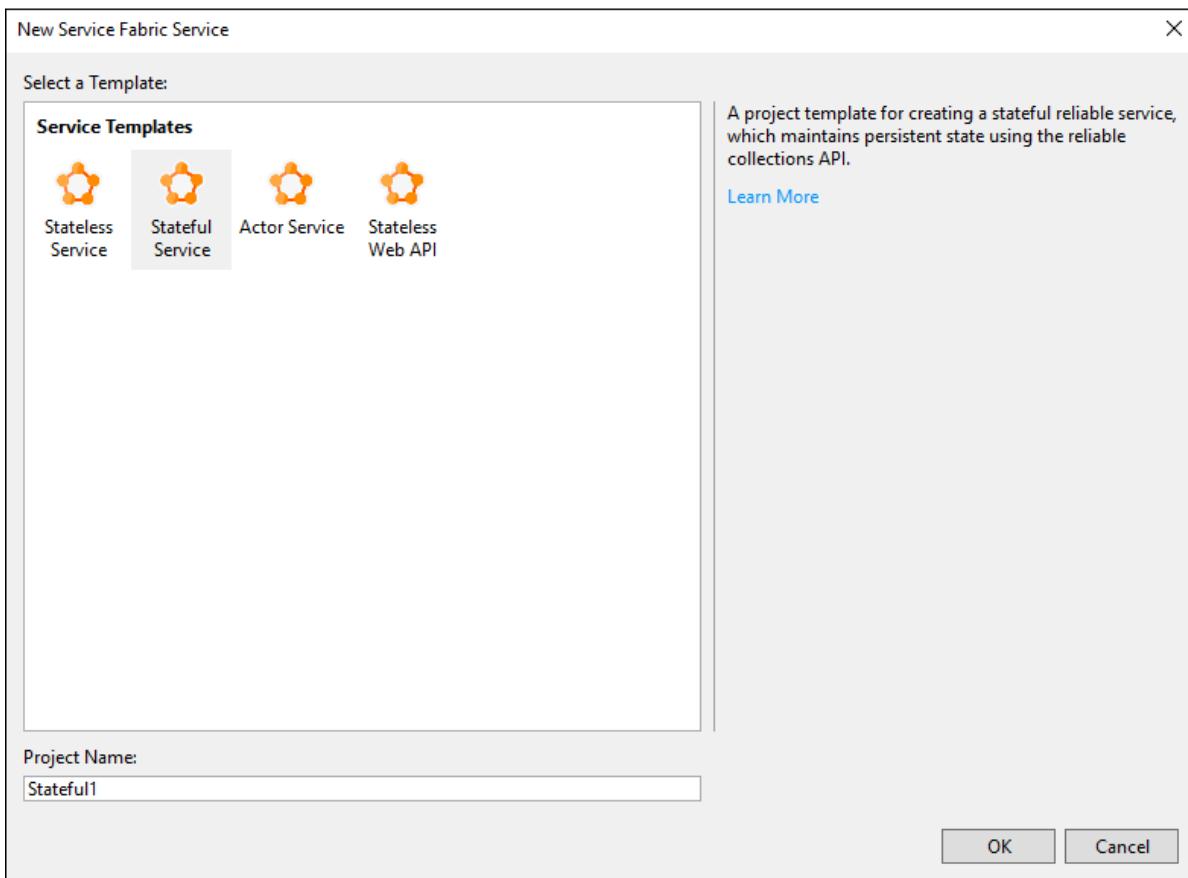
## To create a local secure dev cluster

- Open PowerShell with admin privileges, and run the following commands:

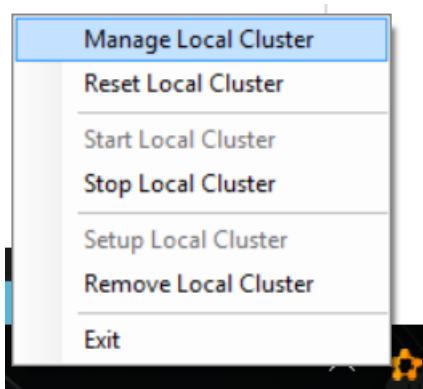
```
C:\Users\punewa> cd "C:\Program Files\Microsoft SDKs\Service Fabric\ClusterSetup"
C:\Program Files\Microsoft SDKs\Service Fabric\ClusterSetup> .\DevClusterSetup.ps1 -AsSecureCluster
```

## To deploy an application and check its health

1. Open Visual Studio as an administrator.
2. Create a project by using the **Stateful Service** template.



3. Press **F5** to run the application in debug mode. The application is deployed to the local cluster.
4. After the application is running, right-click the Local Cluster Manager icon in the notification area and select **Manage Local Cluster** from the shortcut menu to open Service Fabric Explorer.



5. The application health should be displayed as in this image. At this time, the application should be healthy with no errors.

6. You can also check the health by using PowerShell. You can use `Get-ServiceFabricApplicationHealth` to check an application's health, and you can use `Get-ServiceFabricServiceHealth` to check a service's health. The health report for the same application in PowerShell is in this image.

```
PS C:\Users\kunalds> Get-ServiceFabricApplicationHealth -ApplicationName fabric:/Application10

ApplicationName      : fabric:/Application10
AggregatedHealthstate : Ok
ServiceHealthStates  :
  :
    ServiceName       : fabric:/Application10/Stateful1
    AggregatedHealthstate : Ok

DeployedApplicationHealthstates :
  :
    ApplicationName   : fabric:/Application10
    NodeName          : Node.2
    AggregatedHealthstate : Ok

    ApplicationName   : fabric:/Application10
    NodeName          : Node.1
    AggregatedHealthstate : Ok

    ApplicationName   : fabric:/Application10
    NodeName          : Node.4
    AggregatedHealthstate : Ok

HealthEvents         :
  :
    SourceId          : System.CM
    Property           : State
    Healthstate        : Ok
    SequenceNumber     : 264255
    SentAt             : 10/28/2015 5:48:18 AM
    ReceivedAt         : 10/28/2015 5:48:18 AM
    TTL                : Infinite
    Description         : Application has been created.
    RemovewhenExpired  : False
    IsExpired          : False
    Transitions         : ->Ok = 10/28/2015 5:48:18 AM
```

## To add custom health events to your service code

The Service Fabric project templates in Visual Studio contain sample code. The following steps show how you can report custom health events from your service code. Such reports show up automatically in the standard tools for health monitoring that Service Fabric provides, such as Service Fabric Explorer, Azure portal health view, and PowerShell.

1. Reopen the application that you created previously in Visual Studio, or create a new application by using the **Stateful Service** Visual Studio template.

2. Open the Stateful1.cs file, and find the `myDictionary.TryGetValueAsync` call in the `RunAsync` method. You can see that this method returns a `result` that holds the current value of the counter because the key logic in this application is to keep a count running. If this were a real application, and if the lack of result represented a failure, you would want to flag that event.
3. To report a health event when the lack of result represents a failure, add the following steps.

- a. Add the `System.Fabric.Health` namespace to the Stateful1.cs file.

```
using System.Fabric.Health;
```

- b. Add the following code after the `myDictionary.TryGetValueAsync` call

```
if (!result.HasValue)
{
    HealthInformation healthInformation = new HealthInformation("ServiceCode", "StateDictionary",
    HealthState.Error);
    this.Partition.ReportReplicaHealth(healthInformation);
}
```

We report replica health because it's being reported from a stateful service. The `HealthInformation` parameter stores information about the health issue that's being reported.

If you had created a stateless service, use the following code

```
if (!result.HasValue)
{
    HealthInformation healthInformation = new HealthInformation("ServiceCode", "StateDictionary",
    HealthState.Error);
    this.Partition.ReportInstanceHealth(healthInformation);
}
```

4. If your service is running with admin privileges or if the cluster is not [secure](#), you can also use `FabricClient` to report health as shown in the following steps.

- a. Create the `FabricClient` instance after the `var myDictionary` declaration.

```
var fabricClient = new FabricClient(new FabricClientSettings() { HealthReportSendInterval =
TimeSpan.FromSeconds(0) });
```

- b. Add the following code after the `myDictionary.TryGetValueAsync` call.

```
if (!result.HasValue)
{
    var replicaHealthReport = new StatefulServiceReplicaHealthReport(
        this.Context.PartitionId,
        this.Context.ReplicaId,
        new HealthInformation("ServiceCode", "StateDictionary", HealthState.Error));
    fabricClient.HealthManager.ReportHealth(replicaHealthReport);
}
```

5. Let's simulate this failure and see it show up in the health monitoring tools. To simulate the failure, comment out the first line in the health reporting code that you added earlier. After you comment out the first line, the code will look like the following example.

```
//if(!result.HasValue)
{
    HealthInformation healthInformation = new HealthInformation("ServiceCode", "StateDictionary",
    HealthState.Error);
    this.Partition.ReportReplicaHealth(healthInformation);
}
```

This code fires the health report each time `RunAsync` executes. After you make the change, press **F5** to run the application.

- After the application is running, open Service Fabric Explorer to check the health of the application. This time, Service Fabric Explorer shows that the application is unhealthy. This is because of the error that was reported from the code that we added previously.

Name	Application Type
fabric:/TestApplication	TestApplicationType

**ESSENTIALS**

Health State	Version
Error	1.0.0

**DETAILS**

Status
Ready

**MANIFEST**

- If you select the primary replica in the tree view of Service Fabric Explorer, you will see that **Health State** indicates an error, too. Service Fabric Explorer also displays the health report details that were added to the `HealthInformation` parameter in the code. You can see the same health reports in PowerShell and the Azure portal.

Id	Node Name
131027240552591593	_Node_4

**ESSENTIALS**

Health State	Status
Error	Ready

**DETAILS**

Role
Primary

**UNHEALTHY EVALUATIONS**

Kind	Health State	Description
Event	Error	Error event: Sourceld='ServiceCode', Property='StateDictionary'.

This report remains in the health manager until it is replaced by another report or until this replica is deleted. Because we did not set `TimeToLive` for this health report in the `HealthInformation` object, the report never expires.

We recommend that health should be reported on the most granular level, which in this case is the replica. You can also report health on `Partition`.

```
HealthInformation healthInformation = new HealthInformation("ServiceCode", "StateDictionary",
    HealthState.Error);
this.Partition.ReportPartitionHealth(healthInformation);
```

To report health on `Application`, `DeployedApplication`, and `DeployedServicePackage`, use `CodePackageActivationContext`.

```
HealthInformation healthInformation = new HealthInformation("ServiceCode", "StateDictionary",
    HealthState.Error);
var activationContext = FabricRuntime.GetActivationContext();
activationContext.ReportApplicationHealth(healthInformation);
```

## Next steps

- [Deep dive on Service Fabric health](#)
- [REST API for reporting service health](#)
- [REST API for reporting application health](#)

# Add custom Service Fabric health reports

8/8/2017 • 20 min to read • [Edit Online](#)

Azure Service Fabric introduces a [health model](#) designed to flag unhealthy cluster and application conditions on specific entities. The health model uses **health reporters** (system components and watchdogs). The goal is easy and fast diagnosis and repair. Service writers need to think upfront about health. Any condition that can impact health should be reported on, especially if it can help flag problems close to the root. The health information can save time and effort on debugging and investigation. The usefulness is especially clear once the service is up and running at scale in the cloud (private or Azure).

The Service Fabric reporters monitor identified conditions of interest. They report on those conditions based on their local view. The [health store](#) aggregates health data sent by all reporters to determine whether entities are globally healthy. The model is intended to be rich, flexible, and easy to use. The quality of the health reports determines the accuracy of the health view of the cluster. False positives that wrongly show unhealthy issues can negatively impact upgrades or other services that use health data. Examples of such services are repair services and alerting mechanisms. Therefore, some thought is needed to provide reports that capture conditions of interest in the best possible way.

To design and implement health reporting, watchdogs and system components must:

- Define the condition they are interested in, the way it is monitored, and the impact on the cluster or application functionality. Based on this information, decide on the health report property and health state.
- Determine the [entity](#) that the report applies to.
- Determine where the reporting is done, from within the service or from an internal or external watchdog.
- Define a source used to identify the reporter.
- Choose a reporting strategy, either periodically or on transitions. The recommended way is periodically, as it requires simpler code and is less prone to errors.
- Determine how long the report for unhealthy conditions should stay in the health store and how it should be cleared. Using this information, decide the report's time to live and remove-on-expiration behavior.

As mentioned, reporting can be done from:

- The monitored Service Fabric service replica.
- Internal watchdogs deployed as a Service Fabric service (for example, a Service Fabric stateless service that monitors conditions and issues reports). The watchdogs can be deployed on all nodes or can be affinitized to the monitored service.
- Internal watchdogs that run on the Service Fabric nodes but are *not* implemented as Service Fabric services.
- External watchdogs that probe the resource from *outside* the Service Fabric cluster (for example, monitoring service like Gomez).

## NOTE

Out of the box, the cluster is populated with health reports sent by the system components. Read more at [Using system health reports for troubleshooting](#). The user reports must be sent on [health entities](#) that have already been created by the system.

Once the health reporting design is clear, health reports can be sent easily. You can use [FabricClient](#) to report health if the cluster is not [secure](#) or if the fabric client has admin privileges. Reporting can be done through the API by using [FabricClient.HealthManager.ReportHealth](#), through PowerShell, or through REST. Configuration knobs

batch reports for improved performance.

#### NOTE

Report health is synchronous, and it represents only the validation work on the client side. The fact that the report is accepted by the health client or the `Partition` or `CodePackageActivationContext` objects doesn't mean that it is applied in the store. It is sent asynchronously and possibly batched with other reports. The processing on the server may still fail: the sequence number could be stale, the entity on which the report must be applied has been deleted, etc.

## Health client

The health reports are sent to the health store through a health client, which lives inside the fabric client. The health client can be configured with the following settings:

- **HealthReportSendInterval**: The delay between the time the report is added to the client and the time it is sent to the health store. Used to batch reports into a single message, rather than sending one message for each report. The batching improves performance. Default: 30 seconds.
- **HealthReportRetrySendInterval**: The interval at which the health client resends accumulated health reports to the health store. Default: 30 seconds.
- **HealthOperationTimeout**: The timeout period for a report message sent to the health store. If a message times out, the health client retries it until the health store confirms that the report has been processed. Default: two minutes.

#### NOTE

When the reports are batched, the fabric client must be kept alive for at least the `HealthReportSendInterval` to ensure that they are sent. If the message is lost or the health store cannot apply them due to transient errors, the fabric client must be kept alive longer to give it a chance to retry.

The buffering on the client takes the uniqueness of the reports into consideration. For example, if a particular bad reporter is reporting 100 reports per second on the same property of the same entity, the reports are replaced with the last version. At most one such report exists in the client queue. If batching is configured, the number of reports sent to the health store is just one per send interval. This report is the last added report, which reflects the most current state of the entity. Specify configuration parameters when `FabricClient` is created by passing `FabricClientSettings` with the desired values for health-related entries.

The following example creates a fabric client and specifies that the reports should be sent when they are added. On timeouts and errors that can be retried, retries happen every 40 seconds.

```
var clientSettings = new FabricClientSettings()
{
    HealthOperationTimeout = TimeSpan.FromSeconds(120),
    HealthReportSendInterval = TimeSpan.FromSeconds(0),
    HealthReportRetrySendInterval = TimeSpan.FromSeconds(40),
};
var fabricClient = new FabricClient(clientSettings);
```

We recommend keeping the default fabric client settings, which set `HealthReportSendInterval` to 30 seconds. This setting ensures optimal performance due to batching. For critical reports that must be sent as soon as possible, use `HealthReportSendOptions` with `Immediate true` in `FabricClient.HealthClient.ReportHealth` API. Immediate reports bypass the batching interval. Use this flag with care; we want to take advantage of the health client batching whenever possible. Immediate send is also useful when the fabric client is closing (for example, the process has determined invalid state and needs to shut down to prevent side effects). It ensures a best-effort send

of the accumulated reports. When one report is added with `Immediate` flag, the health client batches all the accumulated reports since last send.

Same parameters can be specified when a connection to a cluster is created through PowerShell. The following example starts a connection to a local cluster:

```
PS C:\> Connect-ServiceFabricCluster -HealthOperationTimeoutInSec 120 -HealthReportSendIntervalInSec 0 -  
HealthReportRetrySendIntervalInSec 40  
True  
  
ConnectionEndpoint :  
FabricClientSettings : {  
    ClientFriendlyName : PowerShell-1944858a-4c6d-465f-89c7-9021c12ac0bb  
    PartitionLocationCacheLimit : 100000  
    PartitionLocationCacheBucketCount : 1024  
    ServiceChangePollInterval : 00:02:00  
    ConnectionInitializationTimeout : 00:00:02  
    KeepAliveInterval : 00:00:20  
    HealthOperationTimeout : 00:02:00  
    HealthReportSendInterval : 00:00:00  
    HealthReportRetrySendInterval : 00:00:40  
    NotificationGatewayConnectionTimeout : 00:00:00  
    NotificationCacheUpdateTimeout : 00:00:00  
}  
GatewayInformation : {  
    NodeAddress : localhost:19000  
    NodeId : 1880ec88a3187766a6da323399721f53  
    NodeInstanceId : 130729063464981219  
    NodeName : Node.1  
}
```

Similarly to API, reports can be sent using `-Immediate` switch to be sent immediately, regardless of the `HealthReportSendInterval` value.

For REST, the reports are sent to the Service Fabric gateway, which has an internal fabric client. By default, this client is configured to send reports batched every 30 seconds. You can change the batch interval with the cluster configuration setting `HttpGatewayHealthReportSendInterval` on `HttpGateway`. As mentioned, a better option is to send the reports with `Immediate` true.

#### NOTE

To ensure that unauthorized services can't report health against the entities in the cluster, configure the server to accept requests only from secured clients. The `FabricClient` used for reporting must have security enabled to be able to communicate with the cluster (for example, with Kerberos or certificate authentication). Read more about [cluster security](#).

## Report from within low privilege services

If Service Fabric services do not have admin access to the cluster, you can report health on entities from the current context through `Partition` or `CodePackageActivationContext`.

- For stateless services, use `IStatelessServicePartition.ReportInstanceHealth` to report on the current service instance.
- For stateful services, use `IStatefulServicePartition.ReportReplicaHealth` to report on current replica.
- Use `IServicePartition.ReportPartitionHealth` to report on the current partition entity.
- Use `CodePackageActivationContext.ReportApplicationHealth` to report on current application.
- Use `CodePackageActivationContext.ReportDeployedApplicationHealth` to report on the current application deployed on the current node.

- Use `CodePackageActivationContext.ReportDeployedServicePackageHealth` to report on a service package for the application deployed on the current node.

#### NOTE

Internally, the `Partition` and the `CodePackageActivationContext` hold a health client configured with default settings. As explained for the [health client](#), reports are batched and sent on a timer. The objects should be kept alive to have a chance to send the report.

You can specify `HealthReportSendOptions` when sending reports through `Partition` and `CodePackageActivationContext` health APIs. If you have critical reports that must be sent as soon as possible, use `HealthReportSendOptions` with `Immediate` `true`. Immediate reports bypass the batching interval of the internal health client. As mentioned before, use this flag with care; we want to take advantage of the health client batching whenever possible.

## Design health reporting

The first step in generating high-quality reports is identifying the conditions that can impact the health of the service. Any condition that can help flag problems in the service or cluster when it starts--or even better, before a problem happens--can potentially save billions of dollars. The benefits include less down time, fewer night hours spent investigating and repairing issues, and higher customer satisfaction.

Once the conditions are identified, watchdog writers need to figure out the best way to monitor them for balance between overhead and usefulness. For example, consider a service that does complex calculations that use some temporary files on a share. A watchdog could monitor the share to ensure that enough space is available. It could listen for notifications of file or directory changes. It could report a warning if an upfront threshold is reached, and report an error if the share is full. On a warning, a repair system could start cleaning up older files on the share. On an error, a repair system could move the service replica to another node. Note how the condition states are described in terms of health: the state of the condition that can be considered healthy (ok) or unhealthy (warning or error).

Once the monitoring details are set, a watchdog writer needs to figure out how to implement the watchdog. If the conditions can be determined from within the service, the watchdog can be part of the monitored service itself. For example, the service code can check the share usage, and then report every time it tries to write a file. The advantage of this approach is that reporting is simple. Care must be taken to prevent watchdog bugs from impacting the service functionality.

Reporting from within the monitored service is not always an option. A watchdog within the service may not be able to detect the conditions. It may not have the logic or data to make the determination. The overhead of monitoring the conditions may be high. The conditions also may not be specific to a service, but instead affect interactions between services. Another option is to have watchdogs in the cluster as separate processes. The watchdogs monitor the conditions and report, without affecting the main services in any way. For example, these watchdogs could be implemented as stateless services in the same application, deployed on all nodes or on the same nodes as the service.

Sometimes, a watchdog running in the cluster is not an option either. If the monitored condition is the availability or functionality of the service as users see it, it's best to have the watchdogs in the same place as the user clients. There, they can test the operations in the same way users call them. For example, you can have a watchdog that lives outside the cluster, issues requests to the service, and checks the latency and correctness of the result. (For a calculator service, for example, does  $2+2$  return 4 in a reasonable amount of time?)

Once the watchdog details have been finalized, you should decide on a source ID that uniquely identifies it. If multiple watchdogs of the same type are living in the cluster, they must report on different entities, or, if they report on the same entity, use different source ID or property. This way, their reports can coexist. The property of

the health report should capture the monitored condition. (For the example above, the property could be **ShareSize**.) If multiple reports apply to the same condition, the property should contain some dynamic information that allows reports to coexist. For example, if multiple shares need to be monitored, the property name can be **ShareSize-sharename**.

**NOTE**

*Do not use the health store to keep status information. Only health-related information should be reported as health, as this information impacts the health evaluation of an entity. The health store was not designed as a general-purpose store. It uses health evaluation logic to aggregate all data into the health state. Sending information unrelated to health (like reporting status with a health state of OK) doesn't impact the aggregated health state, but it can negatively affect the performance of the health store.*

The next decision point is which entity to report on. Most of the time, the condition clearly identifies the entity. Choose the entity with best possible granularity. If a condition impacts all replicas in a partition, report on the partition, not on the service. There are corner cases where more thought is needed, though. If the condition impacts an entity, such as a replica, but the desire is to have the condition flagged for more than the duration of replica life, then it should be reported on the partition. Otherwise, when the replica is deleted, the health store cleans up all its reports. Watchdog writers must think about the lifetimes of the entity and the report. It must be clear when a report should be cleaned up from a store (for example, when an error reported on an entity no longer applies).

Let's look at an example that puts together the points I described. Consider a Service Fabric application composed of a master stateful persistent service and secondary stateless services deployed on all nodes (one secondary service type for each type of task). The master has a processing queue that contains commands to be executed by secondaries. The secondaries execute the incoming requests and send back acknowledgement signals. One condition that could be monitored is the length of the master processing queue. If the master queue length reaches a threshold, a warning is reported. The warning indicates that the secondaries can't handle the load. If the queue reaches the maximum length and commands are dropped, an error is reported, as the service can't recover. The reports can be on the property **QueueStatus**. The watchdog lives inside the service, and it's sent periodically on the master primary replica. The time to live is two minutes, and it's sent periodically every 30 seconds. If the primary goes down, the report is cleaned up automatically from store. If the service replica is up, but it is deadlocked or having other issues, the report expires in the health store. In this case, the entity is evaluated at error.

Another condition that can be monitored is task execution time. The master distributes tasks to the secondaries based on the task type. Depending on the design, the master could poll the secondaries for task status. It could also wait for secondaries to send back acknowledgement signals when they are done. In the second case, care must be taken to detect situations where secondaries die or messages are lost. One option is for the master to send a ping request to the same secondary, which sends back its status. If no status is received, the master considers it a failure and reschedules the task. This behavior assumes that the tasks are idempotent.

The monitored condition can be translated as a warning if the task is not done in a certain time (**t1**, for example 10 minutes). If the task is not completed in time (**t2**, for example 20 minutes), the monitored condition can be translated as Error. This reporting can be done in multiple ways:

- The master primary replica reports on itself periodically. You can have one property for all pending tasks in the queue. If at least one task takes longer, the report status on the property **PendingTasks** is a warning or error, as appropriate. If there are no pending tasks or all tasks started execution, the report status is OK. The tasks are persistent. If the primary goes down, the newly promoted primary can continue to report properly.
- Another watchdog process (in the cloud or external) checks the tasks (from outside, based on the desired task result) to see if they are completed. If they do not respect the thresholds, a report is sent on the master service. A report is also sent on each task that includes the task identifier, like **PendingTask+taskId**. Reports should be sent only on unhealthy states. Set time to live to a few minutes, and mark the reports to be removed when they

expire to ensure cleanup.

- The secondary that is executing a task reports when it takes longer than expected to run it. It reports on the service instance on the property **PendingTasks**. The report pinpoints the service instance that has issues, but it doesn't capture the situation where the instance dies. The reports are cleaned up then. It could report on the secondary service. If the secondary completes the task, the secondary instance clears the report from the store. The report doesn't capture the situation where the acknowledgement message is lost and the task is not finished from the master's point of view.

However the reporting is done in the cases described above, the reports are captured in application health when health is evaluated.

## Report periodically vs. on transition

By using the health reporting model, watchdogs can send reports periodically or on transitions. The recommended way for watchdog reporting is periodically, because the code is much simpler and less prone to errors. The watchdogs must strive to be as simple as possible to avoid bugs that trigger incorrect reports. Incorrect *unhealthy* reports impact health evaluations and scenarios based on health, including upgrades. Incorrect *healthy* reports hide issues in the cluster, which is not desired.

For periodic reporting, the watchdog can be implemented with a timer. On a timer callback, the watchdog can check the state and send a report based on the current state. There is no need to see which report was sent previously or make any optimizations in terms of messaging. The health client has batching logic to help with performance. While the health client is kept alive, it retries internally until the report is acknowledged by the health store or the watchdog generates a newer report with the same entity, property, and source.

Reporting on transitions requires careful handling of state. The watchdog monitors some conditions and reports only when the conditions change. The upside of this approach is that fewer reports are needed. The downside is that the logic of the watchdog is complex. The watchdog must maintain the conditions or the reports, so that they can be inspected to determine state changes. On failover, care must be taken with reports added, but not yet sent to the health store. The sequence number must be ever-increasing. If not, the reports are rejected as stale. In the rare cases where data loss is incurred, synchronization may be needed between the state of the reporter and the state of the health store.

Reporting on transitions makes sense for services reporting on themselves, through `Partition` or `CodePackageActivationContext`. When the local object (replica or deployed service package / deployed application) is removed, all its reports are also removed. This automatic cleanup relaxes the need for synchronization between reporter and health store. If the report is for parent partition or parent application, care must be taken on failover to avoid stale reports in the health store. Logic must be added to maintain the correct state and clear the report from store when not needed anymore.

## Implement health reporting

Once the entity and report details are clear, sending health reports can be done through the API, PowerShell, or REST.

### API

To report through the API, you need to create a health report specific to the entity type they want to report on. Give the report to a health client. Alternatively, create a health information and pass it to correct reporting methods on `Partition` or `CodePackageActivationContext` to report on current entities.

The following example shows periodic reporting from a watchdog within the cluster. The watchdog checks whether an external resource can be accessed from within a node. The resource is needed by a service manifest within the application. If the resource is unavailable, the other services within the application can still function properly. Therefore, the report is sent on the deployed service package entity every 30 seconds.

```

private static Uri ApplicationName = new Uri("fabric:/WordCount");
private static string ServiceManifestName = "WordCount.Service";
private static string NodeName = FabricRuntime.GetNodeContext().NodeName;
private static Timer ReportTimer = new Timer(new TimerCallback(SendReport), null, 30 * 1000, 30 * 1000);
private static FabricClient Client = new FabricClient(new FabricClientSettings() { HealthReportSendInterval =
TimeSpan.FromSeconds(0) });

public static void SendReport(object obj)
{
    // Test whether the resource can be accessed from the node
    HealthState healthState = this.TestConnectivityToExternalResource();

    // Send report on deployed service package, as the connectivity is needed by the specific service manifest
    // and can be different on different nodes
    var deployedServicePackageHealthReport = new DeployedServicePackageHealthReport(
        ApplicationName,
        ServiceManifestName,
        NodeName,
        new HealthInformation("ExternalSourceWatcher", "Connectivity", healthState));

    // TODO: handle exception. Code omitted for snippet brevity.
    // Possible exceptions: FabricException with error codes
    // FabricHealthStaleReport (non-retryable; the report is already queued on the health client),
    // FabricHealthMaxReportsReached (retryable; user should retry with exponential delay until the report is
    accepted).
    Client.HealthManager.ReportHealth(deployedServicePackageHealthReport);
}

```

## PowerShell

Send health reports with **Send-ServiceFabricEntityTypeHealthReport**.

The following example shows periodic reporting on CPU values on a node. The reports should be sent every 30 seconds, and they have a time to live of two minutes. If they expire, the reporter has issues, so the node is evaluated at error. When the CPU is above a threshold, the report has a health state of warning. When the CPU remains above a threshold for more than the configured time, it's reported as an error. Otherwise, the reporter sends a health state of OK.

```

PS C:\> Send-ServiceFabricNodeHealthReport -NodeName Node.1 -HealthState Warning -SourceId PowershellWatcher -
HealthProperty CPU -Description "CPU is above 80% threshold" -TimeToLiveSec 120

PS C:\> Get-ServiceFabricNodeHealth -NodeName Node.1
NodeName          : Node.1
AggregatedHealthState : Warning
UnhealthyEvaluations :
    Unhealthy event: SourceId='PowershellWatcher', Property='CPU', HealthState='Warning',
ConsiderWarningAsError=false.

HealthEvents      :
    SourceId          : System.FM
    Property           : State
    HealthState        : Ok
    SequenceNumber     : 5
    SentAt             : 4/21/2015 8:01:17 AM
    ReceivedAt         : 4/21/2015 8:02:12 AM
    TTL                : Infinite
    Description         : Fabric node is up.
    RemoveWhenExpired : False
    IsExpired          : False
    Transitions         : ->Ok = 4/21/2015 8:02:12 AM

    SourceId          : PowershellWatcher
    Property           : CPU
    HealthState        : Warning
    SequenceNumber     : 130741236814913394
    SentAt             : 4/21/2015 9:01:21 PM
    ReceivedAt         : 4/21/2015 9:01:21 PM
    TTL                : 00:02:00
    Description         : CPU is above 80% threshold
    RemoveWhenExpired : False
    IsExpired          : False
    Transitions         : ->Warning = 4/21/2015 9:01:21 PM

```

The following example reports a transient warning on a replica. It first gets the partition ID and then the replica ID for the service it is interested in. It then sends a report from **PowershellWatcher** on the property

**ResourceDependency**. The report is of interest for only two minutes, and it is removed from the store automatically.

```

PS C:\> $partitionId = (Get-ServiceFabricPartition -ServiceName
fabric:/WordCount/WordCount.Service).PartitionId

PS C:\> $replicaId = (Get-ServiceFabricReplica -PartitionId $partitionId | where {$_.ReplicaRole -eq
"Primary"}).ReplicaId

PS C:\> Send-ServiceFabricReplicaHealthReport -PartitionId $partitionId -ReplicaId $replicaId -HealthState
Warning -SourceId PowershellWatcher -HealthProperty ResourceDependency -Description "The external resource
that the primary is using has been rebooted at 4/21/2015 9:01:21 PM. Expect processing delays for a few
minutes." -TimeToLiveSec 120 -RemoveWhenExpired

PS C:\> Get-ServiceFabricReplicaHealth -PartitionId $partitionId -ReplicaOrInstanceId $replicaId

PartitionId      : 8f82daff-eb68-4fd9-b631-7a37629e08c0
ReplicaId       : 130740415594605869
AggregatedHealthState : Warning
UnhealthyEvaluations :
    Unhealthy event: SourceId='PowershellWatcher', Property='ResourceDependency',
HealthState='Warning', ConsiderWarningAsError=false.

HealthEvents      :
    SourceId      : System.RA
    Property       : State
    HealthState    : Ok
    SequenceNumber : 130740768777734943
    SentAt        : 4/21/2015 8:01:17 AM
    ReceivedAt    : 4/21/2015 8:02:12 AM
    TTL           : Infinite
    Description    : Replica has been created.
    RemoveWhenExpired : False
    IsExpired     : False
    Transitions   : ->Ok = 4/21/2015 8:02:12 AM

    SourceId      : PowershellWatcher
    Property       : ResourceDependency
    HealthState    : Warning
    SequenceNumber : 130741243777723555
    SentAt        : 4/21/2015 9:12:57 PM
    ReceivedAt    : 4/21/2015 9:12:57 PM
    TTL           : 00:02:00
    Description    : The external resource that the primary is using has been
rebooted at 4/21/2015 9:01:21 PM. Expect processing delays for a few minutes.
    RemoveWhenExpired : True
    IsExpired     : False
    Transitions   : ->Warning = 4/21/2015 9:12:32 PM

```

## REST

Send health reports using REST with POST requests that go to the desired entity and have in the body the health report description. For example, see how to send REST [cluster health reports](#) or [service health reports](#). All entities are supported.

## Next steps

Based on the health data, service writers and cluster/application administrators can think of ways to consume the information. For example, they can set up alerts based on health status to catch severe issues before they provoke outages. Administrators can also set up repair systems to fix issues automatically.

[Introduction to Service Fabric health Monitoring](#)

[View Service Fabric health reports](#)

[How to report and check service health](#)

[Use system health reports for troubleshooting](#)

[Monitor and diagnose services locally](#)

[Service Fabric application upgrade](#)

# Use system health reports to troubleshoot

10/25/2017 • 24 min to read • [Edit Online](#)

Azure Service Fabric components provide system health reports on all entities in the cluster right out of the box. The [health store](#) creates and deletes entities based on the system reports. It also organizes them in a hierarchy that captures entity interactions.

## NOTE

To understand health-related concepts, read more at [Service Fabric health model](#).

System health reports provide visibility into cluster and application functionality, and flag problems. For applications and services, system health reports verify that entities are implemented and are behaving correctly from the Service Fabric perspective. The reports don't provide any health monitoring of the business logic of the service or detection of hung processes. User services can enrich the health data with information specific to their logic.

## NOTE

Health reports sent by user watchdogs are visible only *after* the system components create an entity. When an entity is deleted, the health store automatically deletes all the health reports associated with it. The same is true when a new instance of the entity is created, for example, when a new stateful persisted service replica instance is created. All reports associated with the old instance are deleted and cleaned up from the store.

The system component reports are identified by the source, which starts with the "**System.**" prefix. Watchdogs can't use the same prefix for their sources, as reports with invalid parameters are rejected.

Let's look at some system reports to understand what triggers them and to learn how to correct the potential problems they represent.

## NOTE

Service Fabric continues to add reports on conditions of interest that improve visibility into what is happening in the cluster and the applications. Existing reports can be enhanced with more details to help troubleshoot the problem faster.

## Cluster system health reports

The cluster health entity is created automatically in the health store. If everything works properly, it doesn't have a system report.

### Neighborhood loss

**System.Federation** reports an error when it detects a neighborhood loss. The report is from individual nodes, and the node ID is included in the property name. If one neighborhood is lost in the entire Service Fabric ring, you can typically expect two events that represent both sides of the gap report. If more neighborhoods are lost, there are more events.

The report specifies the global-lease timeout as the time-to-live (TTL). The report is resent every half of the TTL duration for as long as the condition remains active. The event is automatically removed when it expires. Remove-when-expired behavior ensures that the report is cleaned up from the health store correctly, even if the

reporting node is down.

- **SourceId:** System.Federation
- **Property:** Starts with **Neighborhood** and includes node information.
- **Next steps:** Investigate why the neighborhood is lost, for example, check the communication between cluster nodes.

## Node system health reports

**System.FM**, which represents the Failover Manager service, is the authority that manages information about cluster nodes. Each node should have one report from System.FM showing its state. The node entities are removed when the node state is removed. For more information, see [RemoveNodeStateAsync](#).

### Node up/down

System.FM reports as OK when the node joins the ring (it's up and running). It reports an error when the node departs the ring (it's down, either for upgrading or simply because it has failed). The health hierarchy built by the health store acts on deployed entities in correlation with System.FM node reports. It considers the node a virtual parent of all deployed entities. The deployed entities on that node are exposed through queries if the node is reported as up by System.FM, with the same instance as the instance associated with the entities. When System.FM reports that the node is down or restarted, as a new instance, the health store automatically cleans up the deployed entities that can exist only on the down node or on the previous instance of the node.

- **SourceId:** System.FM
- **Property:** State.
- **Next steps:** If the node is down for an upgrade, it should come back up after it's been upgraded. In this case, the health state should switch back to OK. If the node doesn't come back or it fails, the problem needs more investigation.

The following example shows the System.FM event with a health state of OK for node up:

```
PS C:\> Get-ServiceFabricNodeHealth _Node_0

NodeName          : _Node_0
AggregatedHealthState : Ok
HealthEvents      :
    SourceId      : System.FM
    Property       : State
    HealthState   : Ok
    SequenceNumber: 8
    SentAt        : 7/14/2017 4:54:51 PM
    ReceivedAt    : 7/14/2017 4:55:14 PM
    TTL           : Infinite
    Description   : Fabric node is up.
    RemoveWhenExpired: False
    IsExpired     : False
    Transitions   : Error->Ok = 7/14/2017 4:55:14 PM, LastWarning = 1/1/0001
12:00:00 AM
```

### Certificate expiration

**System.FabricNode** reports a warning when certificates used by the node are near expiration. There are three certificates per node: **Certificate\_cluster**, **Certificate\_server**, and **Certificate\_default\_client**. When the expiration is at least two weeks away, the report health state is OK. When the expiration is within two weeks, the report type is a warning. TTL of these events is infinite, and they are removed when a node leaves the cluster.

- **SourceId:** System.FabricNode
- **Property:** Starts with **Certificate** and contains more information about the certificate type.
- **Next steps:** Update the certificates if they are near expiration.

## Load capacity violation

The Service Fabric Load Balancer reports a warning when it detects a node capacity violation.

- **SourceId:** System.PLB
- **Property:** Starts with **Capacity**.
- **Next steps:** Check the provided metrics and view the current capacity on the node.

## Node capacity mismatch for resource governance metrics

System.Hosting reports a warning if defined node capacities in the cluster manifest are larger than the real node capacities for resource governance metrics (memory and cpu cores). Health report will be shown up when first service package that uses [resource governance](#) registers on a specified node.

- **SourceId:** System.Hosting
- **Property:** ResourceGovernance
- **Next steps:** This can be a problem as governing service packages will not be enforced as expected and [resource governance](#) will not work properly. Update the cluster manifest with correct node capacities for these metrics or do not specify them at all and let Service Fabric to automatically detect available resources.

# Application system health reports

**System.CM**, which represents the Cluster Manager service, is the authority that manages information about an application.

## State

System.CM reports as OK when the application has been created or updated. It informs the health store when the application has been deleted, so that it can be removed from the store.

- **SourceId:** System.CM
- **Property:** State.
- **Next steps:** If the application has been created or updated, it should include the Cluster Manager health report. Otherwise, check the state of the application by issuing a query, for example, the PowerShell cmdlet **Get-ServiceFabricApplication -ApplicationName *applicationName***.

The following example shows the state event on the **fabric:/WordCount** application:

```
PS C:\> Get-ServiceFabricApplicationHealth fabric:/WordCount -ServicesFilter None -  
DeployedApplicationsFilter None -ExcludeHealthStatistics  
  
ApplicationName          : fabric:/WordCount  
AggregatedHealthState   : Ok  
ServiceHealthStates      : None  
DeployedApplicationHealthStates : None  
HealthEvents             :  
    SourceId      : System.CM  
    Property       : State  
    HealthState   : Ok  
    SequenceNumber: 282  
    SentAt        : 7/13/2017 5:57:05 PM  
    ReceivedAt    : 7/14/2017 4:55:10 PM  
    TTL           : Infinite  
    Description   : Application has been created.  
    RemoveWhenExpired: False  
    IsExpired     : False  
    Transitions   : Error->Ok = 7/13/2017 5:57:05 PM, LastWarning =  
1/1/0001 12:00:00 AM
```

# Service system health reports

**System.FM**, which represents the Failover Manager service, is the authority that manages information about services.

## State

System.FM reports as OK when the service has been created. It deletes the entity from the health store when the service has been deleted.

- **SourceId:** System.FM
- **Property:** State.

The following example shows the state event on the service **fabric:/WordCount/WordCountWebService**:

```
PS C:\> Get-ServiceFabricServiceHealth fabric:/WordCount/WordCountWebService -ExcludeHealthStatistics

ServiceName      : fabric:/WordCount/WordCountWebService
AggregatedHealthState : Ok
PartitionHealthStates :
    PartitionId      : 8bbcd03a-3a53-47ec-a5f1-9b77f73c53b2
    AggregatedHealthState : Ok

HealthEvents     :
    SourceId        : System.FM
    Property         : State
    HealthState      : Ok
    SequenceNumber   : 14
    SentAt          : 7/13/2017 5:57:05 PM
    ReceivedAt       : 7/14/2017 4:55:10 PM
    TTL              : Infinite
    Description       : Service has been created.
    RemoveWhenExpired : False
    IsExpired        : False
    Transitions       : Error->Ok = 7/13/2017 5:57:18 PM, LastWarning = 1/1/0001
12:00:00 AM
```

## Service correlation error

**System.PLB** reports an error when it detects that updating a service is correlated with another service that creates an affinity chain. The report is cleared when a successful update happens.

- **SourceId:** System.PLB
- **Property:** ServiceDescription.
- **Next steps:** Check the correlated service descriptions.

# Partition system health reports

**System.FM**, which represents the Failover Manager service, is the authority that manages information about service partitions.

## State

System.FM reports as OK when the partition has been created and is healthy. It deletes the entity from the health store when the partition is deleted.

If the partition is below the minimum replica count, it reports an error. If the partition is not below the minimum replica count, but it's below the target replica count, it reports a warning. If the partition is in quorum loss, System.FM reports an error.

Other notable events include a warning when the reconfiguration takes longer than expected and when the build

takes longer than expected. The expected times for the build and reconfiguration are configurable based on the service scenarios. For example, if a service has a terabyte of state, such as Azure SQL Database, the build takes longer than for a service with a small amount of state.

- **SourceId:** System.FM
- **Property:** State.
- **Next steps:** If the health state is not OK, it's possible that some replicas have not been created, opened, or promoted to primary or secondary correctly.

If the description describes quorum loss, then examining the detailed health report for replicas that are down and bringing them back up helps to bring the partition back online.

If the description describes a partition stuck in [reconfiguration](#), then the health report on the primary replica provides additional information.

For other System.FM health reports, there would be reports on the replicas or the partition or service from other system components.

The following examples describe some of these reports.

The following example shows a healthy partition:

```
PS C:\> Get-ServiceFabricPartition fabric:/WordCount/WordCountWebService | Get-ServiceFabricPartitionHealth -ExcludeHealthStatistics -ReplicasFilter None

PartitionId      : 8bbcd03a-3a53-47ec-a5f1-9b77f73c53b2
AggregatedHealthState : Ok
ReplicaHealthStates   : None
HealthEvents      :
    SourceId      : System.FM
    Property       : State
    HealthState    : Ok
    SequenceNumber : 70
    SentAt        : 7/13/2017 5:57:05 PM
    ReceivedAt    : 7/14/2017 4:55:10 PM
    TTL           : Infinite
    Description    : Partition is healthy.
    RemoveWhenExpired : False
    IsExpired     : False
    Transitions    : Error->Ok = 7/13/2017 5:57:18 PM, LastWarning = 1/1/0001
12:00:00 AM
```

The following example shows the health of a partition that's below target replica count. The next step is to get the partition description, which shows how it's configured: **MinReplicaSetSize** is three and **TargetReplicaSetSize** is seven. Then get the number of nodes in the cluster, which in this case is five. So, in this case, two replicas can't be placed, because the target number of replicas is higher than the number of nodes available.

```
PS C:\> Get-ServiceFabricPartition fabric:/WordCount/WordCountService | Get-ServiceFabricPartitionHealth -ReplicasFilter None -ExcludeHealthStatistics

PartitionId      : af2e3e44-a8f8-45ac-9f31-4093eb897600
AggregatedHealthState : Warning
UnhealthyEvaluations  :
    Unhealthy event: SourceId='System.FM', Property='State', HealthState='Warning',
ConsiderWarningAsError=false.

ReplicaHealthStates   : None
HealthEvents      :
    SourceId      : System.FM
    Property       : State
    HealthState    : Warning
```

```

        SequenceNumber      : 123
        SentAt            : 7/14/2017 4:55:39 PM
        ReceivedAt         : 7/14/2017 4:55:44 PM
        TTL               : Infinite
        Description        : Partition is below target replica or instance count.
fabric:/WordCount/WordCountService 7 2 af2e3e44-a8f8-45ac-9f31-4093eb897600
        N/S Ready _Node_2 13144442226002646
        N/S Ready _Node_4 131444422293113678
        N/S Ready _Node_3 131444422293113679
        N/S Ready _Node_1 131444422293118720
        N/P Ready _Node_0 131444422293118721
        (Showing 5 out of 5 replicas. Total available replicas: 5)

        RemoveWhenExpired   : False
        IsExpired           : False
        Transitions          : Error->Warning = 7/14/2017 4:55:44 PM, LastOk = 1/1/0001
12:00:00 AM

        SourceId             : System.PLB
        Property              : ServiceReplicaUnplacedHealth_Secondary_af2e3e44-a8f8-45ac-
9f31-4093eb897600
        HealthState           : Warning
        SequenceNumber        : 131445250939703027
        SentAt                : 7/14/2017 4:58:13 PM
        ReceivedAt             : 7/14/2017 4:58:14 PM
        TTL                  : 00:01:05
        Description            : The Load Balancer was unable to find a placement for one or
more of the Service's Replicas:
        Secondary replica could not be placed due to the following constraints and
properties:
        TargetReplicaSetSize: 7
        Placement Constraint: N/A
        Parent Service: N/A

        Constraint Elimination Sequence:
        Existing Secondary Replicas eliminated 4 possible node(s) for placement -- 1/5
node(s) remain.
        Existing Primary Replica eliminated 1 possible node(s) for placement -- 0/5 node(s)
remain.

        Nodes Eliminated By Constraints:

        Existing Secondary Replicas -- Nodes with Partition's Existing Secondary
Replicas/Instances:
        --
        FaultDomain:fd:/4 NodeName:_Node_4 NodeType:NodeType4 UpgradeDomain:4 UpgradeDomain:
ud:/4 Deactivation Intent/Status: None/None
        FaultDomain:fd:/3 NodeName:_Node_3 NodeType:NodeType3 UpgradeDomain:3 UpgradeDomain:
ud:/3 Deactivation Intent/Status: None/None
        FaultDomain:fd:/2 NodeName:_Node_2 NodeType:NodeType2 UpgradeDomain:2 UpgradeDomain:
ud:/2 Deactivation Intent/Status: None/None
        FaultDomain:fd:/1 NodeName:_Node_1 NodeType:NodeType1 UpgradeDomain:1 UpgradeDomain:
ud:/1 Deactivation Intent/Status: None/None

        Existing Primary Replica -- Nodes with Partition's Existing Primary Replica or
Secondary Replicas:
        --
        FaultDomain:fd:/0 NodeName:_Node_0 NodeType:NodeType0 UpgradeDomain:0 UpgradeDomain:
ud:/0 Deactivation Intent/Status: None/None

        RemoveWhenExpired   : True
        IsExpired           : False
        Transitions          : Error->Warning = 7/14/2017 4:56:14 PM, LastOk = 1/1/0001
12:00:00 AM

PS C:\> Get-ServiceFabricPartition fabric:/WordCount/WordCountService | select
MinReplicaSetSize,TargetReplicaSetSize

```

```

MinReplicaSetSize TargetReplicaSetSize
-----
2                 7
PS C:\> @(Get-ServiceFabricNode).Count
5

```

The following example shows the health of a partition that's stuck in reconfiguration due to the user not honoring the cancellation token in the **RunAsync** method. Investigating the health report of any replica marked as primary (P) can help to drill down further into the problem.

```

PS C:\utilities\ServiceFabricExplorer\ClientPackage\lib> Get-ServiceFabricPartitionHealth 0e40fd81-284d-4be4-a665-13bc5a6607ec -ExcludeHealthStatistics

PartitionId      : 0e40fd81-284d-4be4-a665-13bc5a6607ec
AggregatedHealthState : Warning
UnhealthyEvaluations :
    Unhealthy event: SourceId='System.FM', Property='State', HealthState='Warning',
    ConsiderWarningAsError=false.

HealthEvents     :
    SourceId          : System.FM
    Property           : State
    HealthState        : Warning
    SequenceNumber     : 7
    SentAt             : 8/27/2017 3:43:09 AM
    ReceivedAt         : 8/27/2017 3:43:32 AM
    TTL                : Infinite
    Description         : Partition reconfiguration is taking longer than expected.
    fabric:/app/test1 3 1 0e40fd81-284d-4be4-a665-13bc5a6607ec
    P/S Ready Node1 131482789658160654
    S/P Ready Node2 131482789688598467
    S/S Ready Node3 131482789688598468
    (Showing 3 out of 3 replicas. Total available replicas: 3)

    For more information see: http://aka.ms/sfhealth
    RemoveWhenExpired   : False
    IsExpired           : False
    Transitions          : Ok->Warning = 8/27/2017 3:43:32 AM, LastError = 1/1/0001
12:00:00 AM

```

This health report shows the state of the replicas of the partition undergoing reconfiguration:

```

P/S Ready Node1 131482789658160654
S/P Ready Node2 131482789688598467
S/S Ready Node3 131482789688598468

```

For each replica, the health report contains:

- Previous configuration role
- Current configuration role
- **Replica state**
- Node on which the replica is running
- Replica ID

In a case like the example, further investigation is needed. Investigate the health of each individual replica starting with the replicas marked as **Primary** and **Secondary** (131482789658160654 and 131482789688598467) in the previous example.

## Replica constraint violation

**System.PLB** reports a warning if it detects a replica constraint violation and can't place all partition replicas. The report details show which constraints and properties prevent the replica placement.

- **Sourceld:** System.PLB
- **Property:** Starts with **ReplicaConstraintViolation**.

## Replica system health reports

**System.RA**, which represents the reconfiguration agent component, is the authority for the replica state.

### State

System.RA reports OK when the replica has been created.

- **Sourceld:** System.RA
- **Property:** State.

The following example shows a healthy replica:

```
PS C:\> Get-ServiceFabricPartition fabric:/WordCount/WordCountService | Get-ServiceFabricReplica | where
{$_.ReplicaRole -eq "Primary"} | Get-ServiceFabricReplicaHealth

PartitionId      : af2e3e44-a8f8-45ac-9f31-4093eb897600
ReplicaId       : 131444422293118721
AggregatedHealthState : Ok
HealthEvents     :
    SourceId        : System.RA
    Property         : State
    HealthState     : Ok
    SequenceNumber   : 131445248920273536
    SentAt          : 7/14/2017 4:54:52 PM
    ReceivedAt       : 7/14/2017 4:55:13 PM
    TTL              : Infinite
    Description      : Replica has been created._Node_0
    RemoveWhenExpired : False
    IsExpired        : False
    Transitions       : Error->Ok = 7/14/2017 4:55:13 PM, LastWarning = 1/1/0001
12:00:00 AM
```

### ReplicaOpenStatus, ReplicaCloseStatus, ReplicaChangeRoleStatus

This property is used to indicate warnings or failures when attempting to open a replica, close a replica, or transition a replica from one role to another. For more information, see [Replica lifecycle](#). The failures might be exceptions thrown from the API calls or crashes of the service host process during this time. For failures due to API calls from C# code, Service Fabric adds the exception and stack trace to the health report.

These health warnings are raised after retrying the action locally some number of times (depending on policy). Service Fabric retries the action up to a maximum threshold. After the maximum threshold is reached, it might try to act to correct the situation. This attempt can cause these warnings to get cleared as it gives up on the action on this node. For example, if a replica is failing to open on a node, Service Fabric raises a health warning. If the replica continues to fail to open, Service Fabric acts to self-repair. This action might involve trying the same operation on another node. This causes the warning raised for this replica to be cleared.

- **Sourceld:** System.RA
- **Property:** **ReplicaOpenStatus**, **ReplicaCloseStatus**, and **ReplicaChangeRoleStatus**.
- **Next steps:** Investigate the service code or crash dumps to identify why the operation is failing.

The following example shows the health of a replica that's throwing `TargetException` from its open method. The description contains the point of failure, `IStatefulServiceReplica.Open`, the exception type

**TargetException**, and the stack trace.

```
PS C:\> Get-ServiceFabricReplicaHealth -PartitionId 337cf1df-6cab-4825-99a9-7595090c0b1b -  
ReplicaOrInstanceId 131483509874784794  
  
PartitionId      : 337cf1df-6cab-4825-99a9-7595090c0b1b  
ReplicaId       : 131483509874784794  
AggregatedHealthState : Warning  
UnhealthyEvaluations :  
    Unhealthy event: SourceId='System.RA', Property='ReplicaOpenStatus',  
HealthState='Warning',  
    ConsiderWarningAsError=false.  
  
HealthEvents      :  
    SourceId          : System.RA  
    Property          : ReplicaOpenStatus  
    HealthState       : Warning  
    SequenceNumber    : 131483510001453159  
    SentAt            : 8/27/2017 11:43:20 PM  
    ReceivedAt        : 8/27/2017 11:43:21 PM  
    TTL               : Infinite  
    Description       : Replica had multiple failures during open on _Node_0 API  
call: IStatefulServiceReplica.Open(); Error = System.Reflection.TargetInvocationException (-2146232828)  
Exception has been thrown by the target of an invocation.  
    at Microsoft.ServiceFabric.Replicator.RecoveryManager.d__31.MoveNext()  
--- End of stack trace from previous location where exception was thrown ---  
    at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)  
    at System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task task)  
    at Microsoft.ServiceFabric.Replicator.LoggingReplicator.d__137.MoveNext()  
--- End of stack trace from previous location where exception was thrown ---  
    at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)  
    at System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task task)  
    at Microsoft.ServiceFabric.Replicator.DynamicStateManager.d__109.MoveNext()  
--- End of stack trace from previous location where exception was thrown ---  
    at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)  
    at System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task task)  
    at Microsoft.ServiceFabric.Replicator.TransactionalReplicator.d__79.MoveNext()  
--- End of stack trace from previous location where exception was thrown ---  
    at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)  
    at System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task task)  
    at Microsoft.ServiceFabric.Replicator.StatefulServiceReplica.d__21.MoveNext()  
--- End of stack trace from previous location where exception was thrown ---  
    at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)  
    at System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task task)  
    at Microsoft.ServiceFabric.Services.Runtime.StatefulServiceReplicaAdapter.d__0.MoveNext()  
  
    For more information see: http://aka.ms/sfhealth  
        RemoveWhenExpired   : False  
        IsExpired          : False  
        Transitions         : Error->Warning = 8/27/2017 11:43:21 PM, LastOk = 1/1/0001  
12:00:00 AM
```

The following example shows a replica that's constantly crashing during close:

```
C:>Get-ServiceFabricReplicaHealth -PartitionId dcafb6b7-9446-425c-8b90-b3fdf3859e64 -ReplicaOrInstanceId
131483565548493142

PartitionId      : dcafb6b7-9446-425c-8b90-b3fdf3859e64
ReplicaId       : 131483565548493142
AggregatedHealthState : Warning
UnhealthyEvaluations :
    Unhealthy event: SourceId='System.RA', Property='ReplicaCloseStatus',
HealthState='Warning',
    ConsiderWarningAsError=false.

HealthEvents     :
    SourceId      : System.RA
    Property       : ReplicaCloseStatus
    HealthState   : Warning
    SequenceNumber: 131483565611258984
    SentAt        : 8/28/2017 1:16:01 AM
    ReceivedAt    : 8/28/2017 1:16:03 AM
    TTL           : Infinite
    Description   : Replica had multiple failures during close on _Node_1. The
application
host has crashed.

For more information see: http://aka.ms/sfhealth
RemoveWhenExpired : False
IsExpired        : False
Transitions       : Error->Warning = 8/28/2017 1:16:03 AM, LastOk = 1/1/0001
12:00:00 AM
```

## Reconfiguration

This property is used to indicate when a replica performing a [reconfiguration](#) detects that the reconfiguration is stalled or stuck. This health report might be on the replica whose current role is primary, except in the cases of a swap primary reconfiguration, where it might be on the replica that's being demoted from primary to active secondary.

The reconfiguration can be stuck for one of the following reasons:

- An action on the local replica, the same replica as the one performing the reconfiguration, is not completing. In this case, investigating the health reports on this replica from other components, System.RAP or System.RE, might provide additional information.
- An action is not completing on a remote replica. Replicas for which actions are pending are listed in the health report. Further investigation should be done on health reports for those remote replicas. There might also be communication problems between this node and the remote node.

In rare cases, the reconfiguration can be stuck due to communication or other problems between this node and the Failover Manager service.

- **Sourceld:** System.RA
- **Property: Reconfiguration.**
- **Next steps:** Investigate local or remote replicas depending on the description of the health report.

The following example shows a health report where a reconfiguration is stuck on the local replica. In this sample, it's due to a service not honoring the cancellation token.

```

PS C:\> Get-ServiceFabricReplicaHealth -PartitionId 9a0cedee-464c-4603-abbc-1cf57c4454f3 -
ReplicaOrInstanceId 131483600074836703

PartitionId      : 9a0cedee-464c-4603-abbc-1cf57c4454f3
ReplicaId       : 131483600074836703
AggregatedHealthState : Warning
UnhealthyEvaluations :
    Unhealthy event: SourceId='System.RA', Property='Reconfiguration',
HealthState='Warning',
    ConsiderWarningAsError=false.

HealthEvents     :
    SourceId      : System.RA
    Property       : Reconfiguration
    HealthState   : Warning
    SequenceNumber : 131483600309264482
    SentAt        : 8/28/2017 2:13:50 AM
    ReceivedAt    : 8/28/2017 2:13:57 AM
    TTL           : Infinite
    Description    : Reconfiguration is stuck. Waiting for response from the
local replica

For more information see: http://aka.ms/sfhealth
RemoveWhenExpired : False
IsExpired        : False
Transitions       : Error->Warning = 8/28/2017 2:13:57 AM, LastOk = 1/1/0001
12:00:00 AM

```

The following example shows a health report where a reconfiguration is stuck waiting for a response from two remote replicas. In this example, there are three replicas in the partition, including the current primary.

```

PS C:\> Get-ServiceFabricReplicaHealth -PartitionId 579d50c6-d670-4d25-af70-d706e4bc19a2 -
ReplicaOrInstanceId 131483956274977415

PartitionId      : 579d50c6-d670-4d25-af70-d706e4bc19a2
ReplicaId       : 131483956274977415
AggregatedHealthState : Warning
UnhealthyEvaluations :
    Unhealthy event: SourceId='System.RA', Property='Reconfiguration',
HealthState='Warning', ConsiderWarningAsError=false.

HealthEvents     :
    SourceId      : System.RA
    Property       : Reconfiguration
    HealthState   : Warning
    SequenceNumber : 131483960376212469
    SentAt        : 8/28/2017 12:13:57 PM
    ReceivedAt    : 8/28/2017 12:14:07 PM
    TTL           : Infinite
    Description    : Reconfiguration is stuck. Waiting for response from 2
replicas

Pending Replicas:
P/I Down 40 131483956244554282
S/S Down 20 131483956274972403

For more information see: http://aka.ms/sfhealth
RemoveWhenExpired : False
IsExpired        : False
Transitions       : Error->Warning = 8/28/2017 12:07:37 PM, LastOk = 1/1/0001
12:00:00 AM

```

This health report shows that the reconfiguration is stuck waiting for a response from two replicas:

```
P/I Down 40 131483956244554282  
S/S Down 20 131483956274972403
```

For each replica the following information is given:

- Previous configuration role
- Current configuration role
- [Replica state](#)
- Node ID
- Replica ID

To unblock the reconfiguration:

- The **down** replicas should be brought up.
- The **inbuild** replicas should complete the build and transition to ready.

### Slow service API call

**System.RAP** and **System.Replicator** report a warning if a call to the user service code takes longer than the configured time. The warning is cleared when the call completes.

- **SourceId:** System.RAP or System.Replicator
- **Property:** The name of the slow API. The description provides more details about the time the API has been pending.
- **Next steps:** Investigate why the call takes longer than expected.

The following example shows the health event from System.RAP for a reliable service that's not honoring the cancellation token in **RunAsync**:

```
PS C:\> Get-ServiceFabricReplicaHealth -PartitionId 5f6060fb-096f-45e4-8c3d-c26444d8dd10 -  
ReplicaOrInstanceId 131483966141404693

PartitionId      : 5f6060fb-096f-45e4-8c3d-c26444d8dd10
ReplicaId       : 131483966141404693
AggregatedHealthState : Warning
UnhealthyEvaluations  :
    Unhealthy event: SourceId='System.RA', Property='Reconfiguration',
    HealthState='Warning', ConsiderWarningAsError=false.

HealthEvents      :
    SourceId      : System.RAP
    Property       : IStatefulServiceReplica.ChangeRole(S)Duration
    HealthState   : Warning
    SequenceNumber: 131483966663476570
    SentAt        : 8/28/2017 12:24:26 PM
    ReceivedAt    : 8/28/2017 12:24:56 PM
    TTL           : Infinite
    Description    : The api IStatefulServiceReplica.ChangeRole(S) on _Node_1 is
stuck. Start Time (UTC): 2017-08-28 12:23:56.347.
    RemoveWhenExpired : False
    IsExpired     : False
    Transitions    : Error->Warning = 8/28/2017 12:24:56 PM, LastOk = 1/1/0001
12:00:00 AM
```

The property and text indicate which API got stuck. The next steps to take for different stuck APIs is different. Any API on the *IStatefulServiceReplica* or *IStatelessServiceInstance* is usually a bug in the service code. The following section describes how these translate to the [Reliable Services model](#):

- **IStatefulServiceReplica.Open**: This warning indicates that a call to `CreateServiceInstanceListeners`, `ICommunicationListener.OpenAsync`, or if overridden, `OnOpenAsync` is stuck.
- **IStatefulServiceReplica.Close** and **IStatefulServiceReplica.Abort**: The most common case is a service not honoring the cancellation token passed in to `RunAsync`. It might also be that `ICommunicationListener.CloseAsync`, or if overridden, `OnCloseAsync` is stuck.
- **IStatefulServiceReplica.ChangeRole(S)** and **IStatefulServiceReplica.ChangeRole(N)**: The most common case is a service not honoring the cancellation token passed in to `RunAsync`.
- **IStatefulServiceReplica.ChangeRole(P)**: The most common case is that the service has not returned a task from `RunAsync`.

Other API calls that can get stuck are on the **IReplicator** interface. For example:

- **IReplicator.CatchupReplicaSet**: This warning indicates one of two things. Either there are insufficient up replicas, which can be determined by looking at the replica status of the replicas in the partition or the System.FM health report for a stuck reconfiguration. Or the replicas are not acknowledging operations. The PowerShell command-let `Get-ServiceFabricDeployedReplicaDetail` can be used to determine the progress of all the replicas. The problem lies with replicas whose `LastAppliedReplicationSequenceNumber` is behind the primary's `CommittedSequenceNumber`.
- **IReplicator.BuildReplica()**: This warning indicates a problem in the build process. For more information, see [Replica lifecycle](#). It might be due to a misconfiguration of the replicator address. For more information, see [Configure stateful Reliable Services](#) and [Specify resources in a service manifest](#). It might also be a problem on the remote node.

### Replication queue full

**System.Replicator** reports a warning when the replication queue is full. On the primary, the replication queue usually becomes full because one or more secondary replicas are slow to acknowledge operations. On the secondary, this usually happens when the service is slow to apply the operations. The warning is cleared when the queue is no longer full.

- **Sourceld**: System.Replicator
- **Property**: **PrimaryReplicationQueueStatus** or **SecondaryReplicationQueueStatus**, depending on the replica role.

### Slow Naming operations

**System.NamingService** reports the health on its primary replica when a Naming operation takes longer than acceptable. Examples of Naming operations are `CreateServiceAsync` or `DeleteServiceAsync`. More methods can be found under FabricClient, for example under [service management methods](#) or [property management methods](#).

#### NOTE

The Naming service resolves service names to a location in the cluster and enables users to manage service names and properties. It's a Service Fabric partitioned-persisted service. One of the partitions represents the *Authority Owner*, which contains metadata about all Service Fabric names and services. The Service Fabric names are mapped to different partitions, called *Name Owner* partitions, so the service is extensible. Read more about the [Naming service](#).

When a Naming operation takes longer than expected, the operation is flagged with a warning report on the primary replica of the Naming service partition that serves the operation. If the operation completes successfully, the warning is cleared. If the operation completes with an error, the health report includes details about the error.

- **Sourceld**: System.NamingService

- **Property:** Starts with prefix "**Duration\_**" and identifies the slow operation and the Service Fabric name on which the operation is applied. For example, if create service at name **fabric:/MyApp/MyService** takes too long, the property is **Duration\_AOCreateService.fabric:/MyApp/MyService**. "AO" points to the role of the Naming partition for this name and operation.
- **Next steps:** Check to see why the Naming operation fails. Each operation can have different root causes. For example, the delete service might be stuck. The service could be stuck because the application host keeps crashing on a node due to a user bug in the service code.

The following example shows a create service operation. The operation took longer than the configured duration. "AO" retries and sends work to "NO." "NO" completed the last operation with TIMEOUT. In this case, the same replica is primary for both the "AO" and "NO" roles.

```

PartitionId      : 00000000-0000-0000-0000-000000001000
ReplicaId       : 131064359253133577
AggregatedHealthState : Warning
UnhealthyEvaluations :
    Unhealthy event: SourceId='System.NamingService',
Property='Duration_AOCreateService.fabric:/MyApp/MyService', HealthState='Warning',
ConsiderWarningAsError=false.

HealthEvents     :
    SourceId      : System.RA
    Property       : State
    HealthState   : Ok
    SequenceNumber: 131064359308715535
    SentAt        : 4/29/2016 8:38:50 PM
    ReceivedAt    : 4/29/2016 8:39:08 PM
    TTL           : Infinite
    Description   : Replica has been created.
    RemoveWhenExpired: False
    IsExpired     : False
    Transitions   : Error->Ok = 4/29/2016 8:39:08 PM, LastWarning = 1/1/0001
12:00:00 AM

    SourceId      : System.NamingService
    Property       : Duration_AOCreateService.fabric:/MyApp/MyService
    HealthState   : Warning
    SequenceNumber: 131064359526778775
    SentAt        : 4/29/2016 8:39:12 PM
    ReceivedAt    : 4/29/2016 8:39:38 PM
    TTL           : 00:05:00
    Description   : The AOCreateService started at 2016-04-29 20:39:08.677 is
taking longer than 30.000.
    RemoveWhenExpired: True
    IsExpired     : False
    Transitions   : Error->Warning = 4/29/2016 8:39:38 PM, LastOk = 1/1/0001
12:00:00 AM

    SourceId      : System.NamingService
    Property       : Duration_NOCreateService.fabric:/MyApp/MyService
    HealthState   : Warning
    SequenceNumber: 131064360657607311
    SentAt        : 4/29/2016 8:41:05 PM
    ReceivedAt    : 4/29/2016 8:41:08 PM
    TTL           : 00:00:15
    Description   : The NOCreateService started at 2016-04-29 20:39:08.689
completed with FABRIC_E_TIMEOUT in more than 30.000.
    RemoveWhenExpired: True
    IsExpired     : False
    Transitions   : Error->Warning = 4/29/2016 8:39:38 PM, LastOk = 1/1/0001
12:00:00 AM

```

# DeployedApplication system health reports

**System.Hosting** is the authority on deployed entities.

## Activation

System.Hosting reports as OK when an application has been successfully activated on the node. Otherwise, it reports an error.

- **SourceId:** System.Hosting
- **Property:** Activation, including the rollout version.
- **Next steps:** If the application is unhealthy, investigate why the activation failed.

The following example shows a successful activation:

```
PS C:\> Get-ServiceFabricDeployedApplicationHealth -NodeName _Node_1 -ApplicationName fabric:/WordCount -ExcludeHealthStatistics

ApplicationName      : fabric:/WordCount
NodeName            : _Node_1
AggregatedHealthState : Ok
DeployedServicePackageHealthStates :
    ServiceManifestName   : WordCountServicePkg
    ServicePackageActivationId :
    NodeName              : _Node_1
    AggregatedHealthState : Ok

HealthEvents       :
    SourceId          : System.Hosting
    Property           : Activation
    HealthState        : Ok
    SequenceNumber     : 131445249083836329
    SentAt             : 7/14/2017 4:55:08 PM
    ReceivedAt         : 7/14/2017 4:55:14 PM
    TTL                : Infinite
    Description         : The application was activated successfully.
    RemoveWhenExpired  : False
    IsExpired          : False
    Transitions         : Error->Ok = 7/14/2017 4:55:14 PM, LastWarning =
1/1/0001 12:00:00 AM
```

## Download

System.Hosting reports an error if the application package download fails.

- **SourceId:** System.Hosting
- **Property: Download:RolloutVersion.**
- **Next steps:** Investigate why the download failed on the node.

# DeployedServicePackage system health reports

**System.Hosting** is the authority on deployed entities.

## Service package activation

System.Hosting reports as OK if the service package activation on the node is successful. Otherwise, it reports an error.

- **SourceId:** System.Hosting
- **Property:** Activation.
- **Next steps:** Investigate why the activation failed.

## Code package activation

System.Hosting reports as OK for each code package if the activation is successful. If the activation fails, it reports a warning as configured. If **CodePackage** fails to activate or terminates with an error greater than the configured **CodePackageHealthErrorThreshold**, hosting reports an error. If a service package contains multiple code packages, an activation report is generated for each one.

- **Sourceld:** System.Hosting
- **Property:** Uses the prefix **CodePackageActivation** and contains the name of the code package and the entry point as **CodePackageActivation:CodePackageName:SetupEntryPoint/EntryPoint**. For example, **CodePackageActivation:Code:SetupEntryPoint**.

## Service type registration

System.Hosting reports as OK if the service type has been registered successfully. It reports an error if the registration wasn't done in time, as configured by using **ServiceTypeRegistrationTimeout**. If the runtime is closed, the service type is unregistered from the node and hosting reports a warning.

- **Sourceld:** System.Hosting
- **Property:** Uses the prefix **ServiceTypeRegistration** and contains the service type name. For example, **ServiceTypeRegistration:FileStoreServiceType**.

The following example shows a healthy deployed service package:

```

PS C:\> Get-ServiceFabricDeployedServicePackageHealth -NodeName _Node_1 -ApplicationName fabric:/WordCount -ServiceManifestName WordCountServicePkg

ApplicationName      : fabric:/WordCount
ServiceManifestName  : WordCountServicePkg
ServicePackageActivationId :
NodeName            : _Node_1
AggregatedHealthState : Ok
HealthEvents        :
    SourceId          : System.Hosting
    Property           : Activation
    HealthState        : Ok
    SequenceNumber     : 131445249084026346
    SentAt             : 7/14/2017 4:55:08 PM
    ReceivedAt         : 7/14/2017 4:55:14 PM
    TTL                : Infinite
    Description         : The ServicePackage was activated successfully.
    RemoveWhenExpired  : False
    IsExpired          : False
    Transitions         : Error->Ok = 7/14/2017 4:55:14 PM, LastWarning =
1/1/0001 12:00:00 AM

    SourceId          : System.Hosting
    Property           : CodePackageActivation:Code:EntryPoint
    HealthState        : Ok
    SequenceNumber     : 131445249084306362
    SentAt             : 7/14/2017 4:55:08 PM
    ReceivedAt         : 7/14/2017 4:55:14 PM
    TTL                : Infinite
    Description         : The CodePackage was activated successfully.
    RemoveWhenExpired  : False
    IsExpired          : False
    Transitions         : Error->Ok = 7/14/2017 4:55:14 PM, LastWarning =
1/1/0001 12:00:00 AM

    SourceId          : System.Hosting
    Property           : ServiceTypeRegistration:WordCountServiceType
    HealthState        : Ok
    SequenceNumber     : 131445249088096842
    SentAt             : 7/14/2017 4:55:08 PM
    ReceivedAt         : 7/14/2017 4:55:14 PM
    TTL                : Infinite
    Description         : The ServiceType was registered successfully.
    RemoveWhenExpired  : False
    IsExpired          : False
    Transitions         : Error->Ok = 7/14/2017 4:55:14 PM, LastWarning =
1/1/0001 12:00:00 AM

```

## Download

System.Hosting reports an error if the service package download fails.

- **SourceId:** System.Hosting
- **Property: Download:RolloutVersion.**
- **Next steps:** Investigate why the download failed on the node.

## Upgrade validation

System.Hosting reports an error if validation during the upgrade fails or if the upgrade fails on the node.

- **SourceId:** System.Hosting
- **Property:** Uses the prefix **FabricUpgradeValidation** and contains the upgrade version.
- **Description:** Points to the error encountered.

## **Undefined node capacity for resource governance metrics**

System.Hosting reports a warning if node capacities are not defined in the cluster manifest and config for automatic detection is turned off. Service Fabric will raise health warning whenever service package that uses [resource governance](#) registers on a specified node.

- **Sourceld:** System.Hosting
- **Property:** ResourceGovernance
- **Next steps:** The preferred way to overcome this problem is to change the cluster manifest to enable automatic detection of available resources. Another way is updating the cluster manifest with correctly specified node capacities for these metrics.

## Next steps

[View Service Fabric health reports](#)

[How to report and check service health](#)

[Monitor and diagnose services locally](#)

[Service Fabric application upgrade](#)

# View Service Fabric health reports

9/25/2017 • 31 min to read • [Edit Online](#)

Azure Service Fabric introduces a [health model](#) with health entities on which system components and watchdogs can report local conditions that they are monitoring. The [health store](#) aggregates all health data to determine whether entities are healthy.

The cluster is automatically populated with health reports sent by the system components. Read more at [Use system health to troubleshoot](#).

Service Fabric provides multiple ways to get the aggregated health of the entities:

- [Service Fabric Explorer](#) or other visualization tools
- Health queries (through PowerShell, API, or REST)
- General queries that return a list of entities that have health as one of the properties (through PowerShell, API, or REST)

To demonstrate these options, let's use a local cluster with five nodes and the [fabric:/WordCount application](#). The **fabric:/WordCount** application contains two default services, a stateful service of type `WordCountServiceType`, and a stateless service of type `WordCountWebServiceType`. I changed the `ApplicationManifest.xml` to require seven target replicas for the stateful service and one partition. Because there are only five nodes in the cluster, the system components report a warning on the service partition because it is below the target count.

```
<Service Name="WordCountService">
  <StatefulService ServiceTypeName="WordCountServiceType" TargetReplicaSetSize="7" MinReplicaSetSize="2">
    <UniformInt64Partition PartitionCount="[WordCountService_PartitionCount]" LowKey="1" HighKey="26" />
  </StatefulService>
</Service>
```

## Health in Service Fabric Explorer

Service Fabric Explorer provides a visual view of the cluster. In the image below, you can see that:

- The application **fabric:/WordCount** is red (in error) because it has an error event reported by **MyWatchdog** for the property **Availability**.
- One of its services, **fabric:/WordCount/WordCountService** is yellow (in warning). The service is configured with seven replicas and the cluster has five nodes, so two replicas can't be placed. Although it's not shown here, the service partition is yellow because of a system report from `System.FM` saying that `Partition is below target replica or instance count`. The yellow partition triggers the yellow service.
- The cluster is red because of the red application.

The evaluation uses default policies from the cluster manifest and application manifest. They are strict policies and do not tolerate any failure.

View of the cluster with Service Fabric Explorer:

#### NOTE

Read more about [Service Fabric Explorer](#).

## Health queries

Service Fabric exposes health queries for each of the supported [entity types](#). They can be accessed through the API, using methods on [FabricClient.HealthManager](#), PowerShell cmdlets, and REST. These queries return complete health information about the entity: the aggregated health state, entity health events, child health states (when applicable), unhealthy evaluations (when the entity is not healthy), and children health statistics (when applicable).

#### NOTE

A health entity is returned when it is fully populated in the health store. The entity must be active (not deleted) and have a system report. Its parent entities on the hierarchy chain must also have system reports. If any of these conditions are not satisfied, the health queries return a [FabricException](#) with [FabricErrorCode FabricHealthEntityNotFound](#) that shows why the entity is not returned.

The health queries must pass in the entity identifier, which depends on the entity type. The queries accept optional health policy parameters. If no health policies are specified, the [health policies](#) from the cluster or application manifest are used for evaluation. If the manifests don't contain a definition for health policies, the default health policies are used for evaluation. The default health policies do not tolerate any failures. The queries also accept filters for returning only partial children or events--the ones that respect the specified filters. Another filter allows excluding the children statistics.

#### NOTE

The output filters are applied on the server side, so the message reply size is reduced. We recommended that you use the output filters to limit the data returned, rather than apply filters on the client side.

An entity's health contains:

- The aggregated health state of the entity. Computed by the health store based on entity health reports, child health states (when applicable), and health policies. Read more about [entity health evaluation](#).
- The health events on the entity.
- The collection of health states of all children for the entities that can have children. The health states contain entity identifiers and the aggregated health state. To get complete health for a child, call the query health for the child entity type and pass in the child identifier.
- The unhealthy evaluations that point to the report that triggered the state of the entity, if the entity is not healthy. The evaluations are recursive, containing the children health evaluations that triggered current health

state. For example, a watchdog reported an error against a replica. The application health shows an unhealthy evaluation due to an unhealthy service; the service is unhealthy due to a partition in error; the partition is unhealthy due to a replica in error; the replica is unhealthy due to the watchdog error health report.

- The health statistics for all children types of the entities that have children. For example, cluster health shows the total number of applications, services, partitions, replicas, and deployed entities in the cluster. Service health shows the total number of partitions and replicas under the specified service.

## Get cluster health

Returns the health of the cluster entity and contains the health states of applications and nodes (children of the cluster). Input:

- [Optional] The cluster health policy used to evaluate the nodes and the cluster events.
- [Optional] The application health policy map, with the health policies used to override the application manifest policies.
- [Optional] Filters for events, nodes, and applications that specify which entries are of interest and should be returned in the result (for example, errors only, or both warnings and errors). All events, nodes, and applications are used to evaluate the entity aggregated health, regardless of the filter.
- [Optional] Filter to exclude health statistics.
- [Optional] Filter to include fabric:/System health statistics in the health statistics. Only applicable when the health statistics are not excluded. By default, the health statistics include only statistics for user applications and not the System application.

### API

To get cluster health, create a `FabricClient` and call the `GetClusterHealthAsync` method on its **HealthManager**.

The following call gets the cluster health:

```
ClusterHealth clusterHealth = await fabricClient.HealthManager.GetClusterHealthAsync();
```

The following code gets the cluster health by using a custom cluster health policy and filters for nodes and applications. It specifies that the health statistics include the fabric:/System statistics. It creates `ClusterHealthQueryDescription`, which contains the input information.

```

var policy = new ClusterHealthPolicy()
{
    MaxPercentUnhealthyNodes = 20
};

var nodesFilter = new NodeHealthStatesFilter()
{
    HealthStateFilterValue = HealthStateFilter.Error | HealthStateFilter.Warning
};

var applicationsFilter = new ApplicationHealthStatesFilter()
{
    HealthStateFilterValue = HealthStateFilter.Error
};

var healthStatisticsFilter = new ClusterHealthStatisticsFilter()
{
    ExcludeHealthStatistics = false,
    IncludeSystemApplicationHealthStatistics = true
};

var queryDescription = new ClusterHealthQueryDescription()
{
    HealthPolicy = policy,
    ApplicationsFilter = applicationsFilter,
    NodesFilter = nodesFilter,
    HealthStatisticsFilter = healthStatisticsFilter
};

ClusterHealth clusterHealth = await fabricClient.HealthManager.GetClusterHealthAsync(queryDescription);

```

## PowerShell

The cmdlet to get the cluster health is [Get-ServiceFabricClusterHealth](#). First, connect to the cluster by using the [Connect-ServiceFabricCluster](#) cmdlet.

The state of the cluster is five nodes, the system application, and fabric:/WordCount configured as described.

The following cmdlet gets cluster health by using default health policies. The aggregated health state is warning, because the fabric:/WordCount application is in warning. Note how the unhealthy evaluations provide details on the conditions that triggered the aggregated health.

```

PS D:\ServiceFabric> Get-ServiceFabricClusterHealth

AggregatedHealthState   : Warning
UnhealthyEvaluations    :
    Unhealthy applications: 100% (1/1), MaxPercentUnhealthyApplications=0%.
    Unhealthy application: ApplicationName='fabric:/WordCount',
    AggregatedHealthState='Warning'.

    Unhealthy services: 100% (1/1), ServiceType='WordCountServiceType',
    MaxPercentUnhealthyServices=0%.
    Unhealthy service: ServiceName='fabric:/WordCount/WordCountService',
    AggregatedHealthState='Warning'.

    Unhealthy partitions: 100% (1/1), MaxPercentUnhealthyPartitionsPerService=0%.
    Unhealthy partition: PartitionId='af2e3e44-a8f8-45ac-9f31-4093eb897600',
    AggregatedHealthState='Warning'.

    Unhealthy event: SourceId='System.FM', Property='State',
    HealthState='Warning', ConsiderWarningAsError=false.

NodeHealthStates         :
    NodeName          : _Node_4
    AggregatedHealthState : Ok

    NodeName          : _Node_3
    AggregatedHealthState : Ok

    NodeName          : _Node_2
    AggregatedHealthState : Ok

    NodeName          : _Node_1
    AggregatedHealthState : Ok

    NodeName          : _Node_0
    AggregatedHealthState : Ok

ApplicationHealthStates :
    ApplicationName     : fabric:/System
    AggregatedHealthState : Ok

    ApplicationName     : fabric:/WordCount
    AggregatedHealthState : Warning

HealthEvents              : None
HealthStatistics          :
    Node               : 5 Ok, 0 Warning, 0 Error
    Replica            : 6 Ok, 0 Warning, 0 Error
    Partition          : 1 Ok, 1 Warning, 0 Error
    Service             : 1 Ok, 1 Warning, 0 Error
    DeployedServicePackage : 6 Ok, 0 Warning, 0 Error
    DeployedApplication  : 5 Ok, 0 Warning, 0 Error
    Application         : 0 Ok, 1 Warning, 0 Error

```

The following PowerShell cmdlet gets the health of the cluster by using a custom application policy. It filters results to get only applications and nodes in error or warning. As a result, no nodes are returned, as they are all healthy. Only the fabric:/WordCount application respects the applications filter. Because the custom policy specifies to consider warnings as errors for the fabric:/WordCount application, the application is evaluated as in error, and so is the cluster.

```

PS D:\ServiceFabric> $appHealthPolicy = New-Object -TypeName System.Fabric.Health.ApplicationHealthPolicy
$appHealthPolicy.ConsiderWarningAsError = $true
$appHealthPolicyMap = New-Object -TypeName System.Fabric.Health.ApplicationHealthPolicyMap
$appUri1 = New-Object -TypeName System.Uri -ArgumentList "fabric:/WordCount"
$appHealthPolicyMap.Add($appUri1, $appHealthPolicy)
Get-ServiceFabricClusterHealth -ApplicationHealthPolicyMap $appHealthPolicyMap -ApplicationsFilter
"Warning,Error" -NodesFilter "Warning,Error" -ExcludeHealthStatistics

AggregatedHealthState : Error
UnhealthyEvaluations :
    Unhealthy applications: 100% (1/1), MaxPercentUnhealthyApplications=0%.
        Unhealthy application: ApplicationName='fabric:/WordCount',
AggregatedHealthState='Error'.

        Unhealthy services: 100% (1/1), ServiceType='WordCountServiceType',
MaxPercentUnhealthyServices=0%.
            Unhealthy service: ServiceName='fabric:/WordCount/WordCountService',
AggregatedHealthState='Error'.

            Unhealthy partitions: 100% (1/1), MaxPercentUnhealthyPartitionsPerService=0%.
                Unhealthy partition: PartitionId='af2e3e44-a8f8-45ac-9f31-4093eb897600',
AggregatedHealthState='Error'.

                Unhealthy event: SourceId='System.FM', Property='State',
HealthState='Warning', ConsiderWarningAsError=true.

NodeHealthStates : None
ApplicationHealthStates :
    ApplicationName : fabric:/WordCount
    AggregatedHealthState : Error

HealthEvents : None

```

## REST

You can get cluster health with a [GET request](#) or a [POST request](#) that includes health policies described in the body.

## Get node health

Returns the health of a node entity and contains the health events reported on the node. Input:

- [Required] The node name that identifies the node.
- [Optional] The cluster health policy settings used to evaluate health.
- [Optional] Filters for events that specify which entries are of interest and should be returned in the result (for example, errors only, or both warnings and errors). All events are used to evaluate the entity aggregated health, regardless of the filter.

## API

To get node health through the API, create a `FabricClient` and call the `GetNodeHealthAsync` method on its `HealthManager`.

The following code gets the node health for the specified node name:

```
NodeHealth nodeHealth = await fabricClient.HealthManager.GetNodeHealthAsync(nodeName);
```

The following code gets the node health for the specified node name and passes in events filter and custom policy through [NodeHealthQueryDescription](#):

```
var queryDescription = new NodeHealthQueryDescription(nodeName)
{
    HealthPolicy = new ClusterHealthPolicy() { ConsiderWarningAsError = true },
    EventsFilter = new HealthEventsFilter() { HealthStateFilterValue = HealthStateFilter.Warning },
};

NodeHealth nodeHealth = await fabricClient.HealthManager.GetNodeHealthAsync(queryDescription);
```

## PowerShell

The cmdlet to get the node health is [Get-ServiceFabricNodeHealth](#). First, connect to the cluster by using the [Connect-ServiceFabricCluster](#) cmdlet. The following cmdlet gets the node health by using default health policies:

```
PS D:\ServiceFabric> Get-ServiceFabricNodeHealth _Node_1

NodeName          : _Node_1
AggregatedHealthState : Ok
HealthEvents      :
    SourceId      : System.FM
    Property       : State
    HealthState    : Ok
    SequenceNumber : 3
    SentAt        : 7/13/2017 4:39:23 PM
    ReceivedAt    : 7/13/2017 4:40:47 PM
    TTL           : Infinite
    Description    : Fabric node is up.
    RemoveWhenExpired : False
    IsExpired     : False
    Transitions    : Error->Ok = 7/13/2017 4:40:47 PM, LastWarning = 1/1/0001
12:00:00 AM
```

The following cmdlet gets the health of all nodes in the cluster:

```
PS D:\ServiceFabric> Get-ServiceFabricNode | Get-ServiceFabricNodeHealth | select NodeName,
AggregatedHealthState | ft -AutoSize

NodeName AggregatedHealthState
-----
_Node_4          Ok
_Node_3          Ok
_Node_2          Ok
_Node_1          Ok
_Node_0          Ok
```

## REST

You can get node health with a [GET request](#) or a [POST request](#) that includes health policies described in the body.

## Get application health

Returns the health of an application entity. It contains the health states of the deployed application and service children. Input:

- [Required] The application name (URI) that identifies the application.
- [Optional] The application health policy used to override the application manifest policies.
- [Optional] Filters for events, services, and deployed applications that specify which entries are of interest and

should be returned in the result (for example, errors only, or both warnings and errors). All events, services, and deployed applications are used to evaluate the entity aggregated health, regardless of the filter.

- [Optional] Filter to exclude the health statistics. If not specified, the health statistics include the ok, warning, and error count for all application children: services, partitions, replicas, deployed applications, and deployed service packages.

## API

To get application health, create a `FabricClient` and call the [GetApplicationHealthAsync](#) method on its `HealthManager`.

The following code gets the application health for the specified application name (URI):

```
ApplicationHealth applicationHealth = await  
fabricClient.HealthManager.GetApplicationHealthAsync(applicationName);
```

The following code gets the application health for the specified application name (URI), with filters and custom policies specified via [ApplicationHealthQueryDescription](#).

```
HealthStateFilter warningAndErrors = HealthStateFilter.Error | HealthStateFilter.Warning;  
var serviceTypePolicy = new ServiceTypeHealthPolicy()  
{  
    MaxPercentUnhealthyPartitionsPerService = 0,  
    MaxPercentUnhealthyReplicasPerPartition = 5,  
    MaxPercentUnhealthyServices = 0,  
};  
var policy = new ApplicationHealthPolicy()  
{  
    ConsiderWarningAsError = false,  
    DefaultServiceTypeHealthPolicy = serviceTypePolicy,  
    MaxPercentUnhealthyDeployedApplications = 0,  
};  
  
var queryDescription = new ApplicationHealthQueryDescription(applicationName)  
{  
    HealthPolicy = policy,  
    EventsFilter = new HealthEventsFilter() { HealthStateFilterValue = warningAndErrors },  
    ServicesFilter = new ServiceHealthStatesFilter() { HealthStateFilterValue = warningAndErrors },  
    DeployedApplicationsFilter = new DeployedApplicationHealthStatesFilter() { HealthStateFilterValue =  
warningAndErrors },  
};  
  
ApplicationHealth applicationHealth = await  
fabricClient.HealthManager.GetApplicationHealthAsync(queryDescription);
```

## PowerShell

The cmdlet to get the application health is [Get-ServiceFabricApplicationHealth](#). First, connect to the cluster by using the [Connect-ServiceFabricCluster](#) cmdlet.

The following cmdlet returns the health of the **fabric:/WordCount** application:

```
PS D:\ServiceFabric> Get-ServiceFabricApplicationHealth fabric:/WordCount  
  
ApplicationName : fabric:/WordCount  
AggregatedHealthState : Warning  
UnhealthyEvaluations :  
    Unhealthy services: 100% (1/1), ServiceType='WordCountServiceType',  
    MaxPercentUnhealthyServices=0%.  
  
    Unhealthy service: ServiceName='fabric:/WordCount/WordCountService',
```

```

AggregatedHealthState='Warning' .

Unhealthy partitions: 100% (1/1),
MaxPercentUnhealthyPartitionsPerService=0%.


Unhealthy partition: PartitionId='af2e3e44-a8f8-45ac-9f31-4093eb897600',
AggregatedHealthState='Warning'.


Unhealthy event: SourceId='System.FM', Property='State',
HealthState='Warning', ConsiderWarningAsError=false.


ServiceHealthStates      :
    ServiceName          : fabric:/WordCount/WordCountWebService
    AggregatedHealthState : Ok

    ServiceName          : fabric:/WordCount/WordCountService
    AggregatedHealthState : Warning


DeployedApplicationHealthStates :
    ApplicationName       : fabric:/WordCount
    NodeName              : _Node_4
    AggregatedHealthState : Ok

    ApplicationName       : fabric:/WordCount
    NodeName              : _Node_3
    AggregatedHealthState : Ok

    ApplicationName       : fabric:/WordCount
    NodeName              : _Node_0
    AggregatedHealthState : Ok

    ApplicationName       : fabric:/WordCount
    NodeName              : _Node_2
    AggregatedHealthState : Ok

    ApplicationName       : fabric:/WordCount
    NodeName              : _Node_1
    AggregatedHealthState : Ok


HealthEvents      :
    SourceId           : System.CM
    Property            : State
    HealthState         : Ok
    SequenceNumber      : 282
    SentAt              : 7/13/2017 5:57:05 PM
    ReceivedAt          : 7/13/2017 5:57:05 PM
    TTL                 : Infinite
    Description          : Application has been created.
    RemoveWhenExpired   : False
    IsExpired            : False
    Transitions          : Error->Ok = 7/13/2017 5:57:05 PM, LastWarning =
1/1/0001 12:00:00 AM


HealthStatistics  :
    Replica             : 6 Ok, 0 Warning, 0 Error
    Partition            : 1 Ok, 1 Warning, 0 Error
    Service              : 1 Ok, 1 Warning, 0 Error
    DeployedServicePackage : 6 Ok, 0 Warning, 0 Error
    DeployedApplication   : 5 Ok, 0 Warning, 0 Error

```

The following PowerShell cmdlet passes in custom policies. It also filters children and events.

```

PS D:\ServiceFabric> Get-ServiceFabricApplicationHealth -ApplicationName fabric:/WordCount -
ConsiderWarningAsError $true -ServicesFilter Error -EventsFilter Error -DeployedApplicationsFilter Error -
ExcludeHealthStatistics

ApplicationName          : fabric:/WordCount
AggregatedHealthState   : Error
UnhealthyEvaluations    :
                           Unhealthy services: 100% (1/1), ServiceType='WordCountServiceType',
MaxPercentUnhealthyServices=0%.
                           Unhealthy service: ServiceName='fabric:/WordCount/WordCountService',
AggregatedHealthState='Error'.
                           Unhealthy partitions: 100% (1/1),
MaxPercentUnhealthyPartitionsPerService=0%.
                           Unhealthy partition: PartitionId='af2e3e44-a8f8-45ac-9f31-4093eb897600',
AggregatedHealthState='Error'.
                           Unhealthy event: SourceId='System.FM', Property='State',
HealthState='Warning', ConsiderWarningAsError=true.

ServiceHealthStates      :
                           ServiceName       : fabric:/WordCount/WordCountService
                           AggregatedHealthState : Error
DeployedApplicationHealthStates : None
HealthEvents              : None

```

## REST

You can get application health with a [GET request](#) or a [POST request](#) that includes health policies described in the body.

## Get service health

Returns the health of a service entity. It contains the partition health states. Input:

- [Required] The service name (URI) that identifies the service.
- [Optional] The application health policy used to override the application manifest policy.
- [Optional] Filters for events and partitions that specify which entries are of interest and should be returned in the result (for example, errors only, or both warnings and errors). All events and partitions are used to evaluate the entity aggregated health, regardless of the filter.
- [Optional] Filter to exclude health statistics. If not specified, the health statistics show the ok, warning, and error count for all partitions and replicas of the service.

## API

To get service health through the API, create a `FabricClient` and call the [GetServiceHealthAsync](#) method on its `HealthManager`.

The following example gets the health of a service with specified service name (URI):

```
ServiceHealth serviceHealth = await fabricClient.HealthManager.GetServiceHealthAsync(serviceName);
```

The following code gets the service health for the specified service name (URI), specifying filters and custom policy via [ServiceHealthQueryDescription](#):

```

var queryDescription = new ServiceHealthQueryDescription(serviceName)
{
    EventsFilter = new HealthEventsFilter() { HealthStateFilterValue = HealthStateFilter.All },
    PartitionsFilter = new PartitionHealthStatesFilter() { HealthStateFilterValue = HealthStateFilter.Error },
};

ServiceHealth serviceHealth = await fabricClient.HealthManager.GetServiceHealthAsync(queryDescription);

```

## PowerShell

The cmdlet to get the service health is [Get-ServiceFabricServiceHealth](#). First, connect to the cluster by using the [Connect-ServiceFabricCluster](#) cmdlet.

The following cmdlet gets the service health by using default health policies:

```

PS D:\ServiceFabric> Get-ServiceFabricServiceHealth -ServiceName fabric:/WordCount/WordCountService

ServiceName          : fabric:/WordCount/WordCountService
AggregatedHealthState : Warning
UnhealthyEvaluations :
    Unhealthy partitions: 100% (1/1), MaxPercentUnhealthyPartitionsPerService=0%.
    Unhealthy partition: PartitionId='af2e3e44-a8f8-45ac-9f31-4093eb897600',
    AggregatedHealthState='Warning'.
    Unhealthy event: SourceId='System.FM', Property='State', HealthState='Warning',
    ConsiderWarningAsError=false.

PartitionHealthStates :
    PartitionId          : af2e3e44-a8f8-45ac-9f31-4093eb897600
    AggregatedHealthState : Warning

HealthEvents          :
    SourceId            : System.FM
    Property             : State
    HealthState          : Ok
    SequenceNumber       : 15
    SentAt               : 7/13/2017 5:57:05 PM
    ReceivedAt           : 7/13/2017 5:57:18 PM
    TTL                 : Infinite
    Description          : Service has been created.
    RemoveWhenExpired   : False
    IsExpired            : False
    Transitions          : Error->Ok = 7/13/2017 5:57:18 PM, LastWarning = 1/1/0001
    12:00:00 AM

HealthStatistics      :
    Replica              : 5 Ok, 0 Warning, 0 Error
    Partition             : 0 Ok, 1 Warning, 0 Error

```

## REST

You can get service health with a [GET request](#) or a [POST request](#) that includes health policies described in the body.

## Get partition health

Returns the health of a partition entity. It contains the replica health states. Input:

- [Required] The partition ID (GUID) that identifies the partition.
- [Optional] The application health policy used to override the application manifest policy.

- [Optional] Filters for events and replicas that specify which entries are of interest and should be returned in the result (for example, errors only, or both warnings and errors). All events and replicas are used to evaluate the entity aggregated health, regardless of the filter.
- [Optional] Filter to exclude health statistics. If not specified, the health statistics show how many replicas are in ok, warning, and error states.

## API

To get partition health through the API, create a `FabricClient` and call the `GetPartitionHealthAsync` method on its `HealthManager`. To specify optional parameters, create `PartitionHealthQueryDescription`.

```
PartitionHealth partitionHealth = await fabricClient.HealthManager.GetPartitionHealthAsync(partitionId);
```

## PowerShell

The cmdlet to get the partition health is `Get-ServiceFabricPartitionHealth`. First, connect to the cluster by using the `Connect-ServiceFabricCluster` cmdlet.

The following cmdlet gets the health for all partitions of the **fabric:/WordCount/WordCountService** service and filters out replica health states:

```
PS D:\ServiceFabric> Get-ServiceFabricPartition fabric:/WordCount/WordCountService | Get-ServiceFabricPartitionHealth -ReplicasFilter None

PartitionId      : af2e3e44-a8f8-45ac-9f31-4093eb897600
AggregatedHealthState : Warning
UnhealthyEvaluations :
    Unhealthy event: SourceId='System.FM', Property='State', HealthState='Warning',
ConsiderWarningAsError=false.

ReplicaHealthStates : None
HealthEvents :
    SourceId      : System.FM
    Property      : State
    HealthState   : Warning
    SequenceNumber : 72
    SentAt        : 7/13/2017 5:57:29 PM
    ReceivedAt    : 7/13/2017 5:57:48 PM
    TTL           : Infinite
    Description   : Partition is below target replica or instance count.
    fabric:/WordCount/WordCountService 7 2 af2e3e44-a8f8-45ac-9f31-4093eb897600
    N/P RD _Node_2 Up 131444422260002646
    N/S RD _Node_4 Up 131444422293113678
    N/S RD _Node_3 Up 131444422293113679
    N/S RD _Node_1 Up 131444422293118720
    N/S RD _Node_0 Up 131444422293118721
    (Showing 5 out of 5 replicas. Total available replicas: 5.)

    RemoveWhenExpired : False
    IsExpired        : False
    Transitions       : Ok->Warning = 7/13/2017 5:57:48 PM, LastError = 1/1/0001
12:00:00 AM

    SourceId      : System.PLB
    Property      : ServiceReplicaUnplacedHealth_Secondary_af2e3e44-a8f8-45ac-
9f31-4093eb897600
    HealthState   : Warning
    SequenceNumber : 131444445174851664
    SentAt        : 7/13/2017 6:35:17 PM
    ReceivedAt    : 7/13/2017 6:35:18 PM
    TTL           : 00:01:05
    Description   : The Load Balancer was unable to find a placement for one or
more of the Service's Replicas:
    Secondary replica could not be placed due to the following constraints and
```

```

properties:
    TargetReplicaSetSize: 7
    Placement Constraint: N/A
    Parent Service: N/A

    Constraint Elimination Sequence:
    Existing Secondary Replicas eliminated 4 possible node(s) for placement -- 1/5
node(s) remain.
    Existing Primary Replica eliminated 1 possible node(s) for placement -- 0/5 node(s)
remain.

    Nodes Eliminated By Constraints:

    Existing Secondary Replicas -- Nodes with Partition's Existing Secondary
Replicas/Instances:
    --
        FaultDomain:fd:/4 NodeName:_Node_4 NodeType:NodeType4 UpgradeDomain:4 UpgradeDomain:
ud:/4 Deactivation Intent/Status: None/None
        FaultDomain:fd:/3 NodeName:_Node_3 NodeType:NodeType3 UpgradeDomain:3 UpgradeDomain:
ud:/3 Deactivation Intent/Status: None/None
        FaultDomain:fd:/1 NodeName:_Node_1 NodeType:NodeType1 UpgradeDomain:1 UpgradeDomain:
ud:/1 Deactivation Intent/Status: None/None
        FaultDomain:fd:/0 NodeName:_Node_0 NodeType:NodeType0 UpgradeDomain:0 UpgradeDomain:
ud:/0 Deactivation Intent/Status: None/None

    Existing Primary Replica -- Nodes with Partition's Existing Primary Replica or
Secondary Replicas:
    --
        FaultDomain:fd:/2 NodeName:_Node_2 NodeType:NodeType2 UpgradeDomain:2 UpgradeDomain:
ud:/2 Deactivation Intent/Status: None/None

    RemoveWhenExpired : True
    IsExpired : False
    Transitions : Error->Warning = 7/13/2017 5:57:48 PM, LastOk = 1/1/0001
12:00:00 AM

    HealthStatistics :
        Replica : 5 Ok, 0 Warning, 0 Error

```

## REST

You can get partition health with a [GET request](#) or a [POST request](#) that includes health policies described in the body.

## Get replica health

Returns the health of a stateful service replica or a stateless service instance. Input:

- [Required] The partition ID (GUID) and replica ID that identifies the replica.
- [Optional] The application health policy parameters used to override the application manifest policies.
- [Optional] Filters for events that specify which entries are of interest and should be returned in the result (for example, errors only, or both warnings and errors). All events are used to evaluate the entity aggregated health, regardless of the filter.

## API

To get the replica health through the API, create a `FabricClient` and call the `GetReplicaHealthAsync` method on its `HealthManager`. To specify advanced parameters, use `ReplicaHealthQueryDescription`.

```
ReplicaHealth replicaHealth = await fabricClient.HealthManager.GetReplicaHealthAsync(partitionId, replicaId);
```

## PowerShell

The cmdlet to get the replica health is [Get-ServiceFabricReplicaHealth](#). First, connect to the cluster by using the [Connect-ServiceFabricCluster](#) cmdlet.

The following cmdlet gets the health of the primary replica for all partitions of the service:

```
PS D:\ServiceFabric> Get-ServiceFabricPartition fabric:/WordCount/WordCountService | Get-ServiceFabricReplica
| where {$_.ReplicaRole -eq "Primary"} | Get-ServiceFabricReplicaHealth

PartitionId      : af2e3e44-a8f8-45ac-9f31-4093eb897600
ReplicaId       : 131444422260002646
AggregatedHealthState : Ok
HealthEvents     :
    SourceId        : System.RA
    Property         : State
    HealthState      : Ok
    SequenceNumber   : 131444422263668344
    SentAt          : 7/13/2017 5:57:06 PM
    ReceivedAt       : 7/13/2017 5:57:18 PM
    TTL              : Infinite
    Description      : Replica has been created._Node_2
    RemoveWhenExpired: False
    IsExpired        : False
    Transitions      : Error->Ok = 7/13/2017 5:57:18 PM, LastWarning = 1/1/0001
12:00:00 AM
```

## REST

You can get replica health with a [GET request](#) or a [POST request](#) that includes health policies described in the body.

## Get deployed application health

Returns the health of an application deployed on a node entity. It contains the deployed service package health states. Input:

- [Required] The application name (URI) and node name (string) that identify the deployed application.
- [Optional] The application health policy used to override the application manifest policies.
- [Optional] Filters for events and deployed service packages that specify which entries are of interest and should be returned in the result (for example, errors only, or both warnings and errors). All events and deployed service packages are used to evaluate the entity aggregated health, regardless of the filter.
- [Optional] Filter to exclude health statistics. If not specified, the health statistics show the number of deployed service packages in ok, warning, and error health states.

## API

To get the health of an application deployed on a node through the API, create a `FabricClient` and call the `GetDeployedApplicationHealthAsync` method on its `HealthManager`. To specify optional parameters, use `DeployedApplicationHealthQueryDescription`.

```
DeployedApplicationHealth health = await fabricClient.HealthManager.GetDeployedApplicationHealthAsync(
    new DeployedApplicationHealthQueryDescription(applicationName, nodeName));
```

## PowerShell

The cmdlet to get the deployed application health is [Get-ServiceFabricDeployedApplicationHealth](#). First, connect to the cluster by using the [Connect-ServiceFabricCluster](#) cmdlet. To find out where an application is deployed, run [Get-ServiceFabricApplicationHealth](#) and look at the deployed application children.

The following cmdlet gets the health of the **fabric:/WordCount** application deployed on **\_Node\_2**.

```
PS D:\ServiceFabric> Get-ServiceFabricDeployedApplicationHealth -ApplicationName fabric:/WordCount -NodeName _Node_0

ApplicationName      : fabric:/WordCount
NodeName            : _Node_0
AggregatedHealthState : Ok
DeployedServicePackageHealthStates :
    ServiceManifestName   : WordCountServicePkg
    ServicePackageActivationId :
        NodeName          : _Node_0
        AggregatedHealthState : Ok

    ServiceManifestName   : WordCountWebServicePkg
    ServicePackageActivationId :
        NodeName          : _Node_0
        AggregatedHealthState : Ok

HealthEvents       :
    SourceId          : System.Hosting
    Property           : Activation
    HealthState        : Ok
    SequenceNumber     : 131444422261848308
    SentAt             : 7/13/2017 5:57:06 PM
    ReceivedAt         : 7/13/2017 5:57:17 PM
    TTL                : Infinite
    Description         : The application was activated successfully.
    RemoveWhenExpired  : False
    IsExpired          : False
    Transitions         : Error->Ok = 7/13/2017 5:57:17 PM, LastWarning =
1/1/0001 12:00:00 AM

HealthStatistics   :
    DeployedServicePackage : 2 Ok, 0 Warning, 0 Error
```

## REST

You can get deployed application health with a [GET request](#) or a [POST request](#) that includes health policies described in the body.

## Get deployed service package health

Returns the health of a deployed service package entity. Input:

- [Required] The application name (URI), node name (string), and service manifest name (string) that identify the deployed service package.
- [Optional] The application health policy used to override the application manifest policy.
- [Optional] Filters for events that specify which entries are of interest and should be returned in the result (for example, errors only, or both warnings and errors). All events are used to evaluate the entity aggregated health, regardless of the filter.

## API

To get the health of a deployed service package through the API, create a `FabricClient` and call the `GetDeployedServicePackageHealthAsync` method on its `HealthManager`. To specify optional parameters, use `DeployedServicePackageHealthQueryDescription`.

```
DeployedServicePackageHealth health = await fabricClient.HealthManager.GetDeployedServicePackageHealthAsync(
    new DeployedServicePackageHealthQueryDescription(applicationName, nodeName, serviceManifestName));
```

## PowerShell

The cmdlet to get the deployed service package health is [Get-ServiceFabricDeployedServicePackageHealth](#). First, connect to the cluster by using the [Connect-ServiceFabricCluster](#) cmdlet. To see where an application is deployed, run [Get-ServiceFabricApplicationHealth](#) and look at the deployed applications. To see which service packages are in an application, look at the deployed service package children in the [Get-ServiceFabricDeployedApplicationHealth](#) output.

The following cmdlet gets the health of the **WordCountServicePkg** service package of the **fabric:/WordCount** application deployed on **\_Node\_2**. The entity has **System.Hosting** reports for successful service-package and entry-point activation, and successful service-type registration.

```
PS D:\ServiceFabric> Get-ServiceFabricDeployedApplication -ApplicationName fabric:/WordCount -NodeName _Node_2 | Get-ServiceFabricDeployedServicePackageHealth -ServiceManifestName WordCountServicePkg

ApplicationName      : fabric:/WordCount
ServiceManifestName  : WordCountServicePkg
ServicePackageActivationId :
NodeName            : _Node_2
AggregatedHealthState : Ok
HealthEvents        :

    SourceId          : System.Hosting
    Property           : Activation
    HealthState        : Ok
    SequenceNumber     : 131444422267693359
    SentAt             : 7/13/2017 5:57:06 PM
    ReceivedAt         : 7/13/2017 5:57:18 PM
    TTL                : Infinite
    Description         : The ServicePackage was activated successfully.
    RemoveWhenExpired  : False
    IsExpired          : False
    Transitions         : Error->Ok = 7/13/2017 5:57:18 PM, LastWarning = 1/1/0001

12:00:00 AM

    SourceId          : System.Hosting
    Property           : CodePackageActivation:Code:EntryPoint
    HealthState        : Ok
    SequenceNumber     : 131444422267903345
    SentAt             : 7/13/2017 5:57:06 PM
    ReceivedAt         : 7/13/2017 5:57:18 PM
    TTL                : Infinite
    Description         : The CodePackage was activated successfully.
    RemoveWhenExpired  : False
    IsExpired          : False
    Transitions         : Error->Ok = 7/13/2017 5:57:18 PM, LastWarning = 1/1/0001

12:00:00 AM

    SourceId          : System.Hosting
    Property           : ServiceTypeRegistration:WordCountServiceType
    HealthState        : Ok
    SequenceNumber     : 131444422272458374
    SentAt             : 7/13/2017 5:57:07 PM
    ReceivedAt         : 7/13/2017 5:57:18 PM
    TTL                : Infinite
    Description         : The ServiceType was registered successfully.
    RemoveWhenExpired  : False
    IsExpired          : False
    Transitions         : Error->Ok = 7/13/2017 5:57:18 PM, LastWarning = 1/1/0001

12:00:00 AM
```

## REST

You can get deployed service package health with a [GET request](#) or a [POST request](#) that includes health policies described in the body.

# Health chunk queries

The health chunk queries can return multi-level cluster children (recursively), per input filters. It supports advanced filters that allow a lot of flexibility in choosing the children to be returned. The filters can specify children by the unique identifier or by other group identifiers and/or health states. By default, no children are included, as opposed to health commands that always include first-level children.

The [health queries](#) return only first-level children of the specified entity per required filters. To get the children of the children, you must call additional health APIs for each entity of interest. Similarly, to get the health of specific entities, you must call one health API for each desired entity. The chunk query advanced filtering allows you to request multiple items of interest in one query, minimizing the message size and the number of messages.

The value of the chunk query is that you can get health state for more cluster entities (potentially all cluster entities starting at required root) in one call. You can express complex health query such as:

- Return only applications in error, and for those applications include all services in warning or error. For returned services, include all partitions.
- Return only the health of four applications, specified by their names.
- Return only the health of applications of a desired application type.
- Return all deployed entities on a node. Returns all applications, all deployed applications on the specified node and all the deployed service packages on that node.
- Return all replicas in error. Returns all applications, services, partitions, and only replicas in error.
- Return all applications. For a specified service, include all partitions.

Currently, the health chunk query is exposed only for the cluster entity. It returns a cluster health chunk, which contains:

- The cluster aggregated health state.
- The health state chunk list of nodes that respect input filters.
- The health state chunk list of applications that respect input filters. Each application health state chunk contains a chunk list with all services that respect input filters and a chunk list with all deployed applications that respect the filters. Same for the children of services and deployed applications. This way, all entities in the cluster can be potentially returned if requested, in a hierarchical fashion.

## Cluster health chunk query

Returns the health of the cluster entity and contains the hierarchical health state chunks of required children.

Input:

- [Optional] The cluster health policy used to evaluate the nodes and the cluster events.
- [Optional] The application health policy map, with the health policies used to override the application manifest policies.
- [Optional] Filters for nodes and applications that specify which entries are of interest and should be returned in the result. The filters are specific to an entity/group of entities or are applicable to all entities at that level. The list of filters can contain one general filter and/or filters for specific identifiers to fine-grain entities returned by the query. If empty, the children are not returned by default. Read more about the filters at [NodeHealthStateFilter](#) and [ApplicationHealthStateFilter](#). The application filters can recursively specify advanced filters for children.

The chunk result includes the children that respect the filters.

Currently, the chunk query does not return unhealthy evaluations or entity events. That extra information can be obtained using the existing cluster health query.

## API

To get cluster health chunk, create a `FabricClient` and call the `GetClusterHealthChunkAsync` method on its

**HealthManager**. You can pass in [ClusterHealthQueryDescription](#) to describe health policies and advanced filters.

The following code gets cluster health chunk with advanced filters.

```
var queryDescription = new ClusterHealthChunkQueryDescription();
queryDescription.ApplicationFilters.Add(new ApplicationHealthStateFilter()
{
    // Return applications only if they are in error
    HealthStateFilter = HealthStateFilter.Error
});

// Return all replicas
var wordCountServiceReplicaFilter = new ReplicaHealthStateFilter()
{
    HealthStateFilter = HealthStateFilter.All
};

// Return all replicas and all partitions
var wordCountServicePartitionFilter = new PartitionHealthStateFilter()
{
    HealthStateFilter = HealthStateFilter.All
};
wordCountServicePartitionFilter.ReplicaFilters.Add(wordCountServiceReplicaFilter);

// For specific service, return all partitions and all replicas
var wordCountServiceFilter = new ServiceHealthStateFilter()
{
    ServiceNameFilter = new Uri("fabric:/WordCount/WordCountService"),
};
wordCountServiceFilter.PartitionFilters.Add(wordCountServicePartitionFilter);

// Application filter: for specific application, return no services except the ones of interest
var wordCountApplicationFilter = new ApplicationHealthStateFilter()
{
    // Always return fabric:/WordCount application
    ApplicationNameFilter = new Uri("fabric:/WordCount"),
};
wordCountApplicationFilter.ServiceFilters.Add(wordCountServiceFilter);

queryDescription.ApplicationFilters.Add(wordCountApplicationFilter);

var result = await fabricClient.HealthManager.GetClusterHealthChunkAsync(queryDescription);
```

## PowerShell

The cmdlet to get the cluster health is [Get-ServiceFabricClusterChunkHealth](#). First, connect to the cluster by using the [Connect-ServiceFabricCluster](#) cmdlet.

The following code gets nodes only if they are in Error except for a specific node, which should always be returned.

```
PS D:\ServiceFabric> $errorFilter = [System.Fabric.Health.HealthStateFilter]::Error;
$errorFilter = [System.Fabric.Health.HealthStateFilter]::All;

$nodeFilter1 = New-Object System.Fabric.Health.NodeHealthStateFilter -Property
@{HealthStateFilter=$errorFilter}
$nodeFilter2 = New-Object System.Fabric.Health.NodeHealthStateFilter -Property
@{NodeNameFilter="_Node_1";HealthStateFilter=$allFilter}
# Create node filter list that will be passed in the cmdlet
$nodeFilters = New-Object System.Collections.Generic.List[System.Fabric.Health.NodeHealthStateFilter]
$nodeFilters.Add($nodeFilter1)
$nodeFilters.Add($nodeFilter2)

Get-ServiceFabricClusterHealthChunk -NodeFilters $nodeFilters

HealthState          : Warning
NodeHealthStateChunks:
                           TotalCount      : 1
                           NodeName       : _Node_1
                           HealthState    : Ok

ApplicationHealthStateChunks : None
```

The following cmdlet gets cluster chunk with application filters.

```

PS D:\ServiceFabric> $errorFilter = [System.Fabric.Health.HealthStateFilter]::Error;
$allFilter = [System.Fabric.Health.HealthStateFilter]::All;

# All replicas
$replicaFilter = New-Object System.Fabric.Health.ReplicaHealthStateFilter -Property
@{HealthStateFilter=$allFilter}

# All partitions
$partitionFilter = New-Object System.Fabric.Health.PartitionHealthStateFilter -Property
@{HealthStateFilter=$allFilter}
$partitionFilter.ReplicaFilters.Add($replicaFilter)

# For WordCountService, return all partitions and all replicas
$svcFilter1 = New-Object System.Fabric.Health.ServiceHealthStateFilter -Property
@{ServiceNameFilter="fabric:/WordCount/WordCountService"}
$svcFilter1.PartitionFilters.Add($partitionFilter)

$svcFilter2 = New-Object System.Fabric.Health.ServiceHealthStateFilter -Property
@{HealthStateFilter=$errorFilter}

$appFilter = New-Object System.Fabric.Health.ApplicationHealthStateFilter -Property
@{ApplicationNameFilter="fabric:/WordCount"}
$appFilter.ServiceFilters.Add($svcFilter1)
$appFilter.ServiceFilters.Add($svcFilter2)

$appFilters = New-Object System.Collections.Generic.List[System.Fabric.Health.ApplicationHealthStateFilter]
$appFilters.Add($appFilter)

Get-ServiceFabricClusterHealthChunk -ApplicationFilters $appFilters

HealthState : Error
NodeHealthStateChunks : None
ApplicationHealthStateChunks :
    TotalCount : 1

        ApplicationName : fabric:/WordCount
        ApplicationTypeName : WordCount
        HealthState : Error
        ServiceHealthStateChunks :
            TotalCount : 1

                ServiceName : fabric:/WordCount/WordCountService
                HealthState : Error
                PartitionHealthStateChunks :
                    TotalCount : 1

                        PartitionId : af2e3e44-a8f8-45ac-9f31-4093eb897600
                        HealthState : Error
                        ReplicaHealthStateChunks :
                            TotalCount : 5

                                ReplicaOrInstanceId : 131444422293118720
                                HealthState : Ok

                                ReplicaOrInstanceId : 131444422293118721
                                HealthState : Ok

                                ReplicaOrInstanceId : 131444422293113678
                                HealthState : Ok

                                ReplicaOrInstanceId : 131444422293113679
                                HealthState : Ok

                                ReplicaOrInstanceId : 131444422260002646
                                HealthState : Error

```

The following cmdlet returns all deployed entities on a node.

```
PS D:\ServiceFabric> $errorFilter = [System.Fabric.Health.HealthStateFilter]::Error;
$errorFilter = [System.Fabric.Health.HealthStateFilter]::All;

$dspFilter = New-Object System.Fabric.Health.DeployedServicePackageHealthStateFilter -Property
@{HealthStateFilter=$allFilter}
$daFilter = New-Object System.Fabric.Health.DeployedApplicationHealthStateFilter -Property
@{HealthStateFilter=$allFilter;NodeNameFilter="_Node_2"}
$daFilter.DeployedServicePackageFilters.Add($dspFilter)

$appFilter = New-Object System.Fabric.Health.ApplicationHealthStateFilter -Property
@{HealthStateFilter=$allFilter}
$appFilter.DeployedApplicationFilters.Add($daFilter)

$appFilters = New-Object System.Collections.Generic.List[System.Fabric.Health.ApplicationHealthStateFilter]
$appFilters.Add($appFilter)
Get-ServiceFabricClusterHealthChunk -ApplicationFilters $appFilters

HealthState          : Error
NodeHealthStateChunks : None
ApplicationHealthStateChunks :
    TotalCount      : 2

        ApplicationName      : fabric:/System
        HealthState          : Ok
        DeployedApplicationHealthStateChunks :
            TotalCount      : 1

                NodeName      : _Node_2
                HealthState    : Ok
                DeployedServicePackageHealthStateChunks :
                    TotalCount      : 1

                    ServiceManifestName   : FAS
                    ServicePackageActivationId :
                    HealthState          : Ok

    ApplicationName      : fabric:/WordCount
    ApplicationTypeName   : WordCount
    HealthState          : Error
    DeployedApplicationHealthStateChunks :
        TotalCount      : 1

        NodeName      : _Node_2
        HealthState    : Ok
        DeployedServicePackageHealthStateChunks :
            TotalCount      : 1

            ServiceManifestName   : WordCountServicePkg
            ServicePackageActivationId :
            HealthState          : Ok
```

## REST

You can get cluster health chunk with a [GET request](#) or a [POST request](#) that includes health policies and advanced filters described in the body.

## General queries

General queries return a list of Service Fabric entities of a specified type. They are exposed through the API (via the methods on **FabricClient.QueryManager**), PowerShell cmdlets, and REST. These queries aggregate subqueries

from multiple components. One of them is the [health store](#), which populates the aggregated health state for each query result.

#### NOTE

General queries return the aggregated health state of the entity and do not contain rich health data. If an entity is not healthy, you can follow up with health queries to get all its health information, including events, child health states, and unhealthy evaluations.

If general queries return an unknown health state for an entity, it's possible that the health store doesn't have complete data about the entity. It's also possible that a subquery to the health store wasn't successful (for example, there was a communication error, or the health store was throttled). Follow up with a health query for the entity. If the subquery encountered transient errors, such as network issues, this follow-up query may succeed. It may also give you more details from the health store about why the entity is not exposed.

The queries that contain **HealthState** for entities are:

- Node list: Returns the list nodes in the cluster (paged).
  - API: [FabricClient.QueryClient.GetNodeListAsync](#)
  - PowerShell: Get-ServiceFabricNode
- Application list: Returns the list of applications in the cluster (paged).
  - API: [FabricClient.QueryClient.GetApplicationListAsync](#)
  - PowerShell: Get-ServiceFabricApplication
- Service list: Returns the list of services in an application (paged).
  - API: [FabricClient.QueryClient.GetServiceListAsync](#)
  - PowerShell: Get-ServiceFabricService
- Partition list: Returns the list of partitions in a service (paged).
  - API: [FabricClient.QueryClient.GetPartitionListAsync](#)
  - PowerShell: Get-ServiceFabricPartition
- Replica list: Returns the list of replicas in a partition (paged).
  - API: [FabricClient.QueryClient.GetReplicaListAsync](#)
  - PowerShell: Get-ServiceFabricReplica
- Deployed application list: Returns the list of deployed applications on a node.
  - API: [FabricClient.QueryClient.GetDeployedApplicationListAsync](#)
  - PowerShell: Get-ServiceFabricDeployedApplication
- Deployed service package list: Returns the list of service packages in a deployed application.
  - API: [FabricClient.QueryClient.GetDeployedServicePackageListAsync](#)
  - PowerShell: Get-ServiceFabricDeployedApplication

#### NOTE

Some of the queries return paged results. The return of these queries is a list derived from [PagedList](#). If the results do not fit a message, only a page is returned and a ContinuationToken that tracks where enumeration stopped. Continue to call the same query and pass in the continuation token from the previous query to get next results.

## Examples

The following code gets the unhealthy applications in the cluster:

```
var applications = fabricClient.QueryManager.GetApplicationListAsync().Result.Where(
    app => app.HealthState == HealthState.Error);
```

The following cmdlet gets the application details for the fabric:/WordCount application. Notice that health state is at warning.

```
PS C:\> Get-ServiceFabricApplication -ApplicationName fabric:/WordCount

ApplicationName      : fabric:/WordCount
ApplicationTypeName  : WordCount
ApplicationTypeVersion: 1.0.0
ApplicationStatus    : Ready
HealthState          : Warning
ApplicationParameters: { "WordCountWebService_InstanceCount" = "1";
                        "_WFDebugParams_" =
[{"ServiceManifestName": "WordCountWebServicePkg", "CodePackageName": "Code", "EntryPointType": "Main", "DebugExePath": "C:\\Program Files (x86)\\Microsoft Visual Studio 14.0\\Common7\\Packages\\Debugger\\VsDebugLaunchNotify.exe", "DebugArguments": "[74f7e5d5-71a9-47e2-a8cd-1878ec4734f1] -p [ProcessId] -tid [ThreadId]", "EnvironmentBlock": "_NO_DEBUG_HEAP=1\u0000"}, {"ServiceManifestName": "WordCountServicePkg", "CodePackageName": "Code", "EntryPointType": "Main", "DebugExePath": "C:\\Program Files (x86)\\Microsoft Visual Studio 14.0\\Common7\\Packages\\Debugger\\VsDebugLaunchNotify.exe", "DebugArguments": "[2ab462e6-e0d1-4fd1-a844-972f561fe751] -p [ProcessId] -tid [ThreadId]", "EnvironmentBlock": "_NO_DEBUG_HEAP=1\u0000"}]
```

The following cmdlet gets the services with a health state of error:

```
PS D:\\ServiceFabric> Get-ServiceFabricApplication | Get-ServiceFabricService | where {$_.HealthState -eq "Error"}

ServiceName      : fabric:/WordCount/WordCountService
ServiceKind      : Stateful
ServiceTypeName  : WordCountServiceType
IsServiceGroup   : False
ServiceManifestVersion: 1.0.0
HasPersistedState: True
ServiceStatus    : Active
HealthState      : Error
```

## Cluster and application upgrades

During a monitored upgrade of the cluster and application, Service Fabric checks health to ensure that everything remains healthy. If an entity is unhealthy as evaluated by using configured health policies, the upgrade applies upgrade-specific policies to determine the next action. The upgrade may be paused to allow user interaction (such as fixing error conditions or changing policies), or it may automatically roll back to the previous good version.

During a *cluster* upgrade, you can get the cluster upgrade status. The upgrade status includes unhealthy evaluations, which point to what is unhealthy in the cluster. If the upgrade is rolled back due to health issues, the upgrade status remembers the last unhealthy reasons. This information can help administrators investigate what went wrong after the upgrade rolled back or stopped.

Similarly, during an *application* upgrade, any unhealthy evaluations are contained in the application upgrade status.

The following shows the application upgrade status for a modified fabric:/WordCount application. A watchdog reported an error on one of its replicas. The upgrade is rolling back because the health checks are not respected.

```

PS C:\> Get-ServiceFabricApplicationUpgrade fabric:/WordCount

ApplicationName          : fabric:/WordCount
ApplicationTypeName       : WordCount
TargetApplicationTypeVersion : 1.0.0.0
ApplicationParameters     : {}
StartTimeStampUtc        : 4/21/2017 5:23:26 PM
FailureTimeStampUtc      : 4/21/2017 5:23:37 PM
FailureReason             : HealthCheck
UpgradeState              : RollingBackInProgress
UpgradeDuration           : 00:00:23
CurrentUpgradeDomainDuration : 00:00:00
CurrentUpgradeDomainProgress : UD1

                           NodeName      : _Node_1
                           UpgradePhase : Upgrading

                           NodeName      : _Node_2
                           UpgradePhase : Upgrading

                           NodeName      : _Node_3
                           UpgradePhase : PreUpgradeSafetyCheck
                           PendingSafetyChecks :
                           EnsurePartitionQuorum - PartitionId: 30db5be6-4e20-4698-8185-4bd7ca744020
NextUpgradeDomain         : UD2
UpgradeDomainsStatus      : { "UD1" = "Completed";
                           "UD2" = "Pending";
                           "UD3" = "Pending";
                           "UD4" = "Pending" }
UnhealthyEvaluations     :
                           Unhealthy services: 100% (1/1), ServiceType='WordCountServiceType',
MaxPercentUnhealthyServices=0%.
                           Unhealthy service: ServiceName='fabric:/WordCount/WordCountService',
AggregatedHealthState='Error'.
                           Unhealthy partitions: 100% (1/1),
MaxPercentUnhealthyPartitionsPerService=0%.
                           Unhealthy partition: PartitionId='a1f83a35-d6bf-4d39-b90d-
28d15f39599b', AggregatedHealthState='Error'.
                           Unhealthy replicas: 20% (1/5),
MaxPercentUnhealthyReplicasPerPartition=0%.
                           Unhealthy replica: PartitionId='a1f83a35-d6bf-4d39-b90d-
28d15f39599b',
                           ReplicaOrInstanceId='131031502346844058', AggregatedHealthState='Error'.
                           Error event: SourceId='DiskWatcher', Property='Disk'.

UpgradeKind               : Rolling
RollingUpgradeMode         : UnmonitoredAuto
ForceRestart               : False
UpgradeReplicaSetCheckTimeout : 00:15:00

```

Read more about the [Service Fabric application upgrade](#).

## Use health evaluations to troubleshoot

Whenever there is an issue with the cluster or an application, look at the cluster or application health to pinpoint what is wrong. The unhealthy evaluations provide details about what triggered the current unhealthy state. If you need to, you can drill down into unhealthy child entities to identify the root cause.

For example, consider an application unhealthy because there is an error report on one of its replicas. The

following Powershell cmdlet shows the unhealthy evaluations:

```
PS D:\ServiceFabric> Get-ServiceFabricApplicationHealth fabric:/WordCount -EventsFilter None -ServicesFilter None -DeployedApplicationsFilter None -ExcludeHealthStatistics

ApplicationName          : fabric:/WordCount
AggregatedHealthState   : Error
UnhealthyEvaluations    :
    Unhealthy services: 100% (1/1), ServiceType='WordCountServiceType',
    MaxPercentUnhealthyServices=0%.
    Unhealthy service: ServiceName='fabric:/WordCount/WordCountService',
    AggregatedHealthState='Error'.
    Unhealthy partitions: 100% (1/1),
    MaxPercentUnhealthyPartitionsPerService=0%.
    Unhealthy partition: PartitionId='af2e3e44-a8f8-45ac-9f31-4093eb897600',
    AggregatedHealthState='Error'.
    Unhealthy replicas: 20% (1/5),
    MaxPercentUnhealthyReplicasPerPartition=0%.
    Unhealthy replica: PartitionId='af2e3e44-a8f8-45ac-9f31-
4093eb897600', ReplicaOrInstanceId='131444422260002646', AggregatedHealthState='Error'.
    Error event: SourceId='MyWatchdog', Property='Memory'.

ServiceHealthStates       : None
DeployedApplicationHealthStates : None
HealthEvents              : None
```

You can look at the replica to get more information:

```
PS D:\ServiceFabric> Get-ServiceFabricReplicaHealth -ReplicaOrInstanceId 131444422260002646 -PartitionId af2e3e44-a8f8-45ac-9f31-4093eb897600
```

```
PartitionId      : af2e3e44-a8f8-45ac-9f31-4093eb897600
ReplicaId       : 131444422260002646
AggregatedHealthState : Error
UnhealthyEvaluations  :
    Error event: SourceId='MyWatchdog', Property='Memory'.

HealthEvents      :
    SourceId          : System.RA
    Property           : State
    HealthState        : Ok
    SequenceNumber     : 131444422263668344
    SentAt             : 7/13/2017 5:57:06 PM
    ReceivedAt         : 7/13/2017 5:57:18 PM
    TTL                : Infinite
    Description         : Replica has been created._Node_2
    RemoveWhenExpired  : False
    IsExpired          : False
    Transitions         : Error->Ok = 7/13/2017 5:57:18 PM, LastWarning = 1/1/0001
12:00:00 AM

    SourceId          : MyWatchdog
    Property           : Memory
    HealthState        : Error
    SequenceNumber     : 131444451657749403
    SentAt             : 7/13/2017 6:46:05 PM
    ReceivedAt         : 7/13/2017 6:46:05 PM
    TTL                : Infinite
    Description         :
    RemoveWhenExpired  : False
    IsExpired          : False
    Transitions         : Warning->Error = 7/13/2017 6:46:05 PM, LastOk = 1/1/0001
12:00:00 AM
```

#### NOTE

The unhealthy evaluations show the first reason the entity is evaluated to current health state. There may be multiple other events that trigger this state, but they are not reflected in the evaluations. To get more information, drill down into the health entities to figure out all the unhealthy reports in the cluster.

## Next steps

[Use system health reports to troubleshoot](#)

[Add custom Service Fabric health reports](#)

[How to report and check service health](#)

[Monitor and diagnose services locally](#)

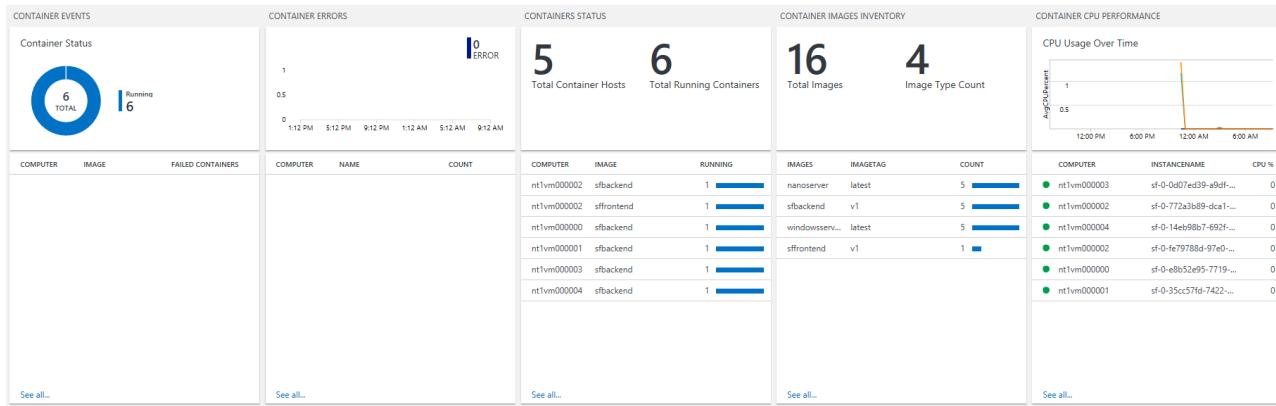
[Service Fabric application upgrade](#)

# Monitoring Windows Server containers with OMS

10/16/2017 • 3 min to read • [Edit Online](#)

## OMS Containers Solution

Operations Management Suite (OMS) Log Analytics has a Containers solution that can be used for monitoring containers. Alongside the Service Fabric solution, this solution is a great tool to monitor container deployments orchestrated on Service Fabric. Here's a simple example of what the dashboard in the solution looks like:



It also collects different kinds of logs that can be queried in the OMS Log Analytics tool, and can be used to visualize any metrics or events being generated. The log types that are collected are:

1. ContainerInventory: shows information about container location, name, and images
2. ContainerImageInventory: information about deployed images, including IDs or sizes
3. ContainerLog: specific error logs, docker logs (stdout, etc.), and other entries
4. ContainerServiceLog: docker daemon commands that have been run
5. Perf: performance counters including container cpu, memory, network traffic, disk i/o, and custom metrics from the host machines

This article covers the steps required to set up container monitoring for your cluster. To learn more about OMS's Containers solution, check out their [documentation](#).

## 1. Set up a Service Fabric cluster

Create a cluster using the Azure Resource Manager template found [here](#). Make sure to add a unique OMS workspace name. This template also defaults to deploying a cluster in the preview build of Service Fabric (v255.255), which means that it cannot be used in production, and cannot be upgraded to a different Service Fabric version. If you decide to use this template for long-term or production use, change the version to a stable version number.

Once the cluster is set up, confirm that you have installed the appropriate certificate, and make sure you are able to connect to the cluster.

Confirm that your Resource Group is set up correctly by heading to the [Azure portal](#) and finding the deployment. The resource group should contain all the Service Fabric resources, and also have a Log Analytics solution as well as a Service Fabric solution.

For modifying an existing Service Fabric cluster:

- Confirm that Diagnostics is enabled (if not, enable it via [updating the virtual machine scale set](#))

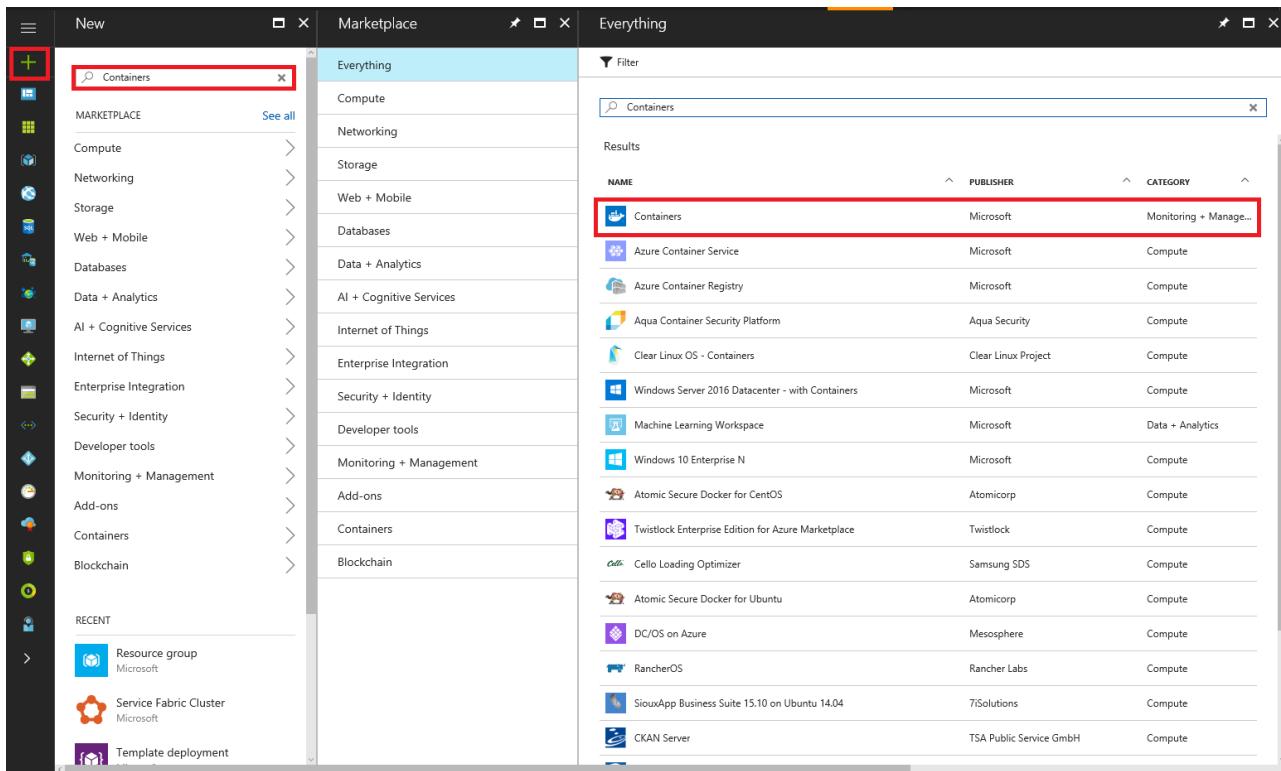
- Add an OMS Workspace by creating a "Service Fabric Analytics" solution via the Azure Marketplace
- Edit the data sources of the Service Fabric solution to pick up data from the appropriate Azure Storage tables (set up by WAD) in the Resource Group that the cluster is in
- Add the agent as an [extension to the virtual machine scale set](#) via PowerShell or through updating the virtual machine scale set (same link as above, to modify the Resource Manager template)

## 2. Deploy a container

Once the cluster is ready and you have confirmed that you can access it, deploy a container to it. If you chose to use the preview version as set by the template, you can also explore Service Fabric's new docker compose functionality. Bear in mind that the first time a container image is deployed to a cluster, it takes several minutes to download the image depending on its size.

## 3. Add the Containers solution

In the Azure portal, create a Containers resource (under the Monitoring + Management category) through Azure Marketplace.



The screenshot shows the Azure portal interface. On the left, there is a sidebar with a 'New' button highlighted with a red box. Below it is a 'Marketplace' section with a search bar containing 'Containers'. The search results page is titled 'Everything' and shows a list of containers. The 'Containers' solution by Microsoft is highlighted with a red box in the results list. Other visible solutions include 'Azure Container Service', 'Azure Container Registry', 'Aqua Container Security Platform', 'Clear Linux OS - Containers', 'Windows Server 2016 Datacenter - with Containers', 'Machine Learning Workspace', 'Windows 10 Enterprise N', 'Atomic Secure Docker for CentOS', 'Twistlock Enterprise Edition for Azure Marketplace', 'Cello Loading Optimizer', 'Atomic Secure Docker for Ubuntu', 'DC/OS on Azure', 'RancherOS', 'SiouxApp Business Suite 15.10 on Ubuntu 14.04', and 'CKAN Server'.

In the creation step, it requests an OMS workspace. Select the one that was created with the deployment above. This step adds a Containers solution within your OMS workspace, and is automatically detected by the OMS agent deployed by the template. The agent will start gathering data on the containers processes in the cluster, and in about 10-15 minutes, you should see the solution light up with data as in the image of the dashboard above.

## 4. Next steps

OMS offers various tools in the workspace to make it more useful for you. Explore the following options to customize the solution to your needs:

- Get familiarized with the [log search and querying](#) features offered as part of Log Analytics
- Configure the OMS agent to pick up specific performance counters (go to the workspace Home > Settings > Data > Windows Performance Counters)
- Configure OMS to set up [automated alerting](#) rules to aid in detecting and diagnostics

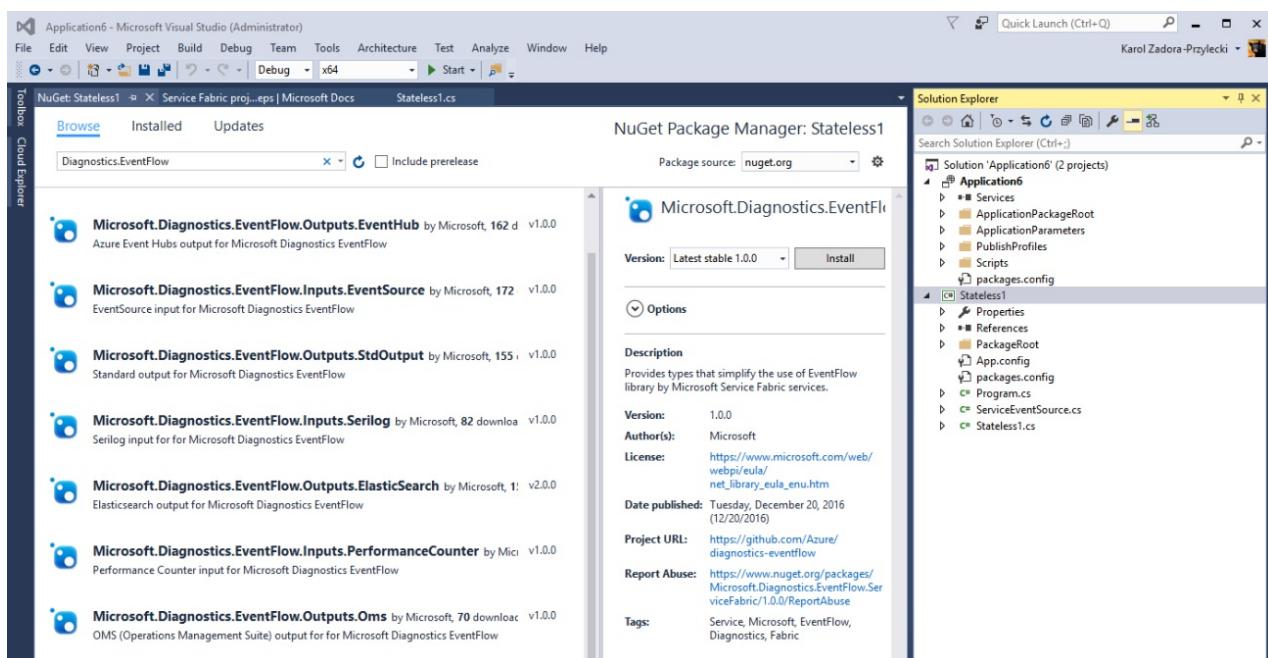
# Event aggregation and collection using EventFlow

10/16/2017 • 4 min to read • [Edit Online](#)

[Microsoft Diagnostics EventFlow](#) can route events from a node to one or more monitoring destinations. Because it is included as a NuGet package in your service project, EventFlow code and configuration travel with the service, eliminating the per-node configuration issue mentioned earlier about Azure Diagnostics. EventFlow runs within your service process, and connects directly to the configured outputs. Because of the direct connection, EventFlow works for Azure, container, and on-premises service deployments. Be careful if you run EventFlow in high-density scenarios, such as in a container, because each EventFlow pipeline makes an external connection. So, if you host several processes, you get several outbound connections! This isn't as much a concern for Service Fabric applications, because all replicas of a `ServiceType` run in the same process, and this limits the number of outbound connections. EventFlow also offers event filtering, so that only the events that match the specified filter are sent.

## Set up EventFlow

EventFlow binaries are available as a set of NuGet packages. To add EventFlow to a Service Fabric service project, right-click the project in the Solution Explorer and choose "Manage NuGet packages." Switch to the "Browse" tab and search for "`Diagnostics.EventFlow`":



You will see a list of various packages show up, labeled with "Inputs" and "Outputs". EventFlow supports various different logging providers and analyzers. The service hosting EventFlow should include appropriate packages depending on the source and destination for the application logs. In addition to the core ServiceFabric package, you also need at least one Input and Output configured. For example, you can add the following packages to sent EventSource events to Application Insights:

- `Microsoft.Diagnostics.EventFlow.Input.EventSource` to capture data from the service's EventSource class, and from standard EventSources such as *Microsoft-ServiceFabric-Services* and *Microsoft-ServiceFabric-Actors*)
- `Microsoft.Diagnostics.EventFlow.Output.ApplicationInsights` (we are going to send the logs to an Azure Application Insights resource)
- `Microsoft.Diagnostics.EventFlow.ServiceFabric` (enables initialization of the EventFlow pipeline from Service Fabric service configuration and reports any problems with sending diagnostic data as Service Fabric health

reports)

#### NOTE

The `Microsoft.Diagnostics.EventFlow.Input.EventSource` package requires the service project to target .NET Framework 4.6 or newer. Make sure you set the appropriate target framework in project properties before installing this package.

After all the packages are installed, the next step is to configure and enable EventFlow in the service.

## Configure and enable log collection

The EventFlow pipeline responsible for sending the logs is created from a specification stored in a configuration file. The `Microsoft.Diagnostics.EventFlow.ServiceFabric` package installs a starting EventFlow configuration file under `PackageRoot\Config` solution folder, named `eventFlowConfig.json`. This configuration file needs to be modified to capture data from the default service `EventSource` class, and any other inputs you want to configure, and send data to the appropriate place.

Here is a sample `eventFlowConfig.json` based on the NuGet packages mentioned above:

```
{  
  "inputs": [  
    {  
      "type": "EventSource",  
      "sources": [  
        { "providerName": "Microsoft-ServiceFabric-Services" },  
        { "providerName": "Microsoft-ServiceFabric-Actors" },  
        // (replace the following value with your service's ServiceEventSource name)  
        { "providerName": "your-service-EventSource-name" }  
      ]  
    }  
  ],  
  "filters": [  
    {  
      "type": "drop",  
      "include": "Level == Verbose"  
    }  
  ],  
  "outputs": [  
    {  
      "type": "ApplicationInsights",  
      // (replace the following value with your AI resource's instrumentation key)  
      "instrumentationKey": "00000000-0000-0000-0000-000000000000"  
    }  
  ],  
  "schemaVersion": "2016-08-11"  
}
```

The name of service's ServiceEventSource is the value of the `Name` property of the `EventSourceAttribute` applied to the `ServiceEventSource` class. It is all specified in the `ServiceEventSource.cs` file, which is part of the service code. For example, in the following code snippet the name of the `ServiceEventSource` is `MyCompany-Application1-Stateless1`:

```
[EventSource(Name = "MyCompany-Application1-Stateless1")]  
internal sealed class ServiceEventSource : EventSource  
{  
    // (rest of ServiceEventSource implementation)  
}
```

Note that `eventFlowConfig.json` file is part of service configuration package. Changes to this file can be included in

full- or configuration-only upgrades of the service, subject to Service Fabric upgrade health checks and automatic rollback if there is upgrade failure. For more information, see [Service Fabric application upgrade](#).

The *filters* section of the config allows you to further customize the information that is going to go through the EventFlow pipeline to the outputs, allowing you to drop or include certain information, or change the structure of the event data. For more information on filtering, see [EventFlow filters](#).

The final step is to instantiate EventFlow pipeline in your service's startup code, located in `Program.cs` file:

```
using System;
using System.Diagnostics;
using System.Threading;
using Microsoft.ServiceFabric;
using Microsoft.ServiceFabric.Services.Runtime;

// **** EventFlow namespace
using Microsoft.Diagnostics.EventFlow.ServiceFabric;

namespace Stateless1
{
    internal static class Program
    {
        /// <summary>
        /// This is the entry point of the service host process.
        /// </summary>
        private static void Main()
        {
            try
            {
                // **** Instantiate log collection via EventFlow
                using (var diagnosticsPipeline =
ServiceFabricDiagnosticPipelineFactory.CreatePipeline("MyApplication-MyService-DiagnosticsPipeline"))
                {

                    ServiceRuntime.RegisterServiceAsync("Stateless1Type",
context => new Stateless1(context)).GetAwaiter().GetResult();

                    ServiceEventSource.Current.ServiceTypeRegistered(Process.GetCurrentProcess().Id,
typeof(Stateless1).Name);

                    Thread.Sleep(Timeout.Infinite);
                }
            }
            catch (Exception e)
            {
                ServiceEventSource.Current.ServiceHostInitializationFailed(e.ToString());
                throw;
            }
        }
    }
}
```

The name passed as the parameter of the `CreatePipeline` method of the `ServiceFabricDiagnosticPipelineFactory` is the name of the *health entity* representing the EventFlow log collection pipeline. This name is used if EventFlow encounters an error and reports it through the Service Fabric health subsystem.

## Use Service Fabric settings and application parameters in eventFlowConfig

EventFlow supports using Service Fabric settings and application parameters to configure EventFlow settings. You can refer to Service Fabric settings parameters using this special syntax for values:

```
servicefabric:/<section-name>/<setting-name>
```

`<section-name>` is the name of the Service Fabric configuration section, and `<setting-name>` is the configuration setting providing the value that will be used to configure an EventFlow setting. To read more about how to do this, go to [Support for Service Fabric settings and application parameters](#).

## Verification

Start your service and observe the Debug output window in Visual Studio. After the service is started, you should start seeing evidence that your service is sending records to the output that you have configured. Navigate to your event analysis and visualization platform and confirm that logs have started to show up (could take a few minutes).

## Next steps

- [Event Analysis and Visualization with Application Insights](#)
- [Event Analysis and Visualization with OMS](#)
- [EventFlow documentation](#)

# Event aggregation and collection using Windows Azure Diagnostics

8/11/2017 • 9 min to read • [Edit Online](#)

When you're running an Azure Service Fabric cluster, it's a good idea to collect the logs from all the nodes in a central location. Having the logs in a central location helps you analyze and troubleshoot issues in your cluster, or issues in the applications and services running in that cluster.

One way to upload and collect logs is to use the Windows Azure Diagnostics (WAD) extension, which uploads logs to Azure Storage, and also has the option to send logs to Azure Application Insights or Event Hubs. You can also use an external process to read the events from storage and place them in an analysis platform product, such as [OMS Log Analytics](#) or another log-parsing solution.

## Prerequisites

These tools are used to perform some of the operations in this document:

- [Azure Diagnostics](#) (related to Azure Cloud Services but has good information and examples)
- [Azure Resource Manager](#)
- [Azure PowerShell](#)
- [Azure Resource Manager client](#)
- [Azure Resource Manager template](#)

## Log and event sources

### Service Fabric platform events

As discussed in [this article](#), Service Fabric sets you up with a few out-of-the-box logging channels, of which the following channels are easily configured with WAD to send monitoring and diagnostics data to a storage table or elsewhere:

- Operational events: higher-level operations that the Service Fabric platform performs. Examples include creation of applications and services, node state changes, and upgrade information. These are emitted as Event Tracing for Windows (ETW) logs
- [Reliable Actors programming model events](#)
- [Reliable Services programming model events](#)

### Application events

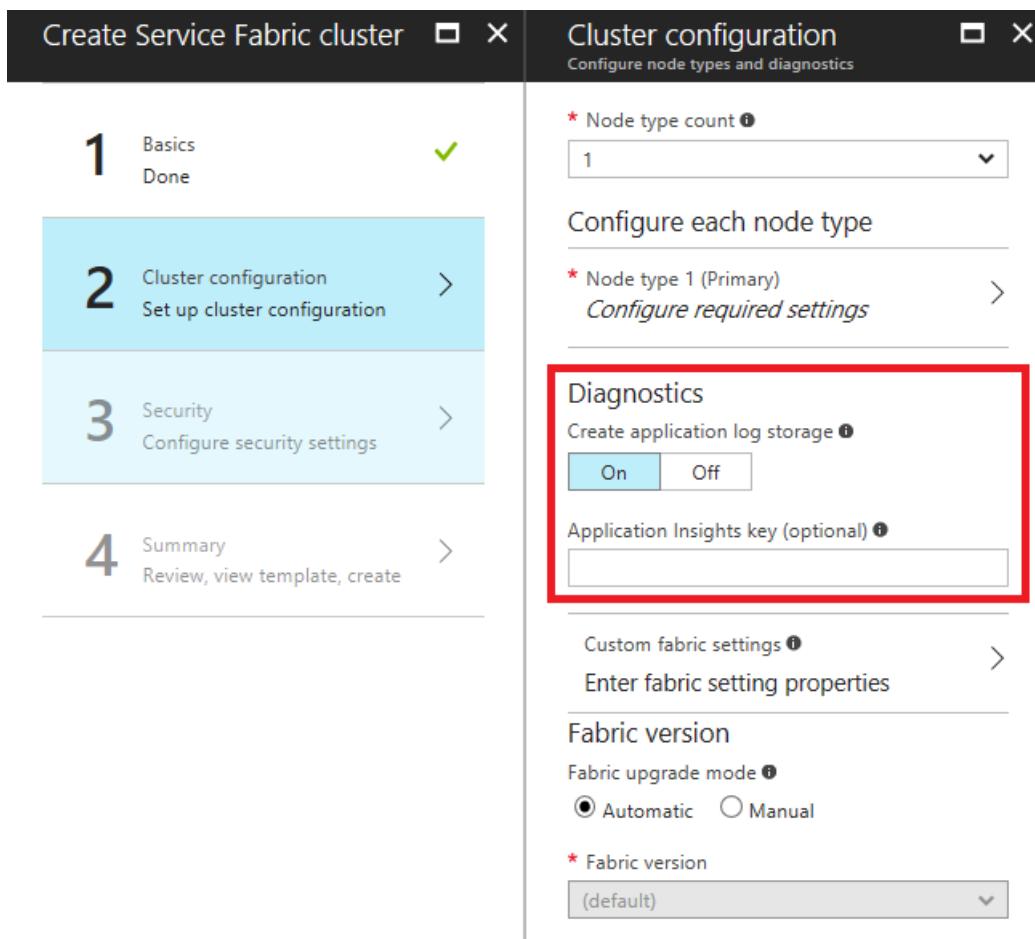
Events emitted from your applications' and services' code and written out by using the EventSource helper class provided in the Visual Studio templates. For more information on how to write EventSource logs from your application, see [Monitor and diagnose services in a local machine development setup](#).

## Deploy the Diagnostics extension

The first step in collecting logs is to deploy the Diagnostics extension on each of the VMs in the Service Fabric cluster. The Diagnostics extension collects logs on each VM and uploads them to the storage account that you specify. The steps vary a little based on whether you use the Azure portal or Azure Resource Manager. The steps also vary based on whether the deployment is part of cluster creation or is for a cluster that already exists. Let's look at the steps for each scenario.

## Deploy the Diagnostics extension as part of cluster creation through Azure portal

To deploy the Diagnostics extension to the VMs in the cluster as part of cluster creation, you use the Diagnostics settings panel shown in the following image - ensure that Diagnostics is set to **On** (the default setting). After you create the cluster, you can't change these settings by using the portal.



When you're creating a cluster by using the portal, we highly recommend that you download the template **before you click OK** to create the cluster. For details, refer to [Set up a Service Fabric cluster by using an Azure Resource Manager template](#). You'll need the template to make changes later, because you can't make some changes by using the portal.

## Deploy the Diagnostics extension as part of cluster creation by using Azure Resource Manager

To create a cluster by using Resource Manager, you need to add the Diagnostics configuration JSON to the full cluster Resource Manager template before you create the cluster. We provide a sample five-VM cluster Resource Manager template with Diagnostics configuration added to it as part of our Resource Manager template samples. You can see it at this location in the Azure Samples gallery: [Five-node cluster with Diagnostics Resource Manager template sample](#).

To see the Diagnostics setting in the Resource Manager template, open the azuredeploy.json file and search for **IaaS.Diagnostics**. To create a cluster by using this template, select the **Deploy to Azure** button available at the previous link.

Alternatively, you can download the Resource Manager sample, make changes to it, and create a cluster with the modified template by using the `New-AzureRmResourceGroupDeployment` command in an Azure PowerShell window. See the following code for the parameters that you pass in to the command. For detailed information on how to deploy a resource group by using PowerShell, see the article [Deploy a resource group with the Azure Resource Manager template](#).

## Deploy the Diagnostics extension to an existing cluster

If you have an existing cluster that doesn't have Diagnostics deployed, or if you want to modify an existing

configuration, you can add or update it. Modify the Resource Manager template that's used to create the existing cluster or download the template from the portal as described earlier. Modify the template.json file by performing the following tasks.

Add a new storage resource to the template by adding to the resources section.

```
{  
  "apiVersion": "2015-05-01-preview",  
  "type": "Microsoft.Storage/storageAccounts",  
  "name": "[parameters('applicationDiagnosticsStorageAccountName')]",  
  "location": "[parameters('computeLocation')]",  
  "properties": {  
    "accountType": "[parameters('applicationDiagnosticsStorageAccountType')]"  
  },  
  "tags": {  
    "resourceType": "Service Fabric",  
    "clusterName": "[parameters('clusterName')]"  
  }  
},
```

Next, add to the parameters section just after the storage account definitions, between

`supportLogStorageAccountName` and `vmNodeType0Name`. Replace the placeholder text *storage account name goes here* with the name of the storage account.

```
"applicationDiagnosticsStorageAccountType": {  
  "type": "string",  
  "allowedValues": [  
    "Standard_LRS",  
    "Standard_GRS"  
  ],  
  "defaultValue": "Standard_LRS",  
  "metadata": {  
    "description": "Replication option for the application diagnostics storage account"  
  }  
,  
  "applicationDiagnosticsStorageAccountName": {  
    "type": "string",  
    "defaultValue": "storage account name goes here",  
    "metadata": {  
      "description": "Name for the storage account that contains application diagnostics data from the  
cluster"  
    }  
},
```

Then, update the `VirtualMachineProfile` section of the template.json file by adding the following code within the extensions array. Be sure to add a comma at the beginning or the end, depending on where it's inserted.

```
{
  "name": "[concat(parameters('vmNodeType0Name'), '_Microsoft.Insights.VMDiagnosticsSettings')]",
  "properties": {
    "type": "IaaSDiagnostics",
    "autoUpgradeMinorVersion": true,
    "protectedSettings": {
      "storageAccountName": "[parameters('applicationDiagnosticsStorageAccountName')]",
      "storageAccountKey": "[listKeys(resourceId('Microsoft.Storage/storageAccounts', parameters('applicationDiagnosticsStorageAccountName')), '2015-05-01-preview').key1]",
      "storageAccountEndPoint": "https://core.windows.net/"
    },
    "publisher": "Microsoft.Azure.Diagnostics",
    "settings": {
      "WadCfg": {
        "DiagnosticMonitorConfiguration": {
          "overallQuotaInMB": "50000",
          "EtwProviders": [
            {
              "EtwEventSourceProviderConfiguration": [
                {
                  "provider": "Microsoft-ServiceFabric-Actors",
                  "scheduledTransferKeywordFilter": "1",
                  "scheduledTransferPeriod": "PT5M",
                  "DefaultEvents": {
                    "eventDestination": "ServiceFabricReliableActorEventTable"
                  }
                },
                {
                  "provider": "Microsoft-ServiceFabric-Services",
                  "scheduledTransferPeriod": "PT5M",
                  "DefaultEvents": {
                    "eventDestination": "ServiceFabricReliableServiceEventTable"
                  }
                }
              ],
              "EtwManifestProviderConfiguration": [
                {
                  "provider": "cbd93bc2-71e5-4566-b3a7-595d8eeaca6e8",
                  "scheduledTransferLogLevelFilter": "Information",
                  "scheduledTransferKeywordFilter": "4611686018427387904",
                  "scheduledTransferPeriod": "PT5M",
                  "DefaultEvents": {
                    "eventDestination": "ServiceFabricSystemEventTable"
                  }
                }
              ]
            }
          ],
          "StorageAccount": "[parameters('applicationDiagnosticsStorageAccountName')]"
        },
        "typeHandlerVersion": "1.5"
      }
    }
  }
}
```

After you modify the template.json file as described, republish the Resource Manager template. If the template was exported, running the deploy.ps1 file republishes the template. After you deploy, ensure that **ProvisioningState** is **Succeeded**.

## Collect health and load events

Starting with the 5.4 release of Service Fabric, health and load metric events are available for collection. These events reflect events generated by the system or your code by using the health or load reporting APIs such as [ReportPartitionHealth](#) or [ReportLoad](#). This allows for aggregating and viewing system health over time and for alerting based on health or load events. To view these events in Visual Studio's Diagnostic Event Viewer add

"Microsoft-ServiceFabric:4:0x4000000000000008" to the list of ETW providers.

To collect the events, modify the Resource Manager template to include

```
"EtwManifestProviderConfiguration": [
  {
    "provider": "cbd93bc2-71e5-4566-b3a7-595d8eeca6e8",
    "scheduledTransferLogLevelFilter": "Information",
    "scheduledTransferKeywordFilter": "4611686018427387912",
    "scheduledTransferPeriod": "PT5M",
    "DefaultEvents": {
      "eventDestination": "ServiceFabricSystemEventTable"
    }
  }
]
```

## Collect reverse proxy events

Starting with the 5.7 release of Service Fabric, [reverse proxy](#) events are available for collection. Reverse proxy emits events into two channels, one containing error events reflecting request processing failures and the other one containing verbose events about all the requests processed at the reverse proxy.

1. Collect error events: To view these events in Visual Studio's Diagnostic Event Viewer add "Microsoft-ServiceFabric:4:0x4000000000000010" to the list of ETW providers. To collect the events from Azure clusters, modify the Resource Manager template to include

```
"EtwManifestProviderConfiguration": [
  {
    "provider": "cbd93bc2-71e5-4566-b3a7-595d8eeca6e8",
    "scheduledTransferLogLevelFilter": "Information",
    "scheduledTransferKeywordFilter": "4611686018427387920",
    "scheduledTransferPeriod": "PT5M",
    "DefaultEvents": {
      "eventDestination": "ServiceFabricSystemEventTable"
    }
  }
]
```

1. Collect all request processing events: In Visual Studio's Diagnostic Event Viewer, update the Microsoft-ServiceFabric entry in the ETW provider list to "Microsoft-ServiceFabric:4:0x4000000000000020". For Azure Service Fabric clusters, modify the resource manager template to include

```
"EtwManifestProviderConfiguration": [
  {
    "provider": "cbd93bc2-71e5-4566-b3a7-595d8eeca6e8",
    "scheduledTransferLogLevelFilter": "Information",
    "scheduledTransferKeywordFilter": "4611686018427387936",
    "scheduledTransferPeriod": "PT5M",
    "DefaultEvents": {
      "eventDestination": "ServiceFabricSystemEventTable"
    }
  }
]
```

It is recommended to judiciously enable collecting events from this channel as this collects all traffic through the reverse proxy and can quickly consume storage capacity.

For Azure Service Fabric clusters, the events from all the nodes are collected and aggregated in the SystemEventTable. For detailed troubleshooting of the reverse proxy events, refer the [reverse proxy diagnostics guide](#).

## Collect from new EventSource channels

To update Diagnostics to collect logs from new EventSource channels that represent a new application that you're about to deploy, perform the same steps as previously described for the setup of Diagnostics for an existing cluster.

Update the `EtwEventSourceProviderConfiguration` section in the template.json file to add entries for the new EventSource channels before you apply the configuration update by using the `New-AzureRmResourceGroupDeployment` PowerShell command. The name of the event source is defined as part of your code in the Visual Studio-generated ServiceEventSource.cs file.

For example, if your event source is named My-Eventsource, add the following code to place the events from My-Eventsource into a table named MyDestinationTableName.

```
{  
    "provider": "My-Eventsource",  
    "scheduledTransferPeriod": "PT5M",  
    "DefaultEvents": {  
        "eventDestination": "MyDestinationTableName"  
    }  
}
```

To collect performance counters or event logs, modify the Resource Manager template by using the examples provided in [Create a Windows virtual machine with monitoring and diagnostics by using an Azure Resource Manager template](#). Then, republish the Resource Manager template.

## Collect Performance Counters

To collect performance metrics from your cluster, add the performance counters to your "WadCfg > DiagnosticMonitorConfiguration" in the Resource Manager template for your cluster. See [Service Fabric Performance Counters](#) for performance counters that we recommend collecting.

For example, here we set one performance counter, sampled every 15 seconds (this can be changed and follows the format of "PT<time><unit>", for example, PT3M would sample at three minute intervals), and transferred to the appropriate storage table every one minute.

```
"PerformanceCounters": {  
    "scheduledTransferPeriod": "PT1M",  
    "PerformanceCounterConfiguration": [  
        {  
            "counterSpecifier": "\Processor(_Total)\% Processor Time",  
            "sampleRate": "PT15S",  
            "unit": "Percent",  
            "annotation": [  
                ],  
            "sinks": ""  
        }  
    ]  
}
```

If you are using an Application Insights sink, as described in the section below, and want these metrics to show up in Application Insights, then make sure to add the sink name in the "sinks" section as shown above.

Additionally, consider creating a separate table to send your Performance Counters to, so they don't crowd out the data coming from the other logging channels you have enabled.

## Send logs to Application Insights

Sending monitoring and diagnostics data to Application Insights (AI) can be done as part of the WAD configuration. If you decide to use AI for event analysis and visualization, read [Event Analysis and Visualization with Application Insights](#) to set up an AI Sink as part of your "WadCfg".

## Next steps

Once you have correctly configured Azure diagnostics, you will see data in your Storage tables from the ETW and EventSource logs. If you choose to use OMS, Kibana, or any other data analytics and visualization platform that is not directly configured in the Resource Manager template, make sure to set up the platform of your choice to read in the data from these storage tables. Doing this for OMS is relatively trivial, and is explained in [Event and log analysis through OMS](#). Application Insights is a bit of a special case in this sense, since it can be configured as part of the Diagnostics extension configuration, so refer to the [appropriate article](#) if you choose to use AI.

### NOTE

There is currently no way to filter or groom the events that are sent to the table. If you don't implement a process to remove events from the table, the table will continue to grow. Currently, there is an example of a data grooming service running in the [Watchdog sample](#), and it is recommended that you write one for yourself as well, unless there is a good reason for you to store logs beyond a 30 or 90 day timeframe.

- [Learn how to collect performance counters or logs by using the Diagnostics extension](#)
- [Event Analysis and Visualization with Application Insights](#)
- [Event Analysis and Visualization with OMS](#)

# Event aggregation and collection using Linux Azure Diagnostics

9/15/2017 • 1 min to read • [Edit Online](#)

When you're running an Azure Service Fabric cluster, it's a good idea to collect the logs from all the nodes in a central location. Having the logs in a central location helps you analyze and troubleshoot issues in your cluster, or issues in the applications and services running in that cluster.

One way to upload and collect logs is to use the Linux Azure Diagnostics (LAD) extension, which uploads logs to Azure Storage, and also has the option to send logs to Azure Application Insights or Event Hubs. You can also use an external process to read the events from storage and place them in an analysis platform product, such as [OMS Log Analytics](#) or another log-parsing solution.

## Log and event sources

### Service Fabric platform events

Service Fabric emits a few out-of-the-box logs via [LTtng](#), including operational events or runtime events. These logs are stored in the location that the cluster's Resource Manager template specifies. To get or set the storage account details, search for the tag **AzureTableWinFabETWQueryable** and look for **StoreConnectionString**.

### Application events

Events emitted from your applications' and services' code as specified by you when instrumenting your software. You can use any logging solution that writes text-based log files--for example, LTtng. For more information, see the LTtng documentation on tracing your application.

[Monitor and diagnose services in a local machine development setup.](#)

## Deploy the Diagnostics extension

The first step in collecting logs is to deploy the Diagnostics extension on each of the VMs in the Service Fabric cluster. The Diagnostics extension collects logs on each VM and uploads them to the storage account that you specify.

To deploy the Diagnostics extension to the VMs in the cluster as part of cluster creation, set **Diagnostics** to **On**. After you create the cluster, you can't change this setting by using the portal, so you will have to make the appropriate changes in the Resource Manager template.

This configures the LAD agent to monitor specified log files. Whenever a new line is appended to the file, it creates a syslog entry that is sent to the storage (table) that you specified.

## Next steps

1. To understand in more detail what events you should examine while troubleshooting issues, see [LTtng documentation](#) and [Using LAD](#).
2. [Set up the OMS agent](#) to help gather metrics, monitor Containers deployed on your cluster, and visualize your logs

# Event analysis and visualization with Application Insights

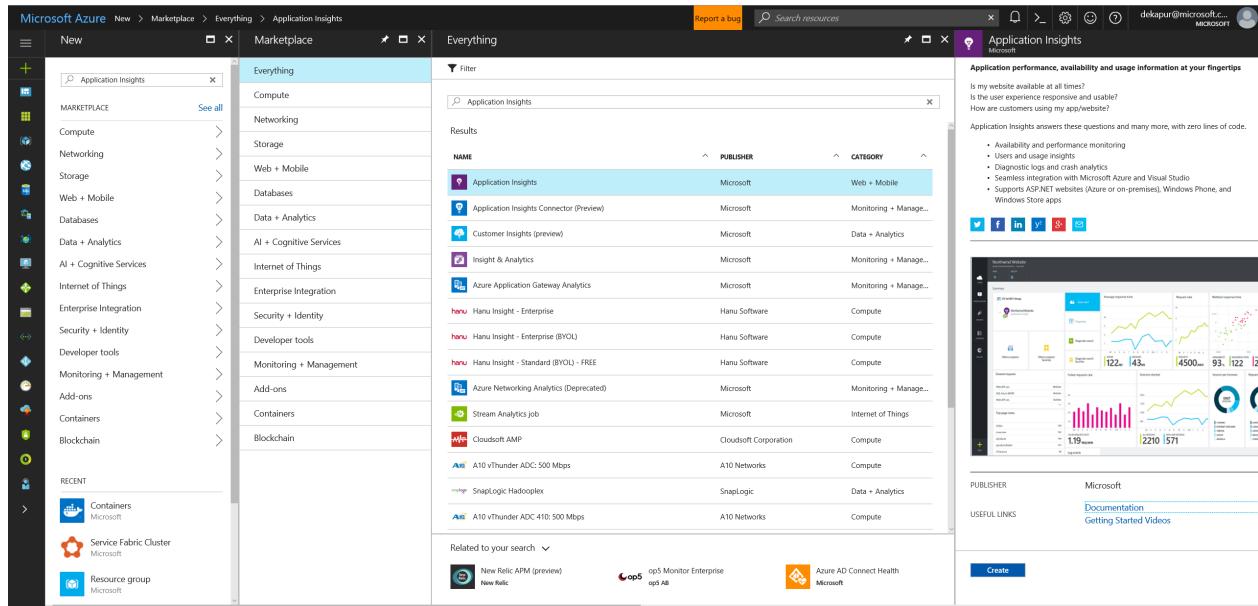
10/16/2017 • 5 min to read • [Edit Online](#)

Azure Application Insights is an extensible platform for application monitoring and diagnostics. It includes a powerful analytics and querying tool, customizable dashboard and visualizations, and further options including automated alerting. It is the recommended platform for monitoring and diagnostics for Service Fabric applications and services.

## Setting up Application Insights

### Creating an AI Resource

To create an AI resource, head over to the Azure Marketplace, and search for "Application Insights". It should show up as the first solution (it is under category "Web + Mobile"). Click **Create** when you are looking at the right resource (confirm that your path matches the image below).



You will need to fill out some information to provision the resource correctly. In the *Application Type* field, use "ASP.NET web application" if you will be using any of Service Fabric's programming models or publishing a .NET application to the cluster. Use "General" if you will be deploying guest executables and containers. In general, default to using "ASP.NET web application" to keep your options open in the future. The name is up to your preference, and both the resource group and subscription are changeable post-deployment of the resource. We recommend that your AI resource is in the same resource group as your cluster. If you need more information, please see [Create an Application Insights resource](#)

You need the AI Instrumentation Key to configure AI with your event aggregation tool. Once your AI resource is set up (takes a few minutes after the deployment is validated), navigate to it and find the **Properties** section on the left navigation bar. A new blade will open up that shows an *INSTRUMENTATION KEY*. If you need to change the subscription or resource group of the resource, it can be done here as well.

### Configuring AI with WAD

## NOTE

This is only applicable to Windows clusters at the moment.

There are two primary ways to send data from WAD to Azure AI, which is achieved by adding an AI sink to the WAD configuration, as detailed in [this article](#).

### Add an AI Instrumentation Key when creating a cluster in Azure portal

The screenshot shows the Azure portal interface for creating a Service Fabric cluster. On the left, the 'Create Service Fabric cluster' wizard is displayed with four steps: 1. Basics (Done), 2. Cluster configuration (Set up cluster configuration), 3. Security (Configure security settings), and 4. Summary (Review, view template, create). Step 2 is currently selected. On the right, the 'Cluster configuration' blade is open, specifically the 'Configure node types and diagnostics' section. This section includes fields for 'Node type count' (set to 1) and 'Diagnostics'. The 'Diagnostics' section contains a 'Create application log storage' field with an 'On' button selected, and an optional 'Application Insights key (optional)' input field. A red box highlights this entire 'Diagnostics' section. Below it, there are sections for 'Custom fabric settings' and 'Fabric version'.

When creating a cluster, if Diagnostics is turned "On", an optional field to enter an Application Insights Instrumentation key will show. If you paste your AI IKey here, the AI sink will be automatically configured for you in the Resource Manager template that is used to deploy your cluster.

### Add the AI Sink to the Resource Manager template

In the "WadCfg" of the Resource Manager template, add a "Sink" by including the following two changes:

1. Add the sink configuration directly after the declaring of the `DiagnosticMonitorConfiguration` is completed:

```
"SinksConfig": {  
    "Sink": [  
        {  
            "name": "applicationInsights",  
            "ApplicationInsights": "***ADD INSTRUMENTATION KEY HERE***"  
        }  
    ]  
}
```

2. Include the Sink in the `DiagnosticMonitorConfiguration` by adding the following line in the `DiagnosticMonitorConfiguration` of the `WadCfg` (right before the `EtwProviders` are declared):

```
"sinks": "applicationInsights"
```

In both the code snippets above, the name "applicationInsights" was used to describe the sink. This is not a requirement and as long as the name of the sink is included in "sinks", you can set the name to any string.

Currently, logs from the cluster will show up as traces in AI's log viewer. Since most of the traces coming from the platform are of level "Informational", you can also consider changing the sink configuration to only send logs of type "Critical" or "Error". This can be done by adding "Channels" to your sink, as demonstrated in [this article](#).

#### NOTE

If you use an incorrect AI IKey either in portal or in your Resource Manager template, you will have to manually change the key and update the cluster / redeploy it.

## Configuring AI with EventFlow

If you are using EventFlow to aggregate events, make sure to import the

`Microsoft.Diagnostics.EventFlow.Output.ApplicationInsights` NuGet package. The following has to be included in the *outputs* section of the *eventFlowConfig.json*:

```
"outputs": [
  {
    "type": "ApplicationInsights",
    // (replace the following value with your AI resource's instrumentation key)
    "instrumentationKey": "00000000-0000-0000-0000-000000000000"
  }
]
```

Make sure to make the required changes in your filters, as well as include any other inputs (along with their respective NuGet packages).

## AI.SDK

It is generally recommended to use EventFlow and WAD as aggregation solutions, because they allow for a more modular approach to diagnostics and monitoring, i.e. if you want to change your outputs from EventFlow, it requires no change to your actual instrumentation, just a simple modification to your config file. If, however, you decide to invest in using Application Insights and are not likely to change to a different platform, you should look into using AI's new SDK for aggregating events and sending them to AI. This means that you will no longer have to configure EventFlow to send your data to AI, but instead will install the ApplicationInsight's Service Fabric NuGet package. Details on the package can be found [here](#).

[Application Insights support for Microservices and Containers](#) shows you some of the new features that are being worked on (currently still in beta), which allow you to have richer out-of-the-box monitoring options with AI. These include dependency tracking (used in building an AppMap of all your services and applications in a cluster and the communication between them), and better correlation of traces coming from your services (helps in better pinpointing an issue in the workflow of an app or service).

If you are developing in .NET and will likely be using some of Service Fabric's programming models, and are willing to use AI as your platform for visualizing and analyzing event and log data, then we recommend that you go via the AI SDK route as your monitoring and diagnostics workflow. Read [this](#) and [this](#) to get started with using AI to collect and display your logs.

## Navigating the AI resource in Azure portal

Once you have configured AI as an output for your events and logs, information should start to show up in your AI resource in a few minutes. Navigate to the AI resource, which will take you to the AI resource dashboard. Click **Search** in the AI taskbar to see the latest traces that it has received, and to be able to filter through them.

*Metrics Explorer* is a useful tool for creating custom dashboards based on metrics that your applications, services, and cluster may be reporting. See [Exploring Metrics in Application Insights](#) to set up a few charts for yourself based on the data you are collecting.

Clicking **Analytics** will take you to the Application Insights Analytics portal, where you can query events and traces with greater scope and optionality. Read more about this at [Analytics in Application Insights](#).

## Next steps

- [Set up Alerts in AI](#) to be notified about changes in performance or usage
- [Smart Detection in Application Insights](#) performs a proactive analysis of the telemetry being sent to AI to warn you of potential performance problems

# Event analysis and visualization with OMS

10/16/2017 • 8 min to read • [Edit Online](#)

Operations Management Suite (OMS) is a collection of management services that help with monitoring and diagnostics for applications and services hosted in the cloud. To get a more detailed overview of OMS and what it offers, read [What is OMS?](#)

## Log Analytics and the OMS workspace

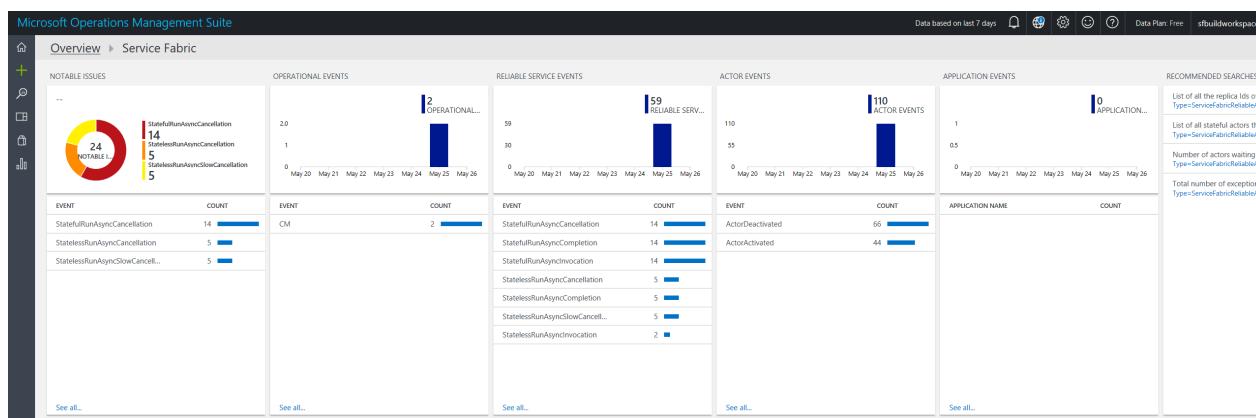
Log Analytics collects data from managed resources, including an Azure storage table or an agent, and maintains it in a central repository. The data can then be used for analysis, alerting, and visualization, or further exporting. Log Analytics supports events, performance data, or any other custom data.

When OMS is configured, you will have access to a specific *OMS workspace*, from where data can be queried or visualized in dashboards.

After data is received by Log Analytics, OMS has several *Management Solutions* that are prepackaged solutions to monitor incoming data, customized to several scenarios. These include a *Service Fabric Analytics* solution and a *Containers* solution, which are the two most relevant ones to diagnostics and monitoring when using Service Fabric clusters. There are several others as well that are worth exploring, and OMS also allows for the creation of custom solutions, which you can read more about [here](#). Each solution that you choose to use for a cluster can be configured in the same OMS workspace, alongside Log Analytics. Workspaces allow for custom dashboards and visualization of data, and modifications to the data you want to collect, process, and analyze.

## Setting up an OMS workspace with the Service Fabric Analytics Solution

It is recommended that you include the Service Fabric Solution in your OMS workspace - it includes a dashboard that shows the various incoming log channels from the platform and application level, and provides the ability to query Service Fabric specific logs. Here is what a relatively simple Service Fabric Solution looks like, with a single application deployed on the cluster:



There are two ways to provision and configure an OMS workspace, either through a Resource Manager template or directly from Azure Marketplace. Use the former when you are deploying a cluster, and the latter if you already have a cluster deployed with Diagnostics enabled.

### Deploying OMS using a Resource Management template

When deploying a cluster using a Resource Manager template, the template can also create a new OMS workspace, add the Service Fabric Solution to it, and configure it to read data from the appropriate storage tables.

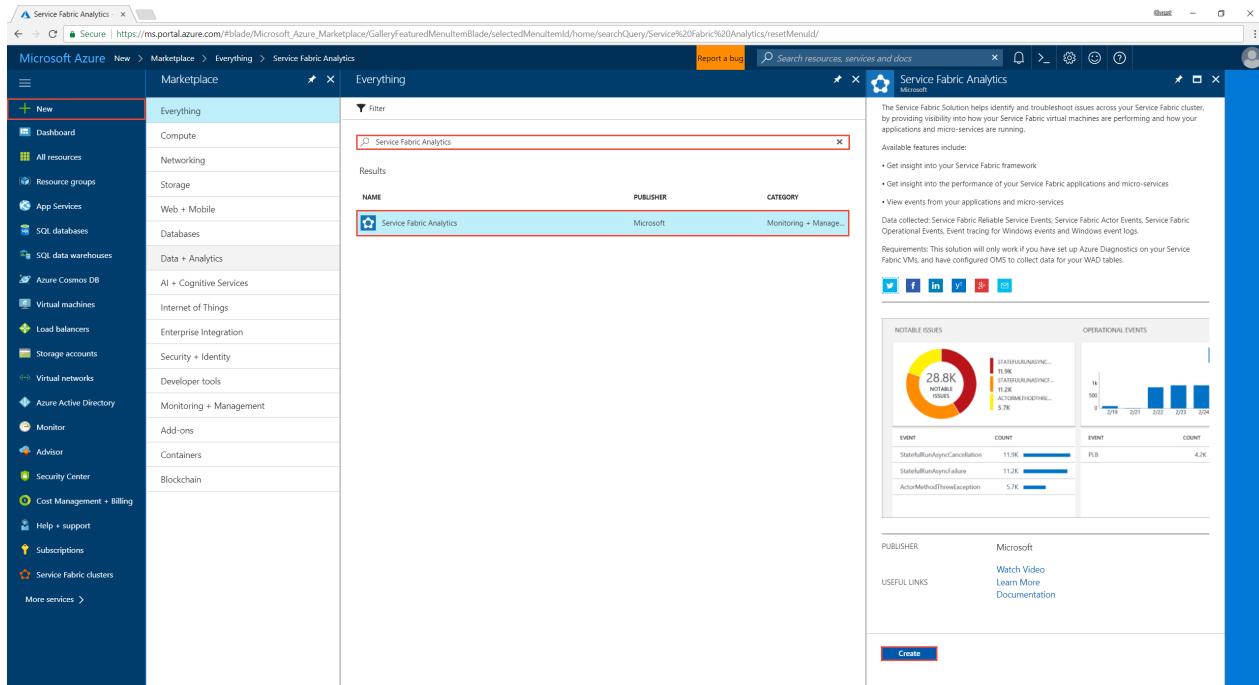
## NOTE

For this to work, Diagnostics has to be enabled in order for the Azure storage tables to exist for OMS / Log Analytics to read information from.

Here is a sample template that you can use and modify as per requirement, which performs above actions. In the case that you want more optionality, there are a few more templates that give you different options depending on where in the process you might be of setting up an OMS workspace - they can be found at [Service Fabric and OMS templates](#).

## Deploying OMS using through Azure Marketplace

If you prefer to add an OMS workspace after you have deployed a cluster, head over to Azure Marketplace and look for "Service Fabric Analytics".



- Click on **Create**
- In the Service Fabric Analytics creation window, click **Select a workspace** for the *OMS Workspace* field, and then **Create a new workspace**. Fill out the required entries - the only requirement here is that the subscription for the Service Fabric cluster and the OMS workspace should be the same. Once your entries have been validated, your OMS workspace will start to deploy. This should only take a few minutes.
- When finished, click **Create** again at the bottom of the Service Fabric Analytics creation window. Make sure that the new workspace shows up under *OMS Workspace*. This will add the solution to the workspace you just created.

Though this adds the solution to the workspace, the workspace still needs to be connected to the diagnostics data coming from your cluster. Navigate to the resource group you created the Service Fabric Analytics solution in. You should see a *ServiceFabric(<nameOfOMSWorkspace>)*.

- Click on the solution to navigate to its overview page, from where you can change solution settings, workspace settings, and navigate to the OMS portal.
- On the left navigation menu, click on **Storage accounts logs**, under *Workspace Data Sources*.

- On the *Storage account logs* page, click **Add** at the top to add your cluster's logs to the workspace.
- Click into **Storage account** to add the appropriate account created in your cluster. If you used the default name, the storage account is named `sfdg<resourceGroupName>`. You can also confirm this by checking the Azure Resource Manager template used to deploy your cluster, by checking the value used for the `applicationDiagnosticsStorageAccountName`. You may also have to scroll down and click **Load more** if the account name does not show up. Click on the right storage account name when it shows up to select it.
- Next, you'll have to specify the *Data Type*, which should be **Service Fabric Events**.
- The *Source* should automatically be set to *WADServiceFabric\*EventTable*.
- Click **OK** to connect your workspace to your cluster's logs.

**Add storage account log**

newsfomsworkspace

\* Storage account  
sfdgnewscluster7770

\* Data Type  
Service Fabric Events

Source  
WADServiceFabric\*EventTable

**OK**

- The account should now show up as part of your *Storage account logs* in your workspace's data sources.

With this you have now added the Service Fabric Analytics solution in an OMS Log Analytics workspace that is now correctly connected to your cluster's platform and application log table. You can add additional sources to

the workspace in the same way.

## Using the OMS Agent

It is recommended to use EventFlow and WAD as aggregation solutions because they allow for a more modular approach to diagnostics and monitoring. For example, if you want to change your outputs from EventFlow, it requires no change to your actual instrumentation, just a simple modification to your config file. If, however, you decide to invest in using OMS and are willing to continue using it for event analysis (does not have to be the only platform you use, but rather that it will be at least one of the platforms), we recommend that you explore setting up the [OMS agent](#). You should also use the OMS agent when deploying containers to your cluster, as discussed below.

The process for doing this is relatively easy, since you just have to add the agent as a virtual machine scale set extension to your Resource Manager template, ensuring that it gets installed on each of your nodes. A sample Resource Manager template that deploys the OMS workspace with the Service Fabric solution (as above) and adds the agent to your nodes can be found for clusters running [Windows](#) or [Linux](#).

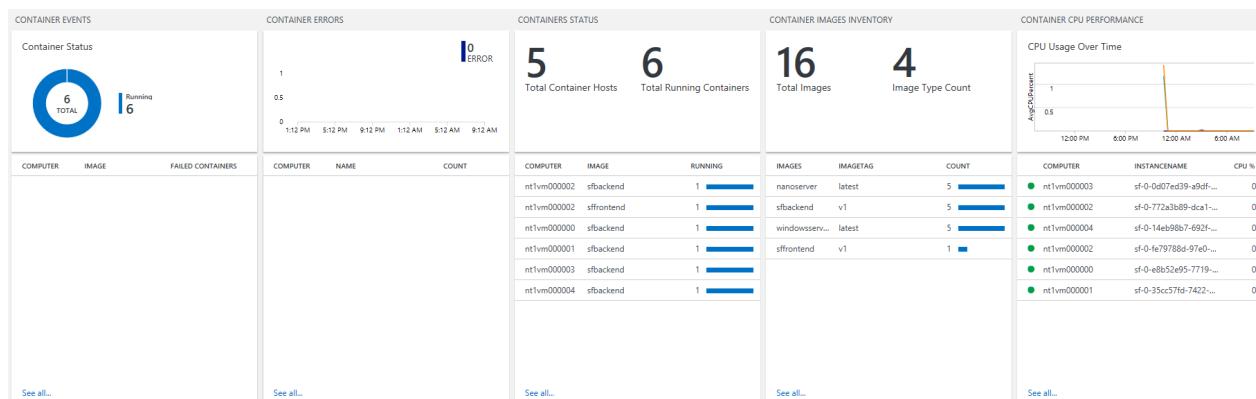
The advantages of this are the following:

- Richer data on the performance counters and metrics side
- Easy to configure metrics being collected from the cluster and without having to update your cluster's configuration. Changes to the agent's settings can be done from the OMS portal and the agent restarts automatically to match the required configuration. To configure the OMS agent to pick up specific performance counters, go to the workspace **Home > Settings > Data > Windows Performance Counters** and pick the data you would like to see collected
- Data shows up faster than it having to be stored before being picked up by OMS / Log Analytics
- Monitoring containers is much easier, since it can pick up docker logs (stdout, stderr) and stats (performance metrics on container and node levels)

The main consideration here is that since the agent will be deployed on your cluster alongside all your applications, there may be some impact to the performance of your applications on the cluster.

## Monitoring Containers

When deploying containers to a Service Fabric cluster, it is recommended that the cluster has been set up with the OMS agent and that the Containers solution has been added to your OMS workspace to enable monitoring and diagnostics. Here is what the containers solution looks like in a workspace:



The agent enables the collection of several container-specific logs that can be queried in OMS, or used to visualize performance indicators. The log types that are collected are:

- ContainerInventory: shows information about container location, name, and images
- ContainerImageInventory: information about deployed images, including IDs or sizes
- ContainerLog: specific error logs, docker logs (stdout, etc.), and other entries

- ContainerServiceLog: docker daemon commands that have been run
- Perf: performance counters including container cpu, memory, network traffic, disk i/o, and custom metrics from the host machines

This article covers the steps required to set up container monitoring for your cluster. To learn more about OMS's Containers solution, check out their [documentation](#).

To set up the Container solution in your workspace, make sure you have the OMS agent deployed on your cluster's nodes by following the steps mentioned above. Once the cluster is ready, deploy a container to it. Bear in mind that the first time a container image is deployed to a cluster, it takes several minutes to download the image depending on its size.

In Azure Marketplace, search for *Container Monitoring Solution* and create the **Container Monitoring Solution** result that should come up, under the Monitoring + Management category.

The screenshot shows the Azure Marketplace interface. On the left, there's a sidebar with various service categories like Dashboard, All resources, Resource groups, etc. The main area shows a search bar with 'Container Monitoring Solution' typed in. Below the search bar is a list of results. The first result, 'Container Monitoring Solution' by Microsoft, is highlighted with a blue background. To the right of the results, there's a detailed view of the solution, including its description, publisher information, and a preview of the monitoring dashboard. The dashboard preview shows various charts and metrics related to container usage and performance.

In the creation step, it requests an OMS workspace. Select the one that was created with the deployment above. This step adds a Containers solution within your OMS workspace, and is automatically detected by the OMS agent deployed by the template. The agent will start gathering data on the containers processes in the cluster, and less than 15 minutes or so, you should see the solution light up with data, as in the image of the dashboard above.

## Next steps

Explore the following OMS tools and options to customize a workspace to your needs:

- For on-premises clusters, OMS offers a Gateway (HTTP Forward Proxy) that can be used to send data to OMS. Read more about that in [Connecting computers without Internet access to OMS using the OMS Gateway](#)
- Configure OMS to set up [automated alerting](#) to aid in detecting and diagnostics
- Get familiarized with the [log search and querying](#) features offered as part of Log Analytics

# Troubleshoot your local development cluster setup

6/27/2017 • 2 min to read • [Edit Online](#)

If you run into an issue while interacting with your local Azure Service Fabric development cluster, review the following suggestions for potential solutions.

## Cluster setup failures

### Cannot clean up Service Fabric logs

#### Problem

While running the DevClusterSetup script, you see an error like this:

```
Cannot clean up C:\SfDevCluster\Log fully as references are likely being held to items in it. Please remove those and run this script again.  
At line:1 char:1 + .\DevClusterSetup.ps1  
+ ~~~~~  
+ CategoryInfo : NotSpecified: (:) [Write-Error], WriteErrorException  
+ FullyQualifiedErrorId : Microsoft.PowerShell.Commands.WriteErrorException,DevClusterSetup.ps1
```

#### Solution

Close the current PowerShell window and open a new PowerShell window as an administrator. You should now be able to successfully run the script.

## Cluster connection failures

### Service Fabric PowerShell cmdlets are not recognized in Azure PowerShell

#### Problem

If you try to run any of the Service Fabric PowerShell cmdlets, such as `Connect-ServiceFabricCluster` in an Azure PowerShell window, it fails, saying that the cmdlet is not recognized. The reason for this is that Azure PowerShell uses the 32-bit version of Windows PowerShell (even on 64-bit OS versions), whereas the Service Fabric cmdlets only work in 64-bit environments.

#### Solution

Always run Service Fabric cmdlets directly from Windows PowerShell.

#### NOTE

The latest version of Azure PowerShell does not create a special shortcut, so this should no longer occur.

### Type Initialization exception

#### Problem

When you are connecting to the cluster in PowerShell, you see the error `TypeInitializationException` for `System.Fabric.Common.AppTrace`.

#### Solution

Your path variable was not correctly set during installation. Sign out of Windows and sign back in. This refreshes your path.

### Cluster connection fails with "Object is closed"

#### Problem

A call to `Connect-ServiceFabricCluster` fails with an error like this:

```
Connect-ServiceFabricCluster : The object is closed.  
At line:1 char:1  
+ Connect-ServiceFabricCluster  
+ ~~~~~~  
+ CategoryInfo : InvalidOperationException: () [Connect-ServiceFabricCluster], FabricObjectClosedException  
+ FullyQualifiedErrorId : CreateClusterConnectionErrorId,Microsoft.ServiceFabric.Powershell.ConnectCluster
```

## Solution

Close the current PowerShell window and open a new PowerShell window as an administrator. You should now be able to successfully connect.

## Fabric Connection Denied exception

### Problem

When debugging from Visual Studio, you get a FabricConnectionDeniedException error.

### Solution

This error usually occurs when you try to start a service host process manually, rather than allowing the Service Fabric runtime to start it for you.

Ensure that you do not have any service projects set as startup projects in your solution. Only Service Fabric application projects should be set as startup projects.

### TIP

If, following setup, your local cluster begins to behave abnormally, you can reset it using the local cluster manager system tray application. This removes the existing cluster and set up a new one. Note that all deployed applications and associated data is removed.

## Next steps

- [Understand and troubleshoot your cluster with system health reports](#)
- [Visualize your cluster with Service Fabric Explorer](#)

Commands for managing Service Fabric clusters and entities. This version is compatible with Service Fabric 6.0 runtime. Commands follow the noun-verb pattern, see the following subgroups for more information.

## Subgroups

SUBGROUP	DESCRIPTION
<a href="#">application</a>	Create, delete, and manage applications and application types.
<a href="#">chaos</a>	Start, stop, and report on the chaos test service.
<a href="#">cluster</a>	Select, manage, and operate Service Fabric clusters.
<a href="#">compose</a>	Create, delete, and manage Docker Compose applications.
<a href="#">is</a>	Query and send commands to the infrastructure service.
<a href="#">node</a>	Manage the nodes that form a cluster.
<a href="#">partition</a>	Query and manage partitions for any service.
<a href="#">rpm</a>	Query and send commands to the repair manager service.
<a href="#">replica</a>	Manage the replicas that belong to service partitions.
<a href="#">service</a>	Create, delete, and manage service, service types and service packages.
<a href="#">store</a>	Perform basic file level operations on the cluster image store.

## Next steps

- [Set up](#) the Service Fabric CLI.
- Learn how to use the Service Fabric CLI using the [sample scripts](#).

# sfctl application

11/2/2017 • 19 min to read • [Edit Online](#)

Create, delete, and manage applications and application types.

## Commands

COMMAND	DESCRIPTION
create	Creates a Service Fabric application using the specified description.
delete	Deletes an existing Service Fabric application.
deployed	Gets the information about an application deployed on a Service Fabric node.
deployed-health	Gets the information about health of an application deployed on a Service Fabric node.
deployed-list	Gets the list of applications deployed on a Service Fabric node.
health	Gets the health of the service fabric application.
info	Gets information about a Service Fabric application.
list	Gets the list of applications created in the Service Fabric cluster that match filters specified as the parameter.
load	Gets load information about a Service Fabric application.
manifest	Gets the manifest describing an application type.
provision	Provisions or registers a Service Fabric application type with the cluster.
report-health	Sends a health report on the Service Fabric application.
type	Gets the list of application types in the Service Fabric cluster matching exactly the specified name.
type-list	Gets the list of application types in the Service Fabric cluster.
unprovision	Removes or unregisters a Service Fabric application type from the cluster.
upgrade	Starts upgrading an application in the Service Fabric cluster.
upgrade-resume	Resumes upgrading an application in the Service Fabric cluster.

COMMAND	DESCRIPTION
upgrade-rollback	Starts rolling back the currently on-going upgrade of an application in the Service Fabric cluster.
upgrade-status	Gets details for the latest upgrade performed on this application.
upload	Copy a Service Fabric application package to the image store.

## sfctl application create

Creates a Service Fabric application using the specified description.

### Arguments

ARGUMENT	DESCRIPTION
--app-name [Required]	The name of the application, including the 'fabric:' URI scheme.
--app-type [Required]	The application type name found in the application manifest.
--app-version [Required]	The version of the application type as defined in the application manifest.
--max-node-count	The maximum number of nodes that Service Fabric reserves capacity for this application. This does not mean that the services of this application are placed on all of those nodes.
--metrics	A JSON encoded list of application capacity metric descriptions. A metric is defined as a name, associated with a set of capacities for each node that the application exists on.
--min-node-count	The minimum number of nodes that Service Fabric reserves capacity for this application. This does not mean that the services of this application are placed on all of those nodes.
--parameters	A JSON encoded list of application parameter overrides to be applied when creating the application.
--timeout -t	Server timeout in seconds. Default: 60.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.

ARGUMENT	DESCRIPTION
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl application delete

Deletes an existing Service Fabric application.

Deletes an existing Service Fabric application. An application must be created before it can be deleted. Deleting an application deletes all services that are part of that application. By default Service Fabric tries to close service replicas in a graceful manner and then delete the service. However if service is having issues closing the replica gracefully, the delete operation may take a long time or get stuck. Use the optional ForceRemove flag to skip the graceful close sequence and forcefully delete the application and all of its services.

### Arguments

ARGUMENT	DESCRIPTION
--application-id [Required]	The identity of the application. This is typically the full name of the application without the 'fabric:' URI scheme. Starting from version 6.0, hierarchical names are delimited with the "~" character. For example, if the application name is "fabric://myapp/app1", the application identity would be "myapp~app1" in 6.0+ and "myapp/app1" in previous versions.
--force-remove	Remove a Service Fabric application or service forcefully without going through the graceful shutdown sequence. This parameter can be used to forcefully delete an application or service for which delete is timing out due to issues in the service code that prevents graceful close of replicas.
--timeout -t	Server timeout in seconds. Default: 60.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl application deployed

Gets the information about an application deployed on a Service Fabric node.

### Arguments

ARGUMENT	DESCRIPTION
--application-id [Required]	The identity of the application. This is typically the full name of the application without the 'fabric' URI scheme. Starting from version 6.0, hierarchical names are delimited with the "~" character. For example, if the application name is "fabric://myapp/app1", the application identity would be "myapp~app1" in 6.0+ and "myapp/app1" in previous versions.
--node-name [Required]	The name of the node.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl application health

Gets the health of the service fabric application.

Returns the health state of the service fabric application. The response reports either Ok, Error or Warning health state. If the entity is not found in the health store, it returns Error.

### Arguments

ARGUMENT	DESCRIPTION
--application-id [Required]	The identity of the application. This is typically the full name of the application without the 'fabric' URI scheme. Starting from version 6.0, hierarchical names are delimited with the "~" character. For example, if the application name is "fabric://myapp/app1", the application identity would be "myapp~app1" in 6.0+ and "myapp/app1" in previous versions.

ARGUMENT	DESCRIPTION
--deployed-applications-health-state-filter	<p>Allows filtering of the deployed applications health state objects returned in the result of application health query based on their health state. The possible values for this parameter include integer value of one of the following health states. Only deployed applications that match the filter will be returned. All deployed applications are used to evaluate the aggregated health state. If not specified, all entries are returned. The state values are flag-based enumeration, so the value could be a combination of these values obtained using bitwise 'OR' operator. For example, if the provided value is 6 then health state of deployed applications with HealthState value of OK (2) and Warning (4) are returned.</p> <ul style="list-style-type: none"> <li>- Default - Default value. Matches any HealthState. The value is zero.</li> <li>- None - Filter that doesn't match any HealthState value. Used in order to return no results on a given collection of states. The value is 1.</li> <li>- Ok - Filter that matches input with HealthState value Ok. The value is 2.</li> <li>- Warning - Filter that matches input with HealthState value Warning. The value is 4.</li> <li>- Error - Filter that matches input with HealthState value Error. The value is 8.</li> <li>- All - Filter that matches input with any HealthState value. The value is 65535.</li> </ul>
--events-health-state-filter	<p>Allows filtering the collection of HealthEvent objects returned based on health state. The possible values for this parameter include integer value of one of the following health states. Only events that match the filter are returned. All events are used to evaluate the aggregated health state. If not specified, all entries are returned. The state values are flag-based enumeration, so the value could be a combination of these values obtained using bitwise 'OR' operator. For example, If the provided value is 6 then all of the events with HealthState value of OK (2) and Warning (4) are returned.</p> <ul style="list-style-type: none"> <li>- Default - Default value. Matches any HealthState. The value is zero.</li> <li>- None - Filter that doesn't match any HealthState value. Used in order to return no results on a given collection of states. The value is 1.</li> <li>- Ok - Filter that matches input with HealthState value Ok. The value is 2.</li> <li>- Warning - Filter that matches input with HealthState value Warning. The value is 4.</li> <li>- Error - Filter that matches input with HealthState value Error. The value is 8.</li> <li>- All - Filter that matches input with any HealthState value. The value is 65535.</li> </ul>
--exclude-health-statistics	<p>Indicates whether the health statistics should be returned as part of the query result. False by default. The statistics show the number of children entities in health state Ok, Warning, and Error.</p>

ARGUMENT	DESCRIPTION
--services-health-state-filter	Allows filtering of the services health state objects returned in the result of services health query based on their health state. The possible values for this parameter include integer value of one of the following health states. Only services that match the filter are returned. All services are used to evaluate the aggregated health state. If not specified, all entries are returned. The state values are flag-based enumeration, so the value could be a combination of these values obtained using bitwise 'OR' operator. For example, if the provided value is 6 then health state of services with HealthState value of OK (2) and Warning (4) will be returned. - Default - Default value. Matches any HealthState. The value is zero. - None - Filter that doesn't match any HealthState value. Used in order to return no results on a given collection of states. The value is 1. - Ok - Filter that matches input with HealthState value Ok. The value is 2. - Warning - Filter that matches input with HealthState value Warning. The value is 4. - Error - Filter that matches input with HealthState value Error. The value is 8. - All - Filter that matches input with any HealthState value. The value is 65535.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl application info

Gets information about a Service Fabric application.

Returns the information about the application that was created or in the process of being created in the Service Fabric cluster and whose name matches the one specified as the parameter. The response includes the name, type, status, parameters, and other details about the application.

### Arguments

ARGUMENT	DESCRIPTION

ARGUMENT	DESCRIPTION
--application-id [Required]	The identity of the application. This is typically the full name of the application without the 'fabric' URI scheme. Starting from version 6.0, hierarchical names are delimited with the "~" character. For example, if the application name is "fabric://myapp/app1", the application identity would be "myapp~app1" in 6.0+ and "myapp/app1" in previous versions.
--exclude-application-parameters	The flag that specifies whether application parameters will be excluded from the result.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl application list

Gets the list of applications created in the Service Fabric cluster that match filters specified as the parameter.

Gets the information about the applications that were created or in the process of being created in the Service Fabric cluster and match filters specified as the parameter. The response includes the name, type, status, parameters, and other details about the application. If the applications do not fit in a page, one page of results is returned as well as a continuation token, which can be used to get the next page.

### Arguments

ARGUMENT	DESCRIPTION
--application-definition-kind-filter	Used to filter on ApplicationDefinitionKind for application query operations. - Default - Default value. Filter that matches input with any ApplicationDefinitionKind value. The value is 0. - All - Filter that matches input with any ApplicationDefinitionKind value. The value is 65535. - ServiceFabricApplicationDescription - Filter that matches input with ApplicationDefinitionKind value ServiceFabricApplicationDescription. The value is 1. - Compose - Filter that matches input with ApplicationDefinitionKind value Compose. The value is 2. Default: 65535.

ARGUMENT	DESCRIPTION
--application-type-name	The application type name used to filter the applications to query for. This value should not contain the application type version.
--continuation-token	The continuation token parameter is used to obtain next set of results. A continuation token with a non empty value is included in the response of the API when the results from the system do not fit in a single response. When this value is passed to the next API call, the API returns next set of results. If there are no further results, then the continuation token does not contain a value. The value of this parameter should not be URL encoded.
--exclude-application-parameters	The flag that specifies whether application parameters are excluded from the result.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl application load

Gets load information about a Service Fabric application.

Returns the load information about the application that was created or in the process of being created in the Service Fabric cluster and whose name matches the one specified as the parameter. The response includes the name, minimum nodes, maximum nodes, the number of nodes the app is occupying currently, and application load metric information about the application.

### Arguments

ARGUMENT	DESCRIPTION
--application-id [Required]	The identity of the application. This is typically the full name of the application without the 'fabric:' URI scheme. Starting from version 6.0, hierarchical names are delimited with the "~" character. For example, if the application name is "fabric://myapp/app1", the application identity would be "myapp~app1" in 6.0+ and "myapp/app1" in previous versions.

ARGUMENT	DESCRIPTION
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl application manifest

Gets the manifest describing an application type.

Gets the manifest describing an application type. The response contains the application manifest XML as a string.

### Arguments

ARGUMENT	DESCRIPTION
--application-type-name [Required]	The name of the application type.
--application-type-version [Required]	The version of the application type.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl application provision

Provisions or registers a Service Fabric application type with the cluster.

Provisions or registers a Service Fabric application type with the cluster. This is required before any new applications can be instantiated.

## Arguments

ARGUMENT	DESCRIPTION
--application-type-build-path [Required]	The relative image store path to the application package.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl application type

Gets the list of application types in the Service Fabric cluster matching exactly the specified name.

Returns the information about the application types that are provisioned or in the process of being provisioned in the Service Fabric cluster. These results are of application types whose name match exactly the one specified as the parameter, and which comply with the given query parameters. All versions of the application type matching the application type name are returned, with each version returned as one application type. The response includes the name, version, status, and other details about the application type. This is a paged query, meaning that if not all of the application types fit in a page, one page of results is returned as well as a continuation token, which can be used to get the next page. For example, if there are 10 application types but a page only fits the first 3 application types, or if max results is set to 3, then 3 is returned. To access the rest of the results, retrieve subsequent pages by using the returned continuation token in the next query. An empty continuation token is returned if there are no subsequent pages.

## Arguments

ARGUMENT	DESCRIPTION
--application-type-name [Required]	The name of the application type.

ARGUMENT	DESCRIPTION
--continuation-token	The continuation token parameter is used to obtain next set of results. A continuation token with a non empty value is included in the response of the API when the results from the system do not fit in a single response. When this value is passed to the next API call, the API returns next set of results. If there are no further results, then the continuation token does not contain a value. The value of this parameter should not be URL encoded.
--exclude-application-parameters	The flag that specifies whether application parameters will be excluded from the result.
--max-results	The maximum number of results to be returned as part of the paged queries. This parameter defines the upper bound on the number of results returned. The results returned can be less than the specified maximum results if they do not fit in the message as per the max message size restrictions defined in the configuration. If this parameter is zero or not specified, the paged query includes as much results as possible that fit in the return message.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl application unprovision

Removes or unregisters a Service Fabric application type from the cluster.

Removes or unregisters a Service Fabric application type from the cluster. This operation can only be performed if all application instance of the application type has been deleted. Once the application type is unregistered, no new application instance can be created for this particular application type.

### Arguments

ARGUMENT	DESCRIPTION
--application-type-name [Required]	The name of the application type.
--application-type-version [Required]	The application type version.

ARGUMENT	DESCRIPTION
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl application upgrade

Starts upgrading an application in the Service Fabric cluster.

Validates the supplied application upgrade parameters and starts upgrading the application if the parameters are valid. Please note that upgrade description replaces the existing application description. This means that if the parameters are not specified, the existing parameters on the applications will be overwritten with the empty parameters list. This results in application using the default value of the parameters from the application manifest.

### Arguments

ARGUMENT	DESCRIPTION
--app-id [Required]	The identity of the application. This is typically the full name of the application without the 'fabric:' URI scheme. Starting from version 6.0, hierarchical names are delimited with the '~' character. For example, if the application name is 'fabric://myapp/app1', the application identity would be 'myapp~app1' in 6.0+ and 'myapp/app1' in previous versions.
--app-version [Required]	Target application version.
--parameters [Required]	A JSON encoded list of application parameter overrides to be applied when upgrading the application.
--default-service-health-policy	JSON encoded specification of the health policy used by default to evaluate the health of a service type.
--failure-action	The action to perform when a Monitored upgrade encounters monitoring policy or health policy violations.
--force-restart	Forcefully restart processes during upgrade even when the code version has not changed.

ARGUMENT	DESCRIPTION
--health-check-retry-timeout	The amount of time to retry health evaluations when the application or cluster is unhealthy before the failure action is executed. Measured in milliseconds. Default: PT0H10M0S.
--health-check-stable-duration	The amount of time that the application or cluster must remain healthy before the upgrade proceeds to the next upgrade domain. Measured in milliseconds. Default: PT0H2M0S.
--health-check-wait-duration	The amount of time to wait after completing an upgrade domain before applying health policies. Measured in milliseconds. Default: 0.
--max-unhealthy-apps	The maximum allowed percentage of unhealthy deployed applications. Represented as a number between 0 and 100.
--mode	The mode used to monitor health during a rolling upgrade. Default: UnmonitoredAuto.
--replica-set-check-timeout	The maximum amount of time to block processing of an upgrade domain and prevent loss of availability when there are unexpected issues. Measured in seconds.
--service-health-policy	JSON encoded map with service type health policy per service type name. The map is empty by default.
--timeout -t	Server timeout in seconds. Default: 60.
--upgrade-domain-timeout	The amount of time each upgrade domain has to complete before FailureAction is executed. Measured in milliseconds. Default: P10675199DT02H48M05.4775807S.
--upgrade-timeout	The amount of time the overall upgrade has to complete before FailureAction is executed. Measured in milliseconds. Default: P10675199DT02H48M05.4775807S.
--warning-as-error	Treat health evaluation warnings with the same severity as errors.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

# sfctl application upload

Copy a Service Fabric application package to the image store.

Optionally display upload progress for each file in the package. Upload progress is sent to `stderr`.

## Arguments

ARGUMENT	DESCRIPTION
--path [Required]	Path to local application package.
--imagestore-string	Destination image store to upload the application package to. Default: fabric\ImageStore.
--show-progress	Show file upload progress for large packages.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## Next steps

- [Setup](#) the Service Fabric CLI.
- Learn how to use the Service Fabric CLI using the [sample scripts](#).

# sfctl chaos

10/10/2017 • 3 min to read • [Edit Online](#)

Start, stop, and report on the chaos test service.

## Commands

COMMAND	DESCRIPTION
report	Gets the next segment of the Chaos report based on the passed-in continuation token or the passed-in time-range.
start	If Chaos is not already running in the cluster, starts running Chaos with the specified in Chaos parameters.
stop	Stops Chaos in the cluster if it is already running, otherwise it does nothing.

## sfctl chaos report

Gets the next segment of the Chaos report based on the passed-in continuation token or the passed-in time-range.

You can either specify the ContinuationToken to get the next segment of the Chaos report or you can specify the time-range through StartTimeUtc and EndTimeUtc, but you cannot specify both the ContinuationToken and the time-range in the same call. When there are more than 100 Chaos events, the Chaos report is returned in segments where a segment contains no more than 100 Chaos events.

### Arguments

ARGUMENT	DESCRIPTION
--continuation-token	The continuation token parameter is used to obtain next set of results. A continuation token with a non empty value is included in the response of the API when the results from the system do not fit in a single response. When this value is passed to the next API call, the API returns next set of results. If there are no further results then the continuation token does not contain a value. The value of this parameter should not be URL encoded.
--end-time-utc	The count of ticks representing the end time of the time range for which a Chaos report is to be generated. Please consult <a href="#">DateTime.Ticks Property</a> for details about tick.
--start-time-utc	The count of ticks representing the start time of the time range for which a Chaos report is to be generated. Please consult <a href="#">DateTime.Ticks Property</a> for details about tick.
--timeout -t	Server timeout in seconds. Default: 60.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl chaos start

If Chaos is not already running in the cluster, starts running Chaos with the specified in Chaos parameters.

### Arguments

ARGUMENT	DESCRIPTION
--app-type-health-policy-map	JSON encoded list with max percentage unhealthy applications for specific application types. Each entry specifies as a key the application type name and as a value an integer that represents the MaxPercentUnhealthyApplications percentage used to evaluate the applications of the specified application type.
--disable-move-replica-faults	Disables the move primary and move secondary faults.
--max-cluster-stabilization	The maximum amount of time to wait for all cluster entities to become stable and healthy. Default: 60.
--max-concurrent-faults	The maximum number of concurrent faults induced per iteration. Default: 1.
--max-percent-unhealthy-apps	When evaluating cluster health during Chaos, the maximum allowed percentage of unhealthy applications before reporting an error.
--max-percent-unhealthy-nodes	When evaluating cluster health during Chaos, the maximum allowed percentage of unhealthy nodes before reporting an error.
--time-to-run	Total time (in seconds) for which Chaos will run before automatically stopping. The maximum allowed value is 4,294,967,295 (System.UInt32.MaxValue). Default: 4294967295.
--timeout -t	Server timeout in seconds. Default: 60.
--wait-time-between-faults	Wait time (in seconds) between consecutive faults within a single iteration. Default: 20.

ARGUMENT	DESCRIPTION
--wait-time-between-iterations	Time-separation (in seconds) between two consecutive iterations of Chaos. Default: 30.
--warning-as-error	When evaluating cluster health during Chaos, treat warnings with the same severity as errors.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl chaos stop

Stops Chaos in the cluster if it is already running, otherwise it does nothing.

Stops Chaos from scheduling further faults; but, the in-flight faults are not affected.

### Arguments

ARGUMENT	DESCRIPTION
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## Next steps

- [Setup](#) the Service Fabric CLI.

- Learn how to use the Service Fabric CLI using the [sample scripts](#).

# sfctl cluster

11/2/2017 • 11 min to read • [Edit Online](#)

Select, manage, and operate Service Fabric clusters.

## Commands

COMMAND	DESCRIPTION
code-versions	Gets a list of fabric code versions that are provisioned in a Service Fabric cluster.
config-versions	Gets a list of fabric config versions that are provisioned in a Service Fabric cluster.
health	Gets the health of a Service Fabric cluster.
manifest	Get the Service Fabric cluster manifest.
operation-cancel	Cancels a user-induced fault operation.
operationgit	Gets a list of user-induced fault operations filtered by provided input.
provision	Provision the code or configuration packages of a Service Fabric cluster.
recover-system	Indicates to the Service Fabric cluster that it should attempt to recover the system services which are currently stuck in quorum loss.
report-health	Sends a health report on the Service Fabric cluster.
select	Connects to a Service Fabric cluster endpoint.
unprovision	Unprovision the code or configuration packages of a Service Fabric cluster.
upgrade	Start upgrading the code or configuration version of a Service Fabric cluster.
upgrade-resume	Make the cluster upgrade move on to the next upgrade domain.
upgrade-rollback	Roll back the upgrade of a Service Fabric cluster.
upgrade-status	Gets the progress of the current cluster upgrade.
upgrade-update	Update the upgrade parameters of a Service Fabric cluster upgrade.

# sfctl cluster health

Gets the health of a Service Fabric cluster.

Gets the health of a Service Fabric cluster. Use EventsHealthStateFilter to filter the collection of health events reported on the cluster based on the health state. Similarly, use NodesHealthStateFilter and ApplicationsHealthStateFilter to filter the collection of nodes and applications returned based on their aggregated health state.

## Arguments

ARGUMENT	DESCRIPTION
--applications-health-state-filter	Allows filtering of the application health state objects returned in the result of cluster health query based on their health state. The possible values for this parameter include integer value obtained from members or bitwise operations on members of HealthStateFilter enumeration. Only applications that match the filter are returned. All applications are used to evaluate the aggregated health state. If not specified, all entries are returned. The state values are flag based enumeration, so the value could be a combination of these values obtained using bitwise 'OR' operator. For example, if the provided value is 6 then health state of applications with HealthState value of OK (2) and Warning (4) are returned. - Default - Default value. Matches any HealthState. The value is zero. - None - Filter that doesn't match any HealthState value. Used in order to return no results on a given collection of states. The value is 1. - Ok - Filter that matches input with HealthState value Ok. The value is 2. - Warning - Filter that matches input with HealthState value Warning. The value is 4. - Error - Filter that matches input with HealthState value Error. The value is 8. - All - Filter that matches input with any HealthState value. The value is 65535.
--events-health-state-filter	Allows filtering the collection of HealthEvent objects returned based on health state. The possible values for this parameter include integer value of one of the following health states. Only events that match the filter are returned. All events are used to evaluate the aggregated health state. If not specified, all entries are returned. The state values are flag- based enumeration, so the value could be a combination of these values obtained using bitwise 'OR' operator. For example, If the provided value is 6 then all of the events with HealthState value of OK (2) and Warning (4) are returned. - Default - Default value. Matches any HealthState. The value is zero. - None - Filter that doesn't match any HealthState value. Used in order to return no results on a given collection of states. The value is 1. - Ok - Filter that matches input with HealthState value Ok. The value is 2. - Warning - Filter that matches input with HealthState value Warning. The value is 4. - Error - Filter that matches input with HealthState value Error. The value is 8. - All - Filter that matches input with any HealthState value. The value is 65535.
--exclude-health-statistics	Indicates whether the health statistics should be returned as part of the query result. False by default. The statistics show the number of children entities in health state Ok, Warning, and Error.

ARGUMENT	DESCRIPTION
--include-system-application-health-statistics	Indicates whether the health statistics should include the fabric:/System application health statistics. False by default. If IncludeSystemApplicationHealthStatistics is set to true, the health statistics include the entities that belong to the fabric:/System application. Otherwise, the query result includes health statistics only for user applications. The health statistics must be included in the query result for this parameter to be applied.
--nodes-health-state-filter	Allows filtering of the node health state objects returned in the result of cluster health query based on their health state. The possible values for this parameter include integer value of one of the following health states. Only nodes that match the filter are returned. All nodes are used to evaluate the aggregated health state. If not specified, all entries are returned. The state values are flag-based enumeration, so the value could be a combination of these values obtained using bitwise 'OR' operator. For example, if the provided value is "6" then health state of nodes with HealthState value of OK (2) and Warning (4) are returned. - Default - Default value. Matches any HealthState. The value is zero. - None - Filter that doesn't match any HealthState value. Used in order to return no results on a given collection of states. The value is 1. - Ok - Filter that matches input with HealthState value Ok. The value is 2. - Warning - Filter that matches input with HealthState value Warning. The value is 4. - Error - Filter that matches input with HealthState value Error. The value is 8. - All - Filter that matches input with any HealthState value. The value is 65535.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl cluster manifest

Get the Service Fabric cluster manifest.

Get the Service Fabric cluster manifest. The cluster manifest contains properties of the cluster that include different node types on the cluster, security configurations, fault, and upgrade domain topologies etc. These properties are specified as part of the ClusterConfig.JSON file while deploying a standalone cluster. However, most of the information in the cluster manifest is generated internally by service fabric during cluster deployment in other

deployment scenarios (for example when using the [Azure portal](#)). The contents of the cluster manifest are for informational purposes only and users are not expected to take a dependency on the format of the file contents or its interpretation.

## Arguments

ARGUMENT	DESCRIPTION
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl cluster provision

Provision the code or configuration packages of a Service Fabric cluster. Validate and provision the code or configuration packages of a Service Fabric cluster.

## Arguments

ARGUMENT	DESCRIPTION
--cluster-manifest-file-path	The cluster manifest file path.
--code-file-path	The cluster code package file path.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.

ARGUMENT	DESCRIPTION
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl cluster select

Connects to a Service Fabric cluster endpoint.

If connecting to secure cluster, specify a cert (.crt) and key file (.key) or a single file with both (.pem). Do not specify both. Optionally, if connecting to a secure cluster, specify also a path to a CA bundle file or directory of trusted CA certs.

### Arguments

ARGUMENT	DESCRIPTION
--endpoint [Required]	Cluster endpoint URL, including port and HTTP or HTTPS prefix.
--aad	Use Azure Active Directory for authentication.
--ca	Path to CA certs directory to treat as valid or CA bundle file.
--cert	Path to a client certificate file.
--key	Path to client certificate key file.
--no-verify	Disable verification for certificates when using HTTPS, note: this is an insecure option and should not be used for production environments.
--pem	Path to client certificate, as a .pem file.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl cluster unprovision

Unprovision the code or configuration packages of a Service Fabric cluster.

Unprovision the code or configuration packages of a Service Fabric cluster.

## Arguments

ARGUMENT	DESCRIPTION
--code-version	The cluster code package version.
--config-version	The cluster manifest version.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl cluster upgrade

Start upgrading the code or configuration version of a Service Fabric cluster. Validate the supplied upgrade parameters and start upgrading the code or configuration version of a Service Fabric cluster if the parameters are valid.

## Arguments

ARGUMENT	DESCRIPTION
--app-health-map	JSON encoded dictionary of pairs of application name and maximum percentage unhealthy before raising error.
--app-type-health-map	JSON encoded dictionary of pairs of application type name and maximum percentage unhealthy before raising error.
--code-version	The cluster code version.
--config-version	The cluster configuration version.
--delta-health-evaluation	Enables delta health evaluation rather than absolute health evaluation after completion of each upgrade domain.

ARGUMENT	DESCRIPTION
--delta-unhealthy-nodes	The maximum allowed percentage of nodes health degradation allowed during cluster upgrades. Default: 10. The delta is measured between the state of the nodes at the beginning of upgrade and the state of the nodes at the time of the health evaluation. The check is performed after every upgrade domain upgrade completion to make sure the global state of the cluster is within tolerated limits.
--failure-action	Possible values include: 'Invalid', 'Rollback', 'Manual'.
--force-restart	Force restart.
--health-check-retry	Health check retry timeout measured in milliseconds.
--health-check-stable	Health check stable duration measured in milliseconds.
--health-check-wait	Health check wait duration measured in milliseconds.
--replica-set-check-timeout	Upgrade replica set check timeout measured in seconds.
--rolling-upgrade-mode	Possible values include: 'Invalid', 'UnmonitoredAuto', 'UnmonitoredManual', 'Monitored'. Default: UnmonitoredAuto.
--timeout -t	Server timeout in seconds. Default: 60.
--unhealthy-applications	The maximum allowed percentage of unhealthy applications before reporting an error. For example, to allow 10% of applications to be unhealthy, this value would be 10. The percentage represents the maximum tolerated percentage of applications that can be unhealthy before the cluster is considered in error. If the percentage is respected but there is at least one unhealthy application, the health is evaluated as Warning. This is calculated by dividing the number of unhealthy applications over the total number of application instances in the cluster, excluding applications of application types that are included in the ApplicationTypeHealthPolicyMap. The computation rounds up to tolerate one failure on small numbers of applications.
--unhealthy-nodes	The maximum allowed percentage of unhealthy nodes before reporting an error. For example, to allow 10% of nodes to be unhealthy, this value would be 10. The percentage represents the maximum tolerated percentage of nodes that can be unhealthy before the cluster is considered in error. If the percentage is respected but there is at least one unhealthy node, the health is evaluated as Warning. The percentage is calculated by dividing the number of unhealthy nodes over the total number of nodes in the cluster. The computation rounds up to tolerate one failure on small numbers of nodes. In large clusters, some nodes will always be down or out for repairs, so this percentage should be configured to tolerate that.

ARGUMENT	DESCRIPTION
--upgrade-domain-delta-unhealthy-nodes	The maximum allowed percentage of upgrade domain nodes health degradation allowed during cluster upgrades. Default: 15. The delta is measured between the state of the upgrade domain nodes at the beginning of upgrade and the state of the upgrade domain nodes at the time of the health evaluation. The check is performed after every upgrade domain upgrade completion for all completed upgrade domains to make sure the state of the upgrade domains is within tolerated limits.
--upgrade-domain-timeout	Upgrade domain timeout measured in milliseconds.
--upgrade-timeout	Upgrade timeout measured in milliseconds.
--warning-as-error	Warnings are treated with the same severity as errors.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## Next steps

- [Setup](#) the Service Fabric CLI.
- Learn how to use the Service Fabric CLI using the [sample scripts](#).

# sfctl compose

10/3/2017 • 5 min to read • [Edit Online](#)

Create, delete, and manage Docker Compose deployments.

## Commands

COMMAND	DESCRIPTION
create	Deploy a Service Fabric application from a Compose file.
list	Gets the list of compose deployments created in the Service Fabric cluster.
remove	Deletes an existing Service Fabric compose deployment from cluster.
status	Gets information about a Service Fabric compose deployment.
upgrade	Starts upgrading a compose deployment in the Service Fabric cluster.
upgrade-status	Gets details for the latest upgrade performed on this Service Fabric compose deployment.

## sfctl compose create

Creates a Service Fabric compose deployment.

### Arguments

ARGUMENT	DESCRIPTION
--file-path [Required]	Path to the target Docker Compose file.
--deployment-name [Required]	The name of the deployment.
--encrypted-pass	Rather than prompting for a container registry password, use an already encrypted passphrase.
--has-pass	Prompts for a password to the container registry.
--timeout -t	Server time-out in seconds. Default: 60.
--user	User name to connect to container registry.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl compose list

Gets the list of compose deployments created in the Service Fabric cluster.

Gets the status about the compose deployments that were created or are in the process of being created in the Service Fabric cluster. The response includes the name, status, and other details about the compose deployment. If the deployments do not fit in a page, one page of results is returned as well as a continuation token, which can be used to get the next page.

### Arguments

ARGUMENT	DESCRIPTION
--continuation-token	The continuation token parameter is used to obtain next set of results. A continuation token with a non empty value is included in the response of the API when the results from the system do not fit in a single response. When this value is passed to the next API call, the API returns next set of results. If there are no further results, then the continuation token does not contain a value. The value of this parameter should not be URL encoded.
--max-results	The maximum number of results to be returned as part of the paged queries. This parameter defines the upper bound on the number of results returned. If they do not fit in the message as per the max message size restrictions defined in the configuration, the results returned can be less than the specified maximum results. If this parameter is zero or not specified, the paged queries include as many results as possible that fit in the return message.
--timeout -t	Server time-out in seconds. Default: 60.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.

ARGUMENT	DESCRIPTION
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl compose remove

Deletes an existing Service Fabric compose deployment from cluster.

Deletes an existing Service Fabric compose deployment.

### Arguments

ARGUMENT	DESCRIPTION
--deployment-name [Required]	The identity of the deployment. This is typically the full name of the application without the 'fabric:' URI scheme.
--timeout -t	Server time-out in seconds. Default: 60.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl compose status

Gets information about a Service Fabric compose deployment.

Returns the status of compose deployment that was created or in the process of being created in the Service Fabric cluster and whose name matches the one specified as the parameter. The response includes the name, status, and other details about the application.

### Arguments

ARGUMENT	DESCRIPTION
--deployment-name [Required]	The identity of the deployment.

ARGUMENT	DESCRIPTION
--timeout -t	Server time-out in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl compose upgrade

Starts upgrading a compose deployment in the Service Fabric cluster.

Validates the supplied upgrade parameters and starts upgrading the deployment.

### Arguments

ARGUMENT	DESCRIPTION
--file-path [Required]	Path to the target Docker Compose file.
--deployment-name [Required]	The name of the deployment.
--default-svc-type-health-map	JSON encoded dictionary that describes the health policy used to evaluate the health of services.
--encrypted-pass	Rather than prompting for a container registry password, use an already encrypted passphrase.
--failure-action	Possible values include: 'Invalid', 'Rollback', 'Manual'.
--force-restart	Force restart.
--has-pass	Prompts for a password to the container registry.
--health-check-retry	Health check retry time-out measured in milliseconds.
--health-check-stable	Health check stable duration measured in milliseconds.
--health-check-wait	Health check wait duration measured in milliseconds.
--replica-set-check	Upgrade replica set check time-out measured in seconds.

ARGUMENT	DESCRIPTION
--svc-type-health-map	JSON encoded list of objects that describe the health policies used to evaluate the health of different service types.
--timeout -t	Server time-out in seconds. Default: 60.
--unhealthy-app	The maximum allowed percentage of unhealthy applications before reporting an error. For example, to allow 10% of applications to be unhealthy, this value would be 10. The percentage represents the maximum tolerated percentage of applications that can be unhealthy before the cluster is considered in error. If the percentage is respected but there is at least one unhealthy application, the health is evaluated as Warning. This percentage is calculated by dividing the number of unhealthy applications over the total number of application instances in the cluster.
--upgrade-domain-timeout	Upgrade domain time-out measured in milliseconds.
--upgrade-kind	Default: Rolling.
--upgrade-mode	Possible values include: 'Invalid', 'UnmonitoredAuto', 'UnmonitoredManual', 'Monitored'. Default: UnmonitoredAuto.
--upgrade-timeout	Upgrade time-out measured in milliseconds.
--user	User name to connect to container registry.
--warning-as-error	Warnings are treated with the same severity as errors.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## Next steps

- [Set up](#) the Service Fabric CLI.
- Learn how to use the Service Fabric CLI using the [sample scripts](#).

# sfctl is

9/27/2017 • 2 min to read • [Edit Online](#)

Query and send commands to the infrastructure service.

## Commands

COMMAND	DESCRIPTION
command	Invokes an administrative command on the given Infrastructure Service instance.
query	Invokes a read-only query on the given infrastructure service instance.

### sfctl is command

Invokes an administrative command on the given Infrastructure Service instance.

For clusters that have one or more instances of the Infrastructure Service configured, this API provides a way to send infrastructure-specific commands to a particular instance of the Infrastructure Service. Available commands and their corresponding response formats vary depending upon the infrastructure on which the cluster is running. This API supports the Service Fabric platform; it is not meant to be used directly from your code. .

#### Arguments

ARGUMENT	DESCRIPTION
--command [Required]	The text of the command to be invoked. The content of the command is infrastructure-specific. Default: is command.
--service-id	The identity of the infrastructure service. This is the full name of the infrastructure service without the 'fabric:' URI scheme. This parameter required only for clusters that have more than one instance of infrastructure service running.
--timeout -t	Server timeout in seconds. Default: 60.

#### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .

ARGUMENT	DESCRIPTION
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl is query

Invokes a read-only query on the given infrastructure service instance.

For clusters that have one or more instances of the Infrastructure Service configured, this API provides a way to send infrastructure-specific queries to a particular instance of the Infrastructure Service. Available commands and their corresponding response formats vary depending upon the infrastructure on which the cluster is running. This API supports the Service Fabric platform; it is not meant to be used directly from your code.

### Arguments

ARGUMENT	DESCRIPTION
--command [Required]	The text of the command to be invoked. The content of the command is infrastructure-specific. Default: is query.
--service-id	The identity of the infrastructure service. This is the full name of the infrastructure service without the 'fabric:' URI scheme. This parameter is required only for clusters that have more than one instance of infrastructure service running.
--timeout -t	Server timeout in seconds. Default: 60.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## Next steps

- [Set up](#) the Service Fabric CLI.
- Learn how to use the Service Fabric CLI using the [sample scripts](#).

# sfctl node

9/27/2017 • 9 min to read • [Edit Online](#)

Manage the nodes that form a cluster.

## Commands

COMMAND	DESCRIPTION
disable	Deactivate a Service Fabric cluster node with the specified deactivation intent.
enable	Activate a Service Fabric cluster node, which is currently deactivated.
health	Gets the health of a Service Fabric node.
info	Gets the list of nodes in the Service Fabric cluster.
list	Gets the list of nodes in the Service Fabric cluster.
load	Gets the load information of a Service Fabric node.
remove-state	Notifies Service Fabric that the persisted state on a node has been permanently removed or lost.
report-health	Sends a health report on the Service Fabric node.
restart	Restarts a Service Fabric cluster node.
transition	Starts or stops a cluster node.
transition-status	Gets the progress of an operation started using StartNodeTransition.

## sfctl node disable

Deactivate a Service Fabric cluster node with the specified deactivation intent.

Deactivate a Service Fabric cluster node with the specified deactivation intent. Once the deactivation is in progress, the deactivation intent can be increased, but not decreased (for example, a node which is deactivated with the Pause intent can be deactivated further with Restart, but not the other way around). Nodes may be reactivated using the Activate a node operation any time after they are deactivated. If the deactivation is not complete this cancels the deactivation. A node that goes down and comes back up while deactivated still needs to be reactivated before services can be placed on that node.

### Arguments

ARGUMENT	DESCRIPTION
--node-name [Required]	The name of the node.
--deactivation-intent	Describes the intent or reason for deactivating the node. The possible values are following. - Pause - Indicates that the node should be paused. The value is 1. - Restart - Indicates that the intent is for the node to be restarted after a short period of time. The value is 2. - RemoveData - Indicates the intent is for the node to remove data. The value is 3. .
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl node enable

Activate a Service Fabric cluster node which is currently deactivated.

Activates a Service Fabric cluster node which is currently deactivated. Once activated, the node again becomes a viable target for placing new replicas, and any deactivated replicas remaining on the node are reactivated.

### Arguments

ARGUMENT	DESCRIPTION
--node-name [Required]	The name of the node.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.

Argument	Description
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl node health

Gets the health of a Service Fabric node.

Gets the health of a Service Fabric node. Use EventsHealthStateFilter to filter the collection of health events reported on the node based on the health state. If the node that you specify by name does not exist in the health store, this returns an error.

### Arguments

Argument	Description
--node-name [Required]	The name of the node.
--events-health-state-filter	Allows filtering the collection of HealthEvent objects returned based on health state. The possible values for this parameter include integer value of one of the following health states. Only events that match the filter are returned. All events are used to evaluate the aggregated health state. If not specified, all entries are returned. The state values are flag-based enumeration, so the value could be a combination of these values obtained using bitwise 'OR' operator. For example, If the provided value is 6 then all of the events with HealthState value of OK (2) and Warning (4) are returned. - Default - Default value. Matches any HealthState. The value is zero. - None - Filter that doesn't match any HealthState value. Used in order to return no results on a given collection of states. The value is 1. - Ok - Filter that matches input with HealthState value Ok. The value is 2. - Warning - Filter that matches input with HealthState value Warning. The value is 4. - Error - Filter that matches input with HealthState value Error. The value is 8. - All - Filter that matches input with any HealthState value. The value is 65535.
--timeout -t	Server timeout in seconds. Default: 60.

### Global Arguments

Argument	Description
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.

ARGUMENT	DESCRIPTION
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl node info

Gets the list of nodes in the Service Fabric cluster.

Gets the information about a specific node in the Service Fabric cluster. The response include the name, status, ID, health, uptime, and other details about the node.

### Arguments

ARGUMENT	DESCRIPTION
--node-name [Required]	The name of the node.
--timeout -t	Server timeout in seconds. Default: 60.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl node list

Gets the list of nodes in the Service Fabric cluster.

The Nodes endpoint returns information about the nodes in the Service Fabric Cluster. The response includes the name, status, ID, health, uptime, and other details about the node.

### Arguments

ARGUMENT	DESCRIPTION
--continuation-token	The continuation token parameter is used to obtain next set of results. A continuation token with a non empty value is included in the response of the API when the results from the system do not fit in a single response. When this value is passed to the next API call, the API returns next set of results. If there are no further results, then the continuation token does not contain a value. The value of this parameter should not be URL encoded.

ARGUMENT	DESCRIPTION
--node-status-filter	Allows filtering the nodes based on the NodeStatus. Only the nodes that are matching the specified filter value are returned. The filter value can be one of the following. - default - This filter value matches all of the nodes excepts the ones with status as Unknown or Removed. - all - This filter value matches all of the nodes. - up - This filter value matches nodes that are Up. - down - This filter value matches nodes that are Down. - enabling - This filter value matches nodes that are in the process of being enabled with status as Enabling. - disabling - This filter value matches nodes that are in the process of being disabled with status as Disabling. - disabled - This filter value matches nodes that are Disabled. - unknown - This filter value matches nodes whose status is Unknown. A node would be in Unknown state if Service Fabric does not have authoritative information about that node. This can happen if the system learns about a node at runtime. - removed - This filter value matches nodes whose status is Removed. These are the nodes that are removed from the cluster using the RemoveNodeState API. . Default: default.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl node load

Gets the load information of a Service Fabric node.

Gets the load information of a Service Fabric node.

### Arguments

ARGUMENT	DESCRIPTION
--node-name [Required]	The name of the node.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl node restart

Restarts a Service Fabric cluster node.

Restarts a Service Fabric cluster node that is already started.

### Arguments

ARGUMENT	DESCRIPTION
--node-name [Required]	The name of the node.
--create-fabric-dump	Specify True to create a dump of the fabric node process. This is case-sensitive. Default: False.
--node-instance-id	The instance ID of the target node. If instance ID is specified the node is restarted only if it matches with the current instance of the node. A default value of "0" would match any instance ID. The instance ID can be obtained using get node query. Default: 0.
--timeout -t	Server timeout in seconds. Default: 60.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl node transition

Starts or stops a cluster node.

Starts or stops a cluster node. A cluster node is a process, not the OS instance itself. To start a node, pass in "Start" for the NodeTransitionType parameter. To stop a node, pass in "Stop" for the NodeTransitionType parameter. This API starts the operation - when the API returns the node may not have finished transitioning yet. Call GetNodeTransitionProgress with the same OperationId to get the progress of the operation..

## Arguments

ARGUMENT	DESCRIPTION
--node-instance-id [Required]	The node instance ID of the target node. This can be determined through GetNodeInfo API.
--node-name [Required]	The name of the node.
--node-transition-type [Required]	Indicates the type of transition to perform. NodeTransitionType.Start starts a stopped node. NodeTransitionType.Stop stops a node that is up. - Invalid - Reserved. Do not pass into API. - Start - Transition a stopped node to up. - Stop - Transition an up node to stopped. .
--operation-id [Required]	A GUID that identifies a call of this API. This is passed into the corresponding GetProgress API.
--stop-duration-in-seconds [Required]	The duration, in seconds, to keep the node stopped. The minimum value is 600, the maximum is 14400. After this time expires, the node automatically comes back up.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## Next steps

- [Setup](#) the Service Fabric CLI.
- Learn how to use the Service Fabric CLI using the [sample scripts](#).

# sfctl partition

10/13/2017 • 8 min to read • [Edit Online](#)

Query and manage partitions for any service.

## Commands

COMMAND	DESCRIPTION
data-loss	This API induces data loss for the specified partition.
data-loss-status	Gets the progress of a partition data loss operation started using the StartDataLoss API.
health	Gets the health of the specified Service Fabric partition.
info	Gets the information about a Service Fabric partition.
list	Gets the list of partitions of a Service Fabric service.
load	Gets the load of the specified Service Fabric partition.
load-reset	Resets the current load of a Service Fabric partition.
quorum-loss	Induces quorum loss for a given stateful service partition.
quorum-loss-status	Gets the progress of a quorum loss operation on a partition started using the StartQuorumLoss API.
recover	Indicates to the Service Fabric cluster that it should attempt to recover a specific partition, which is currently stuck in quorum loss.
recover-all	Indicates to the Service Fabric cluster that it should attempt to recover any services (including system services) which are currently stuck in quorum loss.
report-health	Sends a health report on the Service Fabric partition.
restart	This API restarts some or all replicas or instances of the specified partition.
restart-status	Gets the progress of a PartitionRestart operation started using StartPartitionRestart.
svc-name	Gets the name of the Service Fabric service for a partition.

## sfctl partition health

Gets the health of the specified Service Fabric partition.

Gets the health information of the specified partition. Use EventsHealthStateFilter to filter the collection of health events reported on the service based on the health state. Use ReplicasHealthStateFilter to filter the collection of ReplicaHealthState objects on the partition. If you specify a partition that does not exist in the health store, this cmdlet returns an error..

## Arguments

Argument	Description
--partition-id [Required]	The identity of the partition.
--events-health-state-filter	Allows filtering the collection of HealthEvent objects returned based on health state. The possible values for this parameter include integer value of one of the following health states. Only events that match the filter are returned. All events are used to evaluate the aggregated health state. If not specified, all entries are returned. The state values are flag-based enumeration, so the value could be a combination of these values obtained using bitwise 'OR' operator. For example, If the provided value is 6 then all of the events with HealthState value of OK (2) and Warning (4) are returned. - Default - Default value. Matches any HealthState. The value is zero. - None - Filter that doesn't match any HealthState value. Used in order to return no results on a given collection of states. The value is 1. - Ok - Filter that matches input with HealthState value Ok. The value is 2. - Warning - Filter that matches input with HealthState value Warning. The value is 4. - Error - Filter that matches input with HealthState value Error. The value is 8. - All - Filter that matches input with any HealthState value. The value is 65535.
--exclude-health-statistics	Indicates whether the health statistics should be returned as part of the query result. False by default. The statistics show the number of children entities in health state Ok, Warning, and Error.
--replicas-health-state-filter	Allows filtering the collection of ReplicaHealthState objects on the partition. The value can be obtained from members or bitwise operations on members of HealthStateFilter. Only replicas that match the filter are returned. All replicas are used to evaluate the aggregated health state. If not specified, all entries are returned. The state values are flag-based enumeration, so the value could be a combination of these values obtained using bitwise 'OR' operator. For example, If the provided value is 6 then all of the events with HealthState value of OK (2) and Warning (4) are returned. The possible values for this parameter include integer value of one of the following health states. - Default - Default value. Matches any HealthState. The value is zero. - None - Filter that doesn't match any HealthState value. Used in order to return no results on a given collection of states. The value is 1. - Ok - Filter that matches input with HealthState value Ok. The value is 2. - Warning - Filter that matches input with HealthState value Warning. The value is 4. - Error - Filter that matches input with HealthState value Error. The value is 8. - All - Filter that matches input with any HealthState value. The value is 65535.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl partition info

Gets the information about a Service Fabric partition.

The Partitions endpoint returns information about the specified partition. The response includes the partition ID, partitioning scheme information, keys supported by the partition, status, health, and other details about the partition.

### Arguments

ARGUMENT	DESCRIPTION
--partition-id [Required]	The identity of the partition.
--timeout -t	Server timeout in seconds. Default: 60.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl partition list

Gets the list of partitions of a Service Fabric service.

Gets the list of partitions of a Service Fabric service. The s the partition ID, partitioning scheme information, keys supported by the partition, status, health, and other details about the partition.

### Arguments

ARGUMENT	DESCRIPTION
--service-id [Required]	The identity of the service. This is typically the full name of the service without the 'fabric:' URI scheme. Starting from version 6.0, hierarchical names are delimited with the "~" character. For example, if the service name is "fabric://myapp/app1/svc1", the service identity would be "myapp~app1~svc1" in 6.0+ and "myapp/app1/svc1" in previous versions.
--continuation-token	The continuation token parameter is used to obtain next set of results. A continuation token with a non empty value is included in the response of the API when the results from the system do not fit in a single response. When this value is passed to the next API call, the API returns next set of results. If there are no further results, then the continuation token does not contain a value. The value of this parameter should not be URL encoded.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl partition load

Gets the load of the specified Service Fabric partition.

Returns information about the specified partition. The response includes a list of load information. Each information includes load metric name, value, and last reported time in UTC.

### Arguments

ARGUMENT	DESCRIPTION
--partition-id [Required]	The identity of the partition.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.

ARGUMENT	DESCRIPTION
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl partition recover

Indicates to the Service Fabric cluster that it should attempt to recover a specific partition that is currently stuck in quorum loss.

Indicates to the Service Fabric cluster that it should attempt to recover a specific partition that is currently stuck in quorum loss. This operation should only be performed if it is known that the replicas that are down cannot be recovered. Incorrect use of this API can cause potential data loss.

### Arguments

ARGUMENT	DESCRIPTION
--partition-id [Required]	The identity of the partition.
--timeout -t	Server timeout in seconds. Default: 60.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl partition restart

This API restarts some or all replicas or instances of the specified partition.

This API is useful for testing failover. If used to target a stateless service partition, RestartPartitionMode must be AllReplicasOrInstances. Call the GetPartitionRestartProgress API using the same OperationId to get the progress..

### Arguments

ARGUMENT	DESCRIPTION
--operation-id [Required]	A GUID that identifies a call of this API. This is passed into the corresponding GetProgress API.
--partition-id [Required]	The identity of the partition.
--restart-partition-mode [Required]	- Invalid - Reserved. Do not pass into API. - AllReplicasOrInstances - All replicas or instances in the partition are restarted at once. - OnlyActiveSecondaries - Only the secondary replicas are restarted. .
--service-id [Required]	The identity of the service. This is typically the full name of the service without the 'fabric:' URI scheme. Starting from version 6.0, hierarchical names are delimited with the "~" character. For example, if the service name is "fabric://myapp/app1/svc1", the service identity would be "myapp~app1~svc1" in 6.0+ and "myapp/app1/svc1" in previous versions.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## Next steps

- [Setup](#) the Service Fabric CLI.
- Learn how to use the Service Fabric CLI using the [sample scripts](#).

# sfctl replica

9/26/2017 • 6 min to read • [Edit Online](#)

Manage the replicas that belong to service partitions.

## Commands

COMMAND	DESCRIPTION
deployed	Gets the details of replica deployed on a Service Fabric node.
deployed-list	Gets the list of replicas deployed on a Service Fabric node.
health	Gets the health of a Service Fabric stateful service replica or stateless service
instance.	
info	Gets the information about a replica of a Service Fabric partition.
list	Gets the information about replicas of a Service Fabric service partition.
remove	Removes a service replica running on a node.
report-health	Sends a health report on the Service Fabric replica.
restart	Restarts a service replica of a persisted service running on a node.

## sfctl replica deployed

Gets the details of replica deployed on a Service Fabric node.

Gets the details of the replica deployed on a Service Fabric node. The information includes service kind, service name, current service operation, current service operation start date time, partition ID, replica/instance ID, reported load and other information.

### Arguments

ARGUMENT	DESCRIPTION
--node-name [Required]	The name of the node.
--partition-id [Required]	The identity of the partition.
--replica-id [Required]	The identifier of the replica.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl replica health

Gets the health of a Service Fabric stateful service replica or stateless service instance.

Gets the health of a Service Fabric replica. Use EventsHealthStateFilter to filter the collection of health events reported on the replica based on the health state.. .

### Arguments

ARGUMENT	DESCRIPTION
--partition-id [Required]	The identity of the partition.
--replica-id [Required]	The identifier of the replica.
--events-health-state-filter	Allows filtering the collection of HealthEvent objects returned based on health state. The possible values for this parameter include integer value of one of the following health states. Only events that match the filter are returned. All events are used to evaluate the aggregated health state. If not specified, all entries are returned. The state values are flag-based enumeration, so the value could be a combination of these value obtained using bitwise 'OR' operator. For example, If the provided value is 6 then all of the events with HealthState value of OK (2) and Warning (4) are returned. - Default - Default value. Matches any HealthState. The value is zero. - None - Filter that doesn't match any HealthState value. Used in order to return no results on a given collection of states. The value is 1. - Ok - Filter that matches input with HealthState value Ok. The value is 2. - Warning - Filter that matches input with HealthState value Warning. The value is 4. - Error - Filter that matches input with HealthState value Error. The value is 8. - All - Filter that matches input with any HealthState value. The value is 65535.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl replica info

Gets the information about a replica of a Service Fabric partition.

The responses include the ID, role, status, health, node name, uptime, and other details about the replica.

### Arguments

ARGUMENT	DESCRIPTION
--partition-id [Required]	The identity of the partition.
--replica-id [Required]	The identifier of the replica.
--continuation-token	The continuation token parameter is used to obtain next set of results. A continuation token with a non empty value is included in the response of the API when the results from the system do not fit in a single response. When this value is passed to the next API call, the API returns next set of results. If there are no further results then the continuation token does not contain a value. The value of this parameter should not be URL encoded.
--timeout -t	Server timeout in seconds. Default: 60.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl replica list

Gets the information about replicas of a Service Fabric service partition.

The GetReplicas endpoint returns information about the replicas of the specified partition. The responses include the ID, role, status, health, node name, uptime, and other details about the replica.

### Arguments

ARGUMENT	DESCRIPTION
--partition-id [Required]	The identity of the partition.
--continuation-token	The continuation token parameter is used to obtain next set of results. A continuation token with a non empty value is included in the response of the API when the results from the system do not fit in a single response. When this value is passed to the next API call, the API returns next set of results. If there are no further results then the continuation token does not contain a value. The value of this parameter should not be URL encoded.
--timeout -t	Server timeout in seconds. Default: 60.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl replica remove

Removes a service replica running on a node.

This API simulates a Service Fabric replica failure by removing a replica from a Service Fabric cluster. The removal closes the replica, transitions the replica to the role None, and then removes all of the state information of the replica from the cluster. This API tests the replica state removal path, and simulates the report fault permanent path through client APIs. Warning - There are no safety checks performed when this API is used. Incorrect use of this API can lead to data loss for stateful services. In addition, the forceRemove flag impacts all other replicas hosted in the same process.

### Arguments

ARGUMENT	DESCRIPTION
--node-name [Required]	The name of the node.

ARGUMENT	DESCRIPTION
--partition-id [Required]	The identity of the partition.
--replica-id [Required]	The identifier of the replica.
--force-remove	Remove a Service Fabric application or service forcefully without going through the graceful shutdown sequence. This parameter can be used to forcefully delete an application or service for which delete is timing out due to issues in the service code that prevents graceful close of replicas.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl replica restart

Restarts a service replica of a persisted service running on a node.

Restarts a service replica of a persisted service running on a node. Warning - There are no safety checks performed when this API is used. Incorrect use of this API can lead to availability loss for stateful services.

## Arguments

ARGUMENT	DESCRIPTION
--node-name [Required]	The name of the node.
--partition-id [Required]	The identity of the partition.
--replica-id [Required]	The identifier of the replica.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.

ARGUMENT	DESCRIPTION
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## Next steps

- [Setup](#) the Service Fabric CLI.
- Learn how to use the Service Fabric CLI using the [sample scripts](#).

# sfctl rpm

9/27/2017 • 1 min to read • [Edit Online](#)

Query and send commands to the repair manager service.

## Commands

COMMAND	DESCRIPTION
approve-force	Forces the approval of the given repair task.
delete	Deletes a completed repair task.
list	Gets a list of repair tasks matching the given filters.

## sfctl rpm delete

Deletes a completed repair task.

This API supports the Service Fabric platform; it is not meant to be used directly from your code.

### Arguments

ARGUMENT	DESCRIPTION
--task-id [Required]	The ID of the completed repair task to be deleted.
--version	The current version number of the repair task. If non-zero, then the request will only succeed if this value matches the actual current version of the repair task. If zero, then no version check is performed.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl rpm list

Gets a list of repair tasks matching the given filters.

This API supports the Service Fabric platform; it is not meant to be used directly from your code.

## Arguments

ARGUMENT	DESCRIPTION
--executor-filter	The name of the repair executor whose claimed tasks should be included in the list.
--state-filter	A bitwise-OR of the following values, specifying which task states should be included in the result list. - 1 - Created - 2 - Claimed - 4 - Preparing - 8 - Approved - 16 - Executing - 32 - Restoring - 64 - Completed.
--task-id-filter	The repair task ID prefix to be matched.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## Next steps

- [Set up](#) the Service Fabric CLI.
- Learn how to use the Service Fabric CLI using the [sample scripts](#).

# sfctl service

9/27/2017 • 16 min to read • [Edit Online](#)

Create, delete and manage service, service types and service packages.

## Commands

COMMAND	DESCRIPTION
app-name	Gets the name of the Service Fabric application for a service.
code-package-list	Gets the list of code packages deployed on a Service Fabric node.
create	Creates the specified Service Fabric service from the description.
delete	Deletes an existing Service Fabric service.
deployed-type	Gets the information about a specified service type of the application deployed on a node in a Service Fabric cluster.
deployed-type-list	Gets the list containing the information about service types from the applications deployed on a node in a Service Fabric cluster.
description	Gets the description of an existing Service Fabric service.
health	Gets the health of the specified Service Fabric service.
info	Gets the information about the specific service belonging to a Service Fabric application.
list	Gets the information about all services belonging to the application specified by the application ID.
manifest	Gets the manifest describing a service type.
package-deploy	Downloads packages associated with specified service manifest to the image cache on specified node.
package-health	Gets the information about health of a service package for a specific application deployed for a Service Fabric node and application.
package-info	Gets the list of service packages deployed on a Service Fabric node matching exactly the specified name.
package-list	Gets the list of service packages deployed on a Service Fabric node.

COMMAND	DESCRIPTION
recover	Indicates to the Service Fabric cluster that it should attempt to recover the specified service, which is currently stuck in quorum loss.
report-health	Sends a health report on the Service Fabric service.
resolve	Resolve a Service Fabric partition.
type-list	Gets the list containing the information about service types that are supported by a provisioned application type in a Service Fabric cluster.
update	Updates the specified service using the given update description.

## sfctl service create

Creates the specified Service Fabric service from the description.

### Arguments

ARGUMENT	DESCRIPTION
--app-id [Required]	The identity of the parent application. This is typically the full ID of the application without the 'fabric:' URI scheme. Starting from version 6.0, hierarchical names are delimited with the '~' character. For example, if the application name is 'fabric://myapp/app1', the application identity would be 'myapp~app1' in 6.0+ and 'myapp/app1' in previous versions.
--name [Required]	Name of the service. This should be a child of the application ID. This is the full name including the fabric: URI. For example service fabric:/A/B is a child of application fabric:/A .
--service-type [Required]	Name of the service type.
--activation-mode	The activation mode for the service package.
--constraints	The placement constraints as a string. Placement constraints are boolean expressions on node properties and allow for restricting a service to particular nodes based on the service requirements. For example, to place a service on nodes where NodeType is blue specify the following:"NodeType == blue".
--correlated-service	Name of the target service to correlate with.
--correlation	Correlate the service with an existing service using an alignment affinity.
--dns-name	The DNS name of the service to be created. The Service Fabric DNS system service must be enabled for this setting.
--instance-count	The instance count. This applies to stateless services only.

ARGUMENT	DESCRIPTION
--int-scheme	Indicates the service should be uniformly partitioned across a range of unsigned integers.
--int-scheme-count	The number of partitions inside the integer key range (for a uniform integer partition scheme) to create.
--int-scheme-high	The end of the key integer range, if using a uniform integer partition scheme.
--int-scheme-low	The start of the key integer range, if using a uniform integer partition scheme.
--load-metrics	JSON encoded list of metrics used when load balancing services across nodes.
--min-replica-set-size	The minimum replica set size as a number. This applies to stateful services only.
--move-cost	Specifies the move cost for the service. Possible values are: 'Zero', 'Low', 'Medium', 'High'.
--named-scheme	Indicates the service should have multiple named partitions.
--named-scheme-list	JSON encoded list of names to partition the service across, if using the named partition scheme.
--no-persisted-state	If true, this indicates the service has no persistent state stored on the local disk, or it only stores state in memory.
--placement-policy-list	JSON encoded list of placement policies for the service, and any associated domain names. Policies can be one or more of: <code>NonPartiallyPlaceService</code> , <code>PreferPrimaryDomain</code> , <code>RequireDomain</code> , <code>RequireDomainDistribution</code> .
--quorum-loss-wait	The maximum duration, in seconds, for which a partition is allowed to be in a state of quorum loss. This applies to stateful services only.
--replica-restart-wait	The duration, in seconds, between when a replica goes down and when a new replica is created. This applies to stateful services only.
--singleton-scheme	Indicates the service should have a single partition or be a non- partitioned service.
--stand-by-replica-keep	The maximum duration, in seconds, for which StandBy replicas are maintained before being removed. This applies to stateful services only.
--stateful	Indicates the service is a stateful service.
--stateless	Indicates the service is a stateless service.

ARGUMENT	DESCRIPTION
--target-replica-set-size	The target replica set size as a number. This applies to stateful services only.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl service delete

Deletes an existing Service Fabric service.

Deletes an existing Service Fabric service. A service must be created before it can be deleted. By default Service Fabric tries to close service replicas in a graceful manner and then delete the service. However if service is having issues closing the replica gracefully, the delete operation may take a long time or get stuck. Use the optional ForceRemove flag to skip the graceful close sequence and forcefully delete the service.

### Arguments

ARGUMENT	DESCRIPTION
--service-id [Required]	The identity of the service. This is typically the full name of the service without the 'fabric:' URI scheme. Starting from version 6.0, hierarchical names are delimited with the "~" character. For example, if the service name is fabric://myapp/app1/svc1", the service identity would be "myapp~app1~svc1" in 6.0+ and "myapp/app1/svc1" in previous versions.
--force-remove	Remove a Service Fabric application or service forcefully without going through the graceful shutdown sequence. This parameter can be used to forcefully delete an application or service for which delete is timing out due to issues in the service code that prevents graceful close of replicas.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.

ARGUMENT	DESCRIPTION
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl service description

Gets the description of an existing Service Fabric service.

Gets the description of an existing Service Fabric service. A service must be created before its description can be obtained.

### Arguments

ARGUMENT	DESCRIPTION
--service-id [Required]	The identity of the service. This is typically the full name of the service without the 'fabric:' URL scheme. Starting from version 6.0, hierarchical names are delimited with the "~" character. For example, if the service name is "fabric://myapp/app1/svc1", the service identity would be "myapp~app1~svc1" in 6.0+ and "myapp/app1/svc1" in previous versions.
--timeout -t	Server timeout in seconds. Default: 60.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl service health

Gets the health of the specified Service Fabric service.

Gets the health information of the specified service. Use EventsHealthStateFilter to filter the collection of health

events reported on the service based on the health state. Use PartitionsHealthStateFilter to filter the collection of partitions returned. If you specify a service that does not exist in the health store, this cmdlet returns an error.

## Arguments

ARGUMENT	DESCRIPTION
--service-id [Required]	The identity of the service. This is typically the full name of the service without the 'fabric:' URI scheme. Starting from version 6.0, hierarchical names are delimited with the "~" character. For example, if the service name is "fabric://myapp/app1/svc1", the service identity would be "myapp~app1~svc1" in 6.0+ and "myapp/app1/svc1" in previous versions.
--events-health-state-filter	Allows filtering the collection of HealthEvent objects returned based on health state. The possible values for this parameter include integer value of one of the following health states. Only events that match the filter are returned. All events are used to evaluate the aggregated health state. If not specified, all entries are returned. The state values are flag-based enumeration, so the value could be a combination of these values obtained using bitwise 'OR' operator. For example, If the provided value is 6 then all of the events with HealthState value of OK (2) and Warning (4) are returned. - Default - Default value. Matches any HealthState. The value is zero. - None - Filter that doesn't match any HealthState value. Used in order to return no results on a given collection of states. The value is 1. - Ok - Filter that matches input with HealthState value Ok. The value is 2. - Warning - Filter that matches input with HealthState value Warning. The value is 4. - Error - Filter that matches input with HealthState value Error. The value is 8. - All - Filter that matches input with any HealthState value. The value is 65535.
--exclude-health-statistics	Indicates whether the health statistics should be returned as part of the query result. False by default. The statistics show the number of children entities in health state Ok, Warning, and Error.
--partitions-health-state-filter	Allows filtering of the partitions health state objects returned in the result of service health query based on their health state. The possible values for this parameter include integer value of one of the following health states. Only partitions that match the filter are returned. All partitions are used to evaluate the aggregated health state. If not specified, all entries are returned. The state values are flag-based enumeration, so the value could be a combination of these values obtained using bitwise 'OR' operator. For example, if the provided value is "6" then health state of partitions with HealthState value of OK (2) and Warning (4) are returned. - Default - Default value. Matches any HealthState. The value is zero. - None - Filter that doesn't match any HealthState value. Used in order to return no results on a given collection of states. The value is 1. - Ok - Filter that matches input with HealthState value Ok. The value is 2. - Warning - Filter that matches input with HealthState value Warning. The value is 4. - Error - Filter that matches input with HealthState value Error. The value is 8. - All - Filter that matches input with any HealthState value. The value is 65535.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl service info

Gets the information about the specific service belonging to a Service Fabric application.

Returns the information about specified service belonging to the specified Service Fabric application.

### Arguments

ARGUMENT	DESCRIPTION
--application-id [Required]	The identity of the application. This is typically the full name of the application without the 'fabric:' URI scheme. Starting from version 6.0, hierarchical names are delimited with the "~" character. For example, if the application name is "fabric://myapp/app1", the application identity would be "myapp~app1" in 6.0+ and "myapp/app1" in previous versions.
--service-id [Required]	The identity of the service. This is typically the full name of the service without the 'fabric:' URI scheme. Starting from version 6.0, hierarchical names are delimited with the "~" character. For example, if the service name is "fabric://myapp/app1/svc1", the service identity would be "myapp~app1~svc1" in 6.0+ and "myapp/app1/svc1" in previous versions.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .

ARGUMENT	DESCRIPTION
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl service list

Gets the information about all services belonging to the application specified by the application ID.

Returns the information about all services belonging to the application specified by the application ID.

### Arguments

ARGUMENT	DESCRIPTION
--application-id [Required]	The identity of the application. This is typically the full name of the application without the 'fabric' URI scheme. Starting from version 6.0, hierarchical names are delimited with the "~" character. For example, if the application name is "fabric://myapp/app1", the application identity would be "myapp~app1" in 6.0+ and "myapp/app1" in previous versions.
--continuation-token	The continuation token parameter is used to obtain next set of results. A continuation token with a non empty value is included in the response of the API when the results from the system do not fit in a single response. When this value is passed to the next API call, the API returns next set of results. If there are no further results, then the continuation token does not contain a value. The value of this parameter should not be URL encoded.
--service-type-name	The service type name used to filter the services to query for.
--timeout -t	Server timeout in seconds. Default: 60.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl service manifest

Gets the manifest describing a service type.

Gets the manifest describing a service type. The response contains the service manifest XML as a string.

## Arguments

ARGUMENT	DESCRIPTION
--application-type-name [Required]	The name of the application type.
--application-type-version [Required]	The version of the application type.
--service-manifest-name [Required]	The name of a service manifest registered as part of an application type in a Service Fabric cluster.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl service recover

Indicates to the Service Fabric cluster that it should attempt to recover the specified service, which is currently stuck in quorum loss.

Indicates to the Service Fabric cluster that it should attempt to recover the specified service, which is currently stuck in quorum loss. This operation should only be performed if the replicas that are down cannot be recovered.

Incorrect use of this API can cause potential data loss.

## Arguments

ARGUMENT	DESCRIPTION
--service-id [Required]	The identity of the service. This is typically the full name of the service without the 'fabric:' URI scheme. Starting from version 6.0, hierarchical names are delimited with the "~" character. For example, if the service name is fabric://myapp/app1/svc1", the service identity would be "myapp~app1~svc1" in 6.0+ and "myapp/app1/svc1" in previous versions.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl service resolve

Resolve a Service Fabric partition.

Resolve a Service Fabric service partition, to get the endpoints of the service replicas.

### Arguments

ARGUMENT	DESCRIPTION
--service-id [Required]	The identity of the service. This is typically the full name of the service without the 'fabric:' URI scheme. Starting from version 6.0, hierarchical names are delimited with the "~" character. For example, if the service name is "fabric://myapp/app1/svc1", the service identity would be "myapp~app1~svc1" in 6.0+ and "myapp/app1/svc1" in previous versions.
--partition-key-type	Key type for the partition. This parameter is required if the partition scheme for the service is Int64Range or Named. The possible values are following. - None (1) - Indicates that the PartitionKeyValue parameter is not specified. This is valid for the partitions with partitioning scheme as Singleton. This is the default value. The value is 1. - Int64Range (2) - Indicates that the PartitionKeyValue parameter is an int64 partition key. This is valid for the partitions with partitioning scheme as Int64Range. The value is 2. - Named (3) - Indicates that the PartitionKeyValue parameter is a name of the partition. This is valid for the partitions with partitioning scheme as Named. The value is 3.
--partition-key-value	Partition key. This is required if the partition scheme for the service is Int64Range or Named.
--previous-rsp-version	The value in the Version field of the response that was received previously. This is required if the user knows that the result that was got previously is stale.
--timeout -t	Server timeout in seconds. Default: 60.

### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## sfctl service update

Updates the specified service using the given update description.

### Arguments

ARGUMENT	DESCRIPTION
--service-id [Required]	Target service to update. This is typically the full ID of the service without the 'fabric:' URL scheme. Starting from version 6.0, hierarchical names are delimited with the "~" character. For example, if the service name is 'fabric://myapp/app1/svc1', the service identity would be 'myapp~app1~svc1' in 6.0+ and 'myapp/app1/svc1' in previous versions.
--constraints	The placement constraints as a string. Placement constraints are boolean expressions on node properties and allow for restricting a service to particular nodes based on the service requirements. For example, to place a service on nodes where NodeType is blue specify the following: "NodeColor == blue".
--correlated-service	Name of the target service to correlate with.
--correlation	Correlate the service with an existing service using an alignment affinity.
--instance-count	The instance count. This applies to stateless services only.
--load-metrics	JSON encoded list of metrics used when load balancing across nodes.
--min-replica-set-size	The minimum replica set size as a number. This applies to stateful services only.
--move-cost	Specifies the move cost for the service. Possible values are: 'Zero', 'Low', 'Medium', 'High'.
--placement-policy-list	JSON encoded list of placement policies for the service, and any associated domain names. Policies can be one or more of: <div style="border: 1px solid black; padding: 2px;"><code>NonPartiallyPlaceService</code></div> <div style="border: 1px solid black; padding: 2px;"><code>PreferPrimaryDomain</code></div> <div style="border: 1px solid black; padding: 2px;"><code>RequireDomain</code></div> <div style="border: 1px solid black; padding: 2px;"><code>RequireDomainDistribution</code></div>

ARGUMENT	DESCRIPTION
--quorum-loss-wait	The maximum duration, in seconds, for which a partition is allowed to be in a state of quorum loss. This applies to stateful services only.
--replica-restart-wait	The duration, in seconds, between when a replica goes down and when a new replica is created. This applies to stateful services only.
--stand-by-replica-keep	The maximum duration, in seconds, for which StandBy replicas are maintained before being removed. This applies to stateful services only.
--stateful	Indicates the target service is a stateful service.
--stateless	Indicates the target service is a stateless service.
--target-replica-set-size	The target replica set size as a number. This applies to stateful services only.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## Next steps

- [Set up](#) the Service Fabric CLI.
- Learn how to use the Service Fabric CLI using the [sample scripts](#).

# sfctl store

9/26/2017 • 1 min to read • [Edit Online](#)

Perform basic file level operations on the cluster image store.

## Commands

COMMAND	DESCRIPTION
delete	Deletes existing image store content.
root-info	Gets the content information at the root of the image store.
stat	Gets the image store content information.

### sfctl store delete

Deletes existing image store content.

Deletes existing image store content being found within the given image store relative path. This command can be used to delete uploaded application packages once they are provisioned.

#### Arguments

ARGUMENT	DESCRIPTION
--content-path [Required]	Relative path to file or folder in the image store from its root.
--timeout -t	Server timeout in seconds. Default: 60.

#### Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. For more information and examples, see <a href="http://jmespath.org/">http://jmespath.org/</a> .
--verbose	Increase logging verbosity. Use --debug for full debug logs.

### sfctl store stat

Gets the image store content information.

Returns the information about the image store content at the specified contentPath relative to the root of the

image store.

## Arguments

ARGUMENT	DESCRIPTION
--content-path [Required]	Relative path to file or folder in the image store from its root.
--timeout -t	Server timeout in seconds. Default: 60.

## Global Arguments

ARGUMENT	DESCRIPTION
--debug	Increase logging verbosity to show all debug logs.
--help -h	Show this help message and exit.
--output -o	Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query	JMESPath query string. See <a href="http://jmespath.org/">http://jmespath.org/</a> for more information and examples.
--verbose	Increase logging verbosity. Use --debug for full debug logs.

## Next steps

- [Setup](#) the Service Fabric CLI.
- Learn how to use the Service Fabric CLI using the [sample scripts](#).

# ServiceFabricServiceModel.xsd schema documentation

10/27/2017 • 185 min to read • [Edit Online](#)

This article documents the ServiceFabricServiceModel.xsd schema file installed with the Service Fabric SDK.

## Application Element

Application Instance specific information like application name and application parameter values used to create application. Parameter values in this file overrides the default parameter values defined in Application Manifest.

ATTRIBUTE	VALUE
type	AppInstanceDefinitionType
content	0 element(s), 0 attribute(s)
name	Application

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="Application" type="AppInstanceDefinitionType">
  <xs:annotation>
    <xs:documentation>Application Instance specific information like application name and
    application parameter values used to create application. Parameter values in this file overrides the default
    parameter values defined in Application Manifest.</xs:documentation>
  </xs:annotation>
</xs:element>
```

## ApplicationEndpoints Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	ApplicationEndpoints
minOccurs	0

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ApplicationEndpoints" minOccurs="0">
  <xss:complexType>
    <xss:attribute name="StartPort" type="xs:int" use="required"/>
    <xss:attribute name="EndPort" type="xs:int" use="required"/>
  </xss:complexType>
</xss:element>

```

## Attribute details

### StartPort

ATTRIBUTE	VALUE
name	StartPort
type	xs:int
use	required

### EndPort

ATTRIBUTE	VALUE
name	EndPort
type	xs:int
use	required

## ApplicationInstance Element

Describes an instance of a Microsoft Azure Service Fabric application.

ATTRIBUTE	VALUE
type	ApplicationInstanceType
content	0 element(s), 0 attribute(s)
name	ApplicationInstance

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ApplicationInstance" type="ApplicationInstanceType">
  <xss:annotation>
    <xss:documentation>Describes an instance of a Microsoft Azure Service Fabric application.
  </xss:documentation>
  </xss:annotation>
</xss:element>

```

## ApplicationManifest Element

ATTRIBUTE	VALUE
type	ApplicationManifestType
content	0 element(s), 0 attribute(s)
name	ApplicationManifest

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ApplicationManifest" type="ApplicationManifestType"/>
```

## ApplicationPackage Element

ApplicationPackage represents the versioned Application information required by the node.

ATTRIBUTE	VALUE
type	ApplicationPackageType
content	0 element(s), 0 attribute(s)
name	ApplicationPackage

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ApplicationPackage" type="ApplicationPackageType">
  <xs:annotation>
    <xs:documentation>ApplicationPackage represents the versioned Application information required by the
node.</xs:documentation>
  </xs:annotation>
</xs:element>
```

## ApplicationPackageRef Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	ApplicationPackageRef

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ApplicationPackageRef">
  <xs:complexType>
    <xs:attributeGroup ref="VersionedItemAttrGroup"/>
  </xs:complexType>
</xs:element>
```

## Arguments Element

ATTRIBUTE	VALUE
type	xs:string
content	0 element(s), 0 attribute(s)
name	Arguments
minOccurs	0

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="Arguments" type="xs:string" minOccurs="0"/>
```

## AzureBlob Element

ATTRIBUTE	VALUE
type	AzureBlobType
content	0 element(s), 0 attribute(s)
name	AzureBlob
minOccurs	0
maxOccurs	unbounded

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="AzureBlob" type="AzureBlobType" minOccurs="0" maxOccurs="unbounded"/>
```

## AzureBlob Element

ATTRIBUTE	VALUE
type	AzureBlobETWType
content	0 element(s), 0 attribute(s)
name	AzureBlob
minOccurs	0
maxOccurs	unbounded

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="AzureBlob" type="AzureBlobETWType" minOccurs="0" maxOccurs="unbounded"/>
```

## AzureBlob Element

ATTRIBUTE	VALUE
type	AzureBlobType
content	0 element(s), 0 attribute(s)
name	AzureBlob
minOccurs	0
maxOccurs	unbounded

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="AzureBlob" type="AzureBlobType" minOccurs="0" maxOccurs="unbounded"/>
```

## BackupRestoreServiceReplicatorEndpoint Element

ATTRIBUTE	VALUE
type	InternalEndpointType
content	0 element(s), 0 attribute(s)
name	BackupRestoreServiceReplicatorEndpoint
minOccurs	0

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="BackupRestoreServiceReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
```

## Blackbird Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	Blackbird

### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Blackbird">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Roles">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element
name="Role" type="BlackbirdRoleType" minOccurs="1" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>

```

## Content element details

### Roles

ATTRIBUTE	VALUE
name	Roles

## Capacities Element

The capacities of various metrics for this node type

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	Capacities
minOccurs	0

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Capacities" minOccurs="0">
    <xs:annotation>
        <xs:documentation>The
capacities of various metrics for this node type</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element
name="Capacity" type="KeyValuePairType" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

```

## Content element details

### Capacity

ATTRIBUTE	VALUE
name	Capacity
type	KeyValuePairType
minOccurs	0
maxOccurs	unbounded

## Capacity Element

ATTRIBUTE	VALUE
type	KeyValuePairType
content	0 element(s), 0 attribute(s)
name	Capacity
minOccurs	0
maxOccurs	unbounded

### XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="Capacity" type="KeyValuePairType" minOccurs="0" maxOccurs="unbounded"/>
```

## CertificateRef Element

Specifies information for a certificate which will be exposed to the container.

ATTRIBUTE	VALUE
type	ContainerCertificateType
content	0 element(s), 0 attribute(s)
name	CertificateRef
minOccurs	0
maxOccurs	unbounded

### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="CertificateRef" type="ContainerCertificateType" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>Specifies information for a certificate which will be exposed to the container.
    </xs:documentation>
    </xs:annotation>
</xs:element>

```

## Certificates Element

Describe the certificates associated with this node type

ATTRIBUTE	VALUE
type	<a href="#">CertificatesType</a>
content	0 element(s), 0 attribute(s)
name	Certificates
minOccurs	0

### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Certificates" type="CertificatesType" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Describe the certificates associated with this node type</xs:documentation>
    </xs:annotation>
</xs:element>

```

## Certificates Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	Certificates
minOccurs	0

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Certificates" minOccurs="0">
    <xss:complexType>
        <xss:sequence>
            <xss:element name="SecretsCertificate"
type="FabricCertificateType" minOccurs="0"/>
        </xss:sequence>
    </xss:complexType>
</xss:element>

```

## Content element details

### SecretsCertificate

ATTRIBUTE	VALUE
name	SecretsCertificate
type	FabricCertificateType
minOccurs	0

## Certificates Element

Declares certificates used to secure endpoints or encrypt secrets within the application manifest or a cluster manifest.

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 0 attribute(s)
name	Certificates
minOccurs	0

## XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Certificates" minOccurs="0">
    <xss:annotation>
        <xss:documentation>Declares certificates used to secure endpoints or encrypt secrets within the
application manifest or a cluster manifest.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence maxOccurs="unbounded">
            <xss:element name="SecretsCertificate" type="FabricCertificateType" minOccurs="0">
                <xss:annotation>
                    <xss:documentation>Declares a certificate used to encrypt sensitive information within the
application manifest. The application author uses the Invoke-ServiceFabricEncryptSecret cmdlet to encrypt the
sensitive information, which is copied to a Parameter in the ConfigOverrides section.</xss:documentation>
                </xss:annotation>
            </xss:element>
            <xss:element name="EndpointCertificate" type="EndpointCertificateType" minOccurs="0"/>
        </xss:sequence>
    </xss:complexType>
</xss:element>

```

## Content element details

### SecretsCertificate

Declares a certificate used to encrypt sensitive information within the application manifest. The application author uses the Invoke-ServiceFabricEncryptSecret cmdlet to encrypt the sensitive information, which is copied to a Parameter in the ConfigOverrides section.

ATTRIBUTE	VALUE
name	SecretsCertificate
type	FabricCertificateType
minOccurs	0

### EndpointCertificate

ATTRIBUTE	VALUE
name	EndpointCertificate
type	EndpointCertificateType
minOccurs	0

## Certificates Element

Describe the certificates associated with this node type

ATTRIBUTE	VALUE
type	CertificatesType
content	0 element(s), 0 attribute(s)
name	Certificates
minOccurs	0

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Certificates" type="CertificatesType" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Describe the certificates associated with this node type</xs:documentation>
    </xs:annotation>
</xs:element>
```

## ClientCertificate Element

The default admin role client certificate used to secure client server communication.

ATTRIBUTE	VALUE
type	FabricCertificateType
content	0 element(s), 0 attribute(s)
name	ClientCertificate
minOccurs	0

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ClientCertificate" type="FabricCertificateType" minOccurs="0">
    <xs:annotation>
        <xs:documentation>The default admin role client certificate used to secure client server
communication.</xs:documentation>
    </xs:annotation>
</xs:element>
```

## ClientConnectionEndpoint Element

ATTRIBUTE	VALUE
type	InputEndpointType
content	0 element(s), 0 attribute(s)
name	ClientConnectionEndpoint

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ClientConnectionEndpoint" type="InputEndpointType"/>
```

## ClusterCertificate Element

The certificate used to secure the intra cluster communication.

ATTRIBUTE	VALUE
type	FabricCertificateType
content	0 element(s), 0 attribute(s)
name	ClusterCertificate
minOccurs	0

#### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ClusterCertificate" type="FabricCertificateType" minOccurs="0">
    <xss:annotation>
        <xss:documentation>The certificate used to secure the intra cluster communication.</xss:documentation>
    </xss:annotation>
</xss:element>

```

## ClusterConnectionEndpoint Element

ATTRIBUTE	VALUE
type	InternalEndpointType
content	0 element(s), 0 attribute(s)
name	ClusterConnectionEndpoint

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ClusterConnectionEndpoint" type="InternalEndpointType"/>

```

## ClusterManagerReplicatorEndpoint Element

ATTRIBUTE	VALUE
type	InternalEndpointType
content	0 element(s), 0 attribute(s)
name	ClusterManagerReplicatorEndpoint
minOccurs	0

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ClusterManagerReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>

```

## ClusterManifest Element

Describes a Microsoft Azure Service Fabric Cluster.

ATTRIBUTE	VALUE
type	ClusterManifestType
content	0 element(s), 0 attribute(s)
name	ClusterManifest

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ClusterManifest" type="ClusterManifestType">
    <xs:annotation>
        <xs:documentation>Describes a Microsoft Azure Service Fabric Cluster.
    </xs:documentation>
</xs:element>
```

## CodePackage Element

ATTRIBUTE	VALUE
type	CodePackageType
content	0 element(s), 0 attribute(s)
name	CodePackage
maxOccurs	unbounded

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="CodePackage" type="CodePackageType" maxOccurs="unbounded"/>
```

## CodePackage Element

ATTRIBUTE	VALUE
type	CodePackageType
content	0 element(s), 0 attribute(s)
name	CodePackage

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="CodePackage" type="CodePackageType"/>
```

## Commands Element

ATTRIBUTE	VALUE
type	xs:string
content	0 element(s), 0 attribute(s)
name	Commands

ATTRIBUTE	VALUE
minOccurs	0
maxOccurs	1

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Commands" type="xs:string" minOccurs="0" maxOccurs="1"/>
```

## ConfigOverride Element

ATTRIBUTE	VALUE
type	ConfigOverrideType
content	0 element(s), 0 attribute(s)
name	ConfigOverride
minOccurs	0
maxOccurs	unbounded

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ConfigOverride" type="ConfigOverrideType" minOccurs="0" maxOccurs="unbounded"/>
```

## ConfigOverride Element

ATTRIBUTE	VALUE
type	ConfigOverrideType
content	0 element(s), 0 attribute(s)
name	ConfigOverride
minOccurs	0

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ConfigOverride" type="ConfigOverrideType" minOccurs="0"/>
```

## ConfigOverrides Element

Describes configuration overrides for the imported service manifest. Configuration overrides allow the flexibility of re-using the same service manifests across multiple application types by overriding the service manifest's

configuration only when used with a particular application type. Configuration overrides can change any default configuration in a service manifest as long as default configuration is defined using the Settings.xml in the ConfigPackage folder.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	ConfigOverrides
minOccurs	0

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ConfigOverrides" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Describes configuration overrides for the imported service manifest.
Configuration overrides allow the flexibility of re-using the same service manifests across multiple
application types by overriding the service manifest's configuration only when used with a particular
application type. Configuration overrides can change any default configuration in a service manifest as long as
default configuration is defined using the Settings.xml in the ConfigPackage folder. </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="ConfigOverride" type="ConfigOverrideType" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

## Content element details

### ConfigOverride

ATTRIBUTE	VALUE
name	ConfigOverride
type	ConfigOverrideType
minOccurs	0
maxOccurs	unbounded

## ConfigPackage Element

ATTRIBUTE	VALUE
type	ConfigPackageType
content	0 element(s), 0 attribute(s)
name	ConfigPackage

ATTRIBUTE	VALUE
minOccurs	0
maxOccurs	unbounded

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ConfigPackage" type="ConfigPackageType" minOccurs="0" maxOccurs="unbounded"/>
```

## ConfigPackage Element

ATTRIBUTE	VALUE
type	ConfigPackageType
content	0 element(s), 0 attribute(s)
name	ConfigPackage

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ConfigPackage" type="ConfigPackageType"/>
```

## ConsoleRedirection Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	ConsoleRedirection
minOccurs	0

#### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ConsoleRedirection" minOccurs="0">
  <xss:complexType>
    <xss:attribute name="FileRetentionCount" default="2">
      <xss:simpleType>
        <xss:restriction base="xs:int">
          <xss:minInclusive value="1"/>
        </xss:restriction>
      </xss:simpleType>
    </xss:attribute>
    <xss:attribute name="FileMaxSizeInKb" default="20480">
      <xss:simpleType>
        <xss:restriction base="xs:int">
          <xss:minInclusive value="128"/>
        </xss:restriction>
      </xss:simpleType>
    </xss:attribute>
  </xss:complexType>
</xss:element>

```

## Attribute details

### FileRetentionCount

ATTRIBUTE	VALUE
name	FileRetentionCount
default	2

### FileMaxSizeInKb

ATTRIBUTE	VALUE
name	FileMaxSizeInKb
default	20480

## ContainerEntryPoint Element

Overidden entrypoint for containers so debugger can be launched..

ATTRIBUTE	VALUE
type	xs:string
content	0 element(s), 0 attribute(s)
name	ContainerEntryPoint
minOccurs	0
maxOccurs	unbounded

### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ContainerEntryPoint" type="xs:string" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>Overidden entrypoint for containers so debugger can be launched..
    </xs:documentation>
</xs:annotation>
</xs:element>

```

## ContainerEnvironmentBlock Element

EnvironmentBlock for containers.

ATTRIBUTE	VALUE
type	xs:string
content	0 element(s), 0 attribute(s)
name	ContainerEnvironmentBlock
minOccurs	0
maxOccurs	unbounded

### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ContainerEnvironmentBlock" type="xs:string" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>EnvironmentBlock for containers.</xs:documentation>
    </xs:annotation>
</xs:element>

```

## ContainerHost Element

ATTRIBUTE	VALUE
type	ContainerHostEntryPointType
content	0 element(s), 0 attribute(s)
name	ContainerHost

### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ContainerHost" type="ContainerHostEntryPointType"/>

```

## ContainerHostPolicies Element

Specifies policies for activating container hosts.

ATTRIBUTE	VALUE
type	ContainerHostPoliciesType
content	0 element(s), 0 attribute(s)
name	ContainerHostPolicies
minOccurs	0

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ContainerHostPolicies" type="ContainerHostPoliciesType" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Specifies policies for activating container hosts.</xs:documentation>
    </xs:annotation>
</xs:element>
```

## ContainerHostPolicies Element

Specifies policies for activating container hosts.

ATTRIBUTE	VALUE
type	ContainerHostPoliciesType
content	0 element(s), 0 attribute(s)
name	ContainerHostPolicies
minOccurs	0

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ContainerHostPolicies" type="ContainerHostPoliciesType" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Specifies policies for activating container hosts.</xs:documentation>
    </xs:annotation>
</xs:element>
```

## ContainerMountedVolume Element

Volumes to be mounted inside container.

ATTRIBUTE	VALUE
type	xs:string
content	0 element(s), 0 attribute(s)
name	ContainerMountedVolume

ATTRIBUTE	VALUE
minOccurs	0
maxOccurs	unbounded

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ContainerMountedVolume" type="xs:string" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>Volumes to be mounted inside container.</xs:documentation>
    </xs:annotation>
</xs:element>
```

## CrashDumpSource Element

Specifies crash dump collection. Crash dumps are collected for executables that host the code packages of all services belonging to the application.

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 1 attribute(s)
name	CrashDumpSource
minOccurs	0

#### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="CrashDumpSource" minOccurs="0">
    <xss:annotation>
        <xss:documentation>Specifies crash dump collection. Crash dumps are collected for executables that
host the code packages of all services belonging to the application.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence>
            <xss:element name="Destinations" minOccurs="0">
                <xss:annotation>
                    <xss:documentation>Destinations to which the crash dumps need to be transferred.</xss:documentation>
                </xss:annotation>
            </xss:element>
            <xss:element ref="Parameters" minOccurs="0" maxOccurs="1"/>
        </xss:sequence>
        <xss:attribute name="IsEnabled" type="xs:string">
            <xss:annotation>
                <xss:documentation>Whether or not crash dump collection is enabled. By default, it is not enabled.</xss:documentation>
            </xss:annotation>
        </xss:attribute>
    </xss:complexType>
</xss:element>

```

## Attribute details

### .IsEnabled

ATTRIBUTE	VALUE
name	IsEnabled
type	xs:string

## Content element details

### Destinations

Destinations to which the crash dumps need to be transferred.

ATTRIBUTE	VALUE
name	Destinations
minOccurs	0

### None

ATTRIBUTE	VALUE
ref	Parameters
minOccurs	0

ATTRIBUTE	VALUE
maxOccurs	1

## CurrentInstallation Element

ATTRIBUTE	VALUE
type	WindowsFabricDeploymentInformation
content	0 element(s), 0 attribute(s)
name	CurrentInstallation
minOccurs	0

### XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="CurrentInstallation" type="WindowsFabricDeploymentInformation" minOccurs="0"/>
```

## DataPackage Element

ATTRIBUTE	VALUE
type	DataPackageType
content	0 element(s), 0 attribute(s)
name	DataPackage
minOccurs	0
maxOccurs	unbounded

### XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DataPackage" type="DataPackageType" minOccurs="0" maxOccurs="unbounded"/>
```

## DataPackage Element

ATTRIBUTE	VALUE
type	DataPackageType
content	0 element(s), 0 attribute(s)
name	DataPackage

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DataPackage" type="DataPackageType"/>
```

## DebugParameters Element

ATTRIBUTE	VALUE
type	DebugParametersType
content	0 element(s), 0 attribute(s)
name	DebugParameters
minOccurs	0
maxOccurs	1

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DebugParameters" type="DebugParametersType" minOccurs="0" maxOccurs="1"/>
```

## DebugParameters Element

ATTRIBUTE	VALUE
type	DebugParametersType
content	0 element(s), 0 attribute(s)
name	DebugParameters
minOccurs	0
maxOccurs	1

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DebugParameters" type="DebugParametersType" minOccurs="0" maxOccurs="1"/>
```

## DebugParameters Element

ATTRIBUTE	VALUE
type	DebugParametersType
content	0 element(s), 0 attribute(s)

ATTRIBUTE	VALUE
name	DebugParameters
minOccurs	0
maxOccurs	1

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DebugParameters" type="DebugParametersType" minOccurs="0" maxOccurs="1"/>
```

## DefaultReplicatorEndpoint Element

ATTRIBUTE	VALUE
type	InternalEndpointType
content	0 element(s), 0 attribute(s)
name	DefaultReplicatorEndpoint
minOccurs	0

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DefaultReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
```

## DefaultRunAsPolicy Element

Specify a default user account for all service code packages that don't have a specific RunAsPolicy defined in the ServiceManifestImport section.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)
name	DefaultRunAsPolicy
minOccurs	0

#### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DefaultRunAsPolicy" minOccurs="0">
    <xss:annotation>
        <xss:documentation>Specify a default user account for all service code
packages that don't have a specific RunAsPolicy defined in the ServiceManifestImport section.
    </xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:attribute name="UserRef" type="xs:string" use="required">
            <xss:annotation>
                <xss:documentation>The user account that the service
code packages will run as. The user account must be declared in the Principals section. Often it is preferable
to run the setup entry point using a local system account rather than an administrators account.
            </xss:documentation>
        </xss:annotation>
        </xss:attribute>
    </xss:complexType>
</xss:element>

```

## Attribute details

### UserRef

ATTRIBUTE	VALUE
name	UserRef
type	xs:string
use	required

## DefaultServiceTypeHealthPolicy Element

Specifies the default service type health policy, which will replace the default health policy for all service types in the application.

ATTRIBUTE	VALUE
type	ServiceTypeHealthPolicyType
content	0 element(s), 0 attribute(s)
name	DefaultServiceTypeHealthPolicy
minOccurs	0

## XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DefaultServiceTypeHealthPolicy" type="ServiceTypeHealthPolicyType" minOccurs="0">
    <xss:annotation>
        <xss:documentation>Specifies the default service type health policy, which will replace the default
health policy for all service types in the application.</xss:documentation>
    </xss:annotation>
</xss:element>

```

## DefaultServices Element

ATTRIBUTE	VALUE
type	DefaultServicesType
content	0 element(s), 0 attribute(s)
name	DefaultServices
minOccurs	0

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DefaultServices" type="DefaultServicesType" minOccurs="0">
</xs:element>
```

## DefaultServices Element

ATTRIBUTE	VALUE
type	DefaultServicesType
content	0 element(s), 0 attribute(s)
name	DefaultServices

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DefaultServices" type="DefaultServicesType"/>
```

## Description Element

Text describing this application.

ATTRIBUTE	VALUE
type	xs:string
content	0 element(s), 0 attribute(s)
name	Description
minOccurs	0

#### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Description" type="xs:string" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Text describing this application.</xs:documentation>
    </xs:annotation>
</xs:element>

```

## Description Element

Text describing this service.

ATTRIBUTE	VALUE
type	xs:string
content	0 element(s), 0 attribute(s)
name	Description
minOccurs	0

### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Description" type="xs:string" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Text describing this service.</xs:documentation>
    </xs:annotation>
</xs:element>

```

## Description Element

ATTRIBUTE	VALUE
type	xs:string
content	0 element(s), 0 attribute(s)
name	Description
minOccurs	0

### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Description" type="xs:string" minOccurs="0"/>

```

## Destinations Element

Destinations to which the crash dumps need to be transferred.

ATTRIBUTE	VALUE
type	anonymous complexType
content	3 element(s), 0 attribute(s)
name	Destinations
minOccurs	0

## XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Destinations" minOccurs="0">
    <xss:annotation>
        <xss:documentation>Destinations to which the crash dumps need to be transferred.
    </xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence>
            <xss:element name="LocalStore" type="LocalStoreType" minOccurs="0" maxOccurs="unbounded"/>
            <xss:element name="FileStore" type="FileStoreType" minOccurs="0" maxOccurs="unbounded"/>
            <xss:element name="AzureBlob" type="AzureBlobType" minOccurs="0" maxOccurs="unbounded"/>
        </xss:sequence>
    </xss:complexType>
</xss:element>

```

## Content element details

### LocalStore

ATTRIBUTE	VALUE
name	LocalStore
type	LocalStoreType
minOccurs	0
maxOccurs	unbounded

### FileStore

ATTRIBUTE	VALUE
name	FileStore
type	FileStoreType
minOccurs	0
maxOccurs	unbounded

### AzureBlob

ATTRIBUTE	VALUE
name	AzureBlob
type	AzureBlobType
minOccurs	0
maxOccurs	unbounded

## Destinations Element

Destinations to which the crash dumps need to be transferred.

ATTRIBUTE	VALUE
type	anonymous complexType
content	3 element(s), 0 attribute(s)
name	Destinations
minOccurs	0

### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Destinations" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Destinations to which the crash dumps need to be transferred.
    </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="LocalStore" type="LocalStoreETWType" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="FileStore" type="FileStoreETWType" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="AzureBlob" type="AzureBlobETWType" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

```

### Content element details

#### LocalStore

ATTRIBUTE	VALUE
name	LocalStore
type	LocalStoreETWType
minOccurs	0
maxOccurs	unbounded

#### FileStore

ATTRIBUTE	VALUE
name	FileStore
type	FileStoreETWType
minOccurs	0
maxOccurs	unbounded

#### AzureBlob

ATTRIBUTE	VALUE
name	AzureBlob
type	AzureBlobETWType
minOccurs	0
maxOccurs	unbounded

## Destinations Element

Destinations to which the folder contents need to be transferred.

ATTRIBUTE	VALUE
type	anonymous complexType
content	3 element(s), 0 attribute(s)
name	Destinations
minOccurs	0

#### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Destinations" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Destinations to which the folder contents need to be transferred.
    </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="LocalStore" type="LocalStoreType" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="FileStore" type="FileStoreType" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="AzureBlob" type="AzureBlobType" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

```

#### Content element details

##### LocalStore

ATTRIBUTE	VALUE
name	LocalStore
type	LocalStoreType
minOccurs	0
maxOccurs	unbounded

#### FileStore

ATTRIBUTE	VALUE
name	FileStore
type	FileStoreType
minOccurs	0
maxOccurs	unbounded

#### AzureBlob

ATTRIBUTE	VALUE
name	AzureBlob
type	AzureBlobType
minOccurs	0
maxOccurs	unbounded

## Diagnostics Element

ATTRIBUTE	VALUE
type	DiagnosticsType
content	0 element(s), 0 attribute(s)
name	Diagnostics
minOccurs	0

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Diagnostics" type="DiagnosticsType" minOccurs="0"/>
```

## Diagnostics Element

ATTRIBUTE	VALUE
type	DiagnosticsType
content	0 element(s), 0 attribute(s)
name	Diagnostics

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Diagnostics" type="DiagnosticsType"/>
```

## Diagnostics Element

ATTRIBUTE	VALUE
type	ServiceDiagnosticsType
content	0 element(s), 0 attribute(s)
name	Diagnostics
minOccurs	0

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Diagnostics" type="ServiceDiagnosticsType" minOccurs="0"/>
```

## Diagnostics Element

ATTRIBUTE	VALUE
type	ServiceDiagnosticsType
content	0 element(s), 0 attribute(s)
name	Diagnostics

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Diagnostics" type="ServiceDiagnosticsType"/>
```

## DigestedCertificates Element

ATTRIBUTE	VALUE
type	anonymous complexType

ATTRIBUTE	VALUE
content	2 element(s), 0 attribute(s)
name	DigestedCertificates

## XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DigestedCertificates">
  <xss:complexType>
    <xss:sequence maxOccurs="unbounded">
      <xss:element name="SecretsCertificate" type="FabricCertificateType" minOccurs="0"/>
      <xss:element name="EndpointCertificate" type="EndpointCertificateType" minOccurs="0"/>
    </xss:sequence>
    <xss:attributeGroup ref="VersionedItemAttrGroup"/>
  </xss:complexType>
</xss:element>

```

## Content element details

### SecretsCertificate

ATTRIBUTE	VALUE
name	SecretsCertificate
type	FabricCertificateType
minOccurs	0

### EndpointCertificate

ATTRIBUTE	VALUE
name	EndpointCertificate
type	EndpointCertificateType
minOccurs	0

## DigestedCertificates Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	DigestedCertificates
minOccurs	0
maxOccurs	1

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DigestedCertificates" minOccurs="0" maxOccurs="1">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="EndpointCertificate" type="EndpointCertificateType" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

## Content element details

### EndpointCertificate

ATTRIBUTE	VALUE
name	EndpointCertificate
type	EndpointCertificateType
minOccurs	0
maxOccurs	unbounded

## DigestedCodePackage Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	5 element(s), 2 attribute(s)
name	DigestedCodePackage
maxOccurs	unbounded

## XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DigestedCodePackage" maxOccurs="unbounded">
  <xss:complexType>
    <xss:sequence>
      <xss:element name="CodePackage" type="CodePackageType"/>
      <xss:element name="RunAsPolicy" type="RunAsPolicyType" minOccurs="0" maxOccurs="2"/>
      <xss:element name="DebugParameters" type="DebugParametersType" minOccurs="0" maxOccurs="1"/>
      <xss:element name="ContainerHostPolicies" type="ContainerHostPoliciesType" minOccurs="0">
        <xss:annotation>
          <xss:documentation>Specifies policies for activating container hosts.</xss:documentation>
        </xss:annotation>
      </xss:element>
      <xss:element name="ResourceGovernancePolicy" type="ResourceGovernancePolicyType" minOccurs="0">
        <xss:annotation>
          <xss:documentation>Specifies resource limits for codepackage.</xss:documentation>
        </xss:annotation>
      </xss:element>
    </xss:sequence>
    <xss:attributeGroup ref="VersionedItemAttrGroup"/>
    <xss:attribute name="ContentChecksum" type="xs:string"/>
    <xss:attribute name="IsShared" type="xs:boolean"/>
  </xss:complexType>
</xss:element>

```

## Attribute details

### ContentChecksum

ATTRIBUTE	VALUE
name	ContentChecksum
type	xs:string

### IsShared

ATTRIBUTE	VALUE
name	IsShared
type	xs:boolean

## Content element details

### CodePackage

ATTRIBUTE	VALUE
name	CodePackage
type	CodePackageType

### RunAsPolicy

ATTRIBUTE	VALUE
name	RunAsPolicy
type	RunAsPolicyType

ATTRIBUTE	VALUE
minOccurs	0
maxOccurs	2

#### DebugParameters

ATTRIBUTE	VALUE
name	DebugParameters
type	DebugParametersType
minOccurs	0
maxOccurs	1

#### ContainerHostPolicies

Specifies policies for activating container hosts.

ATTRIBUTE	VALUE
name	ContainerHostPolicies
type	ContainerHostPoliciesType
minOccurs	0

#### ResourceGovernancePolicy

Specifies resource limits for codepackage.

ATTRIBUTE	VALUE
name	ResourceGovernancePolicy
type	ResourceGovernancePolicyType
minOccurs	0

## DigestedConfigPackage Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	3 element(s), 2 attribute(s)
name	DigestedConfigPackage
minOccurs	0

ATTRIBUTE	VALUE
maxOccurs	unbounded

## XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DigestedConfigPackage" minOccurs="0" maxOccurs="unbounded">
<xss:complexType>
<xss:sequence>
<xss:element name="ConfigPackage" type="ConfigPackageType"/>
<xss:element name="ConfigOverride" type="ConfigOverrideType" minOccurs="0"/>
<xss:element name="DebugParameters" type="DebugParametersType" minOccurs="0" maxOccurs="1"/>
</xss:sequence>
<xss:attributeGroup ref="VersionedItemAttrGroup"/>
<xss:attribute name="ContentChecksum" type="xs:string"/>
<xss:attribute name="IsShared" type="xs:boolean"/>
</xss:complexType>
</xss:element>
```

## Attribute details

### ContentChecksum

Specifies resource limits for codepackage.

ATTRIBUTE	VALUE
name	ContentChecksum
type	xs:string

### IsShared

Specifies resource limits for codepackage.

ATTRIBUTE	VALUE
name	IsShared
type	xs:boolean

## Content element details

### ConfigPackage

ATTRIBUTE	VALUE
name	ConfigPackage
type	ConfigPackageType

### ConfigOverride

ATTRIBUTE	VALUE
name	ConfigOverride
type	ConfigOverrideType

ATTRIBUTE	VALUE
minOccurs	0

#### DebugParameters

ATTRIBUTE	VALUE
name	DebugParameters
type	DebugParametersType
minOccurs	0
maxOccurs	1

## DigestedDataPackage Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 2 attribute(s)
name	DigestedDataPackage
minOccurs	0
maxOccurs	unbounded

#### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="DigestedDataPackage" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="DataPackage" type="DataPackageType"/>
      <xs:element name="DebugParameters" type="DebugParametersType" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attributeGroup ref="VersionedItemAttrGroup"/>
    <xs:attribute name="ContentChecksum" type="xs:string"/>
    <xs:attribute name="IsShared" type="xs:boolean"/>
  </xs:complexType>
</xs:element>

```

#### Attribute details

##### ContentChecksum

ATTRIBUTE	VALUE
name	ContentChecksum
type	xs:string

##### IsShared

ATTRIBUTE	VALUE
name	IsShared
type	xs:boolean

## Content element details

### DataPackage

ATTRIBUTE	VALUE
name	DataPackage
type	DataPackageType

### DebugParameters

ATTRIBUTE	VALUE
name	DebugParameters
type	DebugParametersType
minOccurs	0
maxOccurs	1

## DigestedEndpoint Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	4 element(s), 0 attribute(s)
name	DigestedEndpoint
minOccurs	0
maxOccurs	unbounded

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DigestedEndpoint" minOccurs="0" maxOccurs="unbounded">
    <xss:complexType>
        <xss:sequence>
            <xss:element name="Endpoint" type="EndpointType"/>
            <xss:element name="SecurityAccessPolicy" type="SecurityAccessPolicyType" minOccurs="0"/>
            <xss:element name="EndpointBindingPolicy" type="EndpointBindingPolicyType"
minOccurs="0"/>
                <xss:element name="ResourceGovernancePolicy" type="ResourceGovernancePolicyType"
minOccurs="0" maxOccurs="1">
                    </xss:sequence>
                </xss:complexType>
            </xss:element>

```

## Content element details

### Endpoint

ATTRIBUTE	VALUE
name	Endpoint
type	EndpointType

### SecurityAccessPolicy

ATTRIBUTE	VALUE
name	SecurityAccessPolicy
type	SecurityAccessPolicyType
minOccurs	0

### EndpointBindingPolicy

ATTRIBUTE	VALUE
name	EndpointBindingPolicy
type	EndpointBindingPolicyType
minOccurs	0

### ResourceGovernancePolicy

ATTRIBUTE	VALUE
name	ResourceGovernancePolicy
type	ResourceGovernancePolicyType
minOccurs	0
maxOccurs	1

## DigestedEndpoints Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	DigestedEndpoints
minOccurs	0

## XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DigestedEndpoints" minOccurs="0">
    <xss:complexType>
        <xss:sequence>
            <xss:element name="DigestedEndpoint" minOccurs="0" maxOccurs="unbounded">
                <xss:complexType>
                    <xss:sequence>
                        <xss:element name="Endpoint" type="EndpointType"/>
                        <xss:element name="SecurityAccessPolicy" type="SecurityAccessPolicyType" minOccurs="0"/>
                        <xss:element name="EndpointBindingPolicy" type="EndpointBindingPolicyType"
minOccurs="0"/>
                        <xss:element name="ResourceGovernancePolicy" type="ResourceGovernancePolicyType"
minOccurs="0" maxOccurs="1"/>
                    </xss:sequence>
                </xss:complexType>
            </xss:element>
        </xss:sequence>
    </xss:complexType>
</xss:element>

```

## Content element details

### DigestedEndpoint

ATTRIBUTE	VALUE
name	DigestedEndpoint
minOccurs	0
maxOccurs	unbounded

## DigestedEnvironment Element

ATTRIBUTE	VALUE
type	EnvironmentType
content	0 element(s), 0 attribute(s)
name	DigestedEnvironment

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DigestedEnvironment" type="EnvironmentType"/>
```

## DigestedResources Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 0 attribute(s)
name	DigestedResources
minOccurs	1

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DigestedResources" minOccurs="1">
  <xss:complexType>
    <xss:sequence>
      <xss:element name="DigestedEndpoints" minOccurs="0">
        <xss:complexType>
          <xss:sequence>
            <xss:element name="DigestedEndpoint" minOccurs="0" maxOccurs="unbounded">
              <xss:complexType>
                <xss:sequence>
                  <xss:element name="Endpoint" type="EndpointType"/>
                  <xss:element name="SecurityAccessPolicy" type="SecurityAccessPolicyType" minOccurs="0"/>
                  <xss:element name="EndpointBindingPolicy" type="EndpointBindingPolicyType"
minOccurs="0"/>
                  <xss:element name="ResourceGovernancePolicy" type="ResourceGovernancePolicyType"
minOccurs="0" maxOccurs="1"/>
                </xss:sequence>
              </xss:complexType>
            </xss:element>
          </xss:sequence>
        </xss:complexType>
      </xss:element>
      <xss:element name="DigestedCertificates" minOccurs="0" maxOccurs="1">
        <xss:complexType>
          <xss:sequence>
            <xss:element name="EndpointCertificate" type="EndpointCertificateType" minOccurs="0"
maxOccurs="unbounded"/>
          </xss:sequence>
        </xss:complexType>
      </xss:element>
    </xss:sequence>
    <xss:attributeGroup ref="VersionedItemAttrGroup"/>
  </xss:complexType>
</xss:element>
```

### Content element details

#### DigestedEndpoints

ATTRIBUTE	VALUE
name	DigestedEndpoints

ATTRIBUTE	VALUE
minOccurs	0

#### DigestedCertificates

ATTRIBUTE	VALUE
name	DigestedCertificates
minOccurs	0
maxOccurs	1

## DigestedServiceTypes Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	DigestedServiceTypes

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DigestedServiceTypes">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ServiceTypes" type="ServiceTypesType"/>
    </xs:sequence>
    <xs:attributeGroup ref="VersionedItemAttrGroup"/>
  </xs:complexType>
</xs:element>
```

#### Content element details

##### ServiceTypes

ATTRIBUTE	VALUE
name	ServiceTypes
type	ServiceTypesType

## DllHost Element

ATTRIBUTE	VALUE
type	DllHostEntryPointType
content	0 element(s), 0 attribute(s)

ATTRIBUTE	VALUE
name	DllHost

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DllHost" type="DllHostEntryPointType"/>
```

## DomainGroup Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)
name	DomainGroup
minOccurs	0
maxOccurs	unbounded

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DomainGroup" minOccurs="0" maxOccurs="unbounded">
    <xss:complexType>
        <xss:attribute name="Name" type="xs:string"
use="required"/>
    </xss:complexType>
</xss:element>
```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

## DomainUser Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)

ATTRIBUTE	VALUE
name	DomainUser
minOccurs	0
maxOccurs	unbounded

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DomainUser" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
        <xs:attribute name="Name" type="xs:string"
use="required"/>
    </xs:complexType>
</xs:element>
```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

## DriverOption Element

Driver options to be passed to driver.

ATTRIBUTE	VALUE
type	DriverOptionType
content	0 element(s), 0 attribute(s)
name	DriverOption
minOccurs	0
maxOccurs	unbounded

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DriverOption" type="DriverOptionType" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>Driver options to be passed to driver.</xs:documentation>
    </xs:annotation>
</xs:element>
```

## DriverOption Element

Driver options to be passed to driver.

ATTRIBUTE	VALUE
type	DriverOptionType
content	0 element(s), 0 attribute(s)
name	DriverOption
minOccurs	0
maxOccurs	unbounded

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DriverOption" type="DriverOptionType" minOccurs="0" maxOccurs="unbounded">
    <xss:annotation>
        <xss:documentation>Driver options to be passed to driver.</xss:documentation>
    </xss:annotation>
</xss:element>
```

## ETW Element

Describes the ETW settings for the components of this service manifest.

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 0 attribute(s)
name	ETW
minOccurs	0

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ETW" minOccurs="0">
    <xss:annotation>
        <xss:documentation>Describes the ETW settings for the components of this service manifest.
    </xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence>
            <xss:element name="ProviderGuids" minOccurs="0">
                <xss:annotation>
                    <xss:documentation>Lists the ETW provider GUIDs for the components of this service manifest.
                </xss:documentation>
                </xss:annotation>
                <xss:complexType>
                    <xss:sequence>
                        <xss:element name="ProviderGuid" minOccurs="0" maxOccurs="unbounded">
                            <xss:complexType>
                                <xss:attribute name="Value" use="required">
                                    <xss:simpleType>
  <xss:restriction base="xs:string">
  <xss:pattern value="[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{12}">
  </xss:restriction>
                                    </xss:simpleType>
                                </xss:attribute>
                            </xss:complexType>
                            <xss:element>
                                </xss:complexType>
                            </xss:element>
                        </xss:sequence>
                    </xss:complexType>
                </xss:element>
            </xss:sequence>
        </xss:complexType>
    </xss:element>
    <xss:element name="ManifestDataPackages" minOccurs="0">
        <xss:annotation>
            <xss:documentation>Lists the data packages containing ETW manifests for the components of this service manifest. The data package containing ETW manifests should not contain any other files.
        </xss:documentation>
        </xss:annotation>
        <xss:complexType>
            <xss:sequence>
                <xss:element name="ManifestDataPackage" type="DataPackageType" minOccurs="0" maxOccurs="unbounded"/>
            </xss:sequence>
        </xss:complexType>
    </xss:element>
</xss:element>

```

## Content element details

### ProviderGuids

Lists the ETW provider GUIDs for the components of this service manifest.

ATTRIBUTE	VALUE
name	ProviderGuids
minOccurs	0

### ManifestDataPackages

Lists the data packages containing ETW manifests for the components of this service manifest. The data package containing ETW manifests should not contain any other files.

ATTRIBUTE	VALUE
name	ManifestDataPackages
minOccurs	0

## ETWSource Element

Specifies ETW trace collection. ETW traces are collected for the providers that are registered by all services belonging to the application.

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 1 attribute(s)
name	ETWSource
minOccurs	0

### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ETWSource" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Specifies ETW trace collection. ETW traces are collected for the providers that are
registered by all services belonging to the application.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Destinations" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>Destinations to which the crash dumps need to be transferred.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="LocalStore" type="LocalStoreETWType" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element name="FileStore" type="FileStoreETWType" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element name="AzureBlob" type="AzureBlobETWType" minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element ref="Parameters" minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
        <xs:attribute name="IsEnabled" type="xs:string">
            <xs:annotation>
                <xs:documentation>Whether or not ETW trace collection is enabled. By default, it is not enabled.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
</xs:element>

```

### Attribute details

#### .IsEnabled

Lists the data packages containing ETW manifests for the components of this service manifest. The data package containing ETW manifests should not contain any other files.

ATTRIBUTE	VALUE
name	isEnabled
type	xs:string

## Content element details

### Destinations

Destinations to which the crash dumps need to be transferred.

ATTRIBUTE	VALUE
name	Destinations
minOccurs	0

### None

ATTRIBUTE	VALUE
ref	Parameters
minOccurs	0
maxOccurs	1

## Endpoint Element

ATTRIBUTE	VALUE
type	EndpointOverrideType
content	0 element(s), 0 attribute(s)
name	Endpoint
maxOccurs	unbounded

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Endpoint" type="EndpointOverrideType" maxOccurs="unbounded"/>
```

## Endpoint Element

ATTRIBUTE	VALUE
type	EndpointType
content	0 element(s), 0 attribute(s)

ATTRIBUTE	VALUE
name	Endpoint
maxOccurs	unbounded

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Endpoint" type="EndpointType" maxOccurs="unbounded"/>
```

## Endpoint Element

ATTRIBUTE	VALUE
type	EndpointType
content	0 element(s), 0 attribute(s)
name	Endpoint

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Endpoint" type="EndpointType"/>
```

## EndpointBindingPolicy Element

Specifies a certificate that should be returned to a client for an HTTPS endpoint.

ATTRIBUTE	VALUE
type	EndpointBindingPolicyType
content	0 element(s), 0 attribute(s)
name	EndpointBindingPolicy
minOccurs	0

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EndpointBindingPolicy" type="EndpointBindingPolicyType" minOccurs="0">
    <xss:annotation>
        <xss:documentation>Specifies a certificate that should be returned to a client for an HTTPS endpoint.
    </xss:documentation>
    </xss:annotation>
</xss:element>
```

## EndpointBindingPolicy Element

ATTRIBUTE	VALUE
type	EndpointBindingPolicyType
content	0 element(s), 0 attribute(s)
name	EndpointBindingPolicy
minOccurs	0

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="EndpointBindingPolicy" type="EndpointBindingPolicyType" minOccurs="0"/>
```

## EndpointCertificate Element

ATTRIBUTE	VALUE
type	EndpointCertificateType
content	0 element(s), 0 attribute(s)
name	EndpointCertificate
minOccurs	0

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="EndpointCertificate" type="EndpointCertificateType" minOccurs="0"/>
```

## EndpointCertificate Element

ATTRIBUTE	VALUE
type	EndpointCertificateType
content	0 element(s), 0 attribute(s)
name	EndpointCertificate
minOccurs	0

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="EndpointCertificate" type="EndpointCertificateType" minOccurs="0"/>
```

## EndpointCertificate Element

ATTRIBUTE	VALUE
type	EndpointCertificateType
content	0 element(s), 0 attribute(s)
name	EndpointCertificate
minOccurs	0
maxOccurs	unbounded

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EndpointCertificate" type="EndpointCertificateType" minOccurs="0" maxOccurs="unbounded"/>
```

## Endpoints Element

Describe the endpoints associated with this node type

ATTRIBUTE	VALUE
type	FabricEndpointsType
content	0 element(s), 0 attribute(s)
name	Endpoints
minOccurs	0

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Endpoints" type="FabricEndpointsType" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Describe the endpoints associated with this node type</xs:documentation>
    </xs:annotation>
</xs:element>
```

## Endpoints Element

Defines endpoints for the service.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	Endpoints

ATTRIBUTE	VALUE
minOccurs	0

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Endpoints" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Defines endpoints for the service.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Endpoint" type="EndpointOverrideType" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

## Content element details

### Endpoint

ATTRIBUTE	VALUE
name	Endpoint
type	EndpointOverrideType
maxOccurs	unbounded

## Endpoints Element

Defines endpoints for the service.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	Endpoints
minOccurs	0

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Endpoints" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Defines endpoints for the service.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Endpoint" type="EndpointType" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

## Content element details

### Endpoint

ATTRIBUTE	VALUE
name	Endpoint
type	EndpointType
maxOccurs	unbounded

## Endpoints Element

Describe the endpoints associated with this node type

ATTRIBUTE	VALUE
type	FabricEndpointsType
content	0 element(s), 0 attribute(s)
name	Endpoints
minOccurs	0

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="Endpoints" type="FabricEndpointsType" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Describe the endpoints associated with this node type</xs:documentation>
  </xs:annotation>
</xs:element>
```

## EntryPoint Element

ATTRIBUTE	VALUE
type	EntryPointDescriptionType
content	0 element(s), 0 attribute(s)
name	EntryPoint
minOccurs	1

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="EntryPoint" type="EntryPointDescriptionType" minOccurs="1"/>
```

## EntryPoint Element

ATTRIBUTE	VALUE
type	xs:string
content	0 element(s), 0 attribute(s)
name	EntryPoint
minOccurs	0
maxOccurs	1

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EntryPoint" type="xs:string" minOccurs="0" maxOccurs="1"/>
```

## EnvironmentOverrides Element

ATTRIBUTE	VALUE
type	EnvironmentOverridesType
content	0 element(s), 0 attribute(s)
name	EnvironmentOverrides
minOccurs	0
maxOccurs	unbounded

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EnvironmentOverrides" type="EnvironmentOverridesType" minOccurs="0" maxOccurs="unbounded"/>
```

## EnvironmentVariable Element

Environment variable.

ATTRIBUTE	VALUE
type	EnvironmentVariableType
content	0 element(s), 0 attribute(s)
name	EnvironmentVariable
minOccurs	0
maxOccurs	unbounded

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EnvironmentVariable" type="EnvironmentVariableType" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>Environment variable.</xs:documentation>
    </xs:annotation>
</xs:element>
```

## EnvironmentVariable Element

Environment variable.

ATTRIBUTE	VALUE
type	EnvironmentVariableType
content	0 element(s), 0 attribute(s)
name	EnvironmentVariable
minOccurs	0
maxOccurs	unbounded

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EnvironmentVariable" type="EnvironmentVariableType" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>Environment variable.</xs:documentation>
    </xs:annotation>
</xs:element>
```

## EnvironmentVariables Element

ATTRIBUTE	VALUE
type	EnvironmentVariablesType
content	0 element(s), 0 attribute(s)
name	EnvironmentVariables
minOccurs	0
maxOccurs	1

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EnvironmentVariables" type="EnvironmentVariablesType" minOccurs="0" maxOccurs="1"/>
```

## EphemeralEndpoints Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	EphemeralEndpoints
minOccurs	0

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EphemeralEndpoints" minOccurs="0">
    <xs:complexType>
        <xs:attribute name="StartPort" type="xs:int" use="required"/>
        <xs:attribute name="EndPort" type="xs:int" use="required"/>
    </xs:complexType>
</xs:element>
```

### Attribute details

#### StartPort

ATTRIBUTE	VALUE
name	StartPort
type	xs:int
use	required

#### EndPort

ATTRIBUTE	VALUE
name	EndPort
type	xs:int
use	required

## EvictionPolicy Element

Eviction Policy extension for the Service Type.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 2 attribute(s)
name	EvictionPolicy

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EvictionPolicy">
  <xs:annotation>
    <xs:documentation>Eviction Policy extension for the Service Type.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Properties" type="ServiceTypeExtensionPolicyPropertiesType" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="Name" type="xs:string" use="required"/>
    <xs:attribute name="Provider" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

### Provider

ATTRIBUTE	VALUE
name	Provider
type	xs:string
use	required

## Content element details

### Properties

ATTRIBUTE	VALUE
name	Properties
type	ServiceTypeExtensionPolicyPropertiesType
minOccurs	0

## ExeHost Element

ATTRIBUTE	VALUE
type	ExeHostEntryPointType
content	0 element(s), 0 attribute(s)

ATTRIBUTE	VALUE
name	ExeHost

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ExeHost" type="ExeHostEntryPointType"/>
```

## ExeHost Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	ExeHost

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ExeHost">
  <xss:complexType>
    <xss:complexContent>
      <xss:extension base="ExeHostEntryPointType">
        <xss:sequence>
          <xss:element name="RunFrequency" minOccurs="0">
            <xss:complexType>
              <xss:attribute name="IntervalInSeconds" use="required">
                <xss:simpleType>
                  <xss:restriction base="xs:int">
                    <xss:minInclusive value="0"/>
                    <xss:maxInclusive value="2147483647"/>
                  </xss:restriction>
                </xss:simpleType>
              </xss:attribute>
            </xss:complexType>
          </xss:element>
        </xss:sequence>
      </xss:extension>
    </xss:complexContent>
  </xss:complexType>
</xss:element>
```

## Extension Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	Extension

ATTRIBUTE	VALUE
minOccurs	0
maxOccurs	unbounded

## XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Extension" minOccurs="0" maxOccurs="unbounded">
  <xss:complexType>
    <xss:sequence>
      <xss:any namespace="##other" processContents="lax"/>
    </xss:sequence>
    <xss:attribute name="Name" use="required">
      <xss:simpleType>
        <xss:restriction base="xs:string">
          <xss:minLength value="1"/>
        </xss:restriction>
      </xss:simpleType>
    </xss:attribute>
    <xss:attribute name="GeneratedId" type="xs:string" use="optional"/>
  </xss:complexType>
</xss:element>

```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
use	required

### GeneratedId

ATTRIBUTE	VALUE
name	GeneratedId
type	xs:string
use	optional

## Extensions Element

ATTRIBUTE	VALUE
type	ExtensionsType
content	0 element(s), 0 attribute(s)
name	Extensions

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Extensions" type="ExtensionsType"/>
```

## FabricSettings Element

ATTRIBUTE	VALUE
type	SettingsOverridesType
content	0 element(s), 0 attribute(s)
name	FabricSettings
minOccurs	0

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="FabricSettings" type="SettingsOverridesType" minOccurs="0"/>
```

## FailoverManagerReplicatorEndpoint Element

ATTRIBUTE	VALUE
type	InternalEndpointType
content	0 element(s), 0 attribute(s)
name	FailoverManagerReplicatorEndpoint
minOccurs	0

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="FailoverManagerReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
```

## FaultAnalysisServiceReplicatorEndpoint Element

ATTRIBUTE	VALUE
type	InternalEndpointType
content	0 element(s), 0 attribute(s)
name	FaultAnalysisServiceReplicatorEndpoint
minOccurs	0

### XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="FaultAnalysisServiceReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
```

## FileStore Element

ATTRIBUTE	VALUE
type	FileStoreType
content	0 element(s), 0 attribute(s)
name	FileStore
minOccurs	0
maxOccurs	unbounded

### XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="FileStore" type="FileStoreType" minOccurs="0" maxOccurs="unbounded"/>
```

## FileStore Element

ATTRIBUTE	VALUE
type	FileStoreETWType
content	0 element(s), 0 attribute(s)
name	FileStore
minOccurs	0
maxOccurs	unbounded

### XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="FileStore" type="FileStoreETWType" minOccurs="0" maxOccurs="unbounded"/>
```

## FileStore Element

ATTRIBUTE	VALUE
type	FileStoreType
content	0 element(s), 0 attribute(s)

ATTRIBUTE	VALUE
name	FileStore
minOccurs	0
maxOccurs	unbounded

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="FileStore" type="FileStoreType" minOccurs="0" maxOccurs="unbounded"/>
```

## FolderSource Element

Specifies the collection of the contents of a particular folder on the local node.

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 1 attribute(s)
name	FolderSource
minOccurs	0
maxOccurs	unbounded

#### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="FolderSource" minOccurs="0" maxOccurs="unbounded">
    <xss:annotation>
        <xss:documentation>Specifies the collection of the contents of a particular folder on the local node.
    </xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence>
            <xss:element name="Destinations" minOccurs="0">
                <xss:annotation>
                    <xss:documentation>Destinations to which the folder contents need to be transferred.
                </xss:documentation>
            </xss:annotation>
            <xss:complexType>
                <xss:sequence>
                    <xss:element name="LocalStore" type="LocalStoreType" minOccurs="0" maxOccurs="unbounded"/>
                    <xss:element name="FileStore" type="FileStoreType" minOccurs="0" maxOccurs="unbounded"/>
                    <xss:element name="AzureBlob" type="AzureBlobType" minOccurs="0" maxOccurs="unbounded"/>
                </xss:sequence>
            </xss:complexType>
            <xss:element>
                <xss:element ref="Parameters" minOccurs="0" maxOccurs="1"/>
            </xss:element>
            <xss:attribute name="IsEnabled" type="xs:string">
                <xss:annotation>
                    <xss:documentation>Whether or not collection of the contents of this folder is enabled. By
default, it is not enabled.</xss:documentation>
                </xss:annotation>
            </xss:attribute>
            <xss:attributeGroup ref="RelativeFolderPath"/>
            <xss:attributeGroup ref="DataDeletionAgeInDays"/>
        </xss:complexType>
    </xss:element>

```

## Attribute details

### .IsEnabled

ATTRIBUTE	VALUE
name	IsEnabled
type	xs:string

## Content element details

### Destinations

Destinations to which the folder contents need to be transferred.

ATTRIBUTE	VALUE
name	Destinations
minOccurs	0

### None

ATTRIBUTE	VALUE
ref	Parameters

ATTRIBUTE	VALUE
minOccurs	0
maxOccurs	1

## FromSource Element

ATTRIBUTE	VALUE
type	<a href="#">xs:string</a>
content	0 element(s), 0 attribute(s)
name	FromSource
minOccurs	0
maxOccurs	1

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="FromSource" type="xs:string" minOccurs="0" maxOccurs="1"/>
```

## Group Element

Declares a group as a security principal, which can be referenced in policies.

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 1 attribute(s)
name	Group
maxOccurs	unbounded

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Group" maxOccurs="unbounded">
    <xss:annotation>
        <xss:documentation>Declares a group as a security principal, which can be
referenced in policies.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence>
            <xss:element name="NTLMAuthenticationPolicy" minOccurs="0">
                <xss:complexType>
                    <xss:attribute name="IsEnabled" type="xs:boolean" use="optional"
default="true"/>
                </xss:complexType>
            </xss:element>
            <xss:element name="Membership" minOccurs="0">
                <xss:complexType>
                    <xss:choice maxOccurs="unbounded">
                        <xss:element name="DomainGroup" minOccurs="0"
maxOccurs="unbounded">
                            <xss:complexType>
                                <xss:attribute name="Name" type="xs:string"
use="required"/>
                            </xss:complexType>
                        </xss:element>
                        <xss:element name="SystemGroup" minOccurs="0"
maxOccurs="unbounded">
                            <xss:complexType>
                                <xss:attribute name="Name" type="xs:string"
use="required"/>
                            </xss:complexType>
                        </xss:element>
                        <xss:element name="DomainUser" minOccurs="0"
maxOccurs="unbounded">
                            <xss:complexType>
                                <xss:attribute name="Name" type="xs:string"
use="required"/>
                            </xss:complexType>
                        </xss:element>
                    </xss:choice>
                </xss:complexType>
            </xss:element>
        </xss:sequence>
        <xss:attribute name="Name" type="xs:string" use="required">
            <xss:documentation>Name of the local group account. The name will be
prefixed with the application ID.</xss:documentation>
        </xss:attribute>
    </xss:complexType>
</xss:element>

```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

## Content element details

#### NTLMAuthenticationPolicy

ATTRIBUTE	VALUE
name	NTLMAuthenticationPolicy
minOccurs	0

#### Membership

ATTRIBUTE	VALUE
name	Membership
minOccurs	0

## Group Element

The group to add the user to. The group must be defined in the Groups section.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)
name	Group
minOccurs	0
maxOccurs	unbounded

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="Group" minOccurs="0" maxOccurs="unbounded">

  <xs:annotation>
    <xs:documentation>The group to add the user to. The group must be defined in the Groups section.</xs:documentation>
  </xs:annotation>
</xs:annotation>

  <xs:complexType>
    <xs:attribute name="NameRef" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>The name of the group.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
</xs:element>
```

## Attribute details

### NameRef

ATTRIBUTE	VALUE
name	NameRef
type	xs:string
use	required

## Groups Element

Declares a set of groups as security principals, which can be referenced in policies.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	Groups
minOccurs	0

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Groups" minOccurs="0">
    <xss:annotation>
        <xss:documentation>Declares a set of groups as security principals, which can be referenced
in policies.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence>
            <xss:element name="Group" maxOccurs="unbounded">
                <xss:annotation>
                    <xss:documentation>Declares a group as a security principal, which can be
referenced in policies.</xss:documentation>
                </xss:annotation>
                <xss:complexType>
                    <xss:sequence>
                        <xss:element name="NTLMAuthenticationPolicy" minOccurs="0">
                            <xss:complexType>
                                <xss:attribute name="IsEnabled" type="xs:boolean" use="optional"
default="true"/>
                            </xss:complexType>
                        </xss:element>
                        <xss:element name="Membership" minOccurs="0">
                            <xss:complexType>
                                <xss:choice maxOccurs="unbounded">
                                    <xss:element name="DomainGroup" minOccurs="0"
maxOccurs="unbounded">
  <xss:complexType>
  <xss:attribute name="Name" type="xs:string"
use="required"/>
  </xss:complexType>
                                    </xss:element>
                                    <xss:element name="SystemGroup" minOccurs="0"
maxOccurs="unbounded">
  <xss:complexType>
  <xss:attribute name="Name" type="xs:string"
use="required"/>
  </xss:complexType>
                                    </xss:element>
                                    <xss:element name="DomainUser" minOccurs="0"
maxOccurs="unbounded">
  <xss:complexType>
  <xss:attribute name="Name" type="xs:string"
use="required"/>
  </xss:complexType>
                                    </xss:element>
                                </xss:choice>
                            </xss:complexType>
                            <xss:attribute name="Name" type="xs:string" use="required">
                                <xss:annotation>
                                    <xss:documentation>Name of the local group account. The name will be
prefixed with the application ID.</xss:documentation>
                                </xss:annotation>
                                </xss:attribute>
                            </xss:complexType>
                        </xss:element>
                    </xss:sequence>
                    <xss:attribute name="Name" type="xs:string" use="required">
                        <xss:annotation>
                            <xss:documentation>Name of the local group account. The name will be
prefixed with the application ID.</xss:documentation>
                        </xss:annotation>
                        </xss:attribute>
                    </xss:complexType>
                </xss:element>
            </xss:sequence>
        </xss:complexType>
    </xss:element>

```

## Content element details

### Group

Declares a group as a security principal, which can be referenced in policies.

ATTRIBUTE	VALUE
name	Group
maxOccurs	unbounded

## HealthPolicy Element

ATTRIBUTE	VALUE
type	ApplicationHealthPolicyType
content	0 element(s), 0 attribute(s)
name	HealthPolicy
minOccurs	0

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="HealthPolicy" type="ApplicationHealthPolicyType" minOccurs="0"/>
```

## HttpApplicationGatewayEndpoint Element

ATTRIBUTE	VALUE
type	InputEndpointType
content	0 element(s), 0 attribute(s)
name	HttpApplicationGatewayEndpoint
minOccurs	0

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="HttpApplicationGatewayEndpoint" type="InputEndpointType" minOccurs="0"/>
```

## HttpGatewayEndpoint Element

ATTRIBUTE	VALUE
type	InputEndpointType
content	0 element(s), 0 attribute(s)
name	HttpGatewayEndpoint

ATTRIBUTE	VALUE
minOccurs	0

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="HttpGatewayEndpoint" type="InputEndpointType" minOccurs="0"/>
```

## ImageName Element

ATTRIBUTE	VALUE
type	xs:string
content	0 element(s), 0 attribute(s)
name	ImageName

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ImageName" type="xs:string"/>
```

## ImageStoreServiceReplicatorEndpoint Element

ATTRIBUTE	VALUE
type	InternalEndpointType
content	0 element(s), 0 attribute(s)
name	ImageStoreServiceReplicatorEndpoint
minOccurs	0

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ImageStoreServiceReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
```

## Infrastructure Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	6 element(s), 0 attribute(s)
name	Infrastructure

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Infrastructure">
    <xs:complexType>
        <xs:choice>
            <xs:element name="WindowsServer">
                <xs:complexType>
                    <xs:complexContent>
                        <xs:extension
base="WindowsInfrastructureType">
                            <xs:attribute name="IsScaleMin"
type="xs:boolean" default="false"/>
                        </xs:extension>
                    </xs:complexContent>
                </xs:complexType>
            </xs:element>
            <xs:element name="Linux">
                <xs:complexType>
                    <xs:complexContent>
                        <xs:extension base="LinuxInfrastructureType">
                            <xs:attribute name="IsScaleMin" type="xs:boolean" default="false"/>
                        </xs:extension>
                    </xs:complexContent>
                </xs:complexType>
            </xs:element>
            <xs:element name="WindowsAzure">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Roles">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element name="Role"
name="Role" type="AzureRoleType" maxOccurs="unbounded"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="WindowsAzureStaticTopology">
                <xs:complexType>
                    <xs:complexContent>
                        <xs:extension
base="WindowsInfrastructureType"/>
                    </xs:complexContent>
                </xs:complexType>
            </xs:element>
            <xs:element name="Blackbird">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Roles">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element name="Role"
name="Role" type="BlackbirdRoleType" minOccurs="1" maxOccurs="unbounded"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="PaaS">
                <xs:complexType>
                    <xs:all>
                        <xs:element name="Roles">
                            <xs:complexType>
                                <xs:sequence>
```

```

<xs:element name="Role" type="PaaSRoleType" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="Votes">
    <xs:complexType>
        <xs:sequence>
            <xs:element
name="Vote" type="PaaSVoteType" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:all>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>

```

## Content element details

### WindowsServer

ATTRIBUTE	VALUE
name	WindowsServer

### Linux

ATTRIBUTE	VALUE
name	Linux

### WindowsAzure

ATTRIBUTE	VALUE
name	WindowsAzure

### WindowsAzureStaticTopology

ATTRIBUTE	VALUE
name	WindowsAzureStaticTopology

### Blackbird

ATTRIBUTE	VALUE
name	Blackbird

### PaaS

ATTRIBUTE	VALUE
name	PaaS

## InfrastructureInformation Element

Describes the infrastructure on which fabric needs to run.

ATTRIBUTE	VALUE
type	InfrastructureInformationType
content	0 element(s), 0 attribute(s)
name	InfrastructureInformation

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="InfrastructureInformation" type="InfrastructureInformationType">
    <xs:annotation>
        <xs:documentation>Describes the infrastructure on which fabric needs to run.
    </xs:documentation>
</xs:element>
```

## KtlLoggerSettings Element

Describe the KtlLogger information associated with this node type

ATTRIBUTE	VALUE
type	FabricKtlLoggerSettingsType
content	0 element(s), 0 attribute(s)
name	KtlLoggerSettings
minOccurs	0

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="KtlLoggerSettings" type="FabricKtlLoggerSettingsType" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Describe the
KtlLogger information associated with this node type</xs:documentation>
    </xs:annotation>
</xs:element>
```

## LeaseDriverEndpoint Element

ATTRIBUTE	VALUE
type	InternalEndpointType
content	0 element(s), 0 attribute(s)
name	LeaseDriverEndpoint

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LeaseDriverEndpoint" type="InternalEndpointType"/>
```

## Linux Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	Linux

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Linux">
    <xss:complexType>
        <xss:complexContent>
            <xss:extension base="LinuxInfrastructureType">
                <xss:attribute name="IsScaleMin" type="xs:boolean" default="false"/>
            </xss:extension>
        </xss:complexContent>
    </xss:complexType>
</xss:element>
```

## LoadMetric Element

ATTRIBUTE	VALUE
type	LoadMetricType
content	0 element(s), 0 attribute(s)
name	LoadMetric
maxOccurs	unbounded

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LoadMetric" type="LoadMetricType" maxOccurs="unbounded"/>
```

## LoadMetric Element

ATTRIBUTE	VALUE
type	LoadMetricType
content	0 element(s), 0 attribute(s)

ATTRIBUTE	VALUE
name	LoadMetric
maxOccurs	unbounded

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LoadMetric" type="LoadMetricType" maxOccurs="unbounded"/>
```

## LoadMetric Element

ATTRIBUTE	VALUE
type	LoadMetricType
content	0 element(s), 0 attribute(s)
name	LoadMetric
maxOccurs	unbounded

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LoadMetric" type="LoadMetricType" maxOccurs="unbounded"/>
```

## LoadMetric Element

ATTRIBUTE	VALUE
type	LoadMetricType
content	0 element(s), 0 attribute(s)
name	LoadMetric
maxOccurs	unbounded

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LoadMetric" type="LoadMetricType" maxOccurs="unbounded"/>
```

## LoadMetric Element

ATTRIBUTE	VALUE
type	LoadMetricType

ATTRIBUTE	VALUE
content	0 element(s), 0 attribute(s)
name	LoadMetric
maxOccurs	unbounded

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LoadMetric" type="LoadMetricType" maxOccurs="unbounded"/>
```

# LoadMetrics Element

Load metrics reported by this service, used for resource balancing services.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	LoadMetrics
minOccurs	0

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LoadMetrics" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Load metrics reported by this service, used for resource balancing
services.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="LoadMetric" type="LoadMetricType" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

## Content element details

### LoadMetric

ATTRIBUTE	VALUE
name	LoadMetric
type	LoadMetricType
maxOccurs	unbounded

# LoadMetrics Element

Load metrics reported by this service.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	LoadMetrics
minOccurs	0

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LoadMetrics" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Load metrics reported by this service.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="LoadMetric" type="LoadMetricType" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

## Content element details

### LoadMetric

ATTRIBUTE	VALUE
name	LoadMetric
type	LoadMetricType
maxOccurs	unbounded

## LoadMetrics Element

Load metrics reported by this service.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	LoadMetrics
minOccurs	0

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LoadMetrics" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Load metrics reported by this service.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="LoadMetric" type="LoadMetricType" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

## Content element details

### LoadMetric

ATTRIBUTE	VALUE
name	LoadMetric
type	LoadMetricType
maxOccurs	unbounded

## LoadMetrics Element

Load metrics reported by this service.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	LoadMetrics
minOccurs	0

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LoadMetrics" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Load metrics reported by this service.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="LoadMetric" type="LoadMetricType" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

## Content element details

### LoadMetric

ATTRIBUTE	VALUE
name	LoadMetric
type	LoadMetricType
maxOccurs	unbounded

## LoadMetrics Element

Load metrics reported by this service, used for resource balancing services.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	LoadMetrics
minOccurs	0

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LoadMetrics" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Load metrics reported by this service, used for resource balancing services.
    </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="LoadMetric" type="LoadMetricType" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

### Content element details

#### LoadMetric

ATTRIBUTE	VALUE
name	LoadMetric
type	LoadMetricType
maxOccurs	unbounded

## LocalStore Element

ATTRIBUTE	VALUE
type	LocalStoreType

ATTRIBUTE	VALUE
content	0 element(s), 0 attribute(s)
name	LocalStore
minOccurs	0
maxOccurs	unbounded

#### XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LocalStore" type="LocalStoreType" minOccurs="0" maxOccurs="unbounded"/>
```

## LocalStore Element

ATTRIBUTE	VALUE
type	LocalStoreETWType
content	0 element(s), 0 attribute(s)
name	LocalStore
minOccurs	0
maxOccurs	unbounded

#### XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LocalStore" type="LocalStoreETWType" minOccurs="0" maxOccurs="unbounded"/>
```

## LocalStore Element

ATTRIBUTE	VALUE
type	LocalStoreType
content	0 element(s), 0 attribute(s)
name	LocalStore
minOccurs	0
maxOccurs	unbounded

#### XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LocalStore" type="LocalStoreType" minOccurs="0" maxOccurs="unbounded"/>
```

## LogCollectionPolicies Element

Specifies whether log collection is enabled. Works only in an Azure cluster environment

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	LogCollectionPolicies
minOccurs	0

### XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LogCollectionPolicies" minOccurs="0">
    <xss:annotation>
        <xss:documentation>Specifies whether log collection is enabled. Works
only in an Azure cluster environment</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence maxOccurs="unbounded">
            <xss:element name="LogCollectionPolicy">
                <xss:complexType>
                    <xss:attribute name="Path" type="xs:string"
use="optional"/>
                </xss:complexType>
            </xss:element>
        </xss:sequence>
    </xss:complexType>
</xss:element>
```

### Content element details

#### LogCollectionPolicy

ATTRIBUTE	VALUE
name	LogCollectionPolicy

## LogCollectionPolicy Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)
name	LogCollectionPolicy

### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LogCollectionPolicy">
    <xs:complexType>
        <xs:attribute name="Path" type="xs:string"
use="optional"/>
    </xs:complexType>
</xs:element>

```

## Attribute details

### Path

ATTRIBUTE	VALUE
name	Path
type	xs:string
use	optional

## LogConfig Element

Specifies the logging driver for a container.

ATTRIBUTE	VALUE
type	ContainerLoggingDriverType
content	0 element(s), 0 attribute(s)
name	LogConfig
minOccurs	0
maxOccurs	1

### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LogConfig" type="ContainerLoggingDriverType" minOccurs="0" maxOccurs="1">
    <xs:annotation>
        <xs:documentation>Specifies the logging driver for a container.</xs:documentation>
    </xs:annotation>
</xs:element>

```

## LogicalDirectories Element

Describe the LogicalDirectories settings associated with this node type

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)

ATTRIBUTE	VALUE
name	LogicalDirectories
minOccurs	0

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LogicalDirectories" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Describe the
LogicalDirectories settings associated with this node type</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element
name="LogicalDirectory" type="LogicalDirectoryType" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

```

## Content element details

### LogicalDirectory

ATTRIBUTE	VALUE
name	LogicalDirectory
type	LogicalDirectoryType
maxOccurs	unbounded

## LogicalDirectory Element

ATTRIBUTE	VALUE
type	LogicalDirectoryType
content	0 element(s), 0 attribute(s)
name	LogicalDirectory
maxOccurs	unbounded

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LogicalDirectory" type="LogicalDirectoryType" maxOccurs="unbounded"/>

```

## ManagedAssembly Element

ATTRIBUTE	VALUE
type	ManagedAssemblyType
content	0 element(s), 0 attribute(s)
name	ManagedAssembly

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ManagedAssembly" type="ManagedAssemblyType"/>
```

## ManifestDataPackage Element

ATTRIBUTE	VALUE
type	DataPackageType
content	0 element(s), 0 attribute(s)
name	ManifestDataPackage
minOccurs	0
maxOccurs	unbounded

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ManifestDataPackage" type="DataPackageType" minOccurs="0" maxOccurs="unbounded"/>
```

## ManifestDataPackages Element

Lists the data packages containing ETW manifests for the components of this service manifest. The data package containing ETW manifests should not contain any other files.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	ManifestDataPackages
minOccurs	0

#### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ManifestDataPackages" minOccurs="0">
    <xss:annotation>
        <xss:documentation>Lists the data packages containing ETW manifests for the components of this
service manifest. The data package containing ETW manifests should not contain any other files.
    </xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence>
            <xss:element name="ManifestDataPackage" type="DataPackageType" minOccurs="0"
maxOccurs="unbounded"/>
        </xss:sequence>
    </xss:complexType>
</xss:element>

```

## Content element details

### ManifestDataPackage

ATTRIBUTE	VALUE
name	ManifestDataPackage
type	DataPackageType
minOccurs	0
maxOccurs	unbounded

## Member Element

ATTRIBUTE	VALUE
type	ServiceGroupMemberType
content	0 element(s), 0 attribute(s)
name	Member
minOccurs	1
maxOccurs	unbounded

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Member" type="ServiceGroupMemberType" minOccurs="1" maxOccurs="unbounded"/>

```

## Member Element

ATTRIBUTE	VALUE
type	ServiceGroupMemberType

ATTRIBUTE	VALUE
content	0 element(s), 0 attribute(s)
name	Member
minOccurs	1
maxOccurs	unbounded

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Member" type="ServiceGroupMemberType" minOccurs="1" maxOccurs="unbounded"/>
```

## MemberOf Element

Users can be added to any existing membership group, so it can inherit all the properties and security settings of that membership group. The membership group can be used to secure external resources that need to be accessed by different services or the same service (on a different machine).

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 0 attribute(s)
name	MemberOf
minOccurs	0

#### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="MemberOf" minOccurs="0">
    <xss:annotation>
        <xss:documentation>
            Users can be added to any existing membership group, so it can inherit all the
            properties and security settings of that membership group. The membership group can be used to secure external
            resources that need to be accessed by different services or the same service (on a different machine).
        </xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:choice
maxOccurs="unbounded">
            <xss:element
name="SystemGroup" minOccurs="0" maxOccurs="unbounded">
                <xss:annotation>
                    <xss:documentation>The system group to add the user to. The system group must be defined in the Groups section.
                    </xss:documentation>
                </xss:annotation>
                <xss:complexType>
                    <xss:attribute name="Name" type="xs:string" use="required">
                        <xss:annotation>
                            <xss:documentation>The name of the system group.</xss:documentation>
                        </xss:annotation>
                    </xss:attribute>
                </xss:complexType>
            </xss:element>
            <xss:element
name="Group" minOccurs="0" maxOccurs="unbounded">
                <xss:annotation>
                    <xss:documentation>The group to add the user to. The group must be defined in the Groups section.
                    </xss:documentation>
                </xss:annotation>
                <xss:complexType>
                    <xss:attribute name="NameRef" type="xs:string" use="required">
                        <xss:annotation>
                            <xss:documentation>The name of the group.</xss:documentation>
                        </xss:annotation>
                    </xss:attribute>
                </xss:complexType>
            </xss:element>
        </xss:choice>
    </xss:complexType>
</xss:element>

```

## Content element details

### SystemGroup

The system group to add the user to. The system group must be defined in the Groups section.

ATTRIBUTE	VALUE
name	SystemGroup
minOccurs	0
maxOccurs	unbounded

#### Group

The group to add the user to. The group must be defined in the Groups section.

ATTRIBUTE	VALUE
name	Group
minOccurs	0
maxOccurs	unbounded

## Members Element

Member services of this service group

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	Members
minOccurs	1
maxOccurs	1

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Members" minOccurs="1" maxOccurs="1">
    <xs:annotation>
        <xs:documentation>Member services of this service group</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Member" type="ServiceGroupMemberType" minOccurs="1"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

#### Content element details

##### Member

ATTRIBUTE	VALUE
name	Member
type	ServiceGroupMemberType
minOccurs	1
maxOccurs	unbounded

## Members Element

Member services of this service group

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	Members
minOccurs	1
maxOccurs	1

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Members" minOccurs="1" maxOccurs="1">
    <xs:annotation>
        <xs:documentation>Member services of this service group</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Member" type="ServiceGroupMemberType" minOccurs="1"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

### Content element details

#### Member

ATTRIBUTE	VALUE
name	Member
type	ServiceGroupMemberType
minOccurs	1
maxOccurs	unbounded

# Membership Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	3 element(s), 0 attribute(s)
name	Membership
minOccurs	0

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Membership" minOccurs="0">
    <xs:complexType>
        <xs:choice maxOccurs="unbounded">
            <xs:element name="DomainGroup" minOccurs="0"
maxOccurs="unbounded">
                <xs:complexType>
                    <xs:attribute name="Name" type="xs:string"
use="required"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="SystemGroup" minOccurs="0"
maxOccurs="unbounded">
                <xs:complexType>
                    <xs:attribute name="Name" type="xs:string"
use="required"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="DomainUser" minOccurs="0"
maxOccurs="unbounded">
                <xs:complexType>
                    <xs:attribute name="Name" type="xs:string"
use="required"/>
                </xs:complexType>
            </xs:element>
        </xs:choice>
    </xs:complexType>
</xs:element>
```

## Content element details

### DomainGroup

ATTRIBUTE	VALUE
name	DomainGroup
minOccurs	0
maxOccurs	unbounded

### SystemGroup

ATTRIBUTE	VALUE
name	SystemGroup

ATTRIBUTE	VALUE
minOccurs	0
maxOccurs	unbounded

#### DomainUser

ATTRIBUTE	VALUE
name	DomainUser
minOccurs	0
maxOccurs	unbounded

## NTLMAuthenticationPolicy Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)
name	NTLMAuthenticationPolicy
minOccurs	0

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="NTLMAuthenticationPolicy" minOccurs="0">
    <xs:complexType>
        <xs:attribute name="IsEnabled" type="xs:boolean" use="optional"
default="true"/>
    </xs:complexType>
</xs:element>
```

#### Attribute details

##### .IsEnabled

ATTRIBUTE	VALUE
name	isEnabled
type	xs:boolean
use	optional
default	true

## NTLMAuthenticationPolicy Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 6 attribute(s)
name	NTLMAuthenticationPolicy
minOccurs	0

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="NTLMAuthenticationPolicy" minOccurs="0">
    <xs:complexType>
      <xs:attribute name="IsEnabled" type="xs:boolean" use="optional"
        default="true"/>
      <xs:attribute name="PasswordSecret" type="xs:string"
        use="required"/>
      <xs:attribute name="PasswordSecretEncrypted" type="xs:boolean"
        use="optional" default="false"/>
      <xs:attribute name="X509StoreLocation" use="optional"
        default="LocalMachine">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="LocalMachine"/>
            <xs:enumeration value="CurrentUser"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="X509StoreName" default="My">
        <xs:simpleType>
          <xs:restriction base="xs:string"/>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="X509Thumbprint" type="xs:string"/>
    </xs:complexType>
  </xs:element>

```

## Attribute details

### IsEnabled

ATTRIBUTE	VALUE
name	IsEnabled
type	xs:boolean
use	optional
default	true

### PasswordSecret

ATTRIBUTE	VALUE
name	PasswordSecret

ATTRIBUTE	VALUE
type	xs:string
use	required

#### PasswordSecretEncrypted

ATTRIBUTE	VALUE
name	PasswordSecretEncrypted
type	xs:boolean
use	optional
default	false

#### X509StoreLocation

ATTRIBUTE	VALUE
name	X509StoreLocation
use	optional
default	LocalMachine

#### X509StoreName

ATTRIBUTE	VALUE
name	X509StoreName
default	My

#### X509Thumbprint

ATTRIBUTE	VALUE
name	X509Thumbprint
type	xs:string

## NamedPartition Element

Describes a named partitioning scheme based on names for each partition.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)

ATTRIBUTE	VALUE
name	NamedPartition

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="NamedPartition">
    <xs:annotation>
        <xs:documentation>Describes a named partitioning scheme based on names for each partition.
    </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence maxOccurs="unbounded">
            <xs:element name="Partition">
                <xs:annotation>
                    <xs:documentation>Describes a partition by name.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:attribute name="Name" use="required">
                        <xs:annotation>
                            <xs:documentation>The name of the partition</xs:documentation>
                        </xs:annotation>
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:minLength value="1"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:attribute>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>

```

## Content element details

### Partition

Describes a partition by name.

ATTRIBUTE	VALUE
name	Partition

## NamingReplicatorEndpoint Element

ATTRIBUTE	VALUE
type	InternalEndpointType
content	0 element(s), 0 attribute(s)
name	NamingReplicatorEndpoint
minOccurs	0

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="NamingReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
```

## NetworkConfig Element

Specifies the network configuration for a container.

ATTRIBUTE	VALUE
type	ContainerNetworkConfigType
content	0 element(s), 0 attribute(s)
name	NetworkConfig
minOccurs	0
maxOccurs	1

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="NetworkConfig" type="ContainerNetworkConfigType" minOccurs="0" maxOccurs="1">
    <xss:annotation>
        <xss:documentation>Specifies the network configuration for a container.</xss:documentation>
    </xss:annotation>
</xss:element>
```

## Node Element

ATTRIBUTE	VALUE
type	FabricNodeType
content	0 element(s), 0 attribute(s)
name	Node
maxOccurs	unbounded

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Node" type="FabricNodeType" maxOccurs="unbounded"/>
```

## Node Element

ATTRIBUTE	VALUE
type	FabricNodeType

ATTRIBUTE	VALUE
content	0 element(s), 0 attribute(s)
name	Node
maxOccurs	unbounded

## XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Node" type="FabricNodeType" maxOccurs="unbounded"/>
```

## Node Element

ATTRIBUTE	VALUE
type	InfrastructureNodeType
content	0 element(s), 0 attribute(s)
name	Node
maxOccurs	unbounded

## XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Node" type="InfrastructureNodeType" maxOccurs="unbounded"/>
```

## NodeList Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	NodeList

## XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="NodeList">
    <xss:complexType>
        <xss:sequence>
            <xss:element name="Node" type="FabricNodeType"
maxOccurs="unbounded"/>
        </xss:sequence>
    </xss:complexType>
</xss:element>
```

## Content element details

### Node

ATTRIBUTE	VALUE
name	Node
type	FabricNodeType
maxOccurs	unbounded

## NodeList Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	NodeList

### XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="NodeList">
    <xss:complexType>
        <xss:sequence>
            <xss:element name="Node" type="FabricNodeType" maxOccurs="unbounded"/>
        </xss:sequence>
    </xss:complexType>
</xss:element>
```

## Content element details

### Node

ATTRIBUTE	VALUE
name	Node
type	FabricNodeType
maxOccurs	unbounded

## NodeList Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	NodeList

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="NodeList">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Node" type="InfrastructureNodeType" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

## Content element details

### Node

ATTRIBUTE	VALUE
name	Node
type	InfrastructureNodeType
maxOccurs	unbounded

## NodeType Element

Describe a node type.

ATTRIBUTE	VALUE
type	anonymous complexType
content	6 element(s), 1 attribute(s)
name	NodeType
maxOccurs	unbounded

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="NodeType" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>Describe a node type.
    </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:all>
            <xs:element name="Endpoints"
type="FabricEndpointsType" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>Describe the endpoints associated with this node type</xs:documentation>
                    </xs:annotation>
                </xs:element>
            <xs:element name="KtlLoggerSettings"
type="FabricKtlLoggerSettingsType" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>Describe the KtlLogger information associated with this node type</xs:documentation>
                    </xs:annotation>
                </xs:element>
            </xs:all>
        </xs:complexType>
    </xs:element>
```

```

        </xs:element>
        <xs:element name="LogicalDirectories"
minOccurs="0">
            <xs:annotation>
                <xs:documentation>Describe the
LogicalDirectories settings associated with this node type</xs:documentation>
            </xs:annotation>
            <xs:complexType>
                <xs:sequence>
                    <xs:element
name="LogicalDirectory" type="LogicalDirectoryType" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="Certificates"
type="CertificatesType" minOccurs="0">
            <xs:annotation>

<xs:documentation>Describe the certificates associated with this node type</xs:documentation>
            </xs:annotation>
            <xs:element
name="PlacementProperties"
minOccurs="0">
            <xs:annotation>

<xs:documentation>Describe the properties for this NodeType that will be used as placement
constraints</xs:documentation>
            </xs:annotation>
            <xs:complexType>
                <xs:sequence>
                    <xs:element
name="Property" type="KeyValuePairType" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="Capacities"
minOccurs="0">
            <xs:annotation>
                <xs:documentation>The
capacities of various metrics for this node type</xs:documentation>
            </xs:annotation>
            <xs:complexType>
                <xs:sequence>
                    <xs:element
name="Capacity" type="KeyValuePairType" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        </xs:all>
        <xs:attribute name="Name" type="xs:string"
use="required">
            <xs:annotation>
                <xs:documentation>Name of the
NodeType</xs:documentation>
            </xs:annotation>
            </xs:attribute>
        </xs:complexType>
    </xs:element>

```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name

ATTRIBUTE	VALUE
type	xs:string
use	required

## Content element details

### Endpoints

Describe the endpoints associated with this node type

ATTRIBUTE	VALUE
name	Endpoints
type	FabricEndpointsType
minOccurs	0

### KtlLoggerSettings

Describe the KtlLogger information associated with this node type

ATTRIBUTE	VALUE
name	KtlLoggerSettings
type	FabricKtlLoggerSettingsType
minOccurs	0

### LogicalDirectories

Describe the LogicalDirectories settings associated with this node type

ATTRIBUTE	VALUE
name	LogicalDirectories
minOccurs	0

### Certificates

Describe the certificates associated with this node type

ATTRIBUTE	VALUE
name	Certificates
type	CertificatesType
minOccurs	0

### PlacementProperties

Describe the properties for this NodeType that will be used as placement constraints

ATTRIBUTE	VALUE
name	PlacementProperties
minOccurs	0

#### Capacities

The capacities of various metrics for this node type

ATTRIBUTE	VALUE
name	Capacities
minOccurs	0

## NodeTypes Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	NodeTypes
minOccurs	1

#### XML source

```

<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="NodeTypes" minOccurs="1">
    <xss:complexType>
        <xss:sequence>
            <xss:element name="NodeType" maxOccurs="unbounded">
                <xss:annotation>
                    <xss:documentation>Describe a node type.
                </xss:documentation>
                </xss:annotation>
                <xss:complexType>
                    <xss:all>
                        <xss:element name="Endpoints"
type="FabricEndpointsType" minOccurs="0">
                            <xss:annotation>
                                <xss:documentation>Describe the endpoints associated with this node type</xss:documentation>
                                </xss:annotation>
                            </xss:element>
                            <xss:element name="KtlLoggerSettings"
type="FabricKtlLoggerSettingsType" minOccurs="0">
                                <xss:annotation>
                                    <xss:documentation>Describe the KtlLogger information associated with this node type</xss:documentation>
                                    </xss:annotation>
                                </xss:element>
                                <xss:element name="LogicalDirectories"
minOccurs="0">
                                    <xss:annotation>
  <xss:documentation>Describe the LogicalDirectories settings associated with this node type</xss:documentation>
                                    </xss:annotation>
                                </xss:element>
                            </xss:all>
                        </xss:complexType>
                    </xss:all>
                </xss:complexType>
            </xss:element>
        </xss:sequence>
    </xss:complexType>
</xss:element>

```

```

                </xs:annotation>
                <xs:complexType>
                    <xs:sequence>
                        <xs:element
name="LogicalDirectory" type="LogicalDirectoryType" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="Certificates"
type="CertificatesType" minOccurs="0">
                <xs:annotation>

<xs:documentation>Describe the certificates associated with this node type</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="PlacementProperties"
minOccurs="0">
                <xs:annotation>

<xs:documentation>Describe the properties for this NodeType that will be used as placement
constraints</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:sequence>
                        <xs:element
name="Property" type="KeyValuePairType" minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="Capacities"
minOccurs="0">
                <xs:annotation>
                    <xs:documentation>The
capacities of various metrics for this node type</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:sequence>
                        <xs:element
name="Capacity" type="KeyValuePairType" minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            </xs:all>
            <xs:attribute name="Name" type="xs:string"
use="required">
                <xs:annotation>
                    <xs:documentation>Name of the
NodeType</xs:documentation>
                </xs:annotation>
                </xs:attribute>
            </xs:complexType>
        </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>

```

## Content element details

### NodeType

Describe a node type.

ATTRIBUTE	VALUE
name	NodeType

ATTRIBUTE	VALUE
maxOccurs	unbounded

## PaaS Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 0 attribute(s)
name	PaaS

### XML source

```

<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="PaaS">
    <xss:complexType>
        <xss:all>
            <xss:element name="Roles">
                <xss:complexType>
                    <xss:sequence>
                        <xss:element
name="Role" type="PaaSRoleType" maxOccurs="unbounded"/>
                    </xss:sequence>
                </xss:complexType>
            </xss:element>
            <xss:element name="Votes">
                <xss:complexType>
                    <xss:sequence>
                        <xss:element
name="Vote" type="PaaSVoteType" maxOccurs="unbounded"/>
                    </xss:sequence>
                </xss:complexType>
            </xss:element>
        </xss:all>
    </xss:complexType>
</xss:element>

```

### Content element details

#### Roles

ATTRIBUTE	VALUE
name	Roles

#### Votes

ATTRIBUTE	VALUE
name	Votes

## PackageSharingPolicy Element

ATTRIBUTE	VALUE
type	PackageSharingPolicyType
content	0 element(s), 0 attribute(s)
name	PackageSharingPolicy
minOccurs	0

#### XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="PackageSharingPolicy" type="PackageSharingPolicyType" minOccurs="0"/>
```

## Parameter Element

An application parameter to be used in this manifest. The parameter value can be changed during application instantiation, or, if no value is supplied the default value is used.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	Parameter
block	
minOccurs	0
maxOccurs	unbounded

#### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Parameter" block="" minOccurs="0" maxOccurs="unbounded">
    <xss:annotation>
        <xss:documentation>An application parameter to be used in this manifest. The parameter value can
be changed during application instantiation, or, if no value is supplied the default value is used.
    </xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:attribute name="Name" use="required">
            <xss:annotation>
                <xss:documentation>The name of the parameter to be used in the manifest as "[Name]".
            </xss:documentation>
        </xss:attribute>
        <xss:simpleType>
            <xss:restriction base="xs:string">
                <xs:minLength value="1"/>
            </xss:restriction>
        </xss:simpleType>
        </xss:attribute>
        <xss:attribute name="DefaultValue" type="xs:string" use="required">
            <xss:annotation>
                <xss:documentation>Default value for the parameter, used if the parameter value is not
provided during application instantiation.</xss:documentation>
            </xss:annotation>
        </xss:attribute>
    </xss:complexType>
</xss:element>

```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
use	required

### DefaultValue

ATTRIBUTE	VALUE
name	DefaultValue
type	xs:string
use	required

## Parameter Element

The setting to override.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)
name	Parameter

ATTRIBUTE	VALUE
minOccurs	0
maxOccurs	unbounded

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Parameter" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>The setting to override.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:attributeGroup ref="NameValuePair"/>
        <xs:attribute name="IsEncrypted" type="xs:boolean" default="false">
            <xs:annotation>
                <xs:documentation>
                    If true, the value of this parameter is encrypted. The application developer is
                    responsible for creating a certificate and using the Invoke-ServiceFabricEncryptSecret cmdlet to encrypt
                    sensitive information. The certificate information that will be used to encrypt the value is specified in the
                    Certificates section.
                </xs:documentation>
            </xs:annotation>
            </xs:attribute>
        </xs:complexType>
    </xs:element>

```

## Attribute details

### IsEncrypted

ATTRIBUTE	VALUE
name	IsEncrypted
type	xs:boolean
default	false

## Parameter Element

ATTRIBUTE	VALUE
type	ParameterType
content	0 element(s), 0 attribute(s)
name	Parameter
minOccurs	1
maxOccurs	unbounded

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Parameter" type="ParameterType" minOccurs="1" maxOccurs="unbounded"/>
```

## Parameter Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 4 attribute(s)
name	Parameter
minOccurs	0
maxOccurs	unbounded

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Parameter" minOccurs="0" maxOccurs="unbounded">
    <xss:complexType>
        <xss:attribute name="Name" type="xs:string" use="required"/>
        <xss:attribute name="Value" type="xs:string" use="required"/>
        <xss:attribute name="MustOverride" type="xs:boolean" default="false">
            <xss:annotation>
                <xss:documentation>If true, the value of this parameter must be overridden by higher level
configuration.</xss:documentation>
            </xss:annotation>
        </xss:attribute>
        <xss:attribute name="IsEncrypted" type="xs:boolean" default="false">
            <xss:annotation>
                <xss:documentation>If true, the value of this parameter is encrypted.</xss:documentation>
            </xss:annotation>
        </xss:attribute>
    </xss:complexType>
</xss:element>
```

### Attribute details

#### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

#### Value

ATTRIBUTE	VALUE
name	Value

ATTRIBUTE	VALUE
type	xs:string
use	required

#### MustOverride

ATTRIBUTE	VALUE
name	MustOverride
type	xs:boolean
default	false

#### IsEncrypted

ATTRIBUTE	VALUE
name	IsEncrypted
type	xs:boolean
default	false

## Parameter Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	Parameter
minOccurs	0
maxOccurs	unbounded

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Parameter" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
        <xs:attributeGroup ref="NameValuePair"/>
    </xs:complexType>
</xs:element>
```

## Parameters Element

Declares the parameters that are used in this application manifest. The value of these parameters can be supplied when the application is instantiated and can be used to override application or service configuration settings.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	Parameters
minOccurs	0

## XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Parameters" minOccurs="0">
    <xss:annotation>
        <xss:documentation>Declares the parameters that are used in this application manifest. The value of
these parameters can be supplied when the application is instantiated and can be used to override application
or service configuration settings.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence>
            <xss:element name="Parameter" block="" minOccurs="0" maxOccurs="unbounded">
                <xss:annotation>
                    <xss:documentation>An application parameter to be used in this manifest. The parameter value can
be changed during application instantiation, or, if no value is supplied the default value is used.
                </xss:documentation>
                </xss:annotation>
                <xss:complexType>
                    <xss:attribute name="Name" use="required">
                        <xss:annotation>
                            <xss:documentation>The name of the parameter to be used in the manifest as "[Name]".
                        </xss:documentation>
                    </xss:attribute>
                    <xss:attribute name="DefaultValue" type="xs:string" use="required">
                        <xss:annotation>
                            <xss:documentation>Default value for the parameter, used if the parameter value is not
provided during application instantiation.</xss:documentation>
                        </xss:annotation>
                    </xss:attribute>
                </xss:complexType>
            </xss:element>
        </xss:sequence>
    </xss:complexType>
</xss:element>

```

## Content element details

### Parameter

An application parameter to be used in this manifest. The parameter value can be changed during application instantiation, or, if no value is supplied the default value is used.

ATTRIBUTE	VALUE
name	Parameter

ATTRIBUTE	VALUE
block	
minOccurs	0
maxOccurs	unbounded

## Parameters Element

Additional settings specified as name-value pairs

ATTRIBUTE	VALUE
type	ParametersType
content	0 element(s), 0 attribute(s)
name	Parameters

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Parameters" type="ParametersType">
    <xs:annotation>
        <xs:documentation>Additional settings specified as name-value pairs</xs:documentation>
    </xs:annotation>
</xs:element>
```

## Parameters Element

List of parameters for the application as defined in application manifest and their respective values.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	Parameters

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Parameters">
  <xss:annotation>
    <xss:documentation>List of parameters for the application as defined in application manifest and their
respective values.</xss:documentation>
  </xss:annotation>
  <xss:complexType>
    <xss:sequence>
      <xss:element name="Parameter" minOccurs="0" maxOccurs="unbounded">
        <xss:complexType>
          <xss:attributeGroup ref="NameValuePair"/>
        </xss:complexType>
      </xss:element>
    </xss:sequence>
  </xss:complexType>
</xss:element>

```

## Content element details

### Parameter

ATTRIBUTE	VALUE
name	Parameter
minOccurs	0
maxOccurs	unbounded

## Partition Element

Describes a partition by name.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)
name	Partition

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Partition">
    <xss:annotation>
        <xss:documentation>Describes a partition by name.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:attribute name="Name" use="required">
            <xss:annotation>
                <xss:documentation>The name of the partition</xss:documentation>
            </xss:annotation>
            <xss:simpleType>
                <xss:restriction base="xs:string">
                    <xss:minLength value="1"/>
                </xss:restriction>
            </xss:simpleType>
        </xss:attribute>
    </xss:complexType>
</xss:element>

```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
use	required

## PersistencePolicy Element

Persistence Policy extension for the Service Type

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 4 attribute(s)
name	PersistencePolicy

### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="PersistencePolicy">
  <xs:annotation>
    <xs:documentation>Persistence Policy extension for the Service Type</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Properties" type="ServiceTypeExtensionPolicyPropertiesType" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="Name" type="xs:string" use="required"/>
    <xs:attribute name="Mode" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Synchronous"/>
          <xs:enumeration value="Asynchronous"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="WriteBehind" type="xs:string" use="required"/>
    <xs:attribute name="Provider" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

### Mode

ATTRIBUTE	VALUE
name	Mode
use	required

### WriteBehind

ATTRIBUTE	VALUE
name	WriteBehind
type	xs:string
use	required

### Provider

ATTRIBUTE	VALUE
name	Provider

ATTRIBUTE	VALUE
type	xs:string
use	required

## Content element details

### Properties

ATTRIBUTE	VALUE
name	Properties
type	ServiceTypeExtensionPolicyPropertiesType
minOccurs	0

## PlacementConstraints Element

Used to control which nodes in the cluster a service can run on. A key/value pair which describes the node property name and the service's requirements for the value. Individual statements can be grouped together with simple boolean logic to create the necessary constraint. For example, "(FirmwareVersion>12 && InDMZ == True)".

ATTRIBUTE	VALUE
type	xs:string
content	0 element(s), 0 attribute(s)
name	PlacementConstraints
minOccurs	0

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="PlacementConstraints" type="xs:string" minOccurs="0">
    <xss:annotation>
        <xss:documentation>Used to control which nodes in the cluster a service can run on. A
key/value pair which describes the node property name and the service's requirements for the value. Individual
statements can be grouped together with simple boolean logic to create the necessary constraint. For example, "
(FirmwareVersion>12 && InDMZ == True)".</xss:documentation>
    </xss:annotation>
</xss:element>
```

## PlacementConstraints Element

Used to control which nodes in the cluster a service can run on. A key/value pair which describes the node property name and the service's requirements for the value. Individual statements can be grouped together with simple boolean logic to create the necessary constraint. For example, "(FirmwareVersion>12 && InDMZ == True)".

ATTRIBUTE	VALUE
type	xs:string
content	0 element(s), 0 attribute(s)
name	PlacementConstraints
minOccurs	0

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="PlacementConstraints" type="xs:string" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Used to control which nodes in the cluster a service can run on. A key/value pair
which describes the node property name and the service's requirements for the value. Individual statements can
be grouped together with simple boolean logic to create the necessary constraint. For example,
(FirmwareVersion>12 && InDMZ == True)".</xs:documentation>
    </xs:annotation>
</xs:element>
```

## PlacementConstraints Element

Constraints for the placement of services that are part of this package.

ATTRIBUTE	VALUE
type	xs:string
content	0 element(s), 0 attribute(s)
name	PlacementConstraints
minOccurs	0

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="PlacementConstraints" type="xs:string" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Constraints for the placement of services that are part of this package.
    </xs:documentation>
    </xs:annotation>
</xs:element>
```

## PlacementProperties Element

Describe the properties for this NodeType that will be used as placement constraints

ATTRIBUTE	VALUE
type	anonymous complexType

ATTRIBUTE	VALUE
content	1 element(s), 0 attribute(s)
name	PlacementProperties
minOccurs	0

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="PlacementProperties" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Describe the properties for this NodeType that will be used as placement
constraints</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element
name="Property" type="KeyValuePairType" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

## Content element details

### Property

ATTRIBUTE	VALUE
name	Property
type	KeyValuePairType
minOccurs	0
maxOccurs	unbounded

## Policies Element

ATTRIBUTE	VALUE
type	ServiceManifestImportPoliciesType
content	0 element(s), 0 attribute(s)
name	Policies
minOccurs	0

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Policies" type="ServiceManifestImportPoliciesType" minOccurs="0"/>
```

## Policies Element

ATTRIBUTE	VALUE
type	ApplicationPoliciesType
content	0 element(s), 0 attribute(s)
name	Policies
minOccurs	0

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="Policies" type="ApplicationPoliciesType" minOccurs="0"/>
```

## Policies Element

ATTRIBUTE	VALUE
type	ApplicationPoliciesType
content	0 element(s), 0 attribute(s)
name	Policies

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="Policies" type="ApplicationPoliciesType"/>
```

## PortBinding Element

Specifies which endpoint resource to bind container exposed port.

ATTRIBUTE	VALUE
type	PortBindingType
content	0 element(s), 0 attribute(s)
name	PortBinding
minOccurs	0
maxOccurs	unbounded

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="PortBinding" type="PortBindingType" minOccurs="0" maxOccurs="unbounded">
    <xss:annotation>
        <xss:documentation>Specifies which endpoint resource to bind container exposed port.
    </xss:documentation>
    </xss:annotation>
</xss:element>

```

## Principals Element

ATTRIBUTE	VALUE
type	SecurityPrincipalsType
content	0 element(s), 0 attribute(s)
name	Principals
minOccurs	0

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Principals" type="SecurityPrincipalsType" minOccurs="0"/>

```

## Principals Element

ATTRIBUTE	VALUE
type	SecurityPrincipalsType
content	0 element(s), 0 attribute(s)
name	Principals

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Principals" type="SecurityPrincipalsType"/>

```

## Program Element

The executable name. For example, "MySetup.bat" or "MyServiceHost.exe".

ATTRIBUTE	VALUE
type	xs:string
content	0 element(s), 0 attribute(s)
name	Program

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Program" type="xs:string">
  <xs:annotation>
    <xs:documentation>The executable name. For example, "MySetup.bat" or "MyServiceHost.exe".</xs:documentation>
  </xs:annotation>
</xs:element>
```

## Properties Element

ATTRIBUTE	VALUE
type	ServiceTypeExtensionPolicyPropertiesType
content	0 element(s), 0 attribute(s)
name	Properties
minOccurs	0

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Properties" type="ServiceTypeExtensionPolicyPropertiesType" minOccurs="0"/>
```

## Properties Element

ATTRIBUTE	VALUE
type	ServiceTypeExtensionPolicyPropertiesType
content	0 element(s), 0 attribute(s)
name	Properties
minOccurs	0

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Properties" type="ServiceTypeExtensionPolicyPropertiesType" minOccurs="0"/>
```

## Property Element

ATTRIBUTE	VALUE
type	KeyValuePairType
content	0 element(s), 0 attribute(s)

ATTRIBUTE	VALUE
name	Property
minOccurs	0
maxOccurs	unbounded

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Property" type="KeyValuePairType" minOccurs="0" maxOccurs="unbounded"/>
```

## Property Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	Property
minOccurs	0
maxOccurs	unbounded

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Property" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:attribute name="Name" type="xs:string" use="required"/>
    <xs:attribute name="Value" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

### Value

ATTRIBUTE	VALUE
name	Value

ATTRIBUTE	VALUE
type	xs:string
use	required

## ProviderGuid Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)
name	ProviderGuid
minOccurs	0
maxOccurs	unbounded

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ProviderGuid" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
        <xs:attribute name="Value" use="required">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:pattern value="[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-
fA-F0-9]{12}">
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
</xs:element>
```

### Attribute details

#### Value

ATTRIBUTE	VALUE
name	Value
use	required

## ProviderGuids Element

Lists the ETW provider GUIDs for the components of this service manifest.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)

ATTRIBUTE	VALUE
name	ProviderGuids
minOccurs	0

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ProviderGuids" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Lists the ETW provider GUIDs for the components of this service manifest.
    </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="ProviderGuid" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:attribute name="Value" use="required">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:pattern value="[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-
fA-F0-9]{12}">
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:attribute>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>

```

## Content element details

### ProviderGuid

ATTRIBUTE	VALUE
name	ProviderGuid
minOccurs	0
maxOccurs	unbounded

## RepairManagerReplicatorEndpoint Element

ATTRIBUTE	VALUE
type	InternalEndpointType
content	0 element(s), 0 attribute(s)
name	RepairManagerReplicatorEndpoint
minOccurs	0

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="RepairManagerReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
```

## RepositoryCredentials Element

Credentials for container image repository to pull images from.

ATTRIBUTE	VALUE
type	RepositoryCredentialsType
content	0 element(s), 0 attribute(s)
name	RepositoryCredentials
minOccurs	0
maxOccurs	1

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="RepositoryCredentials" type="RepositoryCredentialsType" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>Credentials for container image repository to pull images from.</xs:documentation>
  </xs:annotation>
</xs:element>
```

## ResourceGovernancePolicy Element

Specifies resource limits for codepackage.

ATTRIBUTE	VALUE
type	ResourceGovernancePolicyType
content	0 element(s), 0 attribute(s)
name	ResourceGovernancePolicy
minOccurs	0

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ResourceGovernancePolicy" type="ResourceGovernancePolicyType" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Specifies resource limits for codepackage.</xs:documentation>
  </xs:annotation>
</xs:element>
```

## ResourceGovernancePolicy Element

Specifies resource limits for codepackage.

ATTRIBUTE	VALUE
type	ResourceGovernancePolicyType
content	0 element(s), 0 attribute(s)
name	ResourceGovernancePolicy
minOccurs	0

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ResourceGovernancePolicy" type="ResourceGovernancePolicyType" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Specifies resource limits for codepackage.</xs:documentation>
    </xs:annotation>
</xs:element>
```

## ResourceGovernancePolicy Element

ATTRIBUTE	VALUE
type	ResourceGovernancePolicyType
content	0 element(s), 0 attribute(s)
name	ResourceGovernancePolicy
minOccurs	0
maxOccurs	1

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ResourceGovernancePolicy" type="ResourceGovernancePolicyType" minOccurs="0" maxOccurs="1"/>
```

## ResourceOverrides Element

ATTRIBUTE	VALUE
type	ResourceOverridesType
content	0 element(s), 0 attribute(s)
name	ResourceOverrides
minOccurs	0

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ResourceOverrides" type="ResourceOverridesType" minOccurs="0"/>
```

## Resources Element

ATTRIBUTE	VALUE
type	ResourcesType
content	0 element(s), 0 attribute(s)
name	Resources
minOccurs	0

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Resources" type="ResourcesType" minOccurs="0"/>
```

## Role Element

ATTRIBUTE	VALUE
type	AzureRoleType
content	0 element(s), 0 attribute(s)
name	Role
maxOccurs	unbounded

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Role" type="AzureRoleType" maxOccurs="unbounded"/>
```

## Role Element

ATTRIBUTE	VALUE
type	BlackbirdRoleType
content	0 element(s), 0 attribute(s)
name	Role
minOccurs	1

ATTRIBUTE	VALUE
maxOccurs	unbounded

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Role" type="BlackbirdRoleType" minOccurs="1" maxOccurs="unbounded"/>
```

## Role Element

ATTRIBUTE	VALUE
type	PaaSRoleType
content	0 element(s), 0 attribute(s)
name	Role
maxOccurs	unbounded

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Role" type="PaaSRoleType" maxOccurs="unbounded"/>
```

## Roles Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	Roles

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Roles">
    <xss:complexType>
        <xss:sequence>
            <xss:element
name="Role" type="AzureRoleType" maxOccurs="unbounded"/>
        </xss:sequence>
    </xss:complexType>
</xss:element>
```

## Content element details

### Role

ATTRIBUTE	VALUE
name	Role
type	AzureRoleType
maxOccurs	unbounded

## Roles Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	Roles

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="Roles">
  <xss:complexType>
    <xss:sequence>
      <xss:element
        name="Role" type="BlackbirdRoleType" minOccurs="1" maxOccurs="unbounded"/>
    </xss:sequence>
  </xss:complexType>
</xss:element>
```

## Content element details

### Role

ATTRIBUTE	VALUE
name	Role
type	BlackbirdRoleType
minOccurs	1
maxOccurs	unbounded

## Roles Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	Roles

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Roles">
    <xs:complexType>
        <xs:sequence>
            <xs:element
name="Role" type="PaaSRoleType" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

## Content element details

### Role

ATTRIBUTE	VALUE
name	Role
type	PaaSRoleType
maxOccurs	unbounded

## RunAsPolicy Element

ATTRIBUTE	VALUE
type	RunAsPolicyType
content	0 element(s), 0 attribute(s)
name	RunAsPolicy
minOccurs	0

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="RunAsPolicy" type="RunAsPolicyType" minOccurs="0"/>
```

## RunAsPolicy Element

ATTRIBUTE	VALUE
type	RunAsPolicyType
content	0 element(s), 0 attribute(s)
name	RunAsPolicy
minOccurs	0
maxOccurs	2

## XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric" name="RunAsPolicy" type="RunAsPolicyType" minOccurs="0" maxOccurs="2"/>
```

## RunFrequency Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)
name	RunFrequency
minOccurs	0

## XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric" name="RunFrequency" minOccurs="0">
    <xss:complexType>
        <xss:attribute name="IntervalInSeconds" use="required">
            <xss:simpleType>
                <xss:restriction base="xs:int">
                    <xss:minInclusive value="0"/>
                    <xss:maxInclusive value="2147483647"/>
                </xss:restriction>
            </xss:simpleType>
        </xss:attribute>
    </xss:complexType>
</xss:element>
```

## Attribute details

### IntervalInSeconds

ATTRIBUTE	VALUE
name	IntervalInSeconds
use	required

## SecretsCertificate Element

ATTRIBUTE	VALUE
type	FabricCertificateType
content	0 element(s), 0 attribute(s)
name	SecretsCertificate
minOccurs	0

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="SecretsCertificate" type="FabricCertificateType" minOccurs="0"/>
```

## SecretsCertificate Element

Declares a certificate used to encrypt sensitive information within the application manifest. The application author uses the Invoke-ServiceFabricEncryptSecret cmdlet to encrypt the sensitive information, which is copied to a Parameter in the ConfigOverrides section.

ATTRIBUTE	VALUE
type	FabricCertificateType
content	0 element(s), 0 attribute(s)
name	SecretsCertificate
minOccurs	0

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="SecretsCertificate" type="FabricCertificateType" minOccurs="0">
    <xss:annotation>
        <xss:documentation>Declares a certificate used to encrypt sensitive information within the
application manifest. The application author uses the Invoke-ServiceFabricEncryptSecret cmdlet to encrypt the
sensitive information, which is copied to a Parameter in the ConfigOverrides section.</xss:documentation>
    </xss:annotation>
</xss:element>
```

## SecretsCertificate Element

ATTRIBUTE	VALUE
type	FabricCertificateType
content	0 element(s), 0 attribute(s)
name	SecretsCertificate
minOccurs	0

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="SecretsCertificate" type="FabricCertificateType" minOccurs="0"/>
```

## Section Element

A section in the Settings.xml file to override.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 1 attribute(s)
name	Section
maxOccurs	unbounded

## XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Section" maxOccurs="unbounded">
    <xss:annotation>
        <xss:documentation>A section in the Settings.xml file to override.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence>
            <xss:element name="Parameter" minOccurs="0" maxOccurs="unbounded">
                <xss:annotation>
                    <xss:documentation>The setting to override.</xss:documentation>
                </xss:annotation>
                <xss:complexType>
                    <xss:attributeGroup ref="NameValuePair"/>
                    <xss:attribute name="IsEncrypted" type="xs:boolean" default="false">
                        <xss:annotation>
                            <xss:documentation>
                                If true, the value of this parameter is encrypted. The application developer is
                                responsible for creating a certificate and using the Invoke-ServiceFabricEncryptSecret cmdlet to encrypt
                                sensitive information. The certificate information that will be used to encrypt the value is specified in the
                                Certificates section.
                            </xss:documentation>
                        </xss:annotation>
                    </xss:complexType>
                </xss:element>
            </xss:sequence>
            <xss:attribute name="Name" use="required">
                <xss:annotation>
                    <xss:documentation>The name of the section in the Settings.xml file to override.</xss:documentation>
                </xss:annotation>
                <xss:simpleType>
                    <xss:restriction base="xs:string">
                        <xss:minLength value="1"/>
                    </xss:restriction>
                </xss:simpleType>
            </xss:attribute>
        </xss:complexType>
    </xss:element>

```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
use	required

## Content element details

### Parameter

The setting to override.

ATTRIBUTE	VALUE
name	Parameter
minOccurs	0
maxOccurs	unbounded

## Section Element

A user defined named section.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 1 attribute(s)
name	Section
minOccurs	0
maxOccurs	unbounded

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Section" minOccurs="0" maxOccurs="unbounded">
    <xss:annotation>
        <xss:documentation>A user defined named section.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence>
            <xss:element name="Parameter" minOccurs="0" maxOccurs="unbounded">
                <xss:complexType>
                    <xss:attribute name="Name" type="xs:string" use="required"/>
                    <xss:attribute name="Value" type="xs:string" use="required"/>
                    <xss:attribute name="MustOverride" type="xs:boolean" default="false">
                        <xss:annotation>
                            <xss:documentation>If true, the value of this parameter must be overridden by higher level
configuration.</xss:documentation>
                        </xss:annotation>
                    </xss:attribute>
                    <xss:attribute name="IsEncrypted" type="xs:boolean" default="false">
                        <xss:annotation>
                            <xss:documentation>If true, the value of this parameter is encrypted.</xss:documentation>
                        </xss:annotation>
                    </xss:attribute>
                </xss:complexType>
            </xss:element>
        </xss:sequence>
        <xss:attribute name="Name" type="xs:string" use="required"/>
    </xss:complexType>
</xss:element>
```

## Attribute details

### Name

The setting to override.

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

## Content element details

### Parameter

ATTRIBUTE	VALUE
name	Parameter
minOccurs	0
maxOccurs	unbounded

# SecurityAccessPolicies Element

List of security policies applied to resources at the application level.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	SecurityAccessPolicies
minOccurs	0

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="SecurityAccessPolicies" minOccurs="0">
    <xss:annotation>
        <xss:documentation>List of security policies applied to resources at the
application level.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence maxOccurs="unbounded">
            <xss:element name="SecurityAccessPolicy"
type="SecurityAccessPolicyType"/>
        </xss:sequence>
    </xss:complexType>
</xss:element>
```

## Content element details

### SecurityAccessPolicy

ATTRIBUTE	VALUE
name	SecurityAccessPolicy
type	SecurityAccessPolicyType

## SecurityAccessPolicy Element

ATTRIBUTE	VALUE
type	SecurityAccessPolicyType
content	0 element(s), 0 attribute(s)
name	SecurityAccessPolicy
minOccurs	0

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="SecurityAccessPolicy" type="SecurityAccessPolicyType" minOccurs="0"/>
```

## SecurityAccessPolicy Element

ATTRIBUTE	VALUE
type	SecurityAccessPolicyType
content	0 element(s), 0 attribute(s)
name	SecurityAccessPolicy

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="SecurityAccessPolicy" type="SecurityAccessPolicyType"/>
```

## SecurityAccessPolicy Element

ATTRIBUTE	VALUE
type	SecurityAccessPolicyType
content	0 element(s), 0 attribute(s)
name	SecurityAccessPolicy
minOccurs	0

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="SecurityAccessPolicy" type="SecurityAccessPolicyType" minOccurs="0"/>
```

## SecurityOption Element

Specifies securityoptions for the container.

ATTRIBUTE	VALUE
type	SecurityOptionsType
content	0 element(s), 0 attribute(s)
name	SecurityOption
minOccurs	0
maxOccurs	unbounded

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="SecurityOption" type="SecurityOptionsType" minOccurs="0" maxOccurs="unbounded">
    <xss:annotation>
        <xss:documentation>Specifies securityoptions for the container.</xss:documentation>
    </xss:annotation>
</xss:element>
```

## ServerCertificate Element

The certificate used to secure the intra cluster communication.

ATTRIBUTE	VALUE
type	FabricCertificateType
content	0 element(s), 0 attribute(s)
name	ServerCertificate
minOccurs	0

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServerCertificate" type="FabricCertificateType" minOccurs="0">
    <xss:annotation>
        <xss:documentation>The certificate used to secure the intra cluster communication.</xss:documentation>
    </xss:annotation>
</xss:element>
```

# Service Element

Declares a service to be created automatically when the application is instantiated.

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 4 attribute(s)
name	Service

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Service">
    <xs:annotation>
        <xs:documentation>Declares a service to be created automatically when the application
is instantiated.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:choice minOccurs="0">
                <xs:element name="StatelessService" type="StatelessServiceType"/>
                <xs:element name="StatefulService" type="StatefulServiceType"/>
            </xs:choice>
            <xs:attribute name="Name" type="xs:string" use="required">
                <xs:annotation>
                    <xs:documentation>The service name, used to form the fully qualified
application name URI. The fully qualified name URI of the service would be:
fabric:/ApplicationName/ServiceName.</xs:documentation>
                </xs:annotation>
            </xs:attribute>
            <xs:attribute name="GeneratedIdRef" type="xs:string" use="optional">
                <xs:annotation>
                    <xs:documentation>Reference to the auto generated id used by Visual Studio
tooling.</xs:documentation>
                </xs:annotation>
            </xs:attribute>
            <xs:attribute name="ServiceDnsName" type="xs:string" use="optional">
                <xs:annotation>
                    <xs:documentation>The DNS name of the service.</xs:documentation>
                </xs:annotation>
            </xs:attribute>
            <xs:attribute name="ServicePackageActivationMode" use="optional"
default="SharedProcess">
                <xs:annotation>
                    <xs:documentation>ServicePackageActivationMode to be used when creating the
service. With SharedProcess mode, replica(s) or instance(s) from different partition(s) of service will share
same activation of service package on a node. With ExclusiveProcess mode, each replica or instance of
service will have its own dedicated activation of service package.</xs:documentation>
                </xs:annotation>
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="SharedProcess"/>
                        <xs:enumeration value="ExclusiveProcess"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
        </xs:complexType>
    </xs:element>
```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

#### GeneratedIdRef

ATTRIBUTE	VALUE
name	GeneratedIdRef
type	xs:string
use	optional

#### ServiceDnsName

ATTRIBUTE	VALUE
name	ServiceDnsName
type	xs:string
use	optional

#### ServicePackageActivationMode

ATTRIBUTE	VALUE
name	ServicePackageActivationMode
use	optional
default	SharedProcess

### Content element details

#### StatelessService

ATTRIBUTE	VALUE
name	StatelessService
type	StatelessServiceType

#### StatefulService

ATTRIBUTE	VALUE
name	StatefulService
type	StatefulServiceType

## ServiceConnectionEndpoint Element

ATTRIBUTE	VALUE
type	InternalEndpointType
content	0 element(s), 0 attribute(s)
name	ServiceConnectionEndpoint
minOccurs	0

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="ServiceConnectionEndpoint" type="InternalEndpointType" minOccurs="0"/>
```

## ServiceCorrelation Element

Defines an affinity relationship with another service. Useful when splitting a previously-monolithic application into microservices. One service has a local dependency on another service and both services need to run on the same node in order to work.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	ServiceCorrelation
maxOccurs	unbounded

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceCorrelation" maxOccurs="unbounded">
    <xss:annotation>
        <xss:documentation>Defines an affinity relationship with another service. Useful
when splitting a previously-monolithic application into microservices. One service has a local dependency on
another service and both services need to run on the same node in order to work.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:attribute name="ServiceName" use="required">
            <xss:annotation>
                <xss:documentation>The name of the other service as a URI. Example,
"fabric:/otherApplication/parentService".</xss:documentation>
            </xss:annotation>
            <xss:simpleType>
                <xss:restriction base="xs:string">
                    <xss:minLength value="1"/>
                </xss:restriction>
            </xss:simpleType>
        </xss:attribute>
        <xss:attribute name="Scheme" use="required">
            <xss:annotation>
                <xss:documentation>In NonAlignedAffinity the replicas or instances of
the different services are placed on the same nodes. AlignedAffinity is used with stateful services.
Configuring one stateful service as having aligned affinity with another stateful service ensures that the
primaries of those services are placed on the same nodes as each other, and that each pair of secondaries are
also placed on the same nodes.</xss:documentation>
            </xss:annotation>
            <xss:simpleType>
                <xss:restriction base="xs:string">
                    <xss:enumeration value="Affinity"/>
                    <xss:enumeration value="AlignedAffinity"/>
                    <xss:enumeration value="NonAlignedAffinity"/>
                </xss:restriction>
            </xss:simpleType>
        </xss:attribute>
    </xss:complexType>
</xss:element>

```

## Attribute details

### ServiceName

ATTRIBUTE	VALUE
name	ServiceName
use	required

### Scheme

ATTRIBUTE	VALUE
name	Scheme
use	required

## ServiceCorrelations Element

Defines affinity relationships between services.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	ServiceCorrelations
minOccurs	0

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceCorrelations" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Defines affinity relationships between services.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="ServiceCorrelation" maxOccurs="unbounded">
                <xs:annotation>
                    <xs:documentation>Defines an affinity relationship with another service. Useful
when splitting a previously-monolithic application into microservices. One service has a local dependency on
another service and both services need to run on the same node in order to work.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:attribute name="ServiceName" use="required">
                        <xs:annotation>
                            <xs:documentation>The name of the other service as a URI. Example,
"fabric:/otherApplication/parentService".</xs:documentation>
                        </xs:annotation>
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:minLength value="1"/>
                        </xs:restriction>
                    </xs:simpleType>
                    </xs:attribute>
                    <xs:attribute name="Scheme" use="required">
                        <xs:annotation>
                            <xs:documentation>In NonAlignedAffinity the replicas or instances of
the different services are placed on the same nodes. AlignedAffinity is used with stateful services.
Configuring one stateful service as having aligned affinity with another stateful service ensures that the
primaries of those services are placed on the same nodes as each other, and that each pair of secondaries are
also placed on the same nodes.</xs:documentation>
                        </xs:annotation>
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="Affinity"/>
                            <xs:enumeration value="AlignedAffinity"/>
                            <xs:enumeration value="NonAlignedAffinity"/>
                        </xs:restriction>
                    </xs:simpleType>
                    </xs:attribute>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>

```

## Content element details

### ServiceCorrelation

Defines an affinity relationship with another service. Useful when splitting a previously-monolithic application into

microservices. One service has a local dependency on another service and both services need to run on the same node in order to work.

ATTRIBUTE	VALUE
name	ServiceCorrelation
maxOccurs	unbounded

## ServiceGroup Element

A collection of services that are automatically located together, so they are also moved together during fail-over or resource management.

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 2 attribute(s)
name	ServiceGroup

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceGroup">
    <xss:annotation>
        <xss:documentation>A collection of services that are automatically located together, so
they are also moved together during fail-over or resource management.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:choice minOccurs="0">
            <xss:element name="StatelessServiceGroup" type="StatelessServiceGroupType"/>
            <xss:element name="StatefulServiceGroup" type="StatefulServiceGroupType"/>
        </xss:choice>
        <xss:attribute name="Name" type="xs:string" use="required">
            <xss:annotation>
                <xss:documentation>Name of this service relative to this application Name URI.
Fully qualified Name of the service is a combination of Name Uri of the Application and this Name.
</xss:documentation>
            </xss:annotation>
        </xss:attribute>
        <xss:attribute name="ServicePackageActivationMode" use="optional"
default="SharedProcess">
            <xss:annotation>
                <xss:documentation>ServicePackageActivationMode to be used when creating the
service. With SharedProcess mode, replica(s) or instance(s) from different partition(s) of service will share
same activation of service package on a node. With ExclusiveProcess mode, each replica or instance of
service will have its own dedicated activation of service package.</xss:documentation>
            </xss:annotation>
            <xss:simpleType>
                <xss:restriction base="xs:string">
                    <xss:enumeration value="SharedProcess"/>
                    <xss:enumeration value="ExclusiveProcess"/>
                </xss:restriction>
            </xss:simpleType>
        </xss:attribute>
    </xss:complexType>
</xss:element>
```

## Attribute details

### Name

Defines an affinity relationship with another service. Useful when splitting a previously-monolithic application into microservices. One service has a local dependency on another service and both services need to run on the same node in order to work.

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

### ServicePackageActivationMode

Defines an affinity relationship with another service. Useful when splitting a previously-monolithic application into microservices. One service has a local dependency on another service and both services need to run on the same node in order to work.

ATTRIBUTE	VALUE
name	ServicePackageActivationMode
use	optional
default	SharedProcess

## Content element details

### StatelessServiceGroup

ATTRIBUTE	VALUE
name	StatelessServiceGroup
type	StatelessServiceGroupType

### StatefulServiceGroup

ATTRIBUTE	VALUE
name	StatefulServiceGroup
type	StatefulServiceGroupType

## ServiceGroupMembers Element

Member types of this service group type.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)

ATTRIBUTE	VALUE
name	ServiceGroupMembers
minOccurs	0
maxOccurs	1

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceGroupMembers" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>Member types of this service group type.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="ServiceGroupTypeMember" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

## Content element details

None

ATTRIBUTE	VALUE
ref	ServiceGroupTypeMember
minOccurs	1
maxOccurs	unbounded

## ServiceGroupTypeMember Element

Describes the member type of the service group.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 1 attribute(s)
name	ServiceGroupTypeMember

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceGroupTypeMember">
  <xs:annotation>
    <xs:documentation>Describes the member type of the service group.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="LoadMetrics" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Load metrics reported by this service, used for resource balancing services.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="LoadMetric" type="LoadMetricType" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="ServiceTypeName" use="required">
      <xs:annotation>
        <xs:documentation>User defined type identifier for a Microsoft Azure Service Fabric ServiceGroup
Member, .e.g Actor</xs:documentation>
      </xs:annotation>
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:minLength value="1"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>

```

## Attribute details

### ServiceTypeName

ATTRIBUTE	VALUE
name	ServiceTypeName
use	required

## Content element details

### LoadMetrics

Load metrics reported by this service, used for resource balancing services.

ATTRIBUTE	VALUE
name	LoadMetrics
minOccurs	0

## ServiceManifest Element

ATTRIBUTE	VALUE
type	ServiceManifestType

ATTRIBUTE	VALUE
content	0 element(s), 0 attribute(s)
name	ServiceManifest

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceManifest" type="ServiceManifestType"/>
```

## ServiceManifestImport Element

Imports a service manifest created by the service developer. A service manifest must be imported for each constituent service in the application. Configuration overrides and policies can be declared for the service manifest.

ATTRIBUTE	VALUE
type	anonymous complexType
content	5 element(s), 0 attribute(s)
name	ServiceManifestImport
maxOccurs	unbounded

#### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceManifestImport" maxOccurs="unbounded">
    <xss:annotation>
        <xss:documentation>Imports a service manifest created by the service developer. A service manifest
must be imported for each constituent service in the application. Configuration overrides and policies can be
declared for the service manifest.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence>
            <xss:element name="ServiceManifestRef" type="ServiceManifestRefType"/>
            <xss:element name="ConfigOverrides" minOccurs="0">
                <xss:annotation>
                    <xss:documentation>Describes configuration overrides for the imported service manifest.
Configuration overrides allow the flexibility of re-using the same service manifests across multiple
application types by overriding the service manifest's configuration only when used with a particular
application type. Configuration overrides can change any default configuration in a service manifest as long as
default configuration is defined using the Settings.xml in the ConfigPackage folder. </xss:documentation>
                </xss:annotation>
                <xss:complexType>
                    <xss:sequence>
                        <xss:element name="ConfigOverride" type="ConfigOverrideType" minOccurs="0"
maxOccurs="unbounded"/>
                    </xss:sequence>
                </xss:complexType>
            </xss:element>
            <xss:element name="ResourceOverrides" type="ResourceOverridesType" minOccurs="0"/>
            <xss:element name="EnvironmentOverrides" type="EnvironmentOverridesType" minOccurs="0"
maxOccurs="unbounded"/>
            <xss:element name="Policies" type="ServiceManifestImportPoliciesType" minOccurs="0"/>
        </xss:sequence>
    </xss:complexType>
</xss:element>

```

## Content element details

### ServiceManifestRef

ATTRIBUTE	VALUE
name	ServiceManifestRef
type	ServiceManifestRefType

### ConfigOverrides

Describes configuration overrides for the imported service manifest. Configuration overrides allow the flexibility of re-using the same service manifests across multiple application types by overriding the service manifest's configuration only when used with a particular application type. Configuration overrides can change any default configuration in a service manifest as long as default configuration is defined using the Settings.xml in the ConfigPackage folder.

ATTRIBUTE	VALUE
name	ConfigOverrides
minOccurs	0

### ResourceOverrides

ATTRIBUTE	VALUE
name	ResourceOverrides
type	ResourceOverridesType
minOccurs	0

#### EnvironmentOverrides

ATTRIBUTE	VALUE
name	EnvironmentOverrides
type	EnvironmentOverridesType
minOccurs	0
maxOccurs	unbounded

#### Policies

ATTRIBUTE	VALUE
name	Policies
type	ServiceManifestImportPoliciesType
minOccurs	0

## ServiceManifestRef Element

ATTRIBUTE	VALUE
type	ServiceManifestRefType
content	0 element(s), 0 attribute(s)
name	ServiceManifestRef

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceManifestRef" type="ServiceManifestRefType"/>
```

## ServicePackage Element

ServicePackage represents a versioned unit of deployment and activation. The version of the ServicePackage is determined based on the manifest version and the version of the overrides.

ATTRIBUTE	VALUE
type	ServicePackageType
content	0 element(s), 0 attribute(s)
name	ServicePackage

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServicePackage" type="ServicePackageType">
  <xs:annotation>
    <xs:documentation>ServicePackage represents a versioned unit of deployment and activation. The version of
the ServicePackage is determined based on the manifest version and the version of the overrides.
  </xs:documentation>
  </xs:annotation>
</xs:element>
```

## ServicePackageRef Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)
name	ServicePackageRef
maxOccurs	unbounded

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServicePackageRef" maxOccurs="unbounded">
  <xs:complexType>
    <xs:attribute name="Name" use="required"/>
    <xs:attributeGroup ref="VersionedItemAttrGroup"/>
  </xs:complexType>
</xs:element>
```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
use	required

## ServicePackageResourceGovernancePolicy Element

Defines the resource governance policy that is applied at the level of the entire service package.

ATTRIBUTE	VALUE
type	ServicePackageResourceGovernancePolicyType
content	0 element(s), 0 attribute(s)
name	ServicePackageResourceGovernancePolicy
minOccurs	0
maxOccurs	1

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServicePackageResourceGovernancePolicy" type="ServicePackageResourceGovernancePolicyType" minOccurs="0"
maxOccurs="1">
    <xs:annotation>
        <xs:documentation>Defines the resource governance policy that is applied at the level of the entire
service package.</xs:documentation>
    </xs:annotation>
</xs:element>
```

## ServicePackageResourceGovernancePolicy Element

ATTRIBUTE	VALUE
type	ServicePackageResourceGovernancePolicyType
content	0 element(s), 0 attribute(s)
name	ServicePackageResourceGovernancePolicy
minOccurs	0
maxOccurs	1

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServicePackageResourceGovernancePolicy" type="ServicePackageResourceGovernancePolicyType" minOccurs="0"
maxOccurs="1"/>
```

## ServicePlacementPolicies Element

Declares placement policies for a service. Useful when the cluster spans geographic distances or and/or geopolitical regions.

ATTRIBUTE	VALUE
type	anonymous complexType

ATTRIBUTE	VALUE
content	1 element(s), 0 attribute(s)
name	ServicePlacementPolicies
minOccurs	0

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServicePlacementPolicies" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Declares placement policies for a service. Useful when the cluster spans
geographic distances or and/or geopolitical regions.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="ServicePlacementPolicy" maxOccurs="unbounded">
                <xs:annotation>
                    <xs:documentation>Defines a service placement policy, which specifies that the
service should or should not run in certain cluster fault domains. Useful when the cluster spans geographic
distances or and/or geopolitical regions.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:attribute name="DomainName">
                        <xs:annotation>
                            <xs:documentation>The fault domain where the service should or should
not be placed, depending on the Type value.</xs:documentation>
                        </xs:annotation>
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:minLength value="1"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:attribute>
                    <xs:attribute name="Type" use="required">
                        <xs:annotation>
                            <xs:documentation>InvalidDomain allows you to specify that a particular
Fault Domain is invalid for this workload. RequiredDomain requires that all of the replicas be present in the
specified domain. Multiple required domains can be specified. PreferredPrimaryDomain specifies the preferred
Fault Domain for primary replicas. Useful in geographically spanned clusters where you are using other
locations for redundancy, but would prefer that the primary replicas be placed in a certain location in order
to provider lower latency for operations which go to the primary. RequiredDomainDistribution specifies that
replicas are required to be distributed among the available fault domains. NonPartiallyPlace controls if the
service replicas will be partially place if not all of them can be placed.</xs:documentation>
                        </xs:annotation>
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="InvalidDomain"/>
                                <xs:enumeration value="RequiredDomain"/>
                                <xs:enumeration value="PreferredPrimaryDomain"/>
                                <xs:enumeration value="RequiredDomainDistribution"/>
                                <xs:enumeration value="NonPartiallyPlace"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:attribute>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>

```

## Content element details

#### **ServicePlacementPolicy**

Defines a service placement policy, which specifies that the service should or should not run in certain cluster fault domains. Useful when the cluster spans geographic distances or and/or geopolitical regions.

ATTRIBUTE	VALUE
name	ServicePlacementPolicy
maxOccurs	unbounded

## ServicePlacementPolicies Element

Declares placement policies for a service. Useful when the cluster spans geographic distances or and/or geopolitical regions.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	ServicePlacementPolicies
minOccurs	0

#### **XML source**

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServicePlacementPolicies" minOccurs="0">
    <xss:annotation>
        <xss:documentation>Declares placement policies for a service. Useful when the cluster spans
geographic distances or and/or geopolitical regions.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence>
            <xss:element name="ServicePlacementPolicy" maxOccurs="unbounded">
                <xss:annotation>
                    <xss:documentation>Defines a service placement policy, which specifies that the service should
or should not run in certain cluster fault domains. Useful when the cluster spans geographic distances or
and/or geopolitical regions.</xss:documentation>
                </xss:annotation>
                <xss:complexType>
                    <xss:attribute name="DomainName">
                        <xss:annotation>
                            <xss:documentation>The fault domain where the service should or should not be placed,
depending on the Type value.</xss:documentation>
                        </xss:annotation>
                    </xss:complexType>
                    <xss:restriction base="xs:string">
                        <xs:minLength value="1"/>
                    </xss:restriction>
                </xss:complexType>
                </xss:attribute>
                <xss:attribute name="Type" use="required">
                    <xss:annotation>
                        <xss:documentation>InvalidDomain allows you to specify that a particular Fault Domain is
invalid for this workload. RequiredDomain requires that all of the replicas be present in the specified domain.
Multiple required domains can be specified. PreferredPrimaryDomain specifies the preferred Fault Domain for
primary replicas. Useful in geographically spanned clusters where you are using other locations for redundancy,
but would prefer that the primary replicas be placed in a certain location in order to provider lower latency
for operations which go to the primary. RequiredDomainDistribution specifies that replicas are required to be
distributed among the available fault domains. NonPartiallyPlace controls if the service replicas will be
partially place if not all of them can be placed. </xss:documentation>
                    </xss:annotation>
                </xss:attribute>
                <xss:simpleType>
                    <xss:restriction base="xs:string">
                        <xs:enumeration value="InvalidDomain"/>
                        <xs:enumeration value="RequiredDomain"/>
                        <xs:enumeration value="PreferredPrimaryDomain"/>
                        <xs:enumeration value="RequiredDomainDistribution"/>
                        <xs:enumeration value="NonPartiallyPlace"/>
                    </xss:restriction>
                </xss:simpleType>
                </xss:attribute>
            </xss:complexType>
            </xss:element>
        </xss:sequence>
    </xss:complexType>
</xss:element>

```

## Content element details

### ServicePlacementPolicy

Defines a service placement policy, which specifies that the service should or should not run in certain cluster fault domains. Useful when the cluster spans geographic distances or and/or geopolitical regions.

ATTRIBUTE	VALUE
name	ServicePlacementPolicy
maxOccurs	unbounded

# ServicePlacementPolicy Element

Defines a service placement policy, which specifies that the service should or should not run in certain cluster fault domains. Useful when the cluster spans geographic distances or and/or geopolitical regions.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	ServicePlacementPolicy
maxOccurs	unbounded

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric" name="ServicePlacementPolicy" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>Defines a service placement policy, which specifies that the service should or should not run in certain cluster fault domains. Useful when the cluster spans geographic distances or and/or geopolitical regions.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:attribute name="DomainName">
            <xs:annotation>
                <xs:documentation>The fault domain where the service should or should not be placed, depending on the Type value.</xs:documentation>
            </xs:annotation>
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:minLength value="1"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="Type" use="required">
            <xs:annotation>
                <xs:documentation>InvalidDomain allows you to specify that a particular Fault Domain is invalid for this workload. RequiredDomain requires that all of the replicas be present in the specified domain. Multiple required domains can be specified. PreferredPrimaryDomain specifies the preferred Fault Domain for primary replicas. Useful in geographically spanned clusters where you are using other locations for redundancy, but would prefer that the primary replicas be placed in a certain location in order to provider lower latency for operations which go to the primary. RequiredDomainDistribution specifies that replicas are required to be distributed among the available fault domains. NonPartiallyPlace controls if the service replicas will be partially place if not all of them can be placed.</xs:documentation>
            </xs:annotation>
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="InvalidDomain"/>
                    <xs:enumeration value="RequiredDomain"/>
                    <xs:enumeration value="PreferredPrimaryDomain"/>
                    <xs:enumeration value="RequiredDomainDistribution"/>
                    <xs:enumeration value="NonPartiallyPlace"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
</xs:element>
```

## Attribute details

### DomainName

Defines a service placement policy, which specifies that the service should or should not run in certain cluster fault domains. Useful when the cluster spans geographic distances or and/or geopolitical regions.

ATTRIBUTE	VALUE
name	DomainName

#### Type

Defines a service placement policy, which specifies that the service should or should not run in certain cluster fault domains. Useful when the cluster spans geographic distances or and/or geopolitical regions.

ATTRIBUTE	VALUE
name	Type
use	required

## ServicePlacementPolicy Element

Defines a service placement policy, which specifies that the service should or should not run in certain cluster fault domains. Useful when the cluster spans geographic distances or and/or geopolitical regions.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	ServicePlacementPolicy
maxOccurs	unbounded

#### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServicePlacementPolicy" maxOccurs="unbounded">
    <xss:annotation>
        <xss:documentation>Defines a service placement policy, which specifies that the service should or should not run in certain cluster fault domains. Useful when the cluster spans geographic distances or and/or geopolitical regions.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:attribute name="DomainName">
            <xss:annotation>
                <xss:documentation>The fault domain where the service should or should not be placed, depending on the Type value.</xss:documentation>
            </xss:annotation>
            <xss:simpleType>
                <xss:restriction base="xs:string">
                    <xs:minLength value="1"/>
                </xss:restriction>
            </xss:simpleType>
        </xss:attribute>
        <xss:attribute name="Type" use="required">
            <xss:annotation>
                <xss:documentation>InvalidDomain allows you to specify that a particular Fault Domain is invalid for this workload. RequiredDomain requires that all of the replicas be present in the specified domain. Multiple required domains can be specified. PreferredPrimaryDomain specifies the preferred Fault Domain for primary replicas. Useful in geographically spanned clusters where you are using other locations for redundancy, but would prefer that the primary replicas be placed in a certain location in order to provider lower latency for operations which go to the primary. RequiredDomainDistribution specifies that replicas are required to be distributed among the available fault domains. NonPartiallyPlace controls if the service replicas will be partially place if not all of them can be placed. </xss:documentation>
            </xss:annotation>
            <xss:simpleType>
                <xss:restriction base="xs:string">
                    <xs:enumeration value="InvalidDomain"/>
                    <xs:enumeration value="RequiredDomain"/>
                    <xs:enumeration value="PreferredPrimaryDomain"/>
                    <xs:enumeration value="RequiredDomainDistribution"/>
                    <xs:enumeration value="NonPartiallyPlace"/>
                </xss:restriction>
            </xss:simpleType>
        </xss:attribute>
    </xss:complexType>
</xss:element>

```

## Attribute details

### DomainName

Defines a service placement policy, which specifies that the service should or should not run in certain cluster fault domains. Useful when the cluster spans geographic distances or and/or geopolitical regions.

ATTRIBUTE	VALUE
name	DomainName

### Type

Defines a service placement policy, which specifies that the service should or should not run in certain cluster fault domains. Useful when the cluster spans geographic distances or and/or geopolitical regions.

ATTRIBUTE	VALUE
name	Type
use	required

## ServiceTemplates Element

Declares the set of permitted service types that can be created dynamically inside the application instance. Default configuration values, such as replication factor, are specified and used as a template for creating service instances.

ATTRIBUTE	VALUE
type	ServiceTemplatesType
content	0 element(s), 0 attribute(s)
name	ServiceTemplates
minOccurs	0

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="ServiceTemplates" type="ServiceTemplatesType" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Declares the set of permitted service types that can be created dynamically inside
    the application instance. Default configuration values, such as replication factor, are specified and used as a
    template for creating service instances.</xs:documentation>
  </xs:annotation>
</xs:element>
```

## ServiceTemplates Element

ATTRIBUTE	VALUE
type	ServiceTemplatesType
content	0 element(s), 0 attribute(s)
name	ServiceTemplates

### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="ServiceTemplates" type="ServiceTemplatesType"/>
```

## ServiceTypeHealthPolicy Element

Describes the policy for evaluating health events reported on services, partitions and replicas of a particular service type.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	ServiceTypeHealthPolicy

ATTRIBUTE	VALUE
minOccurs	0
maxOccurs	unbounded

#### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceTypeHealthPolicy" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>Describes the policy for evaluating health events reported on services, partitions
and replicas of a particular service type.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="ServiceTypeHealthPolicyType">
                <xs:attribute name="ServiceTypeName" type="xs:string" use="required">
                    <xs:annotation>
                        <xs:documentation>The name of the service type that the policy will be applied to.
                    </xs:documentation>
                    </xs:annotation>
                </xs:attribute>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>

```

## ServiceTypes Element

Defines what service types are supported by a CodePackage in this manifest. When a service is instantiated against one of these service types, all code packages declared in this manifest are activated by running their entry points. Service types are declared at the manifest level and not the code package level.

ATTRIBUTE	VALUE
type	<a href="#">ServiceAndServiceGroupTypesType</a>
content	0 element(s), 0 attribute(s)
name	ServiceTypes

#### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceTypes" type="ServiceAndServiceGroupTypesType">
    <xs:annotation>
        <xs:documentation>Defines what service types are supported by a CodePackage in this manifest. When a
service is instantiated against one of these service types, all code packages declared in this manifest are
activated by running their entry points. Service types are declared at the manifest level and not the code
package level.</xs:documentation>
        </xs:annotation>
    </xs:element>

```

## ServiceTypes Element

ATTRIBUTE	VALUE
type	ServiceTypesType
content	0 element(s), 0 attribute(s)
name	ServiceTypes

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceTypes" type="ServiceTypesType"/>
```

## Settings Element

Defiles configurable settings for the code packages of a service. Microsoft Azure Service Fabric does not interpret the settings, however it makes them available via Runtime APIs for use by the code components.

ATTRIBUTE	VALUE
type	SettingsType
content	0 element(s), 0 attribute(s)
name	Settings

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Settings" type="SettingsType">
    <xs:annotation>
        <xs:documentation>Defiles configurable settings for the code packages of a service.
Microsoft Azure Service Fabric does not interpret the settings, however it makes them available via Runtime
APIs for use by the code components.</xs:documentation>
    </xs:annotation>
</xs:element>
```

## Settings Element

ATTRIBUTE	VALUE
type	SettingsOverridesType
content	0 element(s), 0 attribute(s)
name	Settings
minOccurs	0

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Settings" type="SettingsOverridesType" minOccurs="0"/>
```

## SetupEntryPoint Element

A privileged entry point that runs with the same credentials as Service Fabric (typically the LocalSystem account) before any other entry point. The executable specified by EntryPoint is typically the long-running service host. The presence of a separate setup entry point avoids having to run the service host with high privileges for extended periods of time.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	SetupEntryPoint
minOccurs	0

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="SetupEntryPoint" minOccurs="0">
    <xss:annotation>
        <xss:documentation>A privileged entry point that runs with the same credentials as Service Fabric
        (typically the LocalSystem account) before any other entry point. The executable specified by EntryPoint is
        typically the long-running service host. The presence of a separate setup entry point avoids having to run the
        service host with high privileges for extended periods of time.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence>
            <xss:element name="ExeHost" type="ExeHostEntryPointType"/>
        </xss:sequence>
    </xss:complexType>
</xss:element>
```

### Content element details

#### ExeHost

ATTRIBUTE	VALUE
name	ExeHost
type	ExeHostEntryPointType

## SharedLogFileId Element

Specific GUID to use as the shared log id.

ATTRIBUTE	VALUE
type	anonymous complexType

ATTRIBUTE	VALUE
content	0 element(s), 1 attribute(s)
name	SharedLogFileId
minOccurs	0

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="SharedLogFileId" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Specific GUID to use as the shared log id.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:attribute name="Value" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:pattern value="[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]
{12}"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>

```

## Attribute details

### Value

ATTRIBUTE	VALUE
name	Value
use	required

## SharedLogFilePath Element

Defines path to shared log.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)
name	SharedLogFilePath
minOccurs	0

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="SharedLogFilePath" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Defines path to shared log.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:attribute name="Value" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

```

## Attribute details

### Value

ATTRIBUTE	VALUE
name	Value
type	xs:string
use	required

## SharedLogFileSizeInMB Element

Defines how large is the shared log.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)
name	SharedLogFileSizeInMB
minOccurs	0

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="SharedLogFileSizeInMB" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Defines how large is the shared log.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:attribute name="Value" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:int">
          <xs:minInclusive value="512"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>

```

## Attribute details

### Value

ATTRIBUTE	VALUE
name	Value
use	required

## SingletonPartition Element

Declares that this service has only one partition.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	SingletonPartition

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="SingletonPartition">
    <xss:annotation>
        <xss:documentation>Declares that this service has only one partition.</xss:documentation>
    </xss:annotation>
    <xss:complexType/>
</xss:element>
```

## StatefulService Element

ATTRIBUTE	VALUE
type	StatefulServiceType
content	0 element(s), 0 attribute(s)
name	StatefulService

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatefulService" type="StatefulServiceType"/>
```

## StatefulService Element

ATTRIBUTE	VALUE
type	StatefulServiceType
content	0 element(s), 0 attribute(s)

ATTRIBUTE	VALUE
name	StatefulService

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatefulService" type="StatefulServiceType"/>
```

## StatefulServiceGroup Element

ATTRIBUTE	VALUE
type	StatefulServiceGroupType
content	0 element(s), 0 attribute(s)
name	StatefulServiceGroup

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatefulServiceGroup" type="StatefulServiceGroupType"/>
```

## StatefulServiceGroup Element

ATTRIBUTE	VALUE
type	StatefulServiceGroupType
content	0 element(s), 0 attribute(s)
name	StatefulServiceGroup

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatefulServiceGroup" type="StatefulServiceGroupType"/>
```

## StatefulServiceGroupType Element

ATTRIBUTE	VALUE
type	StatefulServiceGroupTypeType
content	0 element(s), 0 attribute(s)
name	StatefulServiceGroupType

#### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatefulServiceGroupType" type="StatefulServiceGroupTypeType"/>
```

## StatefulServiceType Element

ATTRIBUTE	VALUE
type	StatefulServiceTypeType
content	0 element(s), 0 attribute(s)
name	StatefulServiceType

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatefulServiceType" type="StatefulServiceTypeType"/>
```

## StatefulServiceType Element

Describes a stateful ServiceType.

ATTRIBUTE	VALUE
type	StatefulServiceTypeType
content	0 element(s), 0 attribute(s)
name	StatefulServiceType

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatefulServiceType" type="StatefulServiceTypeType">
    <xss:annotation>
        <xss:documentation>Describes a stateful ServiceType.</xss:documentation>
    </xss:annotation>
</xss:element>
```

## StatelessService Element

ATTRIBUTE	VALUE
type	StatelessServiceType
content	0 element(s), 0 attribute(s)
name	StatelessService

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatelessService" type="StatelessServiceType"/>
```

## StatelessService Element

ATTRIBUTE	VALUE
type	StatelessServiceType
content	0 element(s), 0 attribute(s)
name	StatelessService

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatelessService" type="StatelessServiceType"/>
```

## StatelessServiceGroup Element

ATTRIBUTE	VALUE
type	StatelessServiceGroupType
content	0 element(s), 0 attribute(s)
name	StatelessServiceGroup

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatelessServiceGroup" type="StatelessServiceGroupType"/>
```

## StatelessServiceGroup Element

ATTRIBUTE	VALUE
type	StatelessServiceGroupType
content	0 element(s), 0 attribute(s)
name	StatelessServiceGroup

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatelessServiceGroup" type="StatelessServiceGroupType"/>
```

## StatelessServiceGroupType Element

ATTRIBUTE	VALUE
type	StatelessServiceGroupTypeType
content	0 element(s), 0 attribute(s)
name	StatelessServiceGroupType

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatelessServiceGroupType" type="StatelessServiceGroupTypeType"/>
```

## StatelessServiceType Element

ATTRIBUTE	VALUE
type	StatelessServiceTypeType
content	0 element(s), 0 attribute(s)
name	StatelessServiceType

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatelessServiceType" type="StatelessServiceTypeType"/>
```

## StatelessServiceType Element

Describes a stateless ServiceType.

ATTRIBUTE	VALUE
type	StatelessServiceTypeType
content	0 element(s), 0 attribute(s)
name	StatelessServiceType

#### XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatelessServiceType" type="StatelessServiceTypeType">
  <xs:annotation>
    <xs:documentation>Describes a stateless ServiceType.</xs:documentation>
  </xs:annotation>
</xs:element>
```

## SystemGroup Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)
name	SystemGroup
minOccurs	0
maxOccurs	unbounded

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="SystemGroup" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
        <xs:attribute name="Name" type="xs:string"
use="required"/>
    </xs:complexType>
</xs:element>
```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

## SystemGroup Element

The system group to add the user to. The system group must be defined in the Groups section.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)
name	SystemGroup
minOccurs	0
maxOccurs	unbounded

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="SystemGroup" minOccurs="0" maxOccurs="unbounded">

<xs:annotation>
<xs:documentation>The system group to add the user to. The system group must be defined in the Groups section.</xs:documentation>
</xs:annotation>

<xs:complexType>
<xs:attribute name="Name" type="xs:string" use="required">
<xs:annotation>
<xs:documentation>The name of the system group.</xs:documentation>
</xs:annotation>
</xs:attribute>
</xs:complexType>
</xs:element>

```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

## TargetInformation Element

Describes the target the FabricDeployer needs to deploy.

ATTRIBUTE	VALUE
type	TargetInformationType
content	0 element(s), 0 attribute(s)
name	TargetInformation

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="TargetInformation" type="TargetInformationType">
    <xs:annotation>
        <xs:documentation>Describes the target the FabricDeployer needs to deploy.</xs:documentation>
    </xs:annotation>
</xs:element>

```

## TargetInstallation Element

ATTRIBUTE	VALUE
type	WindowsFabricDeploymentInformation
content	0 element(s), 0 attribute(s)
name	TargetInstallation
minOccurs	1

### XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="TargetInstallation" type="WindowsFabricDeploymentInformation" minOccurs="1"/>
```

## UniformInt64Partition Element

Describes a uniform partitioning scheme based on Int64 keys.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 3 attribute(s)
name	UniformInt64Partition

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="UniformInt64Partition">
    <xss:annotation>
        <xss:documentation>Describes a uniform partitioning scheme based on Int64 keys.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:attribute name="PartitionCount" type="xs:string" use="required">
            <xss:annotation>
                <xss:documentation>Total number of partitions (positive integer). Each partition is
responsible for a non-overlapping subrange of the overall partition key range.</xss:documentation>
            </xss:annotation>
        </xss:attribute>
        <xss:attribute name="LowKey" type="xs:string" use="required">
            <xss:annotation>
                <xss:documentation>Inclusive low range of the partition key (long).</xss:documentation>
            </xss:annotation>
        </xss:attribute>
        <xss:attribute name="HighKey" type="xs:string" use="required">
            <xss:annotation>
                <xss:documentation>Inclusive high range of the partition key (long).</xss:documentation>
            </xss:annotation>
        </xss:attribute>
    </xss:complexType>
</xss:element>

```

## Attribute details

### PartitionCount

ATTRIBUTE	VALUE
name	PartitionCount
type	xs:string
use	required

### LowKey

ATTRIBUTE	VALUE
name	LowKey
type	xs:string
use	required

### HighKey

ATTRIBUTE	VALUE
name	HighKey
type	xs:string
use	required

## UnmanagedDll Element

ATTRIBUTE	VALUE
type	UnmanagedDllType
content	0 element(s), 0 attribute(s)
name	UnmanagedDll

### XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric" name="UnmanagedDll" type="UnmanagedDllType"/>
```

## UpgradeOrchestrationServiceReplicatorEndpoint Element

ATTRIBUTE	VALUE
type	InternalEndpointType
content	0 element(s), 0 attribute(s)
name	UpgradeOrchestrationServiceReplicatorEndpoint
minOccurs	0

### XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric" name="UpgradeOrchestrationServiceReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
```

## UpgradeServiceReplicatorEndpoint Element

ATTRIBUTE	VALUE
type	InternalEndpointType
content	0 element(s), 0 attribute(s)
name	UpgradeServiceReplicatorEndpoint
minOccurs	0

### XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric" name="UpgradeServiceReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
```

## User Element

Declares a user as a security principal, which can be referenced in policies.

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 5 attribute(s)
name	User
maxOccurs	unbounded

## XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="User" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>Declares a user as a security principal, which can be
referenced in policies.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="NTLMAuthenticationPolicy" minOccurs="0">
                <xs:complexType>
                    <xs:attribute name="IsEnabled" type="xs:boolean" use="optional"
default="true"/>
                    <xs:attribute name="PasswordSecret" type="xs:string"
use="required"/>
                    <xs:attribute name="PasswordSecretEncrypted" type="xs:boolean"
use="optional" default="false"/>
                    <xs:attribute name="X509StoreLocation" use="optional"
default="LocalMachine">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="LocalMachine"/>
                                <xs:enumeration value="CurrentUser"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:attribute>
                    <xs:attribute name="X509StoreName" default="My">
                        <xs:simpleType>
                            <xs:restriction base="xs:string"/>
                        </xs:simpleType>
                    </xs:attribute>
                    <xs:attribute name="X509Thumbprint" type="xs:string"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="MemberOf" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>
                        Users can be added to any existing membership group, so it can inherit all the
properties and security settings of that membership group. The membership group can be used to secure external
resources that need to be accessed by different services or the same service (on a different machine).
                    </xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:choice
maxOccurs="unbounded">
                        <xs:element
name="SystemGroup" minOccurs="0" maxOccurs="unbounded">
                            <xs:annotation>
                                <xs:documentation>The system group to add the user to. The system group must be defined in the Groups section
                                </xs:documentation>
                            </xs:annotation>
                        </xs:element>
                    </xs:choice>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>

```

```

<xs:documentation>The system group to add the user to. The system group must be defined in the Groups section.
</xs:documentation>

</xs:annotation>

<xss:complexType>

<xss:attribute name="Name" type="xs:string" use="required">
<xss:annotation>
<xss:documentation>The name of the system group.</xss:documentation>
</xss:annotation>
</xss:attribute>

<xss:complexType>
<xss:element>
<xss:element name="Group" minOccurs="0" maxOccurs="unbounded">
<xss:annotation>
<xss:documentation>The group to add the user to. The group must be defined in the Groups section.
</xss:documentation>
</xss:annotation>
<xss:complexType>
<xss:attribute name="NameRef" type="xs:string" use="required">
<xss:annotation>
<xss:documentation>The name of the group.</xss:documentation>
</xss:annotation>
</xss:attribute>

<xss:complexType>
<xss:element>
<xss:choice>
<xss:complexType>
<xss:element>
<xss:sequence>
<xss:attribute name="Name" type="xs:string" use="required">
<xss:annotation>
<xss:documentation>Name of the user account.</xss:documentation>
</xss:annotation>
</xss:attribute>
<xss:attribute name="AccountType" use="optional" default="LocalUser">
<xss:annotation>
<xss:documentation>Specifies the type of account. Local user accounts are created on the machines where the application is deployed. By default, these accounts do not have the same names as those specified here. Instead, they are dynamically generated and have random passwords. Supported local system account types are LocalUser, NetworkService, LocalService and LocalSystem. Domain accounts are supported on Windows Server deployments where Azure Active Directory is available.</xss:documentation>
</xss:annotation>
<xss:simpleType>
<xss:restriction base="xs:string">
<xss:enumeration value="LocalUser"/>

```

```

value="DomainUser"/>                                <xs:enumeration>
value="NetworkService"/>                            <xs:enumeration>
value="LocalService"/>                             <xs:enumeration>
value="ManagedServiceAccount"/>                     <xs:enumeration>
value="LocalSystem"/>                               <xs:enumeration>

   </xs:restriction>
   </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="LoadUserProfile">
   <xs:attribute name="PerformInteractiveLogon">
   <xs:attributeGroup>
   <xs:attribute name="PasswordEncrypted">
   <xs:annotation>
   <xs:documentation>True if the
password is encrypted; false if in plain text.</xs:documentation>
   </xs:annotation>
   </xs:attribute>
   </xs:complexType>
   </xs:element>

```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

### AccountType

ATTRIBUTE	VALUE
name	AccountType
use	optional
default	LocalUser

### LoadUserProfile

ATTRIBUTE	VALUE
name	LoadUserProfile
type	xs:boolean
use	optional

ATTRIBUTE	VALUE
default	false

#### PerformInteractiveLogon

ATTRIBUTE	VALUE
name	PerformInteractiveLogon
type	xs:boolean
use	optional
default	false

#### PasswordEncrypted

ATTRIBUTE	VALUE
name	PasswordEncrypted
type	xs:boolean
use	optional

### Content element details

#### NTLMAuthenticationPolicy

ATTRIBUTE	VALUE
name	NTLMAuthenticationPolicy
minOccurs	0

#### MemberOf

Users can be added to any existing membership group, so it can inherit all the properties and security settings of that membership group. The membership group can be used to secure external resources that need to be accessed by different services or the same service (on a different machine).

ATTRIBUTE	VALUE
name	MemberOf
minOccurs	0

## UserRoleClientCertificate Element

The default user role client certificate used to secure client server communication.

ATTRIBUTE	VALUE
type	FabricCertificateType
content	0 element(s), 0 attribute(s)
name	UserRoleClientCertificate
minOccurs	0

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="UserRoleClientCertificate" type="FabricCertificateType" minOccurs="0">
    <xss:annotation>
        <xss:documentation>The default user role client certificate used to secure client server
communication.</xss:documentation>
    </xss:annotation>
</xss:element>
```

## Users Element

Declares a set of users as security principals, which can be referenced in policies.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	Users
minOccurs	0

## XML source

```
<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Users" minOccurs="0">
    <xss:annotation>
        <xss:documentation>Declares a set of users as security principals, which can be referenced
in policies.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence>
            <xss:element name="User" maxOccurs="unbounded">
                <xss:annotation>
                    <xss:documentation>Declares a user as a security principal, which can be
referenced in policies.</xss:documentation>
                </xss:annotation>
                <xss:complexType>
                    <xss:sequence>
                        <xss:element name="NTLMAuthenticationPolicy" minOccurs="0">
                            <xss:complexType>
                                <xss:attribute name="IsEnabled" type="xs:boolean" use="optional"
default="true"/>
                                <xss:attribute name="PasswordSecret" type="xs:string"
use="required"/>
                                <xss:attribute name="PasswordSecretEncrypted" type="xs:boolean"
use="optional" default="false"/>

```

```

<!-- optional attribute -->
<xss:attribute name="X509StoreLocation" use="optional"
default="LocalMachine">
    <xss:simpleType>
        <xss:restriction base="xss:string">
            <xss:enumeration value="LocalMachine"/>
            <xss:enumeration value="CurrentUser"/>
        </xss:restriction>
    </xss:simpleType>
</xss:attribute>
<xss:attribute name="X509StoreName" default="My">
    <xss:simpleType>
        <xss:restriction base="xss:string"/>
    </xss:simpleType>
</xss:attribute>
<xss:attribute name="X509Thumbprint" type="xss:string"/>
</xss:complexType>
</xss:element>
<xss:element name="MemberOf" minOccurs="0">
    <xss:annotation>
        <xss:documentation>
            Users can be added to any existing membership group, so it can inherit all the
            properties and security settings of that membership group. The membership group can be used to secure external
            resources that need to be accessed by different services or the same service (on a different machine).
        </xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:choice maxOccurs="unbounded">
            <xss:element name="SystemGroup" minOccurs="0" maxOccurs="unbounded">
                <xss:annotation>
                    <xss:documentation>The system group to add the user to. The system group must be defined in the Groups section.</xss:documentation>
                </xss:annotation>
                <xss:complexType>
                    <xss:attribute name="Name" type="xss:string" use="required">
                        <xss:annotation>
                            <xss:documentation>The name of the system group.</xss:documentation>
                        </xss:annotation>
                    </xss:attribute>
                </xss:complexType>
            </xss:element>
            <xss:element name="Group" minOccurs="0" maxOccurs="unbounded">
                <xss:annotation>
                    <xss:documentation>The group to add the user to. The group must be defined in the Groups section.</xss:documentation>
                </xss:annotation>
                <xss:complexType>
                    <xss:attribute name="NameRef" type="xss:string" use="required">
                        <xss:annotation>

```

```

<xs:documentation>The name of the group.</xs:documentation>
</xs:annotation>
</xs:attribute>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="Name" type="xs:string"
use="required">
<xs:annotation>
<xs:documentation>Name of the
user account.</xs:documentation>
</xs:annotation>
</xs:attribute>
<xs:attribute name="AccountType" use="optional"
default="LocalUser">
<xs:annotation>
<xs:documentation>Specifies the
type of account. Local user accounts are created on the machines where the application is deployed. By default,
these accounts do not have the same names as those specified here. Instead, they are dynamically generated and
have random passwords. Supported local system account types are LocalUser, NetworkService, LocalService and
LocalSystem. Domain accounts are supported on Windows Server deployments where Azure Active Directory is
available.</xs:documentation>
<xs:annotation>
<xs:simpleType>
<xs:restriction>
<xs:enumeration
base="xs:string">
<xs:enumeration
value="LocalUser"/>
<xs:enumeration
value="DomainUser"/>
<xs:enumeration
value="NetworkService"/>
<xs:enumeration
value="LocalService"/>
<xs:enumeration
value="ManagedServiceAccount"/>
<xs:enumeration
value="LocalSystem"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
<xs:attribute name="LoadUserProfile"
type="xs:boolean" use="optional" default="false"/>
<xs:attribute name="PerformInteractiveLogon"
type="xs:boolean" use="optional" default="false"/>
<xs:attributeGroup
ref="AccountCredentialsGroup"/>
<xs:attribute name="PasswordEncrypted"
type="xs:boolean" use="optional">
<xs:annotation>
<xs:documentation>True if the
password is encrypted; false if in plain text.</xs:documentation>
</xs:annotation>
</xs:attribute>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

```

## Content element details

## User

Declares a user as a security principal, which can be referenced in policies.

ATTRIBUTE	VALUE
name	User
maxOccurs	unbounded

## Volume Element

Specifies the volume to be bound to container.

ATTRIBUTE	VALUE
type	ContainerVolumeType
content	0 element(s), 0 attribute(s)
name	Volume
minOccurs	0
maxOccurs	unbounded

### XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="Volume" type="ContainerVolumeType" minOccurs="0" maxOccurs="unbounded">
  <xss:annotation>
    <xss:documentation>Specifies the volume to be bound to container.</xss:documentation>
  </xss:annotation>
</xss:element>
```

## Vote Element

ATTRIBUTE	VALUE
type	PaaSVoteType
content	0 element(s), 0 attribute(s)
name	Vote
maxOccurs	unbounded

### XML source

```
<xss:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
  name="Vote" type="PaaSVoteType" maxOccurs="unbounded"/>
```

## Votes Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	Votes

## XML source

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Votes">
    <xs:complexType>
        <xs:sequence>
            <xs:element
name="Vote" type="PaaSVoteType" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

## Content element details

### Vote

ATTRIBUTE	VALUE
name	Vote
type	PaaSVoteType
maxOccurs	unbounded

## WindowsAzure Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	WindowsAzure

## XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="WindowsAzure">
    <xss:complexType>
        <xss:sequence>
            <xss:element name="Roles">
                <xss:complexType>
                    <xss:sequence>
                        <xss:element
name="Role" type="AzureRoleType" maxOccurs="unbounded"/>
                    </xss:sequence>
                </xss:complexType>
            </xss:element>
        </xss:sequence>
    </xss:complexType>
</xss:element>

```

## Content element details

### Roles

ATTRIBUTE	VALUE
name	Roles

## WindowsAzureStaticTopology Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	WindowsAzureStaticTopology

### XML source

```

<xss:element xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="WindowsAzureStaticTopology">
    <xss:complexType>
        <xss:complexContent>
            <xss:extension
base="WindowsInfrastructureType"/>
            </xss:complexContent>
        </xss:complexType>
    </xss:element>

```

## WindowsServer Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	WindowsServer

### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="WindowsServer">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension
base="WindowsInfrastructureType">
                <xs:attribute name="IsScaleMin"
type="xs:boolean" default="false"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>

```

## WorkingFolder Element

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	WorkingFolder
default	Work
minOccurs	0

### XML source

```

<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="WorkingFolder" default="Work" minOccurs="0">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="Work"/>
            <xs:enumeration value="CodePackage"/>
            <xs:enumeration value="CodeBase"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>

```

## AppInstanceDefinitionType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 1 attribute(s)
name	AppInstanceDefinitionType

### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="AppInstanceDefinitionType">
  <xss:sequence>
    <xss:element name="Parameters">
      <xss:annotation>
        <xss:documentation>List of parameters for the application as defined in application manifest and their
        respective values.</xss:documentation>
      </xss:annotation>
      <xss:complexType>
        <xss:sequence>
          <xss:element name="Parameter" minOccurs="0" maxOccurs="unbounded">
            <xss:complexType>
              <xss:attributeGroup ref="NameValuePair"/>
            </xss:complexType>
          </xss:element>
        </xss:sequence>
      </xss:complexType>
    </xss:element>
  </xss:sequence>
  <xss:attribute name="Name" type="xs:string" use="required">
    <xss:annotation>
      <xss:documentation>Name of the application to be created.</xss:documentation>
    </xss:annotation>
  </xss:attribute>
</xss:complexType>

```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

## Content element details

### Parameters

List of parameters for the application as defined in application manifest and their respective values.

ATTRIBUTE	VALUE
name	Parameters

## ApplicationHealthPolicyType complexType

Describes the policy for evaluating health events reported on various application related entities. If no policy is specified, an entity is assumed to be unhealthy if the health report is a warning or error.

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 2 attribute(s)
name	ApplicationHealthPolicyType

## XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ApplicationHealthPolicyType">
    <xs:annotation>
        <xs:documentation>Describes the policy for evaluating health events reported on various application
related entities. If no policy is specified, an entity is assumed to be unhealthy if the health report is a
warning or error.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="DefaultServiceTypeHealthPolicy" type="ServiceTypeHealthPolicyType" minOccurs="0">
            <xs:annotation>
                <xs:documentation>Specifies the default service type health policy, which will replace the default
health policy for all service types in the application.</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="ServiceTypeHealthPolicy" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
                <xs:documentation>Describes the policy for evaluating health events reported on services, partitions
and replicas of a particular service type.</xs:documentation>
            </xs:annotation>
            <xs:complexType>
                <xs:complexContent>
                    <xs:extension base="ServiceTypeHealthPolicyType">
                        <xs:attribute name="ServiceTypeName" type="xs:string" use="required">
                            <xs:annotation>
                                <xs:documentation>The name of the service type that the policy will be applied to.
                            </xs:documentation>
                            </xs:annotation>
                        </xs:attribute>
                    </xs:extension>
                </xs:complexContent>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
    <xs:attribute name="ConsiderWarningAsError" type="xs:string" use="optional" default="false">
        <xs:annotation>
            <xs:documentation>Specifies whether to treat warning health reports as errors during health evaluation.
Default: false.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="MaxPercentUnhealthyDeployedApplications" type="xs:string" use="optional" default="0">
        <xs:annotation>
            <xs:documentation>Specifies the maximum tolerated percentage of deployed applications that can be
unhealthy before the application is considered in error. This is calculated by dividing the number of unhealthy
deployed applications over the number of nodes that the applications are currently deployed on in the cluster.
The computation rounds up to tolerate one failure on small numbers of nodes. Default percentage: 0.
</xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:complexType>
```

## Attribute details

### ConsiderWarningAsError

List of parameters for the application as defined in application manifest and their respective values.

ATTRIBUTE	VALUE
name	ConsiderWarningAsError
type	xs:string
use	optional

ATTRIBUTE	VALUE
default	false

#### MaxPercentUnhealthyDeployedApplications

List of parameters for the application as defined in application manifest and their respective values.

ATTRIBUTE	VALUE
name	MaxPercentUnhealthyDeployedApplications
type	xs:string
use	optional
default	0

### Content element details

#### DefaultServiceTypeHealthPolicy

Specifies the default service type health policy, which will replace the default health policy for all service types in the application.

ATTRIBUTE	VALUE
name	DefaultServiceTypeHealthPolicy
type	ServiceTypeHealthPolicyType
minOccurs	0

#### ServiceTypeHealthPolicy

Describes the policy for evaluating health events reported on services, partitions and replicas of a particular service type.

ATTRIBUTE	VALUE
name	ServiceTypeHealthPolicy
minOccurs	0
maxOccurs	unbounded

## ApplicationInstanceType complexType

Describes an instance of a Microsoft Azure Service Fabric application.

ATTRIBUTE	VALUE
type	anonymous complexType
content	4 element(s), 1 attribute(s)

ATTRIBUTE	VALUE
name	ApplicationInstanceType

## XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ApplicationInstanceType">
  <xs:annotation>
    <xs:documentation>Describes an instance of a Microsoft Azure Service Fabric application.
  </xs:documentation>
  <xs:sequence>
    <xs:element name="ApplicationPackageRef">
      <xs:complexType>
        <xs:attributeGroup ref="VersionedItemAttrGroup"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="ServicePackageRef" maxOccurs="unbounded">
      <xs:complexType>
        <xs:attribute name="Name" use="required"/>
        <xs:attributeGroup ref="VersionedItemAttrGroup"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="ServiceTemplates" type="ServiceTemplatesType"/>
    <xs:element name="DefaultServices" type="DefaultServicesType"/>
  </xs:sequence>
  <xs:attribute name="Version" type="xs:int" use="required">
    <xs:annotation>
      <xs:documentation>The version of the ApplicationInstance document.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attributeGroup ref="ApplicationInstanceAttrGroup"/>
  <xs:attributeGroup ref="ApplicationManifestAttrGroup"/>
</xs:complexType>

```

## Attribute details

### Version

Describes the policy for evaluating health events reported on services, partitions and replicas of a particular service type.

ATTRIBUTE	VALUE
name	Version
type	xs:int
use	required

## Content element details

### ApplicationPackageRef

ATTRIBUTE	VALUE
name	ApplicationPackageRef

### ServicePackageRef

ATTRIBUTE	VALUE
name	ServicePackageRef
maxOccurs	unbounded

#### ServiceTemplates

ATTRIBUTE	VALUE
name	ServiceTemplates
type	ServiceTemplatesType

#### DefaultServices

ATTRIBUTE	VALUE
name	DefaultServices
type	DefaultServicesType

## ApplicationManifestType complexType

Declaratively describes the application type and version. One or more service manifests of the constituent services are referenced to compose an application type. Configuration settings of the constituent services can be overridden using parameterized application settings. Default services, service templates, principals, policies, diagnostics set-up, and certificates can also declared at the application level.

ATTRIBUTE	VALUE
type	anonymous complexType
content	9 element(s), 0 attribute(s)
name	ApplicationManifestType

#### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ApplicationManifestType">
  <xs:annotation>
    <xs:documentation>Declaratively describes the application type and version. One or more service manifests
of the constituent services are referenced to compose an application type. Configuration settings of the
constituent services can be overridden using parameterized application settings. Default services, service
templates, principals, policies, diagnostics set-up, and certificates can also declared at the application
level.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="Description" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Text describing this application.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="Parameters" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Declares the parameters that are used in this application manifest. The value of
these parameters can be supplied when the application is instantiated and can be used to override application
settings at runtime.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

```

or service configuration settings.</xs:documentation>
</xs:annotation>
<xss:complexType>
<xss:sequence>
<xss:element name="Parameter" block="" minOccurs="0" maxOccurs="unbounded">
<xss:annotation>
<xss/documentation>An application parameter to be used in this manifest. The parameter value can be changed during application instantiation, or, if no value is supplied the default value is used.
</xss:documentation>
</xss:annotation>
<xss:complexType>
<xss:attribute name="Name" use="required">
<xss:annotation>
<xss/documentation>The name of the parameter to be used in the manifest as "[Name]".
</xss:documentation>
</xss:annotation>
<xss:simpleType>
<xss:restriction base="xs:string">
<xss:minLength value="1"/>
</xss:restriction>
</xss:simpleType>
</xss:attribute>
<xss:attribute name="DefaultValue" type="xs:string" use="required">
<xss:annotation>
<xss/documentation>Default value for the parameter, used if the parameter value is not provided during application instantiation.</xss:documentation>
</xss:annotation>
</xss:attribute>
</xss:complexType>
</xss:element>
</xss:sequence>
</xss:complexType>
</xss:element>
<xss:element name="ServiceManifestImport" maxOccurs="unbounded">
<xss:annotation>
<xss/documentation>Imports a service manifest created by the service developer. A service manifest must be imported for each constituent service in the application. Configuration overrides and policies can be declared for the service manifest.</xss:documentation>
</xss:annotation>
<xss:complexType>
<xss:sequence>
<xss:element name="ServiceManifestRef" type="ServiceManifestRefType"/>
<xss:element name="ConfigOverrides" minOccurs="0">
<xss:annotation>
<xss/documentation>Describes configuration overrides for the imported service manifest. Configuration overrides allow the flexibility of re-using the same service manifests across multiple application types by overriding the service manifest's configuration only when used with a particular application type. Configuration overrides can change any default configuration in a service manifest as long as default configuration is defined using the Settings.xml in the ConfigPackage folder. </xss:documentation>
</xss:annotation>
</xss:complexType>
</xss:sequence>
<xss:element name="ConfigOverride" type="ConfigOverrideType" minOccurs="0" maxOccurs="unbounded"/>
</xss:sequence>
</xss:complexType>
</xss:element>
<xss:element name="ResourceOverrides" type="ResourceOverridesType" minOccurs="0"/>
<xss:element name="EnvironmentOverrides" type="EnvironmentOverridesType" minOccurs="0" maxOccurs="unbounded"/>
<xss:element name="Policies" type="ServiceManifestImportPoliciesType" minOccurs="0"/>
</xss:sequence>
</xss:complexType>
</xss:element>
<xss:element name="ServiceTemplates" type="ServiceTemplatesType" minOccurs="0">
<xss:annotation>
<xss/documentation>Declares the set of permitted service types that can be created dynamically inside the application instance. Default configuration values, such as replication factor, are specified and used as a template for creating service instances.</xss:documentation>

```

```

</xs:annotation>
</xs:element>
<xs:element name="DefaultServices" type="DefaultServicesType" minOccurs="0">

</xs:element>
<xs:element name="Principals" type="SecurityPrincipalsType" minOccurs="0"/>
<xs:element name="Policies" type="ApplicationPoliciesType" minOccurs="0"/>
<xs:element name="Diagnostics" type="DiagnosticsType" minOccurs="0"/>
<xs:element name="Certificates" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Declares certificates used to secure endpoints or encrypt secrets within the application manifest or a cluster manifest.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence maxOccurs="unbounded">
            <xs:element name="SecretsCertificate" type="FabricCertificateType" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>Declares a certificate used to encrypt sensitive information within the application manifest. The application author uses the Invoke-ServiceFabricEncryptSecret cmdlet to encrypt the sensitive information, which is copied to a Parameter in the ConfigOverrides section.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="EndpointCertificate" type="EndpointCertificateType" minOccurs="0"/>
        </xs:sequence>
        <xs:complexType>
    </xs:element>
</xs:sequence>
<xs:attributeGroup ref="ApplicationManifestAttrGroup"/>

</xs:complexType>

```

## Content element details

### Description

Text describing this application.

ATTRIBUTE	VALUE
name	Description
type	xs:string
minOccurs	0

### Parameters

Declares the parameters that are used in this application manifest. The value of these parameters can be supplied when the application is instantiated and can be used to override application or service configuration settings.

ATTRIBUTE	VALUE
name	Parameters
minOccurs	0

### ServiceManifestImport

Imports a service manifest created by the service developer. A service manifest must be imported for each constituent service in the application. Configuration overrides and policies can be declared for the service manifest.

ATTRIBUTE	VALUE
name	ServiceManifestImport
maxOccurs	unbounded

#### ServiceTemplates

Declares the set of permitted service types that can be created dynamically inside the application instance. Default configuration values, such as replication factor, are specified and used as a template for creating service instances.

ATTRIBUTE	VALUE
name	ServiceTemplates
type	ServiceTemplatesType
minOccurs	0

#### DefaultServices

ATTRIBUTE	VALUE
name	DefaultServices
type	DefaultServicesType
minOccurs	0

#### Principals

ATTRIBUTE	VALUE
name	Principals
type	SecurityPrincipalsType
minOccurs	0

#### Policies

ATTRIBUTE	VALUE
name	Policies
type	ApplicationPoliciesType
minOccurs	0

#### Diagnostics

ATTRIBUTE	VALUE
name	Diagnostics
type	DiagnosticsType

ATTRIBUTE	VALUE
minOccurs	0

#### Certificates

Declares certificates used to secure endpoints or encrypt secrets within the application manifest or a cluster manifest.

ATTRIBUTE	VALUE
name	Certificates
minOccurs	0

## ApplicationPackageType complexType

ApplicationPackage represents the versioned Application information required by the node.

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 2 attribute(s)
name	ApplicationPackageType

#### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ApplicationPackageType">
  <xss:annotation>
    <xss:documentation>ApplicationPackage represents the versioned Application information required by the
node.</xss:documentation>
  </xss:annotation>
  <xss:sequence>
    <xss:element name="DigestedEnvironment" type="EnvironmentType"/>
    <xss:element name="DigestedCertificates">
      <xss:complexType>
        <xss:sequence maxOccurs="unbounded">
          <xss:element name="SecretsCertificate" type="FabricCertificateType" minOccurs="0"/>
          <xss:element name="EndpointCertificate" type="EndpointCertificateType" minOccurs="0"/>
        </xss:sequence>
        <xss:attributeGroup ref="VersionedItemAttrGroup"/>
      </xss:complexType>
    </xss:element>
  </xss:sequence>
  <xss:attribute name="ApplicationTypeName" type="xs:string" use="required">
    <xss:annotation>
      <xss:documentation>Type identifier for this application.</xss:documentation>
    </xss:annotation>
  </xss:attribute>
  <xss:attributeGroup ref="VersionedItemAttrGroup"/>
  <xss:attributeGroup ref="ApplicationInstanceAttrGroup"/>
  <xss:attribute name="ContentChecksum" type="xs:string">
    <xss:annotation>
      <xss:documentation>Checksum value of this ApplicationPackage content</xss:documentation>
    </xss:annotation>
  </xss:attribute>
</xss:complexType>

```

## Attribute details

### ApplicationTypeName

Declares certificates used to secure endpoints or encrypt secrets within the application manifest or a cluster manifest.

ATTRIBUTE	VALUE
name	ApplicationTypeName
type	xs:string
use	required

### ContentChecksum

Declares certificates used to secure endpoints or encrypt secrets within the application manifest or a cluster manifest.

ATTRIBUTE	VALUE
name	ContentChecksum
type	xs:string

## Content element details

### DigestedEnvironment

ATTRIBUTE	VALUE
name	DigestedEnvironment
type	EnvironmentType

#### DigestedCertificates

ATTRIBUTE	VALUE
name	DigestedCertificates

## ApplicationPoliciesType complexType

Describes the policies (log collection, default run-as, health, and security access) to be applied at the application level.

ATTRIBUTE	VALUE
type	anonymous complexType
content	4 element(s), 0 attribute(s)
name	ApplicationPoliciesType

#### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ApplicationPoliciesType">
    <xss:annotation>
        <xss:documentation>Describes the policies (log collection, default run-as, health, and
security access) to be applied at the application level.</xss:documentation>
    </xss:annotation>
    <xss:all>
        <xss:element name="LogCollectionPolicies" minOccurs="0">
            <xss:annotation>
                <xss:documentation>Specifies whether log collection is enabled. Works
only in an Azure cluster environment</xss:documentation>
            </xss:annotation>
            <xss:complexType>
                <xss:sequence maxOccurs="unbounded">
                    <xss:element name="LogCollectionPolicy">
                        <xss:complexType>
                            <xss:attribute name="Path" type="xs:string"
use="optional"/>
                        </xss:complexType>
                    </xss:element>
                </xss:sequence>
            </xss:complexType>
        </xss:element>
        <xss:element name="DefaultRunAsPolicy" minOccurs="0">
            <xss:annotation>
                <xss:documentation>Specify a default user account for all service code
packages that don't have a specific RunAsPolicy defined in the ServiceManifestImport section.
</xss:documentation>
            </xss:annotation>
            <xss:complexType>
                <xss:attribute name="UserRef" type="xs:string" use="required">
                    <xss:annotation>
                        <xss:documentation>The user account that the service
code packages will run as. The user account must be declared in the Principals section. Often it is preferable
to run the setup entry point using a local system account rather than an administrators account.
</xss:documentation>
                </xss:annotation>
                </xss:attribute>
            </xss:complexType>
        </xss:element>
        <xss:element name="HealthPolicy" type="ApplicationHealthPolicyType" minOccurs="0"/>
        <xss:element name="SecurityAccessPolicies" minOccurs="0">
            <xss:annotation>
                <xss:documentation>List of security policies applied to resources at the
application level.</xss:documentation>
            </xss:annotation>
            <xss:complexType>
                <xss:sequence maxOccurs="unbounded">
                    <xss:element name="SecurityAccessPolicy"
type="SecurityAccessPolicyType"/>
                </xss:sequence>
            </xss:complexType>
        </xss:element>
    </xss:all>
</xss:complexType>

```

## Content element details

### LogCollectionPolicies

Specifies whether log collection is enabled. Works only in an Azure cluster environment

ATTRIBUTE	VALUE
name	LogCollectionPolicies

ATTRIBUTE	VALUE
minOccurs	0

#### DefaultRunAsPolicy

Specify a default user account for all service code packages that don't have a specific RunAsPolicy defined in the ServiceManifestImport section.

ATTRIBUTE	VALUE
name	DefaultRunAsPolicy
minOccurs	0

#### HealthPolicy

ATTRIBUTE	VALUE
name	HealthPolicy
type	ApplicationHealthPolicyType
minOccurs	0

#### SecurityAccessPolicies

List of security policies applied to resources at the application level.

ATTRIBUTE	VALUE
name	SecurityAccessPolicies
minOccurs	0

## AzureBlobETWType complexType

Describes an Azure blob store destination for ETW events. Works only in Azure environment.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	AzureBlobETWType

#### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="AzureBlobETWType">
  <xs:annotation>
    <xs:documentation>Describes an Azure blob store destination for ETW events. Works only in Azure
environment.</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="AzureBlobType">
      <xs:attributeGroup ref="LevelFilter"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

```

## AzureBlobType complexType

Describes an Azure blob store destination for diagnostics data. Works only in Azure cluster environment.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	AzureBlobType

### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="AzureBlobType">
  <xs:annotation>
    <xs:documentation>Describes an Azure blob store destination for diagnostics data. Works only in Azure
cluster environment.</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="AzureStoreBaseType">
      <xs:attributeGroup ref="ContainerName"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

```

## AzureRoleType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 3 attribute(s)
name	AzureRoleType

### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="AzureRoleType">
  <xs:attribute name="RoleName" type="xs:string" use="required"/>
  <xs:attribute name="NodeTypeRef" type="xs:string" use="required"/>
  <xs:attribute name="SeedNodeCount" type="xs:int" use="optional" default="0"/>
</xs:complexType>

```

## Attribute details

### RoleName

List of security policies applied to resources at the application level.

ATTRIBUTE	VALUE
name	RoleName
type	xs:string
use	required

### NodeTypeRef

List of security policies applied to resources at the application level.

ATTRIBUTE	VALUE
name	NodeTypeRef
type	xs:string
use	required

### SeedNodeCount

List of security policies applied to resources at the application level.

ATTRIBUTE	VALUE
name	SeedNodeCount
type	xs:int
use	optional
default	0

## AzureStore BaseType complexType

Describes a diagnostic store in an Azure storage account.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 1 attribute(s)

ATTRIBUTE	VALUE
name	AzureStoreBaseType

## XML source

```
<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="AzureStoreBaseType">
<xss:annotation>
<xss:documentation>Describes a diagnostic store in an Azure storage account.</xss:documentation>
</xss:annotation>
<xss:sequence>
<xss:element ref="Parameters" minOccurs="0"/>
</xss:sequence>
<xss:attributeGroup ref="Enabled"/>
<xss:attributeGroup ref="ConnectionString"/>
<xss:attribute name="ConnectionStringIsEncrypted" type="xs:string" use="required"/>
<xss:attributeGroup ref="UploadIntervalInMinutes"/>
<xss:attributeGroup ref="DataDeletionAgeInDays"/>
</xss:complexType>
```

## Attribute details

### ConnectionStringIsEncrypted

List of security policies applied to resources at the application level.

ATTRIBUTE	VALUE
name	ConnectionStringIsEncrypted
type	xs:string
use	required

## Content element details

### None

ATTRIBUTE	VALUE
ref	Parameters
minOccurs	0

## BlackbirdRoleType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 4 attribute(s)
name	BlackbirdRoleType

## XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="BlackbirdRoleType">
  <xs:attribute name="EnvironmentName" type="xs:string" use="required"/>
  <xs:attribute name="RoleName" type="xs:string" use="required"/>
  <xs:attribute name="NodeTypeRef" type="xs:string" use="required"/>
  <xs:attribute name="IsSeedNode" type="xs:boolean" use="optional" default="0"/>
</xs:complexType>

```

## Attribute details

### EnvironmentName

ATTRIBUTE	VALUE
name	EnvironmentName
type	xs:string
use	required

### RoleName

ATTRIBUTE	VALUE
name	RoleName
type	xs:string
use	required

### NodeTypeRef

ATTRIBUTE	VALUE
name	NodeTypeRef
type	xs:string
use	required

### IsSeedNode

ATTRIBUTE	VALUE
name	IsSeedNode
type	xs:boolean
use	optional
default	0

## CertificatesType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	4 element(s), 0 attribute(s)
name	CertificatesType

## XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="CertificatesType">
  <xs:all>
    <xs:element name="ClusterCertificate" type="FabricCertificateType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>The certificate used to secure the intra cluster communication.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="ServerCertificate" type="FabricCertificateType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>The certificate used to secure the intra cluster communication.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="ClientCertificate" type="FabricCertificateType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>The default admin role client certificate used to secure client server
communication.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="UserRoleClientCertificate" type="FabricCertificateType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>The default user role client certificate used to secure client server
communication.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:all>
</xs:complexType>

```

## Content element details

### ClusterCertificate

The certificate used to secure the intra cluster communication.

ATTRIBUTE	VALUE
name	ClusterCertificate
type	FabricCertificateType
minOccurs	0

### ServerCertificate

The certificate used to secure the intra cluster communication.

ATTRIBUTE	VALUE
name	ServerCertificate
type	FabricCertificateType

ATTRIBUTE	VALUE
minOccurs	0

#### **ClientCertificate**

The default admin role client certificate used to secure client server communication.

ATTRIBUTE	VALUE
name	ClientCertificate
type	FabricCertificateType
minOccurs	0

#### **UserRoleClientCertificate**

The default user role client certificate used to secure client server communication.

ATTRIBUTE	VALUE
name	UserRoleClientCertificate
type	FabricCertificateType
minOccurs	0

## ClusterManifestType complexType

Describes a Microsoft Azure Service Fabric Cluster.

ATTRIBUTE	VALUE
type	anonymous complexType
content	4 element(s), 3 attribute(s)
name	ClusterManifestType

#### **XML source**

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ClusterManifestType">
    <xs:annotation>
        <xs:documentation>Describes a Microsoft Azure Service Fabric Cluster.</xs:documentation>
    </xs:annotation>
    <xs:documentation>
        </xs:annotation>
        <xs:all>
            <xs:element name="NodeTypes" minOccurs="1">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="NodeType" maxOccurs="unbounded">
                            <xs:annotation>
                                <xs:documentation>Describe a node type.</xs:documentation>
                            </xs:annotation>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:all>
    </xs:documentation>
</xs:complexType>

```

```

                <xs:element name="Endpoints"
type="FabricEndpointsType" minOccurs="0">
                        <xs:annotation>

<xs:documentation>Describe the endpoints associated with this node type</xs:documentation>
                        </xs:annotation>
                </xs:element>
                <xs:element name="KtlLoggerSettings"
type="FabricKtlLoggerSettingsType" minOccurs="0">
                        <xs:annotation>
<xs:documentation>Describe the
KtlLogger information associated with this node type</xs:documentation>
                        </xs:annotation>
                </xs:element>
                <xs:element name="LogicalDirectories"
minOccurs="0">
                        <xs:annotation>
<xs:documentation>Describe the
LogicalDirectories settings associated with this node type</xs:documentation>
                        </xs:annotation>
<xs:complexType>
                        <xs:sequence>
                                <xs:element
name="LogicalDirectory" type="LogicalDirectoryType" maxOccurs="unbounded"/>
                                </xs:sequence>
                        </xs:complexType>
                </xs:element>
                <xs:element name="Certificates"
type="CertificatesType" minOccurs="0">
                        <xs:annotation>

<xs:documentation>Describe the certificates associated with this node type</xs:documentation>
                        </xs:annotation>
                </xs:element>
                <xs:element name="PlacementProperties"
minOccurs="0">
                        <xs:annotation>

<xs:documentation>Describe the properties for this NodeType that will be used as placement
constraints</xs:documentation>
                        </xs:annotation>
<xs:complexType>
                        <xs:sequence>
                                <xs:element
name="Property" type="KeyValuePairType" minOccurs="0" maxOccurs="unbounded"/>
                                </xs:sequence>
                        </xs:complexType>
                </xs:element>
                <xs:element name="Capacities"
minOccurs="0">
                        <xs:annotation>
<xs:documentation>The
capacities of various metrics for this node type</xs:documentation>
                        </xs:annotation>
<xs:complexType>
                        <xs:sequence>
                                <xs:element
name="Capacity" type="KeyValuePairType" minOccurs="0" maxOccurs="unbounded"/>
                                </xs:sequence>
                        </xs:complexType>
                </xs:element>
                </xs:all>
                <xs:attribute name="Name" type="xs:string"
use="required">
                        <xs:annotation>
<xs:documentation>Name of the
NodeType</xs:documentation>
                        </xs:annotation>
                </xs:attribute>

```

```

</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="Infrastructure">
<xs:complexType>
<xs:choice>
<xs:element name="WindowsServer">
<xs:complexType>
<xs:complexContent>
<xs:extension
base="WindowsInfrastructureType">
<xs:attribute name="IsScaleMin"
type="xs:boolean" default="false"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
</xs:element>
<xs:element name="Linux">
<xs:complexType>
<xs:complexContent>
<xs:extension base="LinuxInfrastructureType">
<xs:attribute name="IsScaleMin" type="xs:boolean" default="false"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
</xs:element>
<xs:element name="WindowsAzure">
<xs:complexType>
<xs:sequence>
<xs:element name="Roles">
<xs:complexType>
<xs:sequence>
<xs:element name="Role" type="AzureRoleType" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="WindowsAzureStaticTopology">
<xs:complexType>
<xs:complexContent>
<xs:extension
name="Role" type="AzureRoleType" maxOccurs="unbounded"/>
</xs:complexContent>
</xs:complexType>
</xs:element>
<xs:element name="Blackbird">
<xs:complexType>
<xs:sequence>
<xs:element name="Roles">
<xs:complexType>
<xs:sequence>
<xs:element name="Role" type="BlackbirdRoleType" minOccurs="1" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="PaaS">
<xs:complexType>
<xs:all>
<xs:element name="Roles">

```

```

<xs:complexType>
  <xs:sequence>
    <xs:element name="Role" type="PaaSRoleType" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="Votes">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Vote" type="PaaSVoteType" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:all>
  <xs:complexType>
    <xs:element name="FabricSettings" type="SettingsOverridesType" minOccurs="0"/>
    <xs:element name="Certificates" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="SecretsCertificate" type="FabricCertificateType" minOccurs="0"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:all>
  <xs:attribute name="Name" use="required">
    <xs:annotation>
      <xs:documentation>Name of the Cluster.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="Version" use="required">
    <xs:annotation>
      <xs:documentation>User defined version string for the cluster manifest document.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="Description">
    <xs:annotation>
      <xs:documentation>Description for the Cluster Manifest.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>

```

## Attribute details

### Name

The default user role client certificate used to secure client server communication.

ATTRIBUTE	VALUE
name	Name
use	required

### Version

The default user role client certificate used to secure client server communication.

ATTRIBUTE	VALUE
name	Version
use	required

#### Description

The default user role client certificate used to secure client server communication.

ATTRIBUTE	VALUE
name	Description

#### Content element details

##### NodeTypes

ATTRIBUTE	VALUE
name	NodeTypes
minOccurs	1

##### Infrastructure

ATTRIBUTE	VALUE
name	Infrastructure

##### FabricSettings

ATTRIBUTE	VALUE
name	FabricSettings
type	SettingsOverridesType
minOccurs	0

##### Certificates

ATTRIBUTE	VALUE
name	Certificates
minOccurs	0

## CodePackageType complexType

Describes a code package that supports a defined service type. When a service is instantiated against one of these service types, all code packages declared in this manifest are activated by running their entry points. The resulting processes are expected to register the supported service types at run time. When there are multiple code packages, they are all activated whenever the system looks for any one of the declared service types.

ATTRIBUTE	VALUE
type	anonymous complexType
content	3 element(s), 1 attribute(s)
name	CodePackageType

## XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="CodePackageType">
    <xs:annotation>
        <xs:documentation>Describes a code package that supports a defined service type. When a service is
instantiated against one of these service types, all code packages declared in this manifest are activated by
running their entry points. The resulting processes are expected to register the supported service types at run
time. When there are multiple code packages, they are all activated whenever the system looks for any one of
the declared service types.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="SetupEntryPoint" minOccurs="0">
            <xs:annotation>
                <xs:documentation>A privileged entry point that runs with the same credentials as Service Fabric
(typically the LocalSystem account) before any other entry point. The executable specified by EntryPoint is
typically the long-running service host. The presence of a separate setup entry point avoids having to run the
service host with high privileges for extended periods of time.</xs:documentation>
            </xs:annotation>
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="ExeHost" type="ExeHostEntryPointType"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="EntryPoint" type="EntryPointDescriptionType" minOccurs="1"/>
        <xs:element name="EnvironmentVariables" type="EnvironmentVariablesType" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attributeGroup ref="VersionedName"/>
    <xs:attribute name="IsShared" type="xs:boolean" default="false">
        <xs:annotation>
            <xs:documentation>Indicates if the contents of this code package are shared by other code packages. If
true, on an upgrade of this code package, all code packages will be restarted. This attribute is currently not
supported and it's value will be ignored.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:complexType>

```

## Attribute details

### IsShared

ATTRIBUTE	VALUE
name	IsShared
type	xs:boolean
default	false

## Content element details

### SetupEntryPoint

A privileged entry point that runs with the same credentials as Service Fabric (typically the LocalSystem account) before any other entry point. The executable specified by EntryPoint is typically the long-running service host. The presence of a separate setup entry point avoids having to run the service host with high privileges for extended periods of time.

ATTRIBUTE	VALUE
name	SetupEntryPoint
minOccurs	0

#### EntryPoint

ATTRIBUTE	VALUE
name	EntryPoint
type	EntryPointDescriptionType
minOccurs	1

#### EnvironmentVariables

ATTRIBUTE	VALUE
name	EnvironmentVariables
type	EnvironmentVariablesType
minOccurs	0
maxOccurs	1

## ConfigOverrideType complexType

Describes the configuration overrides for a particular config package in the imported service manifest.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 1 attribute(s)
name	ConfigOverrideType

#### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ConfigOverrideType">
  <xs:annotation>
    <xs:documentation>Describes the configuration overrides for a particular config package in the imported
service manifest.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="Settings" type="SettingsOverridesType" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="Name" use="required">
    <xs:annotation>
      <xs:documentation>The name of the configuration package in the service manifest which contains the
setting(s) to be overridden.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>

```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
use	required

## Content element details

### Settings

ATTRIBUTE	VALUE
name	Settings
type	SettingsOverridesType
minOccurs	0

## ConfigPackageType complexType

Declares a folder, named by the Name attribute, that contains a Settings.xml file. This file contains sections of user-defined, key-value pair settings that the process can read back at run time. During an upgrade, if only the ConfigPackage version has changed, then the running process is not restarted. Instead, a callback notifies the process that configuration settings have changed so they can be reloaded dynamically.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	ConfigPackageType

## XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ConfigPackageType">
    <xs:annotation>
        <xs:documentation>Declares a folder, named by the Name attribute, that contains a Settings.xml file.
This file contains sections of user-defined, key-value pair settings that the process can read back at run
time. During an upgrade, if only the ConfigPackage version has changed, then the running process is not
restarted. Instead, a callback notifies the process that configuration settings have changed so they can be
reloaded dynamically.</xs:documentation>
    </xs:annotation>
    <xs:attributeGroup ref="VersionedName"/>
</xs:complexType>
```

## ContainerCertificateType complexType

Specifies information about an X509 certificate which is to be exposed to the container environment.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 8 attribute(s)
name	ContainerCertificateType

## XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ContainerCertificateType">
    <xss:annotation>
        <xss:documentation>Specifies information about an X509 certificate which is to be exposed to the
container environment.</xss:documentation>
    </xss:annotation>
    <xss:attribute name="X509StoreName" type="xs:string" default="My">
        <xss:annotation>
            <xss:documentation>The store name for the X509 certificate.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="X509FindValue" type="xs:string" use="optional">
        <xss:annotation>
            <xss:documentation>The thumbprint of the X509 certificate.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="DataPackageRef" type="xs:string" use="optional">
        <xss:annotation>
            <xss:documentation>The name of data package that has the certificate files.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="DataPackageVersion" type="xs:string" use="optional">
        <xss:annotation>
            <xss:documentation>The version of data package that has the certificate files.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="RelativePath" type="xs:string" use="optional">
        <xss:annotation>
            <xss:documentation>The relative path to the certificate file inside data package.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="Password" type="xs:string" use="optional">
        <xss:annotation>
            <xss:documentation>Password/Private key for the certificate.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="IsPasswordEncrypted" type="xs:boolean" default="false">
        <xss:annotation>
            <xss:documentation>If true, the value of password is encrypted.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="Name" type="xs:string" use="required">
        <xss:annotation>
            <xss:documentation>Identifier for the specific certificate information. This name is used to set the
environment variable in the container.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
</xss:complexType>

```

## Attribute details

### X509StoreName

ATTRIBUTE	VALUE
name	X509StoreName
type	xs:string
default	My

### X509FindValue

ATTRIBUTE	VALUE
name	X509FindValue
type	xs:string
use	optional

#### DataPackageRef

ATTRIBUTE	VALUE
name	DataPackageRef
type	xs:string
use	optional

#### DataPackageVersion

ATTRIBUTE	VALUE
name	DataPackageVersion
type	xs:string
use	optional

#### RelativePath

ATTRIBUTE	VALUE
name	RelativePath
type	xs:string
use	optional

#### Password

ATTRIBUTE	VALUE
name	Password
type	xs:string
use	optional

#### IsPasswordEncrypted

ATTRIBUTE	VALUE
name	IsPasswordEncrypted
type	xs:boolean

ATTRIBUTE	VALUE
default	false

#### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

## ContainerHostEntryPointType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	4 element(s), 0 attribute(s)
name	ContainerHostEntryPointType

#### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ContainerHostEntryPointType">
  <xs:sequence>
    <!--container image name-->
    <xs:element name="ImageName" type="xs:string"/>
    <!--comma delimited list of commands for container-->
    <xs:element name="Commands" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="EntryPoint" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="FromSource" type="xs:string" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

```

#### Content element details

##### ImageName

ATTRIBUTE	VALUE
name	ImageName
type	xs:string

##### Commands

ATTRIBUTE	VALUE
name	Commands
type	xs:string

ATTRIBUTE	VALUE
minOccurs	0
maxOccurs	1

#### EntryPoint

ATTRIBUTE	VALUE
name	EntryPoint
type	xs:string
minOccurs	0
maxOccurs	1

#### FromSource

ATTRIBUTE	VALUE
name	FromSource
type	xs:string
minOccurs	0
maxOccurs	1

## ContainerHostPoliciesType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	7 element(s), 3 attribute(s)
name	ContainerHostPoliciesType

#### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ContainerHostPoliciesType">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="RepositoryCredentials" type="RepositoryCredentialsType" minOccurs="0" maxOccurs="1">
            <xs:annotation>
                <xs:documentation>Credentials for container image repository to pull images from.</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="PortBinding" type="PortBindingType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
                <xs:documentation>Specifies which endpoint resource to bind container exposed port.</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="CertificateRef" type="ContainerCertificateType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
                <xs:documentation>Specifies information for a certificate which will be exposed to the container.</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="LogConfig" type="ContainerLoggingDriverType" minOccurs="0" maxOccurs="1">
            <xs:annotation>
                <xs:documentation>Specifies the logging driver for a container.</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="NetworkConfig" type="ContainerNetworkConfigType" minOccurs="0" maxOccurs="1">
            <xs:annotation>
                <xs:documentation>Specifies the network configuration for a container.</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="Volume" type="ContainerVolumeType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
                <xs:documentation>Specifies the volume to be bound to container.</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="SecurityOption" type="SecurityOptionsType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
                <xs:documentation>Specifies securityoptions for the container.</xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:choice>
    <xs:attribute name="CodePackageRef" use="required">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:minLength value="1"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="Isolation" use="optional" type="xs:string">
        <xs:annotation>
            <xs:documentation>Isolation mode for container. Valid values are default, process or hyperv (only supported for windows containers).</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="Hostname" use="optional" type="xs:string">
        <xs:annotation>
            <xs:documentation>Specify Hostname for container.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:complexType>

```

## Attribute details

### CodePackageRef

ATTRIBUTE	VALUE
name	CodePackageRef
use	required

#### Isolation

ATTRIBUTE	VALUE
name	Isolation
use	optional
type	xs:string

#### Hostname

ATTRIBUTE	VALUE
name	Hostname
use	optional
type	xs:string

### Content element details

#### RepositoryCredentials

Credentials for container image repository to pull images from.

ATTRIBUTE	VALUE
name	RepositoryCredentials
type	RepositoryCredentialsType
minOccurs	0
maxOccurs	1

#### PortBinding

Specifies which endpoint resource to bind container exposed port.

ATTRIBUTE	VALUE
name	PortBinding
type	PortBindingType
minOccurs	0
maxOccurs	unbounded

#### CertificateRef

Specifies information for a certificate which will be exposed to the container.

ATTRIBUTE	VALUE
name	CertificateRef
type	ContainerCertificateType
minOccurs	0
maxOccurs	unbounded

#### **LogConfig**

Specifies the logging driver for a container.

ATTRIBUTE	VALUE
name	LogConfig
type	ContainerLoggingDriverType
minOccurs	0
maxOccurs	1

#### **NetworkConfig**

Specifies the network configuration for a container.

ATTRIBUTE	VALUE
name	NetworkConfig
type	ContainerNetworkConfigType
minOccurs	0
maxOccurs	1

#### **Volume**

Specifies the volume to be bound to container.

ATTRIBUTE	VALUE
name	Volume
type	ContainerVolumeType
minOccurs	0
maxOccurs	unbounded

#### **SecurityOption**

Specifies securityoptions for the container.

ATTRIBUTE	VALUE
name	SecurityOption
type	SecurityOptionsType
minOccurs	0
maxOccurs	unbounded

## ContainerLoggingDriverType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 1 attribute(s)
name	ContainerLoggingDriverType

### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ContainerLoggingDriverType">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="DriverOption" type="DriverOptionType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
                <xs:documentation>Driver options to be passed to driver.</xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:choice>
    <xs:attribute name="Driver" use="required">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:minLength value="1"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
</xs:complexType>

```

### Attribute details

#### Driver

Specifies securityoptions for the container.

ATTRIBUTE	VALUE
name	Driver
use	required

### Content element details

#### DriverOption

Driver options to be passed to driver.

ATTRIBUTE	VALUE
name	DriverOption
type	DriverOptionType
minOccurs	0
maxOccurs	unbounded

## ContainerNetworkConfigType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)
name	ContainerNetworkConfigType

### XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ContainerNetworkConfigType">
    <xs:attribute name="NetworkType" use="required" type="xs:string">
        <xs:annotation>
            <xs:documentation>NetworkType. Currently only supported type is "Open".</xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:complexType>
```

### Attribute details

#### NetworkType

Driver options to be passed to driver.

ATTRIBUTE	VALUE
name	NetworkType
use	required
type	xs:string

## ContainerVolumeType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 4 attribute(s)
name	ContainerVolumeType

## XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ContainerVolumeType">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="DriverOption" type="DriverOptionType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
                <xs:documentation>Driver options to be passed to driver.</xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:choice>
    <xs:attribute name="Source" use="required">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:minLength value="1"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="Destination" use="required">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:minLength value="1"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="Driver" use="optional">
        <xs:simpleType>
            <xs:restriction base="xs:string">
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="IsReadOnly" type="xs:boolean" default="false"/>
</xs:complexType>
```

## Attribute details

### Source

Driver options to be passed to driver.

ATTRIBUTE	VALUE
name	Source
use	required

### Destination

Driver options to be passed to driver.

ATTRIBUTE	VALUE
name	Destination
use	required

### Driver

Driver options to be passed to driver.

ATTRIBUTE	VALUE
name	Driver

ATTRIBUTE	VALUE
use	optional

#### IsReadOnly

Driver options to be passed to driver.

ATTRIBUTE	VALUE
name	IsReadOnly
type	xs:boolean
default	false

#### Content element details

##### DriverOption

Driver options to be passed to driver.

ATTRIBUTE	VALUE
name	DriverOption
type	DriverOptionType
minOccurs	0
maxOccurs	unbounded

## DataPackageType complexType

Declares a folder, named by the Name attribute, which contains static data files. Service Fabric will recycle all EXEs and DLLHOSTs specified in the host and support packages when any of the data packages listed in the service manifest are upgraded.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	DataPackageType

#### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DataPackageType">
    <xss:annotation>
        <xss:documentation>Declares a folder, named by the Name attribute, which contains static data files.
Service Fabric will recycle all EXEs and DLLHOSTs specified in the host and support packages when any of the
data packages listed in the service manifest are upgraded.</xss:documentation>
    </xss:annotation>
    <xss:attributeGroup ref="VersionedName"/>
</xss:complexType>

```

## DebugParametersType complexType

Specifies information on debugger to attach when activating codepackage

ATTRIBUTE	VALUE
type	anonymous complexType
content	3 element(s), 10 attribute(s)
name	DebugParametersType

### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DebugParametersType">
    <xss:annotation>
        <xss:documentation>Specifies information on debugger to attach when activating
codepackage</xss:documentation>
    </xss:annotation>
    <xss:sequence>
        <xss:element name="ContainerEntryPoint" type="xs:string" minOccurs="0" maxOccurs="unbounded">
            <xss:annotation>
                <xss:documentation>Overidden entrypoint for containers so debugger can be launched..</xss:documentation>
            </xss:annotation>
        </xss:element>
        <xss:element name="ContainerMountedVolume" type="xs:string" minOccurs="0" maxOccurs="unbounded">
            <xss:annotation>
                <xss:documentation>Volumes to be mounted inside container.</xss:documentation>
            </xss:annotation>
        </xss:element>
        <xss:element name="ContainerEnvironmentBlock" type="xs:string" minOccurs="0" maxOccurs="unbounded">
            <xss:annotation>
                <xss:documentation>EnvironmentBlock for containers.</xss:documentation>
            </xss:annotation>
        </xss:element>
    </xss:sequence>
    <xss:attribute name="ProgramExePath">
        <xss:simpleType>
            <xss:restriction base="xs:string">
                <xss:minLength value="1"/>
            </xss:restriction>
        </xss:simpleType>
    </xss:attribute>
    <xss:attribute name="Arguments">
        <xss:simpleType>
            <xss:restriction base="xs:string">
                <xss:minLength value="1"/>
            </xss:restriction>
        </xss:simpleType>
    </xss:attribute>
</xss:complexType>

```

```

<xs:attribute name="EntryPointType" use="optional" default="Main">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Setup"/>
      <xs:enumeration value="Main"/>
      <xs:enumeration value="All"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
<xs:attribute name="CodePackageLinkFolder">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
<xs:attribute name="ConfigPackageLinkFolder">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
<xs:attribute name="DataPackageLinkFolder">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
<xs:attribute name="LockFile">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
<xs:attribute name="WorkingFolder">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
<xs:attribute name="DebugParametersFile">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
<xs:attribute name="EnvironmentBlock">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
</xs:complexType>

```

## Attribute details

### ProgramExePath

Driver options to be passed to driver.

ATTRIBUTE	VALUE
name	ProgramExePath

#### Arguments

Driver options to be passed to driver.

ATTRIBUTE	VALUE
name	Arguments

#### EntryPointType

Driver options to be passed to driver.

ATTRIBUTE	VALUE
name	EntryPointType
use	optional
default	Main

#### CodePackageLinkFolder

Driver options to be passed to driver.

ATTRIBUTE	VALUE
name	CodePackageLinkFolder

#### ConfigPackageLinkFolder

Driver options to be passed to driver.

ATTRIBUTE	VALUE
name	ConfigPackageLinkFolder

#### DataPackageLinkFolder

Driver options to be passed to driver.

ATTRIBUTE	VALUE
name	DataPackageLinkFolder

#### LockFile

Driver options to be passed to driver.

ATTRIBUTE	VALUE
name	LockFile

#### WorkingFolder

Driver options to be passed to driver.

ATTRIBUTE	VALUE
name	WorkingFolder

#### **DebugParametersFile**

Driver options to be passed to driver.

ATTRIBUTE	VALUE
name	DebugParametersFile

#### **EnvironmentBlock**

Driver options to be passed to driver.

ATTRIBUTE	VALUE
name	EnvironmentBlock

### **Content element details**

#### **ContainerEntryPoint**

Overidden entrypoint for containers so debugger can be launched..

ATTRIBUTE	VALUE
name	ContainerEntryPoint
type	xs:string
minOccurs	0
maxOccurs	unbounded

#### **ContainerMountedVolume**

Volumes to be mounted inside container.

ATTRIBUTE	VALUE
name	ContainerMountedVolume
type	xs:string
minOccurs	0
maxOccurs	unbounded

#### **ContainerEnvironmentBlock**

EnvironmentBlock for containers.

ATTRIBUTE	VALUE
name	ContainerEnvironmentBlock
type	xs:string

ATTRIBUTE	VALUE
minOccurs	0
maxOccurs	unbounded

## DefaultServicesType complexType

Declares service instances that are automatically created whenever an application is instantiated against this application type.

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 0 attribute(s)
name	DefaultServicesType

### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DefaultServicesType">
    <xss:annotation>
        <xss:documentation>Declares service instances that are automatically created whenever an application
is instantiated against this application type.</xss:documentation>
    </xss:annotation>
    <xss:sequence>
        <xss:choice minOccurs="0" maxOccurs="unbounded">
            <xss:element name="Service">
                <xss:annotation>
                    <xss:documentation>Declares a service to be created automatically when the application
is instantiated.</xss:documentation>
                </xss:annotation>
                <xss:complexType>
                    <xss:choice minOccurs="0">
                        <xss:element name="StatelessService" type="StatelessServiceType"/>
                        <xss:element name="StatefulService" type="StatefulServiceType"/>
                    </xss:choice>
                    <xss:attribute name="Name" type="xs:string" use="required">
                        <xss:annotation>
                            <xss:documentation>The service name, used to form the fully qualified
application name URI. The fully qualified name URI of the service would be:
fabric:/ApplicationName/ServiceName.</xss:documentation>
                        </xss:annotation>
                    </xss:attribute>
                    <xss:attribute name="GeneratedIdRef" type="xs:string" use="optional">
                        <xss:annotation>
                            <xss:documentation>Reference to the auto generated id used by Visual Studio
tooling.</xss:documentation>
                        </xss:annotation>
                    </xss:attribute>
                    <xss:attribute name="ServiceDnsName" type="xs:string" use="optional">
                        <xss:annotation>
                            <xss:documentation>The DNS name of the service.</xss:documentation>
                        </xss:annotation>
                    </xss:attribute>
                    <xss:attribute name="ServicePackageActivationMode" use="optional">
                        <xss:annotation>
                            <xss:documentation>ServicePackageActivationMode to be used when creating the
service. With SharedProcess mode, replica(s) or instance(s) from different partition(s) of service will share
default="SharedProcess">
                        </xss:annotation>
                    </xss:attribute>
                </xss:complexType>
            </xss:element>
        </xss:choice>
    </xss:sequence>
</xss:complexType>

```

```

<xs:annotation>
    <xs:documentation>A collection of services that are automatically located together, so they are also moved together during fail-over or resource management.</xs:documentation>
</xs:annotation>
<xs:complexType>
    <xs:choice minOccurs="0">
        <xs:element name="StatelessServiceGroup" type="StatelessServiceGroupType"/>
        <xs:element name="StatefulServiceGroup" type="StatefulServiceGroupType"/>
    </xs:choice>
    <xs:attribute name="Name" type="xs:string" use="required">
        <xs:annotation>
            <xs:documentation>Name of this service relative to this application Name URI. Fully qualified Name of the service is a combination of Name Uri of the Application and this Name.</xs:documentation>
        </xs:annotation>
        <xs:attribute name="ServicePackageActivationMode" use="optional" default="SharedProcess">
            <xs:annotation>
                <xs:documentation>ServicePackageActivationMode to be used when creating the service. With SharedProcess mode, replica(s) or instance(s) from different partition(s) of service will share same same activation of service package on a node. With ExclusiveProcess mode, each replica or instance of service will have its own dedicated activation of service package.</xs:documentation>
            </xs:annotation>
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="SharedProcess"/>
                    <xs:enumeration value="ExclusiveProcess"/>
                </xs:restriction>
            </xs:simpleType>
            <xs:attribute>
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="ServiceGroup" type="ServiceGroupType"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:attribute>
        </xs:complexType>
    </xs:element>
</xs:choice>
</xs:sequence>
</xs:complexType>

```

## Content element details

### Service

Declares a service to be created automatically when the application is instantiated.

ATTRIBUTE	VALUE
name	Service

### ServiceGroup

A collection of services that are automatically located together, so they are also moved together during fail-over or resource management.

ATTRIBUTE	VALUE
name	ServiceGroup

## DiagnosticsType complexType

Describes the diagnostic settings for applications.

ATTRIBUTE	VALUE
type	anonymous complexType
content	3 element(s), 0 attribute(s)
name	DiagnosticsType

### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DiagnosticsType">
  <xs:annotation>
    <xs:documentation>Describes the diagnostic settings for applications.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="CrashDumpSource" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Specifies crash dump collection. Crash dumps are collected for executables that
host the code packages of all services belonging to the application.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Destinations" minOccurs="0">
            <xs:annotation>
              <xs:documentation>Destinations to which the crash dumps need to be transferred.
            </xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element ref="Parameters" minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
        <xs:attribute name="IsEnabled" type="xs:string">
          <xs:annotation>
            <xs:documentation>Whether or not crash dump collection is enabled. By default, it is not enabled.
          </xs:documentation>
          </xs:annotation>
        </xs:attribute>
      </xs:complexType>
    </xs:element>
    <xs:element name="ETWSource" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Specifies ETW trace collection. ETW traces are collected for the providers that are
registered by all services belonging to the application.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Destinations" minOccurs="0">
            <xs:annotation>

```

```

<xs:annotation>
    <xs:documentation>Destinations to which the crash dumps need to be transferred.
</xs:documentation>
</xs:annotation>
<xs:complexType>
    <xs:sequence>
        <xs:element name="LocalStore" type="LocalStoreETWType" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="FileStore" type="FileStoreETWType" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="AzureBlob" type="AzureBlobETWType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element ref="Parameters" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
<xs:attribute name="IsEnabled" type="xs:string">
    <xs:annotation>
        <xs:documentation>Whether or not ETW trace collection is enabled. By default, it is not enabled.
</xs:documentation>
    </xs:annotation>
</xs:attribute>
</xs:complexType>
</xs:element>
<xs:element name="FolderSource" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>Specifies the collection of the contents of a particular folder on the local node.
</xs:documentation>
    </xs:annotation>
</xs:complexType>
    <xs:sequence>
        <xs:element name="Destinations" minOccurs="0">
            <xs:annotation>
                <xs:documentation>Destinations to which the folder contents need to be transferred.
</xs:documentation>
            </xs:annotation>
</xs:element>
        <xs:element ref="Parameters" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
<xs:attribute name="IsEnabled" type="xs:string">
    <xs:annotation>
        <xs:documentation>Whether or not collection of the contents of this folder is enabled. By
default, it is not enabled.</xs:documentation>
    </xs:annotation>
</xs:attribute>
<xs:attributeGroup ref="RelativeFolderPath"/>
<xs:attributeGroup ref="DataDeletionAgeInDays"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>

```

## Content element details

### CrashDumpSource

Specifies crash dump collection. Crash dumps are collected for executables that host the code packages of all services belonging to the application.

ATTRIBUTE	VALUE
name	CrashDumpSource

ATTRIBUTE	VALUE
minOccurs	0

#### ETWSource

Specifies ETW trace collection. ETW traces are collected for the providers that are registered by all services belonging to the application.

ATTRIBUTE	VALUE
name	ETWSource
minOccurs	0

#### FolderSource

Specifies the collection of the contents of a particular folder on the local node.

ATTRIBUTE	VALUE
name	FolderSource
minOccurs	0
maxOccurs	unbounded

## DllHostEntryPointType complexType

Unsupported, do not use. DLL hosting support (assembly entry point) is provided through the FWP.exe process. Service Fabric starts the Fabric Worker Process (FWP.exe) and loads the assembly as part of the activation process.

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 1 attribute(s)
name	DllHostEntryPointType

#### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DllHostEntryPointType">
  <xss:annotation>
    <xss:documentation>Unsupported, do not use. DLL hosting support (assembly entry point) is provided
through the FWP.exe process. Service Fabric starts the Fabric Worker Process (FWP.exe) and loads the assembly
as part of the activation process.</xss:documentation>
  </xss:annotation>
  <xss:sequence>
    <xss:choice minOccurs="0" maxOccurs="unbounded">
      <xss:element name="UnmanagedDll" type="UnmanagedDllType"/>
      <xss:element name="ManagedAssembly" type="ManagedAssemblyType"/>
    </xss:choice>
  </xss:sequence>
  <xss:attribute name="IsolationPolicy" use="optional" default="DedicatedProcess">
    <xss:annotation>
      <xss:documentation>Unsupported, do not use. Defines the isolation policy for the Unmanaged DLLs and
Managed Assemblies loaded in the DllHost. </xss:documentation>
    </xss:annotation>
    <xss:simpleType>
      <xss:restriction base="xs:string">
        <xss:enumeration value="SharedDomain"/>
        <xss:enumeration value="DedicatedDomain"/>
        <xss:enumeration value="DedicatedProcess"/>
      </xss:restriction>
    </xss:simpleType>
  </xss:attribute>
</xss:complexType>

```

## Attribute details

### IsolationPolicy

Specifies the collection of the contents of a particular folder on the local node.

ATTRIBUTE	VALUE
name	IsolationPolicy
use	optional
default	DedicatedProcess

## Content element details

### UnmanagedDll

ATTRIBUTE	VALUE
name	UnmanagedDll
type	UnmanagedDllType

### ManagedAssembly

ATTRIBUTE	VALUE
name	ManagedAssembly
type	ManagedAssemblyType

## DriverOptionType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	DriverOptionType

## XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="DriverOptionType">
    <xs:attribute name="Name" type="xs:string" use="required"/>
    <xs:attribute name="Value" type="xs:string" use="required"/>
</xs:complexType>
```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

### Value

ATTRIBUTE	VALUE
name	Value
type	xs:string
use	required

## EndpointBindingPolicyType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	EndpointBindingPolicyType

## XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EndpointBindingPolicyType">
  <xss:attribute name="EndpointRef">
    <xss:annotation>
      <xss:documentation>The name of the endpoint, which must be declared in the Resources section of the
service manifest.</xss:documentation>
    </xss:annotation>
    <xss:simpleType>
      <xss:restriction base="xs:string">
        <xss:minLength value="1"/>
      </xss:restriction>
    </xss:simpleType>
  </xss:attribute>
  <xss:attribute name="CertificateRef" use="required">
    <xss:annotation>
      <xss:documentation>The name of the endpoint certificate, declared in the Certificates section, to return
to the client. </xss:documentation>
    </xss:annotation>
    <xss:simpleType>
      <xss:restriction base="xs:string">
        <xss:minLength value="1"/>
      </xss:restriction>
    </xss:simpleType>
  </xss:attribute>
</xss:complexType>

```

## Attribute details

### EndpointRef

ATTRIBUTE	VALUE
name	EndpointRef

### CertificateRef

ATTRIBUTE	VALUE
name	CertificateRef
use	required

## EndpointCertificateType complexType

Specifies information about an X509 certificate used to secure an endpoint.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 3 attribute(s)
name	EndpointCertificateType

## XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EndpointCertificateType">
    <xss:annotation>
        <xss:documentation>Specifies information about an X509 certificate used to secure an
endpoint.</xss:documentation>
    </xss:annotation>
    <xss:attribute name="X509StoreName" type="xs:string" default="My">
        <xss:annotation>
            <xss:documentation>The store name for the X509 certificate.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="X509FindValue" use="required">
        <xss:annotation>
            <xss:documentation>The thumbprint of the X509 certificate.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="Name" type="xs:string" use="optional"/>
</xss:complexType>

```

## Attribute details

### X509StoreName

ATTRIBUTE	VALUE
name	X509StoreName
type	xs:string
default	My

### X509FindValue

ATTRIBUTE	VALUE
name	X509FindValue
use	required

### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	optional

## EndpointOverrideType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 6 attribute(s)

ATTRIBUTE	VALUE
name	EndpointOverrideType

## XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EndpointOverrideType">
    <xs:attribute name="Name" use="required">
        <xs:annotation>
            <xs:documentation>The name of the endpoint.</xs:documentation>
        </xs:annotation>
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:minLength value="1"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="Port" type="xs:string">
        <xs:annotation>
            <xs:documentation>The port will be overridden in the Service Manifest</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="Protocol" type="xs:string" use="optional">
        <xs:annotation>
            <xs:documentation>The protocol for the endpoint. HTTPS endpoints must also have an EndpointCertificate and an EndpointBindingPolicy declared in the application manifest. The protocol cannot be changed later in an application upgrade. </xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="Type" type="xs:string" use="optional">
        <xs:annotation>
            <xs:documentation>The type of the endpoint. Input endpoints are used to expose the port to the outside, internal endpoints are used for intra-application communication.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="UriScheme" use="optional">
        <xs:annotation>
            <xs:documentation>The URI scheme. For example, "http", "https", or "ftp".</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="PathSuffix" use="optional">
        <xs:annotation>
            <xs:documentation>The path suffix. For example, "/myapp1".</xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:complexType>

```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
use	required

### Port

ATTRIBUTE	VALUE
name	Port

ATTRIBUTE	VALUE
type	xs:string

#### Protocol

ATTRIBUTE	VALUE
name	Protocol
type	xs:string
use	optional

#### Type

ATTRIBUTE	VALUE
name	Type
type	xs:string
use	optional

#### UriScheme

ATTRIBUTE	VALUE
name	UriScheme
use	optional

#### PathSuffix

ATTRIBUTE	VALUE
name	PathSuffix
use	optional

## EndpointType complexType

Defines an endpoint for the service. Specific ports can be requested. If a port is not explicitly specified, a port is assigned from the reserved application port range. Service replicas running on different cluster nodes can be assigned different port numbers, while replicas of the same service running on the same node share the same port. Such ports can be used by the service replicas for various purposes such as replication or listening for client requests.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 8 attribute(s)

ATTRIBUTE	VALUE
name	EndpointType

## XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EndpointType">
  <xs:annotation>
    <xs:documentation>Defines an endpoint for the service. Specific ports can be requested. If a port is not explicitly specified, a port is assigned from the reserved application port range. Service replicas running on different cluster nodes can be assigned different port numbers, while replicas of the same service running on the same node share the same port. Such ports can be used by the service replicas for various purposes such as replication or listening for client requests.</xs:documentation>
  </xs:annotation>
  <xs:attribute name="Name" use="required">
    <xs:annotation>
      <xs:documentation>The name of the endpoint.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="Protocol" use="optional" default="tcp">
    <xs:annotation>
      <xs:documentation>The protocol for the endpoint. HTTPS endpoints must also have an EndpointCertificate and an EndpointBindingPolicy declared in the application manifest. The protocol cannot be changed later in an application upgrade. </xs:documentation>
    </xs:annotation>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="http"/>
        <xs:enumeration value="https"/>
        <xs:enumeration value="tcp"/>
        <xs:enumeration value="udp"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="Type" use="optional" default="Internal">
    <xs:annotation>
      <xs:documentation>The type of the endpoint. Input endpoints are used to expose the port to the outside, internal endpoints are used for intra-application communication.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="Input"/>
        <xs:enumeration value="Internal"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="CodePackageRef" use="optional">
    <xs:annotation>
      <xs:documentation>The name of code Package that will use this endpoint.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="CertificateRef">
    <xs:annotation>
      <xs:documentation>Do not use, this attribute is not supported.</xs:documentation>
    </xs:annotation>
  </xs:attribute>

```

```

</xs:attribute>
<xs:attribute name="Port">
    <xs:annotation>
        <xs:documentation>The port will be replaced with a port determined by Microsoft Azure Service Fabric after registering with Http.sys or BFE.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
        <xs:restriction base="xs:int">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="65535"/>
        </xs:restriction>
    </xs:simpleType>
</xs:attribute>
<xs:attribute name="UriScheme">
    <xs:annotation>
        <xs:documentation>The URI scheme. For example, "http", "https", or "ftp".</xs:documentation>
    </xs:annotation>
</xs:attribute>
<xs:attribute name="PathSuffix">
    <xs:annotation>
        <xs:documentation>The path suffix. For example, "/myapp1".</xs:documentation>
    </xs:annotation>
</xs:attribute>
</xs:complexType>

```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
use	required

### Protocol

ATTRIBUTE	VALUE
name	Protocol
use	optional
default	tcp

### Type

ATTRIBUTE	VALUE
name	Type
use	optional
default	Internal

### CodePackageRef

ATTRIBUTE	VALUE
name	CodePackageRef

ATTRIBUTE	VALUE
use	optional

#### CertificateRef

ATTRIBUTE	VALUE
name	CertificateRef

#### Port

ATTRIBUTE	VALUE
name	Port

#### UriScheme

ATTRIBUTE	VALUE
name	UriScheme

#### PathSuffix

ATTRIBUTE	VALUE
name	PathSuffix

## EntryPointDescriptionType complexType

The executable specified by EntryPoint is typically the long-running service host. The presence of a separate setup entry point avoids having to run the service host with high privileges for extended periods of time. The executable specified by EntryPoint is run after SetupEntryPoint exits successfully. The resulting process is monitored and restarted (beginning again with SetupEntryPoint) if it ever terminates or crashes.

ATTRIBUTE	VALUE
type	anonymous complexType
content	3 element(s), 0 attribute(s)
name	EntryPointDescriptionType

#### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EntryPointDescriptionType">
  <xs:annotation>
    <xs:documentation>The executable specified by EntryPoint is typically the long-running service host. The presence of a separate setup entry point avoids having to run the service host with high privileges for extended periods of time. The executable specified by EntryPoint is run after SetupEntryPoint exits successfully. The resulting process is monitored and restarted (beginning again with SetupEntryPoint) if it ever terminates or crashes.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:choice>
      <xs:element name="ExeHost">
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="ExeHostEntryPointType">
              <xs:sequence>
                <xs:element name="RunFrequency" minOccurs="0">
                  <xs:complexType>
                    <xs:attribute name="IntervalInSeconds" use="required">
                      <xs:simpleType>
                        <xs:restriction base="xs:int">
                          <xs:minInclusive value="0"/>
                          <xs:maxInclusive value="2147483647"/>
                        </xs:restriction>
                      </xs:simpleType>
                    </xs:attribute>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:extension>
          </xs:complexContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="DllHost" type="DllHostEntryPointType"/>
      <xs:element name="ContainerHost" type="ContainerHostEntryPointType"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>

```

## Content element details

### ExeHost

ATTRIBUTE	VALUE
name	ExeHost

### DllHost

ATTRIBUTE	VALUE
name	DllHost
type	DllHostEntryPointType

### ContainerHost

ATTRIBUTE	VALUE
name	ContainerHost
type	ContainerHostEntryPointType

## EnvironmentOverridesType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 1 attribute(s)
name	EnvironmentOverridesType

### XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EnvironmentOverridesType">
  <xs:sequence>
    <xs:element name="EnvironmentVariable" type="EnvironmentVariableType" minOccurs="0"
maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Environment variable.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="CodePackageRef" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>
```

### Attribute details

#### CodePackageRef

ATTRIBUTE	VALUE
name	CodePackageRef
use	required

### Content element details

#### EnvironmentVariable

Environment variable.

ATTRIBUTE	VALUE
name	EnvironmentVariable
type	EnvironmentVariableType
minOccurs	0
maxOccurs	unbounded

## EnvironmentType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	3 element(s), 0 attribute(s)
name	EnvironmentType

## XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EnvironmentType">
<xs:sequence>
    <xs:element name="Principals" type="SecurityPrincipalsType"/>
    <xs:element name="Policies" type="ApplicationPoliciesType"/>
    <xs:element name="Diagnostics" type="DiagnosticsType"/>
</xs:sequence>
<xs:attributeGroup ref="VersionedItemAttrGroup"/>
</xs:complexType>
```

## Content element details

### Principals

ATTRIBUTE	VALUE
name	Principals
type	SecurityPrincipalsType

### Policies

ATTRIBUTE	VALUE
name	Policies
type	ApplicationPoliciesType

### Diagnostics

ATTRIBUTE	VALUE
name	Diagnostics
type	DiagnosticsType

## EnvironmentVariableType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	EnvironmentVariableType

## XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EnvironmentVariableType">
  <xs:attribute name="Name" type="xs:string" use="required">
    <xs:annotation>
      <xs:documentation>Name of environment variable.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="Value">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:minLength value="0"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>
```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

### Value

ATTRIBUTE	VALUE
name	Value

## EnvironmentVariablesType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	EnvironmentVariablesType

## XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="EnvironmentVariablesType">
  <xs:sequence>
    <xs:element name="EnvironmentVariable" type="EnvironmentVariableType" minOccurs="0"
maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Environment variable.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

## Content element details

### EnvironmentVariable

Environment variable.

ATTRIBUTE	VALUE
name	EnvironmentVariable
type	EnvironmentVariableType
minOccurs	0
maxOccurs	unbounded

## ExeHostEntryPointType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	4 element(s), 0 attribute(s)
name	ExeHostEntryPointType

### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ExeHostEntryPointType">

    <xs:sequence>
        <xs:element name="Program" type="xs:string">
            <xs:annotation>
                <xs:documentation>The executable name. For example, "MySetup.bat" or "MyServiceHost.exe".</xs:documentation>
            </xs:annotation></xs:element>
        <xs:element name="Arguments" type="xs:string" minOccurs="0"/>
        <xs:element name="WorkingFolder" default="Work" minOccurs="0">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="Work"/>
                    <xs:enumeration value="CodePackage"/>
                    <xs:enumeration value="CodeBase"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:element>
        <xs:element name="ConsoleRedirection" minOccurs="0">
            <xs:complexType>
                <xs:attribute name="FileRetentionCount" default="2">
                    <xs:simpleType>
                        <xs:restriction base="xs:int">
                            <xs:minInclusive value="1"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="FileMaxSizeInKb" default="20480">
                    <xs:simpleType>
                        <xs:restriction base="xs:int">
                            <xs:minInclusive value="128"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>

```

## Content element details

### Program

The executable name. For example, "MySetup.bat" or "MyServiceHost.exe".

ATTRIBUTE	VALUE
name	Program
type	xs:string

### Arguments

ATTRIBUTE	VALUE
name	Arguments
type	xs:string
minOccurs	0

### WorkingFolder

ATTRIBUTE	VALUE
name	WorkingFolder
default	Work
minOccurs	0

#### ConsoleRedirection

ATTRIBUTE	VALUE
name	ConsoleRedirection
minOccurs	0

## ExtensionsType complexType

Describes extensions that can be applied to other elements.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	ExtensionsType

#### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ExtensionsType">
  <xs:annotation>
    <xs:documentation>Describes extensions that can be applied to other elements.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="Extension" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:any namespace="##other" processContents="lax"/>
        </xs:sequence>
        <xs:attribute name="Name" use="required">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:minLength value="1"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="GeneratedId" type="xs:string" use="optional"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

#### Content element details

##### Extension

ATTRIBUTE	VALUE
name	Extension
minOccurs	0
maxOccurs	unbounded

## FabricCertificateType complexType

This specifies the certificate information.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 5 attribute(s)
name	FabricCertificateType

### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="FabricCertificateType">
    <xs:annotation>
        <xs:documentation>This specifies the certificate information.</xs:documentation>
    </xs:annotation>
    <xs:attribute name="X509StoreName" type="xs:string" default="My">
        <xs:annotation>
            <xs:documentation>The store name for the X509 certificate.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="X509FindType" default="FindByThumbprint">
        <xs:annotation>
            <xs:documentation>This is Used only when credential is X509. This specifies how
to find the certificate whether by the name or the thumbprint </xs:documentation>
        </xs:annotation>
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="FindByThumbprint"/>
                <xs:enumeration value="FindBySubjectName"/>
                <xs:enumeration value="FindByExtension"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="X509FindValue" use="required">
        <xs:annotation>
            <xs:documentation>This is Used only when credential is X509. This is the actual
name or thumbprint of the certificate.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="X509FindValueSecondary" use="optional" default="">
        <xs:annotation>
            <xs:documentation>This is used only when credential is X509. This is the actual
name or thumbprint of the certificate.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="Name" type="xs:string" use="optional"/>
</xs:complexType>
```

## Attribute details

### X509StoreName

ATTRIBUTE	VALUE
name	X509StoreName
type	xs:string
default	My

### X509FindType

ATTRIBUTE	VALUE
name	X509FindType
default	FindByThumbprint

### X509FindValue

ATTRIBUTE	VALUE
name	X509FindValue
use	required

### X509FindValueSecondary

ATTRIBUTE	VALUE
name	X509FindValueSecondary
use	optional
default	

### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	optional

## FabricEndpointsType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	18 element(s), 0 attribute(s)

ATTRIBUTE	VALUE
name	FabricEndpointsType

## XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="FabricEndpointsType">
  <xs:all>
    <xs:element name="ClientConnectionEndpoint" type="InputEndpointType"/>
    <xs:element name="LeaseDriverEndpoint" type="InternalEndpointType"/>
    <xs:element name="ClusterConnectionEndpoint" type="InternalEndpointType"/>
    <xs:element name="HttpGatewayEndpoint" type="InputEndpointType" minOccurs="0"/>
    <xs:element name="HttpApplicationGatewayEndpoint" type="InputEndpointType" minOccurs="0"/>
    <xs:element name="ServiceConnectionEndpoint" type="InternalEndpointType" minOccurs="0"/>
    <xs:element name="ClusterManagerReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
    <xs:element name="RepairManagerReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
    <xs:element name="NamingReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
    <xs:element name="FailoverManagerReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
    <xs:element name="ImageStoreServiceReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
    <xs:element name="UpgradeServiceReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
    <xs:element name="FaultAnalysisServiceReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
    <xs:element name="BackupRestoreServiceReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
    <xs:element name="UpgradeOrchestrationServiceReplicatorEndpoint" type="InternalEndpointType"
minOccurs="0"/>
    <xs:element name="DefaultReplicatorEndpoint" type="InternalEndpointType" minOccurs="0"/>
    <xs:element name="ApplicationEndpoints" minOccurs="0">
      <xs:complexType>
        <xs:attribute name="StartPort" type="xs:int" use="required"/>
        <xs:attribute name="EndPort" type="xs:int" use="required"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="EphemeralEndpoints" minOccurs="0">
      <xs:complexType>
        <xs:attribute name="StartPort" type="xs:int" use="required"/>
        <xs:attribute name="EndPort" type="xs:int" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:all>
</xs:complexType>

```

## Content element details

### ClientConnectionEndpoint

ATTRIBUTE	VALUE
name	ClientConnectionEndpoint
type	InputEndpointType

### LeaseDriverEndpoint

ATTRIBUTE	VALUE
name	LeaseDriverEndpoint
type	InternalEndpointType

### ClusterConnectionEndpoint

ATTRIBUTE	VALUE
name	ClusterConnectionEndpoint
type	InternalEndpointType

#### **HttpGatewayEndpoint**

ATTRIBUTE	VALUE
name	HttpGatewayEndpoint
type	InputEndpointType
minOccurs	0

#### **HttpApplicationGatewayEndpoint**

ATTRIBUTE	VALUE
name	HttpApplicationGatewayEndpoint
type	InputEndpointType
minOccurs	0

#### **ServiceConnectionEndpoint**

ATTRIBUTE	VALUE
name	ServiceConnectionEndpoint
type	InternalEndpointType
minOccurs	0

#### **ClusterManagerReplicatorEndpoint**

ATTRIBUTE	VALUE
name	ClusterManagerReplicatorEndpoint
type	InternalEndpointType
minOccurs	0

#### **RepairManagerReplicatorEndpoint**

ATTRIBUTE	VALUE
name	RepairManagerReplicatorEndpoint
type	InternalEndpointType
minOccurs	0

**NamingReplicatorEndpoint**

ATTRIBUTE	VALUE
name	NamingReplicatorEndpoint
type	InternalEndpointType
minOccurs	0

**FailoverManagerReplicatorEndpoint**

ATTRIBUTE	VALUE
name	FailoverManagerReplicatorEndpoint
type	InternalEndpointType
minOccurs	0

**ImageStoreServiceReplicatorEndpoint**

ATTRIBUTE	VALUE
name	ImageStoreServiceReplicatorEndpoint
type	InternalEndpointType
minOccurs	0

**UpgradeServiceReplicatorEndpoint**

ATTRIBUTE	VALUE
name	UpgradeServiceReplicatorEndpoint
type	InternalEndpointType
minOccurs	0

**FaultAnalysisServiceReplicatorEndpoint**

ATTRIBUTE	VALUE
name	FaultAnalysisServiceReplicatorEndpoint
type	InternalEndpointType
minOccurs	0

**BackupRestoreServiceReplicatorEndpoint**

ATTRIBUTE	VALUE
name	BackupRestoreServiceReplicatorEndpoint

ATTRIBUTE	VALUE
type	InternalEndpointType
minOccurs	0

#### UpgradeOrchestrationServiceReplicatorEndpoint

ATTRIBUTE	VALUE
name	UpgradeOrchestrationServiceReplicatorEndpoint
type	InternalEndpointType
minOccurs	0

#### DefaultReplicatorEndpoint

ATTRIBUTE	VALUE
name	DefaultReplicatorEndpoint
type	InternalEndpointType
minOccurs	0

#### ApplicationEndpoints

ATTRIBUTE	VALUE
name	ApplicationEndpoints
minOccurs	0

#### EphemeralEndpoints

ATTRIBUTE	VALUE
name	EphemeralEndpoints
minOccurs	0

## FabricKtlLoggerSettingsType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	3 element(s), 0 attribute(s)
name	FabricKtlLoggerSettingsType

#### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="FabricKtlLoggerSettingsType">
  <xs:all>
    <xs:element name="SharedLogFilePath" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Defines path to shared log.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:attribute name="Value" type="xs:string" use="required"/>
      </xs:complexType>
    </xs:element>

    <xs:element name="SharedLogFileDialog" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Specific GUID to use as the shared log id.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:attribute name="Value" use="required">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:pattern value="[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{12}"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:complexType>
    </xs:element>

    <xs:element name="SharedLogFileSizeInMB" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Defines how large is the shared log.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:attribute name="Value" use="required">
          <xs:simpleType>
            <xs:restriction base="xs:int">
              <xs:minInclusive value="512"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:complexType>
    </xs:element>

  </xs:all>
</xs:complexType>

```

## Content element details

### **SharedLogFilepath**

Defines path to shared log.

ATTRIBUTE	VALUE
name	SharedLogFilepath
minOccurs	0

### **SharedLogFileDialog**

Specific GUID to use as the shared log id.

ATTRIBUTE	VALUE
name	SharedLogFileDialog

ATTRIBUTE	VALUE
minOccurs	0

#### SharedLogFileSizeInMB

Defines how large is the shared log.

ATTRIBUTE	VALUE
name	SharedLogFileSizeInMB
minOccurs	0

## FabricNodeType complexType

Describes a Microsoft Azure Service Fabric Node.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 6 attribute(s)
name	FabricNodeType

#### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="FabricNodeType">
    <xs:annotation>
        <xs:documentation>Describes a Microsoft Azure Service Fabric Node.</xs:documentation>
    </xs:annotation>
    <xs:attribute name="NodeName" type="xs:string" use="required">
        <xs:annotation>
            <xs:documentation>The name of the node instance.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="IPAddressOrFQDN" type="xs:string" use="required">
        <xs:annotation>
            <xs:documentation>The IP address or the FQDN of the machine on which to place
this node.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="IsSeedNode" type="xs:boolean" default="false">
        <xs:annotation>
            <xs:documentation>A flag indicating whether or not this node is a seed node.
</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="NodeTypeRef" type="xs:string" use="required">
        <xs:annotation>
            <xs:documentation>Name of the nodetype defined in the NodeTypes section.
</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="FaultDomain" type="xs:anyURI" use="optional">
        <xs:annotation>
            <xs:documentation>
The fault domain of this node.
</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="UpgradeDomain" type="xs:anyURI" use="optional">
        <xs:annotation>
            <xs:documentation>
The upgrade domain of this node.
</xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:complexType>

```

## Attribute details

### NodeName

Defines how large is the shared log.

ATTRIBUTE	VALUE
name	NodeName
type	xs:string
use	required

### IPAddressOrFQDN

Defines how large is the shared log.

ATTRIBUTE	VALUE
name	IPAddressOrFQDN
type	xs:string
use	required

#### **IsSeedNode**

Defines how large is the shared log.

ATTRIBUTE	VALUE
name	IsSeedNode
type	xs:boolean
default	false

#### **NodeTypeRef**

Defines how large is the shared log.

ATTRIBUTE	VALUE
name	NodeTypeRef
type	xs:string
use	required

#### **FaultDomain**

Defines how large is the shared log.

ATTRIBUTE	VALUE
name	FaultDomain
type	xs:anyURI
use	optional

#### **UpgradeDomain**

Defines how large is the shared log.

ATTRIBUTE	VALUE
name	UpgradeDomain
type	xs:anyURI
use	optional

## FileStoreETWType complexType

Describes a file store destination for ETW events. Works only in on-premise environment.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	FileStoreETWType

#### XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="FileStoreETWType">
<xs:annotation>
<xs:documentation>Describes a file store destination for ETW events. Works only in on-premise
environment.</xs:documentation>
</xs:annotation>
<xs:complexContent>
<xs:extension base="FileStoreType">
<xs:attributeGroup ref="LevelFilter"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

## FileStoreType complexType

Describes a file store destination for diagnostics data. Works only in a standalone cluster environment.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 2 attribute(s)
name	FileStoreType

#### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="FileStoreType">
  <xss:annotation>
    <xss:documentation>Describes a file store destination for diagnostics data. Works only in a standalone
cluster environment.</xss:documentation>
  </xss:annotation>
  <xss:sequence>
    <xss:element ref="Parameters" minOccurs="0"/>
  </xss:sequence>
  <xss:attributeGroup ref="IsEnabled"/>
  <xss:attributeGroup ref="Path"/>
  <xss:attributeGroup ref="UploadIntervalInMinutes"/>
  <xss:attributeGroup ref="DataDeletionAgeInDays"/>
  <xss:attribute name="AccountType" type="xs:string">
    <xss:annotation>
      <xss:documentation>Specifies the type of account.</xss:documentation>
    </xss:annotation>
  </xss:attribute>
  <xss:attributeGroup ref="AccountCredentialsGroup"/>
  <xss:attribute name="PasswordEncrypted" type="xs:string">
    <xss:annotation>
      <xss:documentation>Specifies if password is encrypted or plain text.</xss:documentation>
    </xss:annotation>
  </xss:attribute>
</xss:complexType>

```

## Attribute details

### AccountType

Defines how large is the shared log.

ATTRIBUTE	VALUE
name	AccountType
type	xs:string

### PasswordEncrypted

Defines how large is the shared log.

ATTRIBUTE	VALUE
name	PasswordEncrypted
type	xs:string

## Content element details

### None

ATTRIBUTE	VALUE
ref	Parameters
minOccurs	0

## InfrastructureInformationType complexType

Contains the infrastructure information for this Microsoft Azure Service Fabric cluster.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	InfrastructureInformationType

## XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="InfrastructureInformationType">
  <xs:annotation>
    <xs:documentation>Contains the infrastructure information for this Microsoft Azure Service Fabric
cluster.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="NodeList">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Node" type="InfrastructureNodeType" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

## Content element details

### NodeList

ATTRIBUTE	VALUE
name	NodeList

## InfrastructureNodeType complexType

Describes a Infrastructure information needed.

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 7 attribute(s)
name	InfrastructureNodeType

## XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="InfrastructureNodeType">
    <xss:annotation>
        <xss:documentation>Describes a Infrastructure information needed.</xss:documentation>
    </xss:annotation>
    <xss:sequence>
        <xss:element name="Endpoints" type="FabricEndpointsType" minOccurs="0">
            <xss:annotation>
                <xss:documentation>Describe the endpoints associated with this node type</xss:documentation>
            </xss:annotation>
        </xss:element>
        <xss:element name="Certificates" type="CertificatesType" minOccurs="0">
            <xss:annotation>
                <xss:documentation>Describe the certificates associated with this node type</xss:documentation>
            </xss:annotation>
        </xss:element>
    </xss:sequence>
    <xss:attribute name="NodeName" type="xs:string" use="required">
        <xss:annotation>
            <xss:documentation>The name of the node instance.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="IPAddressOrFQDN" type="xs:string" use="required">
        <xss:annotation>
            <xss:documentation>The IP address or the FQDN of the machine on which to place this node.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="RoleOrTierName" type="xs:string" use="required">
        <xss:annotation>
            <xss:documentation>Name of the role which links to node type ref which is defined in the NodeTypes section.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="NodeTypeRef" type="xs:string" use="required">
        <xss:annotation>
            <xss:documentation>Name of the node type which is defined in the NodeTypes section.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="IsSeedNode" type="xs:boolean" use="optional" default="false">
        <xss:annotation>
            <xss:documentation>Indicates whether the node is a seed node.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="FaultDomain" type="xs:anyURI" use="optional">
        <xss:annotation>
            <xss:documentation> The fault domain of this node. </xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="UpgradeDomain" type="xs:anyURI" use="optional">
        <xss:annotation>
            <xss:documentation>The upgrade domain of this node. </xss:documentation>
        </xss:annotation>
    </xss:attribute>
</xss:complexType>

```

## Attribute details

### NodeName

ATTRIBUTE	VALUE
name	NodeName
type	xs:string

ATTRIBUTE	VALUE
use	required

#### IPAddressOrFQDN

ATTRIBUTE	VALUE
name	IPAddressOrFQDN
type	xs:string
use	required

#### RoleOrTierName

ATTRIBUTE	VALUE
name	RoleOrTierName
type	xs:string
use	required

#### NodeTypeRef

ATTRIBUTE	VALUE
name	NodeTypeRef
type	xs:string
use	required

#### IsSeedNode

ATTRIBUTE	VALUE
name	IsSeedNode
type	xs:boolean
use	optional
default	false

#### FaultDomain

ATTRIBUTE	VALUE
name	FaultDomain
type	xs:anyURI
use	optional

## UpgradeDomain

ATTRIBUTE	VALUE
name	UpgradeDomain
type	xs:anyURI
use	optional

## Content element details

### Endpoints

Describe the endpoints associated with this node type

ATTRIBUTE	VALUE
name	Endpoints
type	FabricEndpointsType
minOccurs	0

### Certificates

Describe the certificates associated with this node type

ATTRIBUTE	VALUE
name	Certificates
type	CertificatesType
minOccurs	0

## InputEndpointType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	InputEndpointType

### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="InputEndpointType">
  <xs:attribute name="Port" type="xs:positiveInteger" use="required"/>
  <xs:attribute name="Protocol" use="optional" default="tcp">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="http"/>
        <xs:enumeration value="https"/>
        <xs:enumeration value="tcp"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>

```

## Attribute details

### Port

Describe the certificates associated with this node type

ATTRIBUTE	VALUE
name	Port
type	xs:positiveInteger
use	required

### Protocol

Describe the certificates associated with this node type

ATTRIBUTE	VALUE
name	Protocol
use	optional
default	tcp

## InternalEndpointType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	InternalEndpointType

### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="InternalEndpointType">
  <xss:attribute name="Port" type="xs:positiveInteger" use="required"/>
  <xss:attribute name="Protocol" use="optional" default="tcp">
    <xss:simpleType>
      <xss:restriction base="xs:string">
        <xss:enumeration value="http"/>
        <xss:enumeration value="https"/>
        <xss:enumeration value="tcp"/>
      </xss:restriction>
    </xss:simpleType>
  </xss:attribute>
</xss:complexType>

```

## Attribute details

### Port

Describe the certificates associated with this node type

ATTRIBUTE	VALUE
name	Port
type	xs:positiveInteger
use	required

### Protocol

Describe the certificates associated with this node type

ATTRIBUTE	VALUE
name	Protocol
use	optional
default	tcp

## KeyValuePairType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	KeyValuePairType

### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="KeyValuePairType">
  <xss:attributeGroup ref="NameValuePair"/>
</xss:complexType>

```

## LinuxInfrastructureType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	LinuxInfrastructureType

### XML source

```
<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LinuxInfrastructureType">
  <xss:sequence>
    <xss:element name="NodeList">
      <xss:complexType>
        <xss:sequence>
          <xss:element name="Node" type="FabricNodeType" maxOccurs="unbounded"/>
        </xss:sequence>
      </xss:complexType>
    </xss:element>
  </xss:sequence>
</xss:complexType>
```

### Content element details

#### NodeList

ATTRIBUTE	VALUE
name	NodeList

## LoadMetricType complexType

A resource that this service should be balanced on, such as memory or CPU usage. Includes information about how much of that resource each replica or instance of this service consumes by default.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 5 attribute(s)
name	LoadMetricType

### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LoadMetricType">
    <xss:annotation>
        <xss:documentation>A resource that this service should be balanced on, such as memory or CPU usage.
Includes information about how much of that resource each replica or instance of this service consumes by
default.</xss:documentation>
    </xss:annotation>
    <xss:attribute name="Name" use="required">
        <xss:annotation>
            <xss:documentation>A unique identifier for the metric within the cluster from the Cluster Resource
Manager's perspective.</xss:documentation>
        </xss:annotation>
        <xss:simpleType>
            <xss:restriction base="xs:string">
                <xss:minLength value="1"/>
            </xss:restriction>
        </xss:simpleType>
    </xss:attribute>
    <xss:attribute name="DefaultLoad" type="xs:long" use="optional" default="0">
        <xss:annotation>
            <xss:documentation>The default amount of load that this stateless service creates for this metric.
</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="PrimaryDefaultLoad" type="xs:long" use="optional" default="0">
        <xss:annotation>
            <xss:documentation>The default amount of load that this service will exert for this metric when it's a
primary replica.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="SecondaryDefaultLoad" type="xs:long" use="optional" default="0">
        <xss:annotation>
            <xss:documentation>The default amount of load that this service will exert for this metric when it's a
secondary replica.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="Weight">
        <xss:annotation>
            <xss:documentation>Determines the metric weight relative to the other metrics that are configured for
this service. During runtime, if two metrics end up in conflict, the Cluster Resource Manager prefers the
metric with the higher weight. Zero disables load balancing for this metric.</xss:documentation>
        </xss:annotation>
        <xss:simpleType>
            <xss:restriction base="xs:string">
                <xss:enumeration value="Zero"/>
                <xss:enumeration value="Low"/>
                <xss:enumeration value="Medium"/>
                <xss:enumeration value="High"/>
            </xss:restriction>
        </xss:simpleType>
    </xss:attribute>
</xss:complexType>

```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
use	required

### DefaultLoad

ATTRIBUTE	VALUE
name	DefaultLoad
type	xs:long
use	optional
default	0

#### PrimaryDefaultLoad

ATTRIBUTE	VALUE
name	PrimaryDefaultLoad
type	xs:long
use	optional
default	0

#### SecondaryDefaultLoad

ATTRIBUTE	VALUE
name	SecondaryDefaultLoad
type	xs:long
use	optional
default	0

#### Weight

ATTRIBUTE	VALUE
name	Weight

## LocalStoreETWType complexType

Describes a store destination within the node for ETW events.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	LocalStoreETWType

#### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LocalStoreETWType">
  <xss:annotation>
    <xss:documentation>Describes a store destination within the node for ETW events.</xss:documentation>
  </xss:annotation>
  <xss:complexContent>
    <xss:extension base="LocalStoreType">
      <xss:attributeGroup ref="LevelFilter"/>
    </xss:extension>
  </xss:complexContent>
</xss:complexType>

```

## LocalStoreType complexType

Describes a store destination within the node for diagnostic data.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	LocalStoreType

### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LocalStoreType">
  <xss:annotation>
    <xss:documentation>Describes a store destination within the node for diagnostic data.</xss:documentation>
  </xss:annotation>
  <xss:sequence>
    <xss:element ref="Parameters" minOccurs="0"/>
  </xss:sequence>
  <xss:attributeGroup ref=".IsEnabled"/>
  <xss:attributeGroup ref="RelativeFolderPath"/>
  <xss:attributeGroup ref="DataDeletionAgeInDays"/>
</xss:complexType>

```

### Content element details

None

ATTRIBUTE	VALUE
ref	Parameters
minOccurs	0

## LogicalDirectoryType complexType

Describes a LogicalDirectoryType.

ATTRIBUTE	VALUE
type	anonymous complexType

ATTRIBUTE	VALUE
content	0 element(s), 3 attribute(s)
name	LogicalDirectoryType

## XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LogicalDirectoryType">
  <xs:annotation>
    <xs:documentation>Describes a LogicalDirectoryType.</xs:documentation>
  </xs:annotation>
  <xs:attribute name="LogicalDirectoryName" type="xs:string" use="required">
    <xs:annotation>
      <xs:documentation>The name of the LogicalDirectory.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="MappedTo" type="xs:string" use="required">
    <xs:annotation>
      <xs:documentation>The path of the LogicalDirectory.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="Context" use="optional" default="application">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="application"/>
        <xs:enumeration value="node"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>

```

## Attribute details

### LogicalDirectoryName

ATTRIBUTE	VALUE
name	LogicalDirectoryName
type	xs:string
use	required

### MappedTo

ATTRIBUTE	VALUE
name	MappedTo
type	xs:string
use	required

### Context

ATTRIBUTE	VALUE
name	Context
use	optional
default	application

## ManagedAssemblyType complexType

Unsupported, do not use. The name of managed assembly (for example, Queue.dll), to host.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	ManagedAssemblyType

### XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ManagedAssemblyType">
  <xs:annotation>
    <xs:documentation>Unsupported, do not use. The name of managed assembly (for example, Queue.dll), to
host.</xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="xs:string"/>
  </xs:simpleContent>
</xs:complexType>
```

## PaaSRoleType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 3 attribute(s)
name	PaaSRoleType

### XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="PaaSRoleType">
  <xs:attribute name="RoleName" type="xs:string" use="required"/>
  <xs:attribute name="NodeTypeRef" type="xs:string" use="required"/>
  <xs:attribute name="RoleNodeCount" type="xs:int" use="required"/>
</xs:complexType>
```

### Attribute details

#### RoleName

ATTRIBUTE	VALUE
name	RoleName
type	xs:string
use	required

#### NodeTypeRef

ATTRIBUTE	VALUE
name	NodeTypeRef
type	xs:string
use	required

#### RoleNodeCount

ATTRIBUTE	VALUE
name	RoleNodeCount
type	xs:int
use	required

## PaaSVoteType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 3 attribute(s)
name	PaaSVoteType

#### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="PaaSVoteType">
    <xs:attribute name="NodeName" use="required"/>
    <xs:attribute name="IPAddressOrFQDN" use="required"/>
    <xs:attribute name="Port" type="xs:int" use="required"/>
</xs:complexType>

```

#### Attribute details

##### NodeName

ATTRIBUTE	VALUE
name	NodeName

ATTRIBUTE	VALUE
use	required

#### IPAddressOrFQDN

ATTRIBUTE	VALUE
name	IPAddressOrFQDN
use	required

#### Port

ATTRIBUTE	VALUE
name	Port
type	xs:int
use	required

## PackageSharingPolicyType complexType

Indicates if a code, config or data package should be shared.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	PackageSharingPolicyType

#### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="PackageSharingPolicyType">
  <xss:annotation>
    <xss:documentation>Indicates if a code, config or data package should be shared.</xss:documentation>
  </xss:annotation>
  <xss:attribute name="PackageRef">
    <xss:annotation>
      <xss:documentation>The name of the code, config, or data package to be shared. Must match the name of
the package defined in the service manifest.</xss:documentation>
    </xss:annotation>
    <xss:simpleType>
      <xss:restriction base="xs:string">
        <xss:minLength value="1"/>
      </xss:restriction>
    </xss:simpleType>
  </xss:attribute>
  <xss:attribute name="Scope" default="None">
    <xss:simpleType>
      <xss:restriction base="xs:string">
        <xss:enumeration value="None"/>
        <xss:enumeration value="All"/>
        <xss:enumeration value="Code"/>
        <xss:enumeration value="Config"/>
        <xss:enumeration value="Data"/>
      </xss:restriction>
    </xss:simpleType>
  </xss:attribute>
</xss:complexType>

```

## Attribute details

### PackageRef

ATTRIBUTE	VALUE
name	PackageRef

### Scope

ATTRIBUTE	VALUE
name	Scope
default	None

## ParameterType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)
name	ParameterType

### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ParameterType">
    <xss:attributeGroup ref="NameValuePair"/>
    <xss:attribute name="IsEncrypted" type="xs:string">
        <xss:annotation>
            <xss:documentation>If true, the value of this parameter is
encrypted</xss:documentation>
        </xss:annotation>
    </xss:attribute>
</xss:complexType>

```

## Attribute details

### IsEncrypted

ATTRIBUTE	VALUE
name	IsEncrypted
type	xs:string

## ParametersType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	ParametersType

## XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ParametersType">
    <xss:sequence>
        <xss:element name="Parameter" type="ParameterType" minOccurs="1" maxOccurs="unbounded"/>
    </xss:sequence>
</xss:complexType>

```

## Content element details

### Parameter

ATTRIBUTE	VALUE
name	Parameter
type	ParameterType
minOccurs	1
maxOccurs	unbounded

## PortBindingType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	PortBindingType

## XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="PortBindingType">
  <xs:attribute name="ContainerPort" type="xs:int" use="required">
    <xs:annotation>
      <xs:documentation>Container port number.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="EndpointRef">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>
```

## Attribute details

### ContainerPort

ATTRIBUTE	VALUE
name	ContainerPort
type	xs:int
use	required

### EndpointRef

ATTRIBUTE	VALUE
name	EndpointRef

## RepositoryCredentialsType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	RepositoryCredentialsType

## XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="RepositoryCredentialsType">
  <xss:attributeGroup ref="AccountCredentialsGroup"/>
  <xss:attribute name="PasswordEncrypted" type="xs:boolean" use="optional">
    <xss:annotation>
      <xss:documentation>Specifies if password is encrypted or plain text.</xss:documentation>
    </xss:annotation>
  </xss:attribute>
  <xss:attribute name="Email">
    <xss:simpleType>
      <xss:restriction base="xs:string">
        <xss:minLength value="1"/>
      </xss:restriction>
    </xss:simpleType>
  </xss:attribute>
</xss:complexType>

```

## Attribute details

### PasswordEncrypted

ATTRIBUTE	VALUE
name	PasswordEncrypted
type	xs:boolean
use	optional

### Email

ATTRIBUTE	VALUE
name	Email

## ResourceGovernancePolicyType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 9 attribute(s)
name	ResourceGovernancePolicyType

## XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ResourceGovernancePolicyType">
  <xss:attribute name="CodePackageRef" use="required">
    <xss:simpleType>
      <xss:restriction base="xs:string">
        <xss:minLength value="1"/>
      </xss:restriction>
    </xss:simpleType>
  </xss:attribute>
  <xss:attribute name="MemoryInMB" use="optional" default="0">
    <xss:annotation>
      <xss:documentation>Memory limits in MB. Must be a positive integer.</xss:documentation>
    </xss:annotation>
  </xss:attribute>
</xss:complexType>

```

```

</xs:annotation>
<xs:simpleType>
  <xs:restriction base="xs:nonNegativeInteger"/>
</xs:simpleType>
</xs:attribute>
<xs:attribute name="MemorySwapInMB" use="optional" default="0">
  <xs:annotation>
    <xs:documentation>Total memory (memory + swap) limits in MB.</xs:documentation>
  </xs:annotation>
<xs:simpleType>
  <xs:restriction base="xs:nonNegativeInteger"/>
</xs:simpleType>
</xs:attribute>
<xs:attribute name="MemoryReservationInMB" use="optional" default="0">
  <xs:annotation>
    <xs:documentation>Memory soft limits in MB. Must be a positive integer.</xs:documentation>
  </xs:annotation>
<xs:simpleType>
  <xs:restriction base="xs:nonNegativeInteger"/>
</xs:simpleType>
</xs:attribute>
<xs:attribute name="CpuShares" use="optional" default="0">
  <xs:annotation>
    <xs:documentation>Relative CPU weight. Must be a positive integer.</xs:documentation>
  </xs:annotation>
<xs:simpleType>
  <xs:restriction base="xs:nonNegativeInteger"/>
</xs:simpleType>
</xs:attribute>
<xs:attribute name="CpuPercent" use="optional" default="0">
  <xs:annotation>
    <xs:documentation>Usable percentage of available CPUs (windows only). Must be a positive integer.
</xs:documentation>
  </xs:annotation>
<xs:simpleType>
  <xs:restriction base="xs:nonNegativeInteger"/>
</xs:simpleType>
</xs:attribute>
<xs:attribute name="MaximumIOps" use="optional" default="0">
  <xs:annotation>
    <xs:documentation>Maximum IO rate in terms of IOps. Must be a positive integer.</xs:documentation>
  </xs:annotation>
<xs:simpleType>
  <xs:restriction base="xs:nonNegativeInteger"/>
</xs:simpleType>
</xs:attribute>
<xs:attribute name="MaximumIOBandwidth" use="optional" default="0">
  <xs:annotation>
    <xs:documentation>Maximum IO bandwidth. Must be a positive integer.</xs:documentation>
  </xs:annotation>
<xs:simpleType>
  <xs:restriction base="xs:nonNegativeInteger"/>
</xs:simpleType>
</xs:attribute>
<xs:attribute name="BlockIOWeight" use="optional" default="0">
  <xs:annotation>
    <xs:documentation>Relative block IO weight. Must be a positive integer between 10 and 1000.
</xs:documentation>
  </xs:annotation>
<xs:simpleType>
  <xs:restriction base="xs:nonNegativeInteger"/>
</xs:simpleType>
</xs:attribute>
</xs:complexType>

```

## Attribute details

[CodePackageRef](#)

ATTRIBUTE	VALUE
name	CodePackageRef
use	required

#### MemoryInMB

ATTRIBUTE	VALUE
name	MemoryInMB
use	optional
default	0

#### MemorySwapInMB

ATTRIBUTE	VALUE
name	MemorySwapInMB
use	optional
default	0

#### MemoryReservationInMB

ATTRIBUTE	VALUE
name	MemoryReservationInMB
use	optional
default	0

#### CpuShares

ATTRIBUTE	VALUE
name	CpuShares
use	optional
default	0

#### CpuPercent

ATTRIBUTE	VALUE
name	CpuPercent
use	optional
default	0

**MaximumIOps**

ATTRIBUTE	VALUE
name	MaximumIOps
use	optional
default	0

**MaximumIOBandwidth**

ATTRIBUTE	VALUE
name	MaximumIOBandwidth
use	optional
default	0

**BlockIOWeight**

ATTRIBUTE	VALUE
name	BlockIOWeight
use	optional
default	0

## ResourceOverridesType complexType

Describes the resource overrides for endpoints in servicemanifest resources.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	ResourceOverridesType

**XML source**

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ResourceOverridesType">
  <xss:annotation>
    <xss:documentation>Describes the resource overrides for endpoints in servicemanifest resources.
  </xss:documentation>
  <xss:sequence>
    <xss:element name="Endpoints" minOccurs="0">
      <xss:annotation>
        <xss:documentation>Defines endpoints for the service.</xss:documentation>
      </xss:annotation>
      <xss:complexType>
        <xss:sequence>
          <xss:element name="Endpoint" type="EndpointOverrideType" maxOccurs="unbounded"/>
        </xss:sequence>
      </xss:complexType>
    </xss:element>
  </xss:sequence>
</xss:complexType>

```

## Content element details

### Endpoints

Defines endpoints for the service.

ATTRIBUTE	VALUE
name	Endpoints
minOccurs	0

## ResourcesType complexType

Describes the resources used by this service, which can be declared without modifying compiled code and changed when the service is deployed. Access to these resources is controlled through the Principals and Policies sections of the application manifest.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	ResourcesType

### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ResourcesType">
  <xss:annotation>
    <xss:documentation>Describes the resources used by this service, which can be declared without modifying
    compiled code and changed when the service is deployed. Access to these resources is controlled through the
    Principals and Policies sections of the application manifest.</xss:documentation>
  </xss:annotation>
  <xss:sequence>
    <xss:element name="Endpoints" minOccurs="0">
      <xss:annotation>
        <xss:documentation>Defines endpoints for the service.</xss:documentation>
      </xss:annotation>
      <xss:complexType>
        <xss:sequence>
          <xss:element name="Endpoint" type="EndpointType" maxOccurs="unbounded"/>
        </xss:sequence>
      </xss:complexType>
    </xss:element>
  </xss:sequence>
</xss:complexType>

```

## Content element details

### Endpoints

Defines endpoints for the service.

ATTRIBUTE	VALUE
name	Endpoints
minOccurs	0

## RunAsPolicyType complexType

Specifies the local user or local system account that a service code package will run as. Domain accounts are supported on Windows Server deployments where Azure Active Directory is available. By default, applications run under the account that the Fabric.exe process runs under. Applications can also run as other accounts, which must be declared in the Principals section. If you apply a RunAs policy to a service, and the service manifest declares endpoint resources with the HTTP protocol, you must also specify a SecurityAccessPolicy to ensure that ports allocated to these endpoints are correctly access-control listed for the RunAs user account that the service runs under. For an HTTPS endpoint, you also have define a EndpointBindingPolicy to indicate the name of the certificate to return to the client.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 3 attribute(s)
name	RunAsPolicyType

### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="RunAsPolicyType">
    <xss:annotation>
        <xss:documentation>Specifies the local user or local system account that a service code package will run
as. Domain accounts are supported on Windows Server deployments where Azure Active Directory is available. By
default, applications run under the account that the Fabric.exe process runs under. Applications can also run
as other accounts, which must be declared in the Principals section. If you apply a RunAs policy to a service,
and the service manifest declares endpoint resources with the HTTP protocol, you must also specify a
SecurityAccessPolicy to ensure that ports allocated to these endpoints are correctly access-control listed for
the RunAs user account that the service runs under. For an HTTPS endpoint, you also have define a
EndpointBindingPolicy to indicate the name of the certificate to return to the client.</xss:documentation>
    </xss:annotation>
    <xss:attribute name="CodePackageRef" use="required">
        <xss:annotation>
            <xss:documentation>The name of the code package. Must match the name of the CodePackage specified in the
service manifest.</xss:documentation>
        </xss:annotation>
        <xss:simpleType>
            <xss:restriction base="xs:string">
                <xss:minLength value="1"/>
            </xss:restriction>
        </xss:simpleType>
    </xss:attribute>
    <xss:attribute name="UserRef" use="required">
        <xss:annotation>
            <xss:documentation>The user account that the service code package will run as. The user account must be
declared in the Principals section. Often it is preferable to run the setup entry point using a local system
account rather than an administrators account.</xss:documentation>
        </xss:annotation>
        <xss:simpleType>
            <xss:restriction base="xs:string">
                <xss:minLength value="1"/>
            </xss:restriction>
        </xss:simpleType>
    </xss:attribute>
    <xss:attribute name="EntryPointType" use="optional" default="Main">
        <xss:annotation>
            <xss:documentation>Setup is the SetupEntryPoint declared in the service manifest, the privileged entry
point that runs before any other entry point. Main is the EntryPoint declared in the service manifest,
typically the long-running service host. All is all entry points.</xss:documentation>
        </xss:annotation>
        <xss:simpleType>
            <xss:restriction base="xs:string">
                <xss:enumeration value="Setup"/>
                <xss:enumeration value="Main"/>
                <xss:enumeration value="All"/>
            </xss:restriction>
        </xss:simpleType>
    </xss:attribute>
</xss:complexType>

```

## Attribute details

### CodePackageRef

Defines endpoints for the service.

ATTRIBUTE	VALUE
name	CodePackageRef
use	required

### UserRef

Defines endpoints for the service.

ATTRIBUTE	VALUE
name	UserRef
use	required

#### EntryPointType

Defines endpoints for the service.

ATTRIBUTE	VALUE
name	EntryPointType
use	optional
default	Main

## SecurityAccessPolicyType complexType

Grants access permissions to a principal on a resource (such as an endpoint) defined in a service manifest. Typically, it is very useful to control and restrict access of services to different resources in order to minimize security risks. This is especially important when the application is built from a collection of services from a marketplace which are developed by different developers.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 4 attribute(s)
name	SecurityAccessPolicyType

#### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="SecurityAccessPolicyType">
  <xss:annotation>
    <xss:documentation>Grants access permissions to a principal on a resource (such as an endpoint) defined in a service manifest. Typically, it is very useful to control and restrict access of services to different resources in order to minimize security risks. This is especially important when the application is built from a collection of services from a marketplace which are developed by different developers.</xss:documentation>
  </xss:annotation>
  <xss:attribute name="ResourceRef" use="required">
    <xss:annotation>
      <xss:documentation>The resource being granted access to, declared and configured in the service manifest.</xss:documentation>
    </xss:annotation>
    <xss:simpleType>
      <xss:restriction base="xs:string">
        <xss:minLength value="1"/>
      </xss:restriction>
    </xss:simpleType>
  </xss:attribute>
  <xss:attribute name="PrincipalRef" use="required">
    <xss:annotation>
      <xss:documentation>The user or group being assigned access rights to a resource, must be declared in the Principals section.</xss:documentation>
    </xss:annotation>
    <xss:simpleType>
      <xss:restriction base="xs:string">
        <xss:minLength value="1"/>
      </xss:restriction>
    </xss:simpleType>
  </xss:attribute>
  <xss:attribute name="GrantRights" default="Read">
    <xss:annotation>
      <xss:documentation>The rights to grant, default is Read.</xss:documentation>
    </xss:annotation>
    <xss:simpleType>
      <xss:restriction base="xs:string">
        <xss:enumeration value="Read"/>
        <xss:enumeration value="Change"/>
        <xss:enumeration value="Full"/>
      </xss:restriction>
    </xss:simpleType>
  </xss:attribute>
  <xss:attribute name="ResourceType" use="optional" default="Endpoint">
    <xss:annotation>
      <xss:documentation>The type of resource, defined in the service manifest, either Endpoint or Certificate.</xss:documentation>
    </xss:annotation>
    <xss:simpleType>
      <xss:restriction base="xs:string">
        <xss:enumeration value="Endpoint"/>
        <xss:enumeration value="Certificate"/>
      </xss:restriction>
    </xss:simpleType>
  </xss:attribute>
</xss:complexType>

```

## Attribute details

### ResourceRef

Defines endpoints for the service.

ATTRIBUTE	VALUE
name	ResourceRef

ATTRIBUTE	VALUE
use	required

#### PrincipalRef

Defines endpoints for the service.

ATTRIBUTE	VALUE
name	PrincipalRef
use	required

#### GrantRights

Defines endpoints for the service.

ATTRIBUTE	VALUE
name	GrantRights
default	Read

#### ResourceType

Defines endpoints for the service.

ATTRIBUTE	VALUE
name	ResourceType
use	optional
default	Endpoint

## SecurityOptionsType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 1 attribute(s)
name	SecurityOptionsType

#### XML source

```
<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="SecurityOptionsType">
    <xss:attribute name="Value" use="required">
        <xss:simpleType>
            <xss:restriction base="xss:string">
                <xss:minLength value="1"/>
            </xss:restriction>
        </xss:simpleType>
    </xss:attribute>
</xss:complexType>
```

## Attribute details

## Value

Defines endpoints for the service.

ATTRIBUTE	VALUE
name	Value
use	required

## SecurityPrincipalsType complexType

Describes the security principals (users, groups) required for this application to run services and secure resources. Principals are referenced in the policies sections.

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 0 attribute(s)
name	SecurityPrincipalsType

## XML source

```

                                <xs:attribute name="IsEnabled" type="xs:boolean" use="optional"
default="true"/>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="Membership" minOccurs="0">
                                <xs:complexType>
                                    <xs:choice maxOccurs="unbounded">
  <xs:element name="DomainGroup" minOccurs="0"
maxOccurs="unbounded">
  <xs:complexType>
  <xs:attribute name="Name" type="xs:string"
use="required"/>
  </xs:complexType>
  </xs:element>
  <xs:element name="SystemGroup" minOccurs="0"
maxOccurs="unbounded">
  <xs:complexType>
  <xs:attribute name="Name" type="xs:string"
use="required"/>
  </xs:complexType>
  </xs:element>
  <xs:element name="DomainUser" minOccurs="0"
maxOccurs="unbounded">
  <xs:complexType>
  <xs:attribute name="Name" type="xs:string"
use="required"/>
  </xs:complexType>
  </xs:element>
                                    </xs:choice>
                                </xs:complexType>
                            </xs:element>
                            <xs:sequence>
                                <xs:attribute name="Name" type="xs:string" use="required">
                                    <xs:documentation>Name of the local group account. The name will be
prefixed with the application ID.</xs:documentation>
                                </xs:annotation>
                                </xs:attribute>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="Users" minOccurs="0">
                                <xs:annotation>
                                    <xs:documentation>Declares a set of users as security principals, which can be referenced
in policies.</xs:documentation>
                                </xs:annotation>
                                <xs:complexType>
                                    <xs:sequence>
  <xs:element name="User" maxOccurs="unbounded">
  <xs:annotation>
  <xs:documentation>Declares a user as a security principal, which can be
referenced in policies.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
  <xs:sequence>
  <xs:element name="NTLMAuthenticationPolicy" minOccurs="0">
  <xs:complexType>
  <xs:attribute name="IsEnabled" type="xs:boolean" use="optional"
default="true"/>
  <xs:attribute name="PasswordSecret" type="xs:string"
use="required"/>
  <xs:attribute name="PasswordSecretEncrypted" type="xs:boolean"
use="optional" default="false"/>
  <xs:attribute name="X509StoreLocation" use="optional"
default="LocalMachine">
  <xs:simpleType>
  <xs:restriction base="xs:string">

```

```

                <xs:enumeration value="LocalMachine"/>
                <xs:enumeration value="CurrentUser"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>

```

<xs:attribute

name="X509StoreName" default="My">

```

        <xs:simpleType>
            <xs:restriction base="xs:string"/>
        </xs:simpleType>
    </xs:attribute>

```

<xs:attribute name="X509Thumbprint" type="xs:string"/>

```

</xs:complexType>
</xs:element>

```

<xs:element name="MemberOf" minOccurs="0">

```

        <xs:annotation>
            <xs:documentation>
                Users can be added to any existing membership group, so it can inherit all the properties and security settings of that membership group. The membership group can be used to secure external resources that need to be accessed by different services or the same service (on a different machine).
            </xs:documentation>

```

</xs:annotation>

```

        </xs:complexType>
        <xs:choice>

```

maxOccurs="unbounded">

<xs:element

name="SystemGroup" minOccurs="0" maxOccurs="unbounded">

```

        <xs:annotation>
            <xs:documentation>The system group to add the user to. The system group must be defined in the Groups section.</xs:documentation>
        </xs:annotation>

```

</xs:annotation>

```

        <xs:complexType>

```

<xs:attribute name="Name" type="xs:string" use="required">

```

        <xs:annotation>
            <xs:documentation>The name of the system group.</xs:documentation>
        </xs:annotation>

```

</xs:annotation>

```

        </xs:attribute>

```

</xs:complexType>

</xs:element>

name="Group" minOccurs="0" maxOccurs="unbounded">

```

        <xs:annotation>
            <xs:documentation>The group to add the user to. The group must be defined in the Groups section.</xs:documentation>
        </xs:annotation>

```

</xs:annotation>

```

        <xs:complexType>

```

<xs:attribute name="NameRef" type="xs:string" use="required">

```

        <xs:annotation>
            <xs:documentation>The name of the group.</xs:documentation>
        </xs:annotation>

```

</xs:annotation>

```

</xs:attribute>

</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="Name" type="xs:string"
use="required">
<xs:annotation>
<xs:documentation>Name of the
user account.</xs:documentation>
<xs:annotation>
<xs:attribute name="AccountType" use="optional"
default="LocalUser">
<xs:annotation>
<xs:documentation>Specifies the
type of account. Local user accounts are created on the machines where the application is deployed. By default,
these accounts do not have the same names as those specified here. Instead, they are dynamically generated and
have random passwords. Supported local system account types are LocalUser, NetworkService, LocalService and
LocalSystem. Domain accounts are supported on Windows Server deployments where Azure Active Directory is
available.</xs:documentation>
<xs:annotation>
<xs:simpleType>
<xs:restriction>
<xs:enumeration
base="xs:string">
<xs:enumeration
value="LocalUser"/>
<xs:enumeration
value="DomainUser"/>
<xs:enumeration
value="NetworkService"/>
<xs:enumeration
value="LocalService"/>
<xs:enumeration
value="ManagedServiceAccount"/>
<xs:enumeration
value="LocalSystem"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
<xs:attribute name="LoadUserProfile"
type="xs:boolean" use="optional" default="false"/>
<xs:attribute name="PerformInteractiveLogon"
type="xs:boolean" use="optional" default="false"/>
<xss:attributeGroup
ref="AccountCredentialsGroup"/>
<xs:attribute name="PasswordEncrypted"
type="xs:boolean" use="optional">
<xs:annotation>
<xs:documentation>True if the
password is encrypted; false if in plain text.</xs:documentation>
<xs:annotation>
<xs:attribute>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>

```

## Content element details

### Groups

Declares a set of groups as security principals, which can be referenced in policies.

ATTRIBUTE	VALUE
name	Groups
minOccurs	0

#### Users

Declares a set of users as security principals, which can be referenced in policies.

ATTRIBUTE	VALUE
name	Users
minOccurs	0

## ServiceAndServiceGroupTypesType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	4 element(s), 0 attribute(s)
name	ServiceAndServiceGroupTypesType

#### XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceAndServiceGroupTypesType">
  <xs:choice maxOccurs="unbounded">
    <xs:element name="StatefulServiceType" type="StatefulServiceTypeType"/>
    <xs:element name="StatelessServiceType" type="StatelessServiceTypeType"/>
    <xs:element name="StatefulServiceGroupType" type="StatefulServiceGroupTypeType"/>
    <xs:element name="StatelessServiceGroupType" type="StatelessServiceGroupTypeType"/>
  </xs:choice>
</xs:complexType>
```

#### Content element details

##### StatefulServiceType

ATTRIBUTE	VALUE
name	StatefulServiceType
type	StatefulServiceTypeType

##### StatelessServiceType

ATTRIBUTE	VALUE
name	StatelessServiceType

ATTRIBUTE	VALUE
type	StatelessServiceTypeType

#### **StatefulServiceGroupType**

ATTRIBUTE	VALUE
name	StatefulServiceGroupType
type	StatefulServiceGroupTypeType

#### **StatelessServiceGroupType**

ATTRIBUTE	VALUE
name	StatelessServiceGroupType
type	StatelessServiceGroupTypeType

## ServiceDiagnosticsType complexType

Describes the diagnostic settings for the components of this service manifest.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	ServiceDiagnosticsType

#### **XML source**

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceDiagnosticsType">
    <xs:annotation>
        <xs:documentation>Describes the diagnostic settings for the components of this service manifest.
    </xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="ETW" minOccurs="0">
            <xs:annotation>
                <xs:documentation>Describes the ETW settings for the components of this service manifest.
            </xs:documentation>
            </xs:annotation>
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="ProviderGuids" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>Lists the ETW provider GUIDs for the components of this service manifest.
                        </xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="ProviderGuid" minOccurs="0" maxOccurs="unbounded">
                                    <xs:complexType>
  <xs:attribute name="Value" use="required">
  <xs:simpleType>
  <xs:restriction base="xs:string">
  <xs:pattern value="[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{12}"/>
  </xs:restriction>
  </xs:simpleType>
  </xs:attribute>
                                    </xs:complexType>
                                </xs:element>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="ManifestDataPackages" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>Lists the data packages containing ETW manifests for the components of this service manifest. The data package containing ETW manifests should not contain any other files.
                        </xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="ManifestDataPackage" type="DataPackageType" minOccurs="0" maxOccurs="unbounded"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>

```

## Content element details

### ETW

Describes the ETW settings for the components of this service manifest.

ATTRIBUTE	VALUE
name	ETW

ATTRIBUTE	VALUE
minOccurs	0

## ServiceGroupMemberType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 2 attribute(s)
name	ServiceGroupMemberType

### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceGroupMemberType">
    <xss:sequence>
        <xss:element name="LoadMetrics" minOccurs="0">
            <xss:annotation>
                <xss:documentation>Load metrics reported by this service.</xss:documentation>
            </xss:annotation>
            <xss:complexType>
                <xss:sequence>
                    <xss:element name="LoadMetric" type="LoadMetricType" maxOccurs="unbounded"/>
                </xss:sequence>
            </xss:complexType>
        </xss:element>
    </xss:sequence>
    <xss:attribute name="ServiceTypeName" use="required">
        <xss:annotation>
            <xss:documentation>Type of the service group member.</xss:documentation>
        </xss:annotation>
        <xss:simpleType>
            <xss:restriction base="xs:string">
                <xss:minLength value="1"/>
            </xss:restriction>
        </xss:simpleType>
    </xss:attribute>
    <xss:attribute name="Name" use="required">
        <xss:annotation>
            <xss:documentation>Name of the service group member relative to the name of the service group.</xss:documentation>
        </xss:annotation>
        <xss:simpleType>
            <xss:restriction base="xs:string">
                <xss:minLength value="1"/>
            </xss:restriction>
        </xss:simpleType>
    </xss:attribute>
</xss:complexType>

```

### Attribute details

#### ServiceTypeName

Describes the ETW settings for the components of this service manifest.

ATTRIBUTE	VALUE
name	ServiceTypeName
use	required

#### Name

Describes the ETW settings for the components of this service manifest.

ATTRIBUTE	VALUE
name	Name
use	required

#### Content element details

##### LoadMetrics

Load metrics reported by this service.

ATTRIBUTE	VALUE
name	LoadMetrics
minOccurs	0

## ServiceGroupTypeType complexType

Base type that describes a stateful or a stateless ServiceGroupType.

ATTRIBUTE	VALUE
type	anonymous complexType
content	4 element(s), 2 attribute(s)
name	ServiceGroupTypeType

#### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceGroupTypeType">
  <xss:annotation>
    <xss:documentation>Base type that describes a stateful or a stateless ServiceGroupType.</xss:documentation>
  </xss:annotation>
  <xss:sequence>
    <xss:element name="LoadMetrics" minOccurs="0">
      <xss:annotation>
        <xss:documentation>Load metrics reported by this service.</xss:documentation>
      </xss:annotation>
      <xss:complexType>
        <xss:sequence>
          <xss:element name="LoadMetric" type="LoadMetricType" maxOccurs="unbounded"/>
        </xss:sequence>
      </xss:complexType>
    </xss:element>
    <xss:element name="PlacementConstraints" type="xs:string" minOccurs="0">
      <xss:annotation>
        <xss:documentation>Constraints for the placement of services that are part of this package.
      </xss:annotation>
    </xss:element>
    <xss:element name="ServiceGroupMembers" minOccurs="0" maxOccurs="1">
      <xss:annotation>
        <xss:documentation>Member types of this service group type.</xss:documentation>
      </xss:annotation>
      <xss:complexType>
        <xss:sequence>
          <xss:element ref="ServiceGroupTypeMember" minOccurs="1" maxOccurs="unbounded"/>
        </xss:sequence>
      </xss:complexType>
    </xss:element>
    <xss:element ref="Extensions" minOccurs="0"/>
  </xss:sequence>
  <xss:attribute name="ServiceGroupName" use="required">
    <xss:annotation>
      <xss:documentation>User defined type identifier for a service group, For example, "ActorQueueSGType". This value is used in the ApplicationManifest.xml file to identify the service group.</xss:documentation>
    </xss:annotation>
    <xss:simpleType>
      <xss:restriction base="xs:string">
        <xss:minLength value="1"/>
      </xss:restriction>
    </xss:simpleType>
  </xss:attribute>
  <xss:attribute name="UseImplicitFactory" type="xs:boolean" use="optional">
    <xss:annotation>
      <xss:documentation>Specifies whether the service group instance is created by the implicit factory. If false (default), one of the code packages must register the service group factory</xss:documentation>
    </xss:annotation>
  </xss:attribute>
</xss:complexType>

```

## Attribute details

### ServiceGroupName

Load metrics reported by this service.

ATTRIBUTE	VALUE
name	ServiceGroupName
use	required

### UseImplicitFactory

Load metrics reported by this service.

ATTRIBUTE	VALUE
name	UseImplicitFactory
type	xs:boolean
use	optional

## Content element details

### LoadMetrics

Load metrics reported by this service.

ATTRIBUTE	VALUE
name	LoadMetrics
minOccurs	0

### PlacementConstraints

Constraints for the placement of services that are part of this package.

ATTRIBUTE	VALUE
name	PlacementConstraints
type	xs:string
minOccurs	0

### ServiceGroupMembers

Member types of this service group type.

ATTRIBUTE	VALUE
name	ServiceGroupMembers
minOccurs	0
maxOccurs	1

### None

ATTRIBUTE	VALUE
ref	Extensions
minOccurs	0

## ServiceManifestImportPoliciesType complexType

Describes policies (end-point binding, package sharing, run-as, and security access) to be applied on the imported service manifest.

ATTRIBUTE	VALUE
type	anonymous complexType
content	7 element(s), 0 attribute(s)
name	ServiceManifestImportPoliciesType

## XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceManifestImportPoliciesType">
  <xs:annotation>
    <xs:documentation>Describes policies (end-point binding, package sharing, run-as, and security access) to
be applied on the imported service manifest.</xs:documentation>
  </xs:annotation>
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="RunAsPolicy" type="RunAsPolicyType" minOccurs="0"/>
    <xs:element name="SecurityAccessPolicy" type="SecurityAccessPolicyType" minOccurs="0"/>
    <xs:element name="PackageSharingPolicy" type="PackageSharingPolicyType" minOccurs="0"/>
    <xs:element name="EndpointBindingPolicy" type="EndpointBindingPolicyType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Specifies a certificate that should be returned to a client for an HTTPS endpoint.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="ServicePackageResourceGovernancePolicy"
type="ServicePackageResourceGovernancePolicyType" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation>Defines the resource governance policy that is applied at the level of the entire
service package.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="ResourceGovernancePolicy" type="ResourceGovernancePolicyType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Specifies resource limits for codepackage.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="ContainerHostPolicies" type="ContainerHostPoliciesType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Specifies policies for activating container hosts.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:choice>
</xs:complexType>

```

## Content element details

### RunAsPolicy

ATTRIBUTE	VALUE
name	RunAsPolicy
type	RunAsPolicyType
minOccurs	0

### SecurityAccessPolicy

ATTRIBUTE	VALUE
name	SecurityAccessPolicy
type	SecurityAccessPolicyType
minOccurs	0

#### PackageSharingPolicy

ATTRIBUTE	VALUE
name	PackageSharingPolicy
type	PackageSharingPolicyType
minOccurs	0

#### EndpointBindingPolicy

Specifies a certificate that should be returned to a client for an HTTPS endpoint.

ATTRIBUTE	VALUE
name	EndpointBindingPolicy
type	EndpointBindingPolicyType
minOccurs	0

#### ServicePackageResourceGovernancePolicy

Defines the resource governance policy that is applied at the level of the entire service package.

ATTRIBUTE	VALUE
name	ServicePackageResourceGovernancePolicy
type	ServicePackageResourceGovernancePolicyType
minOccurs	0
maxOccurs	1

#### ResourceGovernancePolicy

Specifies resource limits for codepackage.

ATTRIBUTE	VALUE
name	ResourceGovernancePolicy
type	ResourceGovernancePolicyType
minOccurs	0

#### ContainerHostPolicies

Specifies policies for activating container hosts.

ATTRIBUTE	VALUE
name	ContainerHostPolicies
type	ContainerHostPoliciesType
minOccurs	0

## ServiceManifestRefType complexType

Imports the service manifest by reference. Currently the service manifest file (ServiceManifest.xml) must be present in the build package.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	ServiceManifestRefType

### XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceManifestRefType">
  <xs:annotation>
    <xs:documentation>Imports the service manifest by reference. Currently the service manifest file
(ServiceManifest.xml) must be present in the build package.</xs:documentation>
  </xs:annotation>
  <xs:attributeGroup ref="ServiceManifestIdentifier"/>
</xs:complexType>
```

## ServiceManifestType complexType

Declaratively describes the service type and version. It lists the independently upgradeable code, configuration, and data packages that together compose a service package to support one or more service types. Resources, diagnostics settings, and service metadata, such as service type, health properties, and load-balancing metrics, are also specified.

ATTRIBUTE	VALUE
type	anonymous complexType
content	7 element(s), 1 attribute(s)
name	ServiceManifestType

### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceManifestType">
  <xss:annotation>
    <xss:documentation>Declaratively describes the service type and version. It lists the independently
upgradeable code, configuration, and data packages that together compose a service package to support one or
more service types. Resources, diagnostics settings, and service metadata, such as service type, health
properties, and load-balancing metrics, are also specified.</xss:documentation>
  </xss:annotation>
  <xss:sequence>
    <xss:element name="Description" type="xs:string" minOccurs="0">
      <xss:annotation>
        <xss:documentation>Text describing this service.</xss:documentation>
      </xss:annotation>
    </xss:element>
    <xss:element name="ServiceTypes" type="ServiceAndServiceGroupTypesType">
      <xss:annotation>
        <xss:documentation>Defines what service types are supported by a CodePackage in this manifest. When a
service is instantiated against one of these service types, all code packages declared in this manifest are
activated by running their entry points. Service types are declared at the manifest level and not the code
package level.</xss:documentation>
      </xss:annotation>
    </xss:element>
    <xss:element name="CodePackage" type="CodePackageType" maxOccurs="unbounded"/>
    <xss:element name="ConfigPackage" type="ConfigPackageType" minOccurs="0" maxOccurs="unbounded"/>
    <xss:element name="DataPackage" type="DataPackageType" minOccurs="0" maxOccurs="unbounded"/>
    <xss:element name="Resources" type="ResourcesType" minOccurs="0"/>
    <xss:element name="Diagnostics" type="ServiceDiagnosticsType" minOccurs="0"/>
  </xss:sequence>
  <xss:attribute name="ManifestId" use="optional" default="" type="xs:string">
    <xss:annotation>
      <xss:documentation>The identifier of this service manifest, an un-structured string.</xss:documentation>
    </xss:annotation>
  </xss:attribute>
  <xss:attributeGroup ref="VersionedName"/>
  <xss:anyAttribute processContents="skip"/> <!-- Allow unknown attributes to be used. -->
</xss:complexType>

```

## Attribute details

### ManifestId

Specifies policies for activating container hosts.

ATTRIBUTE	VALUE
name	ManifestId
use	optional
default	
type	xs:string

## Content element details

### Description

Text describing this service.

ATTRIBUTE	VALUE
name	Description

ATTRIBUTE	VALUE
type	xs:string
minOccurs	0

#### ServiceTypes

Defines what service types are supported by a CodePackage in this manifest. When a service is instantiated against one of these service types, all code packages declared in this manifest are activated by running their entry points. Service types are declared at the manifest level and not the code package level.

ATTRIBUTE	VALUE
name	ServiceTypes
type	ServiceAndServiceGroupTypesType

#### CodePackage

ATTRIBUTE	VALUE
name	CodePackage
type	CodePackageType
maxOccurs	unbounded

#### ConfigPackage

ATTRIBUTE	VALUE
name	ConfigPackage
type	ConfigPackageType
minOccurs	0
maxOccurs	unbounded

#### DataPackage

ATTRIBUTE	VALUE
name	DataPackage
type	DataPackageType
minOccurs	0
maxOccurs	unbounded

#### Resources

ATTRIBUTE	VALUE
name	Resources
type	ResourcesType
minOccurs	0

#### Diagnostics

ATTRIBUTE	VALUE
name	Diagnostics
type	ServiceDiagnosticsType
minOccurs	0

## ServicePackageResourceGovernancePolicyType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 2 attribute(s)
name	ServicePackageResourceGovernancePolicyType

#### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServicePackageResourceGovernancePolicyType">
    <xs:attribute name="CpuCores" use="optional" default="0">
        <xs:annotation>
            <xs:documentation>CPU limit in number of logical cores. Must be a positive integer.</xs:documentation>
        </xs:annotation>
        <xs:simpleType>
            <xs:restriction base="xs:decimal">
                <xs:minInclusive value="0"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="MemoryInMB" use="optional" default="0">
        <xs:annotation>
            <xs:documentation>Memory limits in MB. Must be a positive integer.</xs:documentation>
        </xs:annotation>
        <xs:simpleType>
            <xs:restriction base="xs:nonNegativeInteger"/>
        </xs:simpleType>
    </xs:attribute>
</xs:complexType>

```

#### Attribute details

##### CpuCores

ATTRIBUTE	VALUE
name	CpuCores
use	optional
default	0

#### MemoryInMB

ATTRIBUTE	VALUE
name	MemoryInMB
use	optional
default	0

## ServicePackageType complexType

ServicePackage represents a versioned unit of deployment and activation. The version of the ServicePackage is determined based on the manifest version and the version of the overrides.

ATTRIBUTE	VALUE
type	anonymous complexType
content	8 element(s), 4 attribute(s)
name	ServicePackageType

#### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServicePackageType">
    <xs:annotation>
        <xs:documentation>ServicePackage represents a versioned unit of deployment and activation. The version of
the ServicePackage is determined based on the manifest version and the version of the overrides.
</xs:documentation>
        <xs:annotation>
            <xs:sequence>
                <xs:element name="Description" type="xs:string" minOccurs="0"/>
                <xs:element name="ServicePackageResourceGovernancePolicy"
type="ServicePackageResourceGovernancePolicyType" minOccurs="0" maxOccurs="1">
                    <xs:element name="DigestedServiceTypes">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="ServiceTypes" type="ServiceTypesType"/>
                            </xs:sequence>
                            <xs:attributeGroup ref="VersionedItemAttrGroup"/>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="DigestedCodePackage" maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="CodePackage" type="CodePackageType"/>
                                <xs:element name="RunAsPolicy" type="RunAsPolicyType" minOccurs="0" maxOccurs="2"/>
                                <xs:element name="DebugParameters" type="DebugParametersType" minOccurs="0" maxOccurs="1"/>
                                <xs:element name="ContainerHostPolicies" type="ContainerHostPoliciesType" minOccurs="0">

```

```

<xs:annotation>
    <xs:documentation>Specifies policies for activating container hosts.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="ResourceGovernancePolicy" type="ResourceGovernancePolicyType" minOccurs="0">
    <xs:annotation>
        <xs:documentation>Specifies resource limits for codepackage.</xs:documentation>
    </xs:annotation>
</xs:element>
</xs:sequence>
<xs:attributeGroup ref="VersionedItemAttrGroup"/>
<xs:attribute name="ContentChecksum" type="xs:string"/>
<xs:attribute name="IsShared" type="xs:boolean"/>
</xs:complexType>
</xs:element>
<xs:element name="DigestedConfigPackage" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="ConfigPackage" type="ConfigPackageType"/>
            <xs:element name="ConfigOverride" type="ConfigOverrideType" minOccurs="0"/>
            <xs:element name="DebugParameters" type="DebugParametersType" minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
        <xs:attributeGroup ref="VersionedItemAttrGroup"/>
        <xs:attribute name="ContentChecksum" type="xs:string"/>
        <xs:attribute name="IsShared" type="xs:boolean"/>
    </xs:complexType>
</xs:element>
<xs:element name="DigestedDataPackage" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="DataPackage" type="DataPackageType"/>
            <xs:element name="DebugParameters" type="DebugParametersType" minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
        <xs:attributeGroup ref="VersionedItemAttrGroup"/>
        <xs:attribute name="ContentChecksum" type="xs:string"/>
        <xs:attribute name="IsShared" type="xs:boolean"/>
    </xs:complexType>
</xs:element>
<xs:element name="DigestedResources" minOccurs="1">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="DigestedEndpoints" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="DigestedEndpoint" minOccurs="0" maxOccurs="unbounded">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element name="Endpoint" type="EndpointType"/>
                                    <xs:element name="SecurityAccessPolicy" type="SecurityAccessPolicyType" minOccurs="0"/>
                                    <xs:element name="EndpointBindingPolicy" type="EndpointBindingPolicyType" minOccurs="0"/>
                                </xs:sequence>
                            </xs:complexType>
                            <xs:element name="ResourceGovernancePolicy" type="ResourceGovernancePolicyType" minOccurs="0" maxOccurs="1"/>
                            </xs:sequence>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="DigestedCertificates" minOccurs="0" maxOccurs="1">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="EndpointCertificate" type="EndpointCertificateType" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:sequence>
<xs:attributeGroup ref="VersionedItemAttrGroup"/>

```

```

</xs:complexType>
</xs:element>
<xs:element name="Diagnostics" type="ServiceDiagnosticsType"/>
</xs:sequence>
<xs:attribute name="Name" type="xs:string" use="required"/>
<xs:attribute name="ManifestVersion" type="xs:string" use="required"/>
<xs:attributeGroup ref="VersionedItemAttrGroup"/>
<xs:attribute name="ManifestChecksum" type="xs:string">
    <xs:annotation>
        <xs:documentation>Checksum value of the ServiceManifest file</xs:documentation>
    </xs:annotation>
</xs:attribute>
<xs:attribute name="ContentChecksum" type="xs:string">
    <xs:annotation>
        <xs:documentation>Checksum value of this ServicePackage content</xs:documentation>
    </xs:annotation>
</xs:attribute>
</xs:complexType>

```

## Attribute details

### Name

ATTRIBUTE	VALUE
name	Name
type	xs:string
use	required

### ManifestVersion

ATTRIBUTE	VALUE
name	ManifestVersion
type	xs:string
use	required

### ManifestChecksum

ATTRIBUTE	VALUE
name	ManifestChecksum
type	xs:string

### ContentChecksum

ATTRIBUTE	VALUE
name	ContentChecksum
type	xs:string

## Content element details

### Description

ATTRIBUTE	VALUE
name	Description
type	xs:string
minOccurs	0

#### ServicePackageResourceGovernancePolicy

ATTRIBUTE	VALUE
name	ServicePackageResourceGovernancePolicy
type	ServicePackageResourceGovernancePolicyType
minOccurs	0
maxOccurs	1

#### DigestedServiceTypes

ATTRIBUTE	VALUE
name	DigestedServiceTypes

#### DigestedCodePackage

ATTRIBUTE	VALUE
name	DigestedCodePackage
maxOccurs	unbounded

#### DigestedConfigPackage

ATTRIBUTE	VALUE
name	DigestedConfigPackage
minOccurs	0
maxOccurs	unbounded

#### DigestedDataPackage

ATTRIBUTE	VALUE
name	DigestedDataPackage
minOccurs	0
maxOccurs	unbounded

#### DigestedResources

ATTRIBUTE	VALUE
name	DigestedResources
minOccurs	1

#### Diagnostics

ATTRIBUTE	VALUE
name	Diagnostics
type	ServiceDiagnosticsType

## ServiceTemplatesType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	4 element(s), 0 attribute(s)
name	ServiceTemplatesType

#### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceTemplatesType">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="StatelessService" type="StatelessServiceType"/>
        <xs:element name="StatefulService" type="StatefulServiceType"/>
        <xs:element name="StatelessServiceGroup" type="StatelessServiceGroupType"/>
        <xs:element name="StatefulServiceGroup" type="StatefulServiceGroupType"/>
    </xs:choice>
</xs:complexType>

```

#### Content element details

##### StatelessService

ATTRIBUTE	VALUE
name	StatelessService
type	StatelessServiceType

##### StatefulService

ATTRIBUTE	VALUE
name	StatefulService
type	StatefulServiceType

##### StatelessServiceGroup

ATTRIBUTE	VALUE
name	StatelessServiceGroup
type	StatelessServiceGroupType

#### StatefulServiceGroup

ATTRIBUTE	VALUE
name	StatefulServiceGroup
type	StatefulServiceGroupType

## ServiceType complexType

Base type that defines a Microsoft Azure Service Fabric service.

ATTRIBUTE	VALUE
type	anonymous complexType
content	4 element(s), 2 attribute(s)
name	ServiceType

#### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceType">
    <xs:annotation>
        <xs:documentation>Base type that defines a Microsoft Azure Service Fabric service.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:group ref="PartitionDescriptionGroup"/>
        <xs:element name="LoadMetrics" minOccurs="0">
            <xs:annotation>
                <xs:documentation>Load metrics reported by this service, used for resource balancing
services.</xs:documentation>
            </xs:annotation>
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="LoadMetric" type="LoadMetricType" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="PlacementConstraints" type="xs:string" minOccurs="0">
            <xs:annotation>
                <xs:documentation>Used to control which nodes in the cluster a service can run on. A
key/value pair which describes the node property name and the service's requirements for the value. Individual
statements can be grouped together with simple boolean logic to create the necessary constraint. For example, "
(FirmwareVersion>12 && InDMZ == True)".</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="ServiceCorrelations" minOccurs="0">
            <xs:annotation>
                <xs:documentation>Defines affinity relationships between services.</xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>

```

```

<xss:sequence>
    <xss:element name="ServiceCorrelation" maxOccurs="unbounded">
        <xss:annotation>
            <xss:documentation>Defines an affinity relationship with another service. Useful when splitting a previously-monolithic application into microservices. One service has a local dependency on another service and both services need to run on the same node in order to work.</xss:documentation>
        </xss:annotation>
        <xss:complexType>
            <xss:attribute name="ServiceName" use="required">
                <xss:annotation>
                    <xss:documentation>The name of the other service as a URI. Example, "fabric:/otherApplication/parentService".</xss:documentation>
                </xss:annotation>
                <xss:simpleType>
                    <xss:restriction base="xs:string">
                        <xss:minLength value="1"/>
                    </xss:restriction>
                </xss:simpleType>
            </xss:attribute>
            <xss:attribute name="Scheme" use="required">
                <xss:annotation>
                    <xss:documentation>In NonAlignedAffinity the replicas or instances of the different services are placed on the same nodes. AlignedAffinity is used with stateful services. Configuring one stateful service as having aligned affinity with another stateful service ensures that the primaries of those services are placed on the same nodes as each other, and that each pair of secondaries are also placed on the same nodes.</xss:documentation>
                </xss:annotation>
                <xss:simpleType>
                    <xss:restriction base="xs:string">
                        <xss:enumeration value="Affinity"/>
                        <xss:enumeration value="AlignedAffinity"/>
                        <xss:enumeration value="NonAlignedAffinity"/>
                    </xss:restriction>
                </xss:simpleType>
            </xss:attribute>
        </xss:complexType>
    </xss:element>
</xss:sequence>
<xss:complexType>
</xss:element>
<xss:element name="ServicePlacementPolicies" minOccurs="0">
    <xss:annotation>
        <xss:documentation>Declares placement policies for a service. Useful when the cluster spans geographic distances or and/or geopolitical regions.</xss:documentation>
    </xss:annotation>
    <xss:complexType>
        <xss:sequence>
            <xss:element name="ServicePlacementPolicy" maxOccurs="unbounded">
                <xss:annotation>
                    <xss:documentation>Defines a service placement policy, which specifies that the service should or should not run in certain cluster fault domains. Useful when the cluster spans geographic distances or and/or geopolitical regions.</xss:documentation>
                </xss:annotation>
                <xss:complexType>
                    <xss:attribute name="DomainName">
                        <xss:annotation>
                            <xss:documentation>The fault domain where the service should or should not be placed, depending on the Type value.</xss:documentation>
                        </xss:annotation>
                        <xss:simpleType>
                            <xss:restriction base="xs:string">
                                <xss:minLength value="1"/>
                            </xss:restriction>
                        </xss:simpleType>
                    </xss:attribute>
                    <xss:attribute name="Type" use="required">
                        <xss:annotation>
                            <xss:documentation>InvalidDomain allows you to specify that a particular Fault Domain is invalid for this workload. RequiredDomain requires that all of the replicas be present in the</xss:documentation>
                        </xss:annotation>
                    </xss:attribute>
                </xss:complexType>
            </xss:element>
        </xss:sequence>
    </xss:complexType>
</xss:element>

```

specified domain. Multiple required domains can be specified. PreferredPrimaryDomain specifies the preferred Fault Domain for primary replicas. Useful in geographically spanned clusters where you are using other locations for redundancy, but would prefer that the primary replicas be placed in a certain location in order to provider lower latency for operations which go to the primary. RequiredDomainDistribution specifies that replicas are required to be distributed among the available fault domains. NonPartiallyPlace controls if the service replicas will be partially place if not all of them can be placed.</xs:documentation>

```

        </xs:annotation>
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="InvalidDomain"/>
                <xs:enumeration value="RequiredDomain"/>
                <xs:enumeration value="PreferredPrimaryDomain"/>
                <xs:enumeration value="RequiredDomainDistribution"/>
                <xs:enumeration value="NonPartiallyPlace"/>
            </xs:restriction>
        </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
    </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="ServiceTypeName" use="required">
    <xs:annotation>
        <xs:documentation>Name of the service type, declared in the service manifest, that will be instantiated.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
        </xs:restriction>
    </xs:simpleType>
</xs:attribute>
<xs:attribute name="DefaultMoveCost">
    <xs:annotation>
        <xs:documentation>Specifies default move cost for this service.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="Zero"/>
            <xs:enumeration value="Low"/>
            <xs:enumeration value="Medium"/>
            <xs:enumeration value="High"/>
        </xs:restriction>
    </xs:simpleType>
</xs:attribute>
</xs:complexType>

```

## Attribute details

### ServiceTypeName

ATTRIBUTE	VALUE
name	ServiceTypeName
use	required

### DefaultMoveCost

ATTRIBUTE	VALUE
name	DefaultMoveCost

## Content element details

### LoadMetrics

Load metrics reported by this service, used for resource balancing services.

ATTRIBUTE	VALUE
name	LoadMetrics
minOccurs	0

### PlacementConstraints

Used to control which nodes in the cluster a service can run on. A key/value pair which describes the node property name and the service's requirements for the value. Individual statements can be grouped together with simple boolean logic to create the necessary constraint. For example, "(FirmwareVersion>12 && InDMZ == True)".

ATTRIBUTE	VALUE
name	PlacementConstraints
type	xs:string
minOccurs	0

### ServiceCorrelations

Defines affinity relationships between services.

ATTRIBUTE	VALUE
name	ServiceCorrelations
minOccurs	0

### ServicePlacementPolicies

Declares placement policies for a service. Useful when the cluster spans geographic distances or and/or geopolitical regions.

ATTRIBUTE	VALUE
name	ServicePlacementPolicies
minOccurs	0

## ServiceTypeExtensionPolicyPropertiesType complexType

Defines Properties for the Persistence and Eviction policies.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	ServiceTypeExtensionPolicyPropertiesType

## XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceTypeExtensionPolicyPropertiesType">
<xs:annotation>
<xs:documentation>Defines Properties for the Persistence and Eviction policies.</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="Property" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:attribute name="Name" type="xs:string" use="required"/>
<xs:attribute name="Value" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
```

## Content element details

### Property

ATTRIBUTE	VALUE
name	Property
minOccurs	0
maxOccurs	unbounded

## ServiceTypeHealthPolicyType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 3 attribute(s)
name	ServiceTypeHealthPolicyType

## XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceTypeHealthPolicyType">
  <xss:attribute name="MaxPercentUnhealthyServices" type="xs:string" use="optional" default="0">
    <xss:annotation>
      <xss:documentation>Specifies the maximum tolerated percentage of unhealthy services before the
application is considered unhealthy. Default percentage: 0.</xss:documentation>
    </xss:annotation>
  </xss:attribute>
  <xss:attribute name="MaxPercentUnhealthyPartitionsPerService" type="xs:string" use="optional" default="0">
    <xss:annotation>
      <xss:documentation>Specifies the maximum tolerated percentage of unhealthy partitions before a service
is considered unhealthy. Default percentage: 0.</xss:documentation>
    </xss:annotation>
  </xss:attribute>
  <xss:attribute name="MaxPercentUnhealthyReplicasPerPartition" type="xs:string" use="optional" default="0">
    <xss:annotation>
      <xss:documentation>Specifies the maximum tolerated percentage of unhealthy replicas before a partition
is considered unhealthy. Default percentage: 0.</xss:documentation>
    </xss:annotation>
  </xss:attribute>
</xss:complexType>

```

## Attribute details

### MaxPercentUnhealthyServices

ATTRIBUTE	VALUE
name	MaxPercentUnhealthyServices
type	xs:string
use	optional
default	0

### MaxPercentUnhealthyPartitionsPerService

ATTRIBUTE	VALUE
name	MaxPercentUnhealthyPartitionsPerService
type	xs:string
use	optional
default	0

### MaxPercentUnhealthyReplicasPerPartition

ATTRIBUTE	VALUE
name	MaxPercentUnhealthyReplicasPerPartition
type	xs:string
use	optional

ATTRIBUTE	VALUE
default	0

## ServiceTypeType complexType

Base type that describes a stateful or a stateless ServiceType.

ATTRIBUTE	VALUE
type	anonymous complexType
content	4 element(s), 1 attribute(s)
name	ServiceTypeType

### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceTypeType">
  <xs:annotation>
    <xs:documentation>Base type that describes a stateful or a stateless ServiceType.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="LoadMetrics" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Load metrics reported by this service.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="LoadMetric" type="LoadMetricType" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="PlacementConstraints" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Used to control which nodes in the cluster a service can run on. A key/value pair
which describes the node property name and the service's requirements for the value. Individual statements can
be grouped together with simple boolean logic to create the necessary constraint. For example,
"(FirmwareVersion>12 && InDMZ == True)".</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="ServicePlacementPolicies" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Declares placement policies for a service. Useful when the cluster spans
geographic distances or and/or geopolitical regions.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="ServicePlacementPolicy" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>Defines a service placement policy, which specifies that the service should
or should not run in certain cluster fault domains. Useful when the cluster spans geographic distances or
and/or geopolitical regions.</xs:documentation>
            </xs:annotation>
            <xs:complexType>
              <xs:attribute name="DomainName">
                <xs:annotation>
                  <xs:documentation>The fault domain where the service should or should not be placed,
depending on the Type value.</xs:documentation>
                </xs:annotation>
                <xs:simpleType>
                  <xs:restriction base="xs:string">

```

```

<xs:restriction base="xs:string">
    <xss:minLength value="1"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
<xs:attribute name="Type" use="required">
    <xss:annotation>
        <xss/documentation>InvalidDomain allows you to specify that a particular Fault Domain is invalid for this workload. RequiredDomain requires that all of the replicas be present in the specified domain. Multiple required domains can be specified. PreferredPrimaryDomain specifies the preferred Fault Domain for primary replicas. Useful in geographically spanned clusters where you are using other locations for redundancy, but would prefer that the primary replicas be placed in a certain location in order to provider lower latency for operations which go to the primary. RequiredDomainDistribution specifies that replicas are required to be distributed among the available fault domains. NonPartiallyPlace controls if the service replicas will be partially place if not all of them can be placed. </xss/documentation>
    </xss:annotation>
</xs:simpleType>
<xs:restriction base="xs:string">
    <xss:enumeration value="InvalidDomain"/>
    <xss:enumeration value="RequiredDomain"/>
    <xss:enumeration value="PreferredPrimaryDomain"/>
    <xss:enumeration value="RequiredDomainDistribution"/>
    <xss:enumeration value="NonPartiallyPlace"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element ref="Extensions" minOccurs="0"/>
</xs:sequence>
<xs:attribute name="ServiceTypeName" use="required">
    <xss:annotation>
        <xss/documentation>User defined type identifier for a service. For example, "QueueType" or "CalculatorType". This value is used in the ApplicationManifest.xml file to identify the service. </xss/documentation>
    </xss:annotation>
</xs:attribute>
</xs:complexType>

```

## Attribute details

### ServiceTypeName

ATTRIBUTE	VALUE
name	ServiceTypeName
use	required

## Content element details

### LoadMetrics

Load metrics reported by this service.

ATTRIBUTE	VALUE
name	LoadMetrics

ATTRIBUTE	VALUE
minOccurs	0

#### PlacementConstraints

Used to control which nodes in the cluster a service can run on. A key/value pair which describes the node property name and the service's requirements for the value. Individual statements can be grouped together with simple boolean logic to create the necessary constraint. For example, "(FirmwareVersion>12 && InDMZ == True)".

ATTRIBUTE	VALUE
name	PlacementConstraints
type	xs:string
minOccurs	0

#### ServicePlacementPolicies

Declares placement policies for a service. Useful when the cluster spans geographic distances or and/or geopolitical regions.

ATTRIBUTE	VALUE
name	ServicePlacementPolicies
minOccurs	0

#### None

ATTRIBUTE	VALUE
ref	Extensions
minOccurs	0

## ServiceTypesType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 0 attribute(s)
name	ServiceTypesType

#### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceTypesType">
  <xss:choice maxOccurs="unbounded">
    <xss:element name="StatefulServiceType" type="StatefulServiceTypeType">
      <xss:annotation>
        <xss:documentation>Describes a stateful ServiceType.</xss:documentation>
      </xss:annotation>
    </xss:element>
    <xss:element name="StatelessServiceType" type="StatelessServiceTypeType">
      <xss:annotation>
        <xss:documentation>Describes a stateless ServiceType.</xss:documentation>
      </xss:annotation>
    </xss:element>
  </xss:choice>
</xss:complexType>

```

## Content element details

### **StatefulServiceType**

Describes a stateful ServiceType.

ATTRIBUTE	VALUE
name	StatefulServiceType
type	StatefulServiceTypeType

### **StatelessServiceType**

Describes a stateless ServiceType.

ATTRIBUTE	VALUE
name	StatelessServiceType
type	StatelessServiceTypeType

## SettingsOverridesType complexType

Declares configuration settings in a service manifest to be overridden. It consists of one or more sections of key-value pairs. Parameter values can be encrypted using the Invoke-ServiceFabricEncryptSecret cmdlet.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	SettingsOverridesType

## XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="SettingsOverridesType">
  <xs:annotation>
    <xs:documentation>Declares configuration settings in a service manifest to be overridden. It consists of one or more sections of key-value pairs. Parameter values can be encrypted using the Invoke-ServiceFabricEncryptSecret cmdlet.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="Section" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>A section in the Settings.xml file to override.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Parameter" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>The setting to override.</xs:documentation>
            </xs:annotation>
            <xs:complexType>
              <xs:attributeGroup ref="NameValuePair"/>
              <xs:attribute name="IsEncrypted" type="xs:boolean" default="false">
                <xs:annotation>
                  <xs:documentation>
                    If true, the value of this parameter is encrypted. The application developer is responsible for creating a certificate and using the Invoke-ServiceFabricEncryptSecret cmdlet to encrypt sensitive information. The certificate information that will be used to encrypt the value is specified in the Certificates section.
                  </xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:boolean">
                  <xs:minLength value="1"/>
                </xs:restriction>
              </xs:complexType>
            </xs:element>
            <xs:attribute name="Name" use="required">
              <xs:annotation>
                <xs:documentation>The name of the section in the Settings.xml file to override.</xs:documentation>
              </xs:annotation>
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:minLength value="1"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

```

## Content element details

### Section

A section in the Settings.xml file to override.

ATTRIBUTE	VALUE
name	Section
maxOccurs	unbounded

## SettingsType complexType

Describes user defined settings for a ServiceComponent or an Application. It consists of one or more sections of key-value pairs. Parameter values can be encrypted using the Invoke-ServiceFabricEncryptSecret cmdlet.

key-value pairs.

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	SettingsType

## XML source

```
<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric" name="SettingsType">
    <xss:annotation>
        <xss:documentation>Describes user defined settings for a ServiceComponent or an Application. It consists of one or more sections of key-value pairs.</xss:documentation>
    </xss:annotation>
    <xss:sequence>
        <xss:element name="Section" minOccurs="0" maxOccurs="unbounded">
            <xss:annotation>
                <xss:documentation>A user defined named section.</xss:documentation>
            </xss:annotation>
            <xss:complexType>
                <xss:sequence>
                    <xss:element name="Parameter" minOccurs="0" maxOccurs="unbounded">
                        <xss:complexType>
                            <xss:attribute name="Name" type="xs:string" use="required"/>
                            <xss:attribute name="Value" type="xs:string" use="required"/>
                            <xss:attribute name="MustOverride" type="xs:boolean" default="false">
                                <xss:annotation>
                                    <xss:documentation>If true, the value of this parameter must be overridden by higher level configuration.</xss:documentation>
                                </xss:annotation>
                            </xss:attribute>
                            <xss:attribute name="IsEncrypted" type="xs:boolean" default="false">
                                <xss:annotation>
                                    <xss:documentation>If true, the value of this parameter is encrypted.</xss:documentation>
                                </xss:annotation>
                            </xss:attribute>
                        </xss:complexType>
                    </xss:element>
                </xss:sequence>
                <xss:attribute name="Name" type="xs:string" use="required"/>
            </xss:complexType>
        </xss:element>
    </xss:sequence>
</xss:complexType>
```

## Content element details

### Section

A user defined named section.

ATTRIBUTE	VALUE
name	Section
minOccurs	0
maxOccurs	unbounded

## StatefulServiceGroupType complexType

Defines a stateful service group.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	StatefulServiceGroupType

### XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric" name="StatefulServiceGroupType">
    <xs:annotation>
        <xs:documentation>Defines a stateful service group.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="StatefulServiceType">
            <xs:sequence>
                <xs:element name="Members" minOccurs="1" maxOccurs="1">
                    <xs:annotation>
                        <xs:documentation>Member services of this service group</xs:documentation>
                    </xs:annotation>
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Member" type="ServiceGroupMemberType" minOccurs="1" maxOccurs="unbounded"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

## StatefulServiceGroupTypeType complexType

Describes a stateful service group type.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	StatefulServiceGroupTypeType

### XML source

```

<xss:complexType xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatefulServiceGroupTypeType">
  <xss:annotation>
    <xss:documentation>Describes a stateful service group type.</xss:documentation>
  </xss:annotation>
  <xss:complexContent>
    <xss:extension base="ServiceGroupTypeType">
      <xss:attribute name="HasPersistedState" type="xs:boolean" default="false">
        <xss:annotation>
          <xss:documentation>True if the service group has state that needs to be persisted.
        </xss:documentation>
        </xss:annotation>
      </xss:attribute>
    </xss:extension>
  </xss:complexContent>
</xss:complexType>

```

## StatefulServiceType complexType

Defines a stateful service.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	StatefulServiceType

[XML source](#)

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatefulServiceType">
    <xs:annotation>
        <xs:documentation>Defines a stateful service.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="ServiceType">
            <xs:attribute name="TargetReplicaSetSize" type="xs:string" default="1">
                <xs:annotation>
                    <xs:documentation>Desired replica set size for the partitions of this stateful service.
                    Must be a positive integer. This is a target size; a replica set is still functional with less members. The
                    quorum is a majority based quorum.</xs:documentation>
                </xs:annotation>
            </xs:attribute>
            <xs:attribute name="MinReplicaSetSize" type="xs:string" default="1">
                <xs:annotation>
                    <xs:documentation>Minimum number of replicas required in the replica set to allow
                    writes. Must be positive integer less than TargetReplicaSetSize. </xs:documentation>
                </xs:annotation>
            </xs:attribute>
            <xs:attribute name="ReplicaRestartWaitDurationSeconds" type="xs:string" use="optional"
default="">
                <xs:annotation>
                    <xs:documentation>The duration between when a replica goes down and when a new replica
                    is created. When a persistent replica goes down, this timer starts. When it expires Service Fabric will create
                    a new replica on any node in the cluster.</xs:documentation>
                </xs:annotation>
            </xs:attribute>
            <xs:attribute name="QuorumLossWaitDurationSeconds" type="xs:string" use="optional" default="">
                <xs:annotation>
                    <xs:documentation>The maximum duration for which a partition is allowed to be in a
                    state of quorum loss. If the partition is still in quorum loss after this duration, the partition is recovered
                    from quorum loss by considering the down replicas as lost. Note that this can potentially incur data loss.
                    </xs:documentation>
                </xs:annotation>
            </xs:attribute>
            <xs:attribute name="StandByReplicaKeepDurationSeconds" type="xs:string" use="optional"
default="">
                <xs:annotation>
                    <xs:documentation>How long StandBy replicas should be maintained before being removed.
                    Sometimes a replica will be down for longer than the ReplicaRestartWaitDuration. In these cases a new replica
                    will be built to replace it. Sometimes however the loss is not permanent and the persistent replica is
                    eventually recovered. This now constitutes a StandBy replica.</xs:documentation>
                </xs:annotation>
            </xs:attribute>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

```

## StatefulServiceTypeType complexType

Describes a stateful service type.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	StatefulServiceTypeType

### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatefulServiceTypeType">
    <xs:annotation>
        <xs:documentation>Describes a stateful service type.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="ServiceTypeType">
            <xs:attribute name="HasPersistedState" type="xs:boolean" default="false">
                <xs:annotation>
                    <xs:documentation>True if the service has state that needs to be persisted on the local disk.</xs:documentation>
                </xs:annotation>
            </xs:attribute>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

```

## StatelessServiceGroupType complexType

Defines a stateless service group.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	StatelessServiceGroupType

### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatelessServiceGroupType">
    <xs:annotation>
        <xs:documentation>Defines a stateless service group.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="StatelessServiceType">
            <xs:sequence>
                <xs:element name="Members" minOccurs="1" maxOccurs="1">
                    <xs:annotation>
                        <xs:documentation>Member services of this service group</xs:documentation>
                    </xs:annotation>
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Member" type="ServiceGroupMemberType" minOccurs="1"
maxOccurs="unbounded"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

```

## StatelessServiceGroupTypeType complexType

Describes a stateless service group type.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	StatelessServiceGroupTypeType

## XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatelessServiceGroupTypeType">
<xs:annotation>
    <xs:documentation>Describes a stateless service group type.</xs:documentation>
</xs:annotation>
<xs:complexContent>
    <xs:extension base="ServiceGroupTypeType"/>
</xs:complexContent>
</xs:complexType>
```

## StatelessServiceType complexType

Defines a stateless service.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	StatelessServiceType

## XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatelessServiceType">
<xs:annotation>
    <xs:documentation>Defines a stateless service.</xs:documentation>
</xs:annotation>
<xs:complexContent>
    <xs:extension base="ServiceType">
        <xs:attribute name="InstanceCount" type="xs:string" default="1">
            <xs:annotation>
                <xs:documentation>Number of instances required for this stateless service (positive
integer).</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:extension>
</xs:complexContent>
</xs:complexType>
```

## StatelessServiceTypeType complexType

Describes a stateless service type.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	StatelessServiceTypeType

## XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="StatelessServiceTypeType">
  <xs:annotation>
    <xs:documentation>Describes a stateless service type.</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="ServiceTypeType">
      <xs:attribute name="UseImplicitHost" type="xs:boolean" default="false">
        <xs:annotation>
          <xs:documentation>Specifies if the service type should be implemented implicitly as a guest
executable. Guest executables are used for hosting any type of applications (such as Node.js or Java) or legacy
applications that do not implement the Service Fabric service interfaces.</xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

## TargetInformationType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	2 element(s), 0 attribute(s)
name	TargetInformationType

## XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="TargetInformationType">
  <xs:all>
    <xs:element name="CurrentInstallation" type="WindowsFabricDeploymentInformation" minOccurs="0"/>
    <xs:element name="TargetInstallation" type="WindowsFabricDeploymentInformation" minOccurs="1"/>
  </xs:all>
</xs:complexType>
```

## Content element details

### CurrentInstallation

ATTRIBUTE	VALUE
name	CurrentInstallation
type	WindowsFabricDeploymentInformation

ATTRIBUTE	VALUE
minOccurs	0

#### TargetInstallation

ATTRIBUTE	VALUE
name	TargetInstallation
type	WindowsFabricDeploymentInformation
minOccurs	1

## UnmanagedDllType complexType

Unsupported, do not use. The name of unmanaged assembly (for example, Queue.dll), to host.

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 0 attribute(s)
name	UnmanagedDllType

#### XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="UnmanagedDllType">
  <xs:annotation>
    <xs:documentation>Unsupported, do not use. The name of unmanaged assembly (for example, Queue.dll), to
host.</xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="xs:string"/>
  </xs:simpleContent>
</xs:complexType>
```

## WindowsFabricDeploymentInformation complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	0 element(s), 11 attribute(s)
name	WindowsFabricDeploymentInformation

#### XML source

```

<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="WindowsFabricDeploymentInformation">
    <xs:attribute name="InstanceId" type="xs:string" use="optional">
        <xs:annotation>
            <xs:documentation>This is the target instance of the node.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="MSILocation" type="xs:string" use="optional">
        <xs:annotation>
            <xs:documentation>This is the full path to the MSI location.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="ClusterManifestLocation" type="xs:string" use="optional">
        <xs:annotation>
            <xs:documentation>This is the full path to the Cluster Manifest Location.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="InfrastructureManifestLocation" type="xs:string" use="optional">
        <xs:annotation>
            <xs:documentation>This location of the infrastructure manifest that is internally generated.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="TargetVersion" type="xs:string" use="optional">
        <xs:annotation>
            <xs:documentation>This is the Target Version of the deployment.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="NodeName" type="xs:string" use="optional">
        <xs:annotation>
            <xs:documentation>This is the name of the Node to which the Fabric Upgrade is to
happe</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="RemoveNodeState" type="xs:boolean" use="optional" default="false">
        <xs:annotation>
            <xs:documentation>A flag indicating if RemoveNodeState Api should be called after removing node
configuration.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="UpgradeEntryPointExe" type="xs:string" use="optional">
        <xs:annotation>
            <xs:documentation>Name of the exe used by the installer service to upgrade </xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="UpgradeEntryPointExeParameters" type="xs:string" use="optional">
        <xs:annotation>
            <xs:documentation>Parameters to the Setup Entry point exe</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="UndoUpgradeEntryPointExe" type="xs:string" use="optional">
        <xs:annotation>
            <xs:documentation>Name of the exe used by the installer service to undo the upgrade</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="UndoUpgradeEntryPointExeParameters" type="xs:string" use="optional">
        <xs:annotation>
            <xs:documentation>Parameters to the Setup Entry point exe</xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:complexType>

```

## Attribute details

### InstanceId

ATTRIBUTE	VALUE
name	Instanceld
type	xs:string
use	optional

#### MSILocation

ATTRIBUTE	VALUE
name	MSILocation
type	xs:string
use	optional

#### ClusterManifestLocation

ATTRIBUTE	VALUE
name	ClusterManifestLocation
type	xs:string
use	optional

#### InfrastructureManifestLocation

ATTRIBUTE	VALUE
name	InfrastructureManifestLocation
type	xs:string
use	optional

#### TargetVersion

ATTRIBUTE	VALUE
name	TargetVersion
type	xs:string
use	optional

#### NodeName

ATTRIBUTE	VALUE
name	NodeName
type	xs:string

ATTRIBUTE	VALUE
use	optional

#### RemoveNodeState

ATTRIBUTE	VALUE
name	RemoveNodeState
type	xs:boolean
use	optional
default	false

#### UpgradeEntryPointExe

ATTRIBUTE	VALUE
name	UpgradeEntryPointExe
type	xs:string
use	optional

#### UpgradeEntryPointExeParameters

ATTRIBUTE	VALUE
name	UpgradeEntryPointExeParameters
type	xs:string
use	optional

#### UndoUpgradeEntryPointExe

ATTRIBUTE	VALUE
name	UndoUpgradeEntryPointExe
type	xs:string
use	optional

#### UndoUpgradeEntryPointExeParameters

ATTRIBUTE	VALUE
name	UndoUpgradeEntryPointExeParameters
type	xs:string
use	optional

## WindowsInfrastructureType complexType

ATTRIBUTE	VALUE
type	anonymous complexType
content	1 element(s), 0 attribute(s)
name	WindowsInfrastructureType

### XML source

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="WindowsInfrastructureType">
    <xss:sequence>
        <xss:element name="NodeList">
            <xss:complexType>
                <xss:sequence>
                    <xss:element name="Node" type="FabricNodeType"
maxOccurs="unbounded"/>
                </xss:sequence>
            </xss:complexType>
        </xss:element>
    </xss:sequence>
</xs:complexType>
```

### Content element details

#### NodeList

ATTRIBUTE	VALUE
name	NodeList

## AccountCredentialsGroup attributeGroup

ATTRIBUTE	VALUE
content	2 attribute(s)
name	AccountCredentialsGroup

### XML source

```
<xs:attributeGroup xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.microsoft.com/2011/01/fabric" name="AccountCredentialsGroup">
    <xss:attribute name="AccountName" type="xs:string" use="optional">
        <xss:annotation>
            <xss:documentation>User name or Service Account Name (i.e., MyMachine\JohnDoe or
John.Doe@department.contoso.com).</xss:documentation>
        </xss:annotation>
    </xss:attribute>
    <xss:attribute name="Password" type="xs:string" use="optional">
        <xss:annotation>
            <xss:documentation>Password for the user account.</xss:documentation>
        </xss:annotation>
    </xss:attribute>
</xs:attributeGroup>
```

## Attribute details

### AccountName

User name or Service Account Name (i.e., MyMachine\JohnDoe or John.Doe@department.contoso.com).

ATTRIBUTE	VALUE
name	AccountName
type	xs:string
use	optional

#### XML source

```
<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="AccountName" type="xs:string" use="optional">
    <xss:annotation>
        <xss:documentation>User name or Service Account Name (i.e., MyMachine\JohnDoe or
John.Doe@department.contoso.com).</xss:documentation>
    </xss:annotation>
</xs:attribute>
```

### Password

Password for the user account.

ATTRIBUTE	VALUE
name	Password
type	xs:string
use	optional

#### XML source

```
<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Password" type="xs:string" use="optional">
    <xss:annotation>
        <xss:documentation>Password for the user account.</xss:documentation>
    </xss:annotation>
</xs:attribute>
```

## ApplicationInstanceAttrGroup attributeGroup

Attribute group for application instance.

ATTRIBUTE	VALUE
content	2 attribute(s)
name	ApplicationInstanceAttrGroup

#### XML source

```

<xs:attributeGroup xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.microsoft.com/2011/01/fabric" name="ApplicationInstanceAttrGroup">
  <xs:annotation>
    <xs:documentation>Attribute group for application instance.</xs:documentation>
  </xs:annotation>
  <xs:attribute name="NameUri" type="FabricUri" use="required">
    <xs:annotation>
      <xs:documentation>Fully qualified name of the application.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="ApplicationId" type="xs:string" use="required">
    <xs:annotation>
      <xs:documentation>Id of this application.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:attributeGroup>

```

## Attribute details

### NameUri

Fully qualified name of the application.

ATTRIBUTE	VALUE
name	NameUri
type	FabricUri
use	required

### XML source

```

<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="NameUri" type="FabricUri" use="required">
  <xs:annotation>
    <xs:documentation>Fully qualified name of the application.</xs:documentation>
  </xs:annotation>
</xs:attribute>

```

### ApplicationId

Id of this application.

ATTRIBUTE	VALUE
name	ApplicationId
type	xs:string
use	required

### XML source

```

<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ApplicationId" type="xs:string" use="required">
  <xs:annotation>
    <xs:documentation>Id of this application.</xs:documentation>
  </xs:annotation>
</xs:attribute>

```

# ApplicationManifestAttrGroup attributeGroup

Attribute group for application manifest.

ATTRIBUTE	VALUE
content	3 attribute(s)
name	ApplicationManifestAttrGroup

## XML source

```
<xs:attributeGroup xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.microsoft.com/2011/01/fabric" name="ApplicationManifestAttrGroup">
  <xs:annotation>
    <xs:documentation>Attribute group for application manifest.</xs:documentation>
  </xs:annotation>
  <xs:attribute name="ApplicationTypeName" use="required">
    <xs:annotation>
      <xs:documentation>The type identifier for this application.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="ApplicationTypeVersion" use="required">
    <xs:annotation>
      <xs:documentation>The version of this application type, an un-structured string.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="ManifestId" use="optional" default="" type="xs:string">
    <xs:annotation>
      <xs:documentation>The identifier of this application manifest, an un-structured string.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:anyAttribute processContents="skip"/> <!-- Allow unknown attributes to be used. -->
</xs:attributeGroup>
```

## Attribute details

### ApplicationTypeName

The type identifier for this application.

ATTRIBUTE	VALUE
name	ApplicationTypeName
use	required

## XML source

```

<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ApplicationTypeName" use="required">
    <xs:annotation>
        <xs:documentation>The type identifier for this application.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
        </xs:restriction>
    </xs:simpleType>
</xs:attribute>

```

### ApplicationTypeVersion

The version of this application type, an un-structured string.

ATTRIBUTE	VALUE
name	ApplicationTypeVersion
use	required

#### XML source

```

<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ApplicationTypeVersion" use="required">
    <xs:annotation>
        <xs:documentation>The version of this application type, an un-structured string.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
        </xs:restriction>
    </xs:simpleType>
</xs:attribute>

```

### ManifestId

The identifier of this application manifest, an un-structured string.

ATTRIBUTE	VALUE
name	ManifestId
use	optional
default	
type	xs:string

#### XML source

```

<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ManifestId" use="optional" default="" type="xs:string">
    <xs:annotation>
        <xs:documentation>The identifier of this application manifest, an un-structured string.</xs:documentation>
    </xs:annotation>
</xs:attribute>

```

# ConfigOverridesIdentifier attributeGroup

Identifies configuration overrides for a service package.

ATTRIBUTE	VALUE
content	2 attribute(s)
name	ConfigOverridesIdentifier

## XML source

```
<xs:attributeGroup xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.microsoft.com/2011/01/fabric" name="ConfigOverridesIdentifier">
    <xs:annotation>
        <xs:documentation>Identifies configuration overrides for a service package.</xs:documentation>
    </xs:annotation>
    <xs:attribute name="ServicePackageName" type="xs:string" use="required"/>
    <xs:attribute name="RolloutVersion" type="xs:string" use="required">
        <xs:annotation>
            <xs:documentation>ID of the rollout in which changes were made to the overrides element.</xs:documentation>
        </xs:annotation>
        </xs:attribute>
    </xs:attributeGroup>
```

## Attribute details

### ServicePackageName

ATTRIBUTE	VALUE
name	ServicePackageName
type	xs:string
use	required

## XML source

```
<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServicePackageName" type="xs:string" use="required"/>
```

### RolloutVersion

ID of the rollout in which changes were made to the overrides element.

ATTRIBUTE	VALUE
name	RolloutVersion
type	xs:string
use	required

## XML source

```

<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="RolloutVersion" type="xs:string" use="required">
    <xs:annotation>
        <xs:documentation>ID of the rollout in which changes were made to the overrides element.
    </xs:documentation>
    </xs:annotation>
</xs:attribute>

```

## ConnectionString attributeGroup

ATTRIBUTE	VALUE
content	1 attribute(s)
name	ConnectionString

### XML source

```

<xs:attributeGroup xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.microsoft.com/2011/01/fabric" name="ConnectionString">
    <xs:attribute name="ConnectionString" type="xs:string" use="required">
        <xs:annotation>
            <xs:documentation>
                Connection string to the Azure storage account. Format:
                DefaultEndpointsProtocol=https;AccountName=[ ];AccountKey=[ ]
            </xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:attributeGroup>

```

## Attribute details

### ConnectionString

Connection string to the Azure storage account. Format:  
DefaultEndpointsProtocol=https;AccountName=[ ];AccountKey=[ ]

ATTRIBUTE	VALUE
name	ConnectionString
type	xs:string
use	required

### XML source

```

<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ConnectionString" type="xs:string" use="required">
    <xs:annotation>
        <xs:documentation>
            Connection string to the Azure storage account. Format:
            DefaultEndpointsProtocol=https;AccountName=[ ];AccountKey=[ ]
        </xs:documentation>
    </xs:annotation>
</xs:attribute>

```

## ContainerName attributeGroup

ATTRIBUTE	VALUE
content	1 attribute(s)
name	ContainerName

### XML source

```
<xs:attributeGroup xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.microsoft.com/2011/01/fabric" name="ContainerName">
<xs:attribute name="ContainerName" type="xs:string">
<xs:annotation>
<xs:documentation>The name of the container in Azure blob storage where data is uploaded.
</xs:documentation>
</xs:annotation>
</xs:attribute>
</xs:attributeGroup>
```

### Attribute details

#### ContainerName

The name of the container in Azure blob storage where data is uploaded.

ATTRIBUTE	VALUE
name	ContainerName
type	xs:string

### XML source

```
<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ContainerName" type="xs:string">
<xs:annotation>
<xs:documentation>The name of the container in Azure blob storage where data is uploaded.
</xs:documentation>
</xs:annotation>
</xs:attribute>
```

## DataDeletionAgeInDays attributeGroup

ATTRIBUTE	VALUE
content	1 attribute(s)
name	DataDeletionAgeInDays

### XML source

```

<xs:attributeGroup xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.microsoft.com/2011/01/fabric" name="DataDeletionAgeInDays">
    <xs:attribute name="DataDeletionAgeInDays" type="xs:string">
        <xs:annotation>
            <xs:documentation>Number of days after which old data is deleted from this location.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:attributeGroup>

```

## Attribute details

### DataDeletionAgeInDays

Number of days after which old data is deleted from this location.

ATTRIBUTE	VALUE
name	DataDeletionAgeInDays
type	xs:string

### XML source

```

<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
    name="DataDeletionAgeInDays" type="xs:string">
    <xs:annotation>
        <xs:documentation>Number of days after which old data is deleted from this location.</xs:documentation>
    </xs:annotation>
</xs:attribute>

```

## IsEnabled attributeGroup

ATTRIBUTE	VALUE
content	1 attribute(s)
name	isEnabled

### XML source

```

<xs:attributeGroup xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.microsoft.com/2011/01/fabric" name=".IsEnabled">
    <xs:attribute name=".IsEnabled" type="xs:string">
        <xs:annotation>
            <xs:documentation>Whether or not data transfer to this destination is enabled.  
By default, it is not enabled.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:attributeGroup>

```

## Attribute details

### .IsEnabled

Whether or not data transfer to this destination is enabled. By default, it is not enabled.

ATTRIBUTE	VALUE
name	isEnabled

ATTRIBUTE	VALUE
type	xs:string

#### XML source

```
<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="IsEnabled" type="xs:string">
    <xs:annotation>
        <xs:documentation>Whether or not data transfer to this destination is enabled.
By default, it is not enabled.</xs:documentation>
    </xs:annotation>
</xs:attribute>
```

## LevelFilter attributeGroup

ATTRIBUTE	VALUE
content	1 attribute(s)
name	LevelFilter

#### XML source

```
<xs:attributeGroup xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.microsoft.com/2011/01/fabric" name="LevelFilter">
    <xs:attribute name="LevelFilter" type="xs:string">
        <xs:annotation>
            <xs:documentation>Level at which ETW events should be filtered. All events at the same or lower level
than the specified level are included.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:attributeGroup>
```

## Attribute details

### LevelFilter

Level at which ETW events should be filtered. All events at the same or lower level than the specified level are included.

ATTRIBUTE	VALUE
name	LevelFilter
type	xs:string

#### XML source

```
<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="LevelFilter" type="xs:string">
    <xs:annotation>
        <xs:documentation>Level at which ETW events should be filtered. All events at the same or lower level
than the specified level are included.</xs:documentation>
    </xs:annotation>
</xs:attribute>
```

## NameValuePair attributeGroup

Name and Value defined as an attribute.

ATTRIBUTE	VALUE
content	2 attribute(s)
name	NameValuePair

### XML source

```
<xs:attributeGroup xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.microsoft.com/2011/01/fabric" name="NameValuePair">
  <xs:annotation>
    <xs:documentation>Name and Value defined as an attribute.</xs:documentation>
  </xs:annotation>
  <xs:attribute name="Name" use="required">
    <xs:annotation>
      <xs:documentation>The name of the setting to override.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="Value" type="xs:string" use="required">
    <xs:annotation>
      <xs:documentation>The new value of the setting.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:attributeGroup>
```

### Attribute details

#### Name

The name of the setting to override.

ATTRIBUTE	VALUE
name	Name
use	required

### XML source

```
<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Name" use="required">
  <xs:annotation>
    <xs:documentation>The name of the setting to override.</xs:documentation>
  </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
```

#### Value

The new value of the setting.

ATTRIBUTE	VALUE
name	Value
type	xs:string
use	required

#### XML source

```
<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Value" type="xs:string" use="required">
    <xs:annotation>
        <xs:documentation>The new value of the setting.</xs:documentation>
    </xs:annotation>
</xs:attribute>
```

## Path attributeGroup

ATTRIBUTE	VALUE
content	1 attribute(s)
name	Path

#### XML source

```
<xs:attributeGroup xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.microsoft.com/2011/01/fabric" name="Path">
    <xs:attribute name="Path" type="xs:string" use="required">
        <xs:annotation>
            <xs:documentation>
                Path to the file share. Format:
                file:[]
            </xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:attributeGroup>
```

## Attribute details

### Path

Path to the file share. Format:  
file:[]

ATTRIBUTE	VALUE
name	Path
type	xs:string
use	required

#### XML source

```

<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Path" type="xs:string" use="required">
    <xs:annotation>
        <xs:documentation>
            Path to the file share. Format:
            file:[]
        </xs:documentation>
    </xs:annotation>
</xs:attribute>

```

## RelativeFolderPath attributeGroup

ATTRIBUTE	VALUE
content	1 attribute(s)
name	RelativeFolderPath

### XML source

```

<xs:attributeGroup xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.microsoft.com/2011/01/fabric" name="RelativeFolderPath">
    <xs:attribute name="RelativeFolderPath" type="xs:string" use="required">
        <xs:annotation>
            <xs:documentation>Path to the folder, relative to the application log
directory.</xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:attributeGroup>

```

### Attribute details

#### RelativeFolderPath

Path to the folder, relative to the application log directory.

ATTRIBUTE	VALUE
name	RelativeFolderPath
type	xs:string
use	required

### XML source

```

<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="RelativeFolderPath" type="xs:string" use="required">
    <xs:annotation>
        <xs:documentation>Path to the folder, relative to the application log
directory.</xs:documentation>
    </xs:annotation>
</xs:attribute>

```

## ServiceManifestIdentifier attributeGroup

Identifies a service manifest.

ATTRIBUTE	VALUE
content	2 attribute(s)
name	ServiceManifestIdentifier

## XML source

```

<xs:attributeGroup xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.microsoft.com/2011/01/fabric" name="ServiceManifestIdentifier">
  <xs:annotation>
    <xs:documentation>Identifies a service manifest.</xs:documentation>
  </xs:annotation>
  <xs:attribute name="ServiceManifestName" use="required">
    <xs:annotation>
      <xs:documentation>The name of the service manifest being referenced. The name must match the Name declared in the ServiceManifest element of the service manifest.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="ServiceManifestVersion" use="required">
    <xs:annotation>
      <xs:documentation>The version of the service manifest being referenced. The version must match the version declared in the service manifest.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:attributeGroup>

```

## Attribute details

### ServiceManifestName

The name of the service manifest being referenced. The name must match the Name declared in the ServiceManifest element of the service manifest.

ATTRIBUTE	VALUE
name	ServiceManifestName
use	required

## XML source

```

<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceManifestName" use="required">
  <xs:annotation>
    <xs:documentation>The name of the service manifest being referenced. The name must match the Name
declared in the ServiceManifest element of the service manifest.</xs:documentation>
  </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>

```

#### ServiceManifestVersion

The version of the service manifest being referenced. The version must match the version declared in the service manifest.

ATTRIBUTE	VALUE
name	ServiceManifestVersion
use	required

#### XML source

```

<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="ServiceManifestVersion" use="required">
  <xs:annotation>
    <xs:documentation>The version of the service manifest being referenced. The version must match the
version declared in the service manifest.</xs:documentation>
  </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>

```

## UploadIntervalInMinutes attributeGroup

ATTRIBUTE	VALUE
content	1 attribute(s)
name	UploadIntervalInMinutes

#### XML source

```

<xs:attributeGroup xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.microsoft.com/2011/01/fabric" name="UploadIntervalInMinutes">
  <xs:attribute name="UploadIntervalInMinutes" type="xs:string">
    <xs:annotation>
      <xs:documentation>Interval in minutes at which data is uploaded to this destination.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:attributeGroup>

```

#### Attribute details

## **UploadIntervalInMinutes**

Interval in minutes at which data is uploaded to this destination.

ATTRIBUTE	VALUE
name	UploadIntervalInMinutes
type	xs:string

### **XML source**

```
<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="UploadIntervalInMinutes" type="xs:string">
<xs:annotation>
<xs:documentation>Interval in minutes at which data is uploaded to this destination.</xs:documentation>
</xs:annotation>
</xs:attribute>
```

## **VersionedItemAttrGroup attributeGroup**

Attribute group for versioning sections in ApplicationInstance and ServicePackage documents.

ATTRIBUTE	VALUE
content	1 attribute(s)
name	VersionedItemAttrGroup

### **XML source**

```
<xs:attributeGroup xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.microsoft.com/2011/01/fabric" name="VersionedItemAttrGroup">
<xs:annotation>
<xs:documentation>Attribute group for versioning sections in ApplicationInstance and ServicePackage
documents.</xs:documentation>
</xs:annotation>
<xs:attribute name="RolloutVersion" type="xs:string" use="required"/>
</xs:attributeGroup>
```

## **Attribute details**

### **RolloutVersion**

ATTRIBUTE	VALUE
name	RolloutVersion
type	xs:string
use	required

### **XML source**

```
<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="RolloutVersion" type="xs:string" use="required"/>
```

## VersionedName attributeGroup

Attribute group that includes a Name and a Version.

ATTRIBUTE	VALUE
content	2 attribute(s)
name	VersionedName

### XML source

```
<xs:attributeGroup xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.microsoft.com/2011/01/fabric" name="VersionedName">
  <xs:annotation>
    <xs:documentation>Attribute group that includes a Name and a Version.</xs:documentation>
  </xs:annotation>
  <xs:attribute name="Name" use="required">
    <xs:annotation>
      <xs:documentation>Name of the versioned item.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="Version" use="required">
    <xs:annotation>
      <xs:documentation>Version of the versioned item, an unstructured string.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:attributeGroup>
```

### Attribute details

#### Name

Name of the versioned item.

ATTRIBUTE	VALUE
name	Name
use	required

[XML source](#)

```

<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Name" use="required">
  <xs:annotation>
    <xs:documentation>Name of the versioned item.</xs:documentation>
  </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>

```

## Version

Version of the versioned item, an unstructured string.

ATTRIBUTE	VALUE
name	Version
use	required

## XML source

```

<xs:attribute xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/2011/01/fabric"
name="Version" use="required">
  <xs:annotation>
    <xs:documentation>Version of the versioned item, an unstructured string.</xs:documentation>
  </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>

```

# Commonly asked Service Fabric questions

8/18/2017 • 10 min to read • [Edit Online](#)

There are many commonly asked questions about what Service Fabric can do and how it should be used. This document covers many of those common questions and their answers.

## Cluster setup and management

### **Can I create a cluster that spans multiple Azure regions or my own datacenters?**

Yes.

The core Service Fabric clustering technology can be used to combine machines running anywhere in the world, so long as they have network connectivity to each other. However, building and running such a cluster can be complicated.

If you are interested in this scenario, we encourage you to get in contact either through the [Service Fabric Github Issues List](#) or through your support representative in order to obtain additional guidance. The Service Fabric team is working to provide additional clarity, guidance, and recommendations for this scenario.

Some things to consider:

1. The Service Fabric cluster resource in Azure is regional today, as are the virtual machine scale sets that the cluster is built on. This means that in the event of a regional failure you may lose the ability to manage the cluster via the Azure Resource Manager or the Azure Portal. This can happen even though the cluster remains running and you'd be able to interact with it directly. In addition, Azure today does not offer the ability to have a single virtual network that is usable across regions. This means that a multi-region cluster in Azure requires either [Public IP Addresses for each VM in the VM Scale Sets](#) or [Azure VPN Gateways](#). These networking choices have different impacts on costs, performance, and to some degree application design, so careful analysis and planning is required before standing up such an environment.
2. The maintenance, management, and monitoring of these machines can become complicated, especially when spanned across *types* of environments, such as between different cloud providers or between on-premises resources and Azure. Care must be taken to ensure that upgrades, monitoring, management, and diagnostics are understood for both the cluster and the applications before running production workloads in such an environment. If you already have lots of experience solving these problems in Azure or within your own datacenters, then it is likely that those same solutions can be applied when building out or running your Service Fabric cluster.

### **Do Service Fabric nodes automatically receive OS updates?**

Not today, but this is also a common request that Azure intends to deliver.

In the interim, we have [provided an application](#) that the operating systems underneath your Service Fabric nodes stay patched and up to date.

The challenge with OS updates is that they typically require a reboot of the machine, which results in temporary availability loss. By itself, that is not a problem, since Service Fabric will automatically redirect traffic for those services to other nodes. However, if OS updates are not coordinated across the cluster, there is the potential that many nodes go down at once. Such simultaneous reboots can cause complete availability loss for a service, or at least for a specific partition (for a stateful service).

In the future, we plan to support an OS update policy that is fully automated and coordinated across update domains, ensuring that availability is maintained despite reboots and other unexpected failures.

## **Can I use Large Virtual Machine Scale Sets in my SF cluster?**

**Short answer** - No.

**Long Answer** - Although the Large Virtual Machine Scale Sets allow you to scale a virtual machine scale set upto 1000 VM instances, it does so by the use of Placement Groups (PGs). Fault domains (FDs) and upgrade domains (UDs) are only consistent within a placement group Service fabric uses FDs and UDs to make placement decisions of your service replicas/Service instances. Since the FDs and UDs are comparable only within a placement group SF cannot use it. For example, If VM1 in PG1 has a topology of FD=0 and VM9 in PG2 has a topology of FD=4 , it does not mean that VM1 and VM2 are on two different Hardware Racks, hence SF cannot use the FD values in this case to make placement decisions.

There are other issues with Large virtual machine scale sets currently, like the lack of level-4 Load balancing support. Refer to for [details on Large scale sets](#)

## **What is the minimum size of a Service Fabric cluster? Why can't it be smaller?**

The minimum supported size for a Service Fabric cluster running production workloads is five nodes. For dev/test scenarios, we support three node clusters.

These minimums exist because the Service Fabric cluster runs a set of stateful system services, including the naming service and the failover manager. These services, which track what services have been deployed to the cluster and where they're currently hosted, depend on strong consistency. That strong consistency, in turn, depends on the ability to acquire a *quorum* for any given update to the state of those services, where a quorum represents a strict majority of the replicas ( $N/2 + 1$ ) for a given service.

With that background, let's examine some possible cluster configurations:

**One node:** this option does not provide high availability since the loss of the single node for any reason means the loss of the entire cluster.

**Two nodes:** a quorum for a service deployed across two nodes ( $N = 2$ ) is 2 ( $2/2 + 1 = 2$ ). When a single replica is lost, it is impossible to create a quorum. Since performing a service upgrade requires temporarily taking down a replica, this is not a useful configuration.

**Three nodes:** with three nodes ( $N=3$ ), the requirement to create a quorum is still two nodes ( $3/2 + 1 = 2$ ). This means that you can lose an individual node and still maintain quorum.

The three node cluster configuration is supported for dev/test because you can safely perform upgrades and survive individual node failures, as long as they don't happen simultaneously. For production workloads, you must be resilient to such a simultaneous failure, so five nodes are required.

## **Can I turn off my cluster at night/weekends to save costs?**

In general, no. Service Fabric stores state on local, ephemeral disks, meaning that if the virtual machine is moved to a different host, the data does not move with it. In normal operation, that is not a problem as the new node is brought up to date by other nodes. However, if you stop all nodes and restart them later, there is a significant possibility that most of the nodes start on new hosts and make the system unable to recover.

If you would like to create clusters for testing your application before it is deployed, we recommend that you dynamically create those clusters as part of your [continuous integration/continuous deployment pipeline](#).

## **How do I upgrade my Operating System (for example from Windows Server 2012 to Windows Server 2016)?**

While we're working on an improved experience, today, you are responsible for the upgrade. You must upgrade the OS image on the virtual machines of the cluster one VM at a time.

## **Container Support**

### **Why are my containers that are deployed to SF unable to resolve DNS addresses?**

This issue has been reported on clusters that are on 5.6.204.9494 version

**Mitigation** : Follow [this document](#) to enable the DNS service fabric service in your cluster.

**Fix** : Upgrade to a supported cluster version that is higher than 5.6.204.9494, when it is available. If your cluster is set to automatic upgrades, then the cluster will automatically upgrade to the version that has this issue fixed.

## Application Design

### What's the best way to query data across partitions of a Reliable Collection?

Reliable collections are typically [partitioned](#) to enable scale out for greater performance and throughput. That means that the state for a given service may be spread across 10s or 100s of machines. To perform operations over that full data set, you have a few options:

- Create a service that queries all partitions of another service to pull in the required data.
- Create a service that can receive data from all partitions of another service.
- Periodically push data from each service to an external store. This approach is only appropriate if the queries you're performing are not part of your core business logic.

### What's the best way to query data across my actors?

Actors are designed to be independent units of state and compute, so it is not recommended to perform broad queries of actor state at runtime. If you have a need to query across the full set of actor state, you should consider either:

- Replacing your actor services with stateful reliable services, such that the number of network requests to gather all data from the number of actors to the number of partitions in your service.
- Designing your actors to periodically push their state to an external store for easier querying. As above, this approach is only viable if the queries you're performing are not required for your runtime behavior.

### How much data can I store in a Reliable Collection?

Reliable services are typically partitioned, so the amount you can store is only limited by the number of machines you have in the cluster, and the amount of memory available on those machines.

As an example, suppose that you have a reliable collection in a service with 100 partitions and 3 replicas, storing objects that average 1kb in size. Now suppose that you have a 10 machine cluster with 16gb of memory per machine. For simplicity and to be very conservative, assume that the operating system and system services, the Service Fabric runtime, and your services consume 6gb of that, leaving 10gb available per machine, or 100gb for the cluster.

Keeping in mind that each object must be stored three times (one primary and two replicas), you would have sufficient memory for approximately 35 million objects in your collection when operating at full capacity. However, we recommend being resilient to the simultaneous loss of a failure domain and an upgrade domain, which represents about 1/3 of capacity, and would reduce the number to roughly 23 million.

Note that this calculation also assumes:

- That the distribution of data across the partitions is roughly uniform or that you're reporting load metrics to the Cluster Resource Manager. By default, Service Fabric will load balance based on replica count. In our example above, that would put 10 primary replicas and 20 secondary replicas on each node in the cluster. That works well for load that is evenly distributed across the partitions. If load is not even, you must report load so that the Resource Manager can pack smaller replicas together and allow larger replicas to consume more memory on an individual node.
- That the reliable service in question is the only one storing state in the cluster. Since you can deploy multiple services to a cluster, you need to be mindful of the resources that each will need to run and manage its state.

- That the cluster itself is not growing or shrinking. If you add more machines, Service Fabric will rebalance your replicas to leverage the additional capacity until the number of machines surpasses the number of partitions in your service, since an individual replica cannot span machines. By contrast, if you reduce the size of the cluster by removing machines, your replicas will be packed more tightly and have less overall capacity.

#### **How much data can I store in an actor?**

As with reliable services, the amount of data that you can store in an actor service is only limited by the total disk space and memory available across the nodes in your cluster. However, individual actors are most effective when they are used to encapsulate a small amount of state and associated business logic. As a general rule, an individual actor should have state that is measured in kilobytes.

## Other questions

#### **How does Service Fabric relate to containers?**

Containers offer a simple way to package services and their dependencies such that they run consistently in all environments and can operate in an isolated fashion on a single machine. Service Fabric offers a way to deploy and manage services, including [services that have been packaged in a container](#).

#### **Are you planning to open source Service Fabric?**

We intend to open source the reliable services and reliable actors frameworks on GitHub and will accept community contributions to those projects. Please follow the [Service Fabric blog](#) for more details as they're announced.

The are currently no plans to open source the Service Fabric runtime.

## Next steps

- [Learn about core Service Fabric concepts and best practices](#)

# Azure Service Fabric support options

10/13/2017 • 3 min to read • [Edit Online](#)

To deliver the appropriate support for your Service Fabric clusters that you are running your application work loads on, we have set up various options for you. Depending on the level of support needed and the severity of the issue, you get to pick the right options.

## Report production issues or request paid support for Azure

For reporting issues on your Service Fabric cluster deployed on Azure, open a ticket for support on [Azure portal](#) or [Microsoft support portal](#).

Learn more about:

- [Support from Microsoft for Azure](#).
- [Microsoft premier support](#).

## Report production issues or request paid support for standalone Service Fabric clusters

For reporting issues on your Service Fabric cluster deployed on-premises or on other clouds, open a ticket for professional support on [Microsoft support portal](#).

Learn more about:

- [Professional Support from Microsoft for on-premises](#).
- [Microsoft premier support](#).

## Report Azure Service Fabric issues

We have set up a GitHub repo for reporting Service Fabric issues. We are also actively monitoring the following forums.

### GitHub repo

Report Azure Service Fabric Issues on [Service-Fabric-issues git repo](#). This repo is intended for reporting and tracking issues with Azure Service Fabric and for making small feature requests. **Do not use this to report live-site issues.**

### StackOverflow and MSDN forums

The [Service Fabric tag on StackOverflow](#) and the [Service Fabric forum on MSDN](#) are best used for asking questions about how the platform works and how you might accomplish certain tasks with it.

### Azure Feedback forum

The [Azure Feedback Forum for Service Fabric](#) is the best place for submitting big feature ideas you have for the product as we review the most popular requests are part of our medium to long-term planning. We encourage you to rally support for your suggestions within the community.

## Supported Service Fabric versions.

Make sure that your cluster is always running a supported Service Fabric version. As and when we announce the release of a new version of Service Fabric, the previous version is marked for end of support after a minimum of

60 days from that date. The new releases are announced [on the Service Fabric team blog](#).

Refer to the following documents on details on how to keep your cluster running a supported Service Fabric version.

- [Upgrade Service Fabric version on an Azure cluster](#)
- [Upgrade Service Fabric version on a standalone windows server cluster](#)

Here are the list of the Service Fabric versions that are supported and their support end dates.

SERVICE FABRIC RUNTIME CLUSTER	COMPATIBLE SDK / NUGET PACKAGE VERSIONS	END OF SUPPORT DATE
All cluster versions prior to 5.3.121	Less than or equal to version 2.3	January 20, 2017
5.3.*	Less than or equal to version 2.3	February 24, 2017
5.4.*	Less than or equal to version 2.4	May 10,2017
5.5.*	Less than or equal to version 2.5	August 10,2017
5.6.*	Less than or equal to version 2.6	October 13,2017
5.7.*	Less than or equal to version 2.7	December 15,2017
6.0.*	Less than or equal to version 2.8	Current version and so no end date

## Service Fabric Preview Versions - unsupported for production use.

From time to time, we release versions that have significant features we want feedback on, which are released as previews. These preview versions should only be used for test purposes. Your production cluster should always be running a supported, stable, Service Fabric version. A preview version always begins with a major and minor version number of 255. For example, if you see a Service Fabric version 255.255.5703.949, that release version is only to be used in test clusters and is in preview. These preview releases are also announced on the [Service Fabric team blog](#) and will have details on the features included.

There is no paid support option for these preview releases. Use one of the options listed under [Report Azure Service Fabric issues](#) to ask questions or provide feedback.

## Next steps

- [Upgrade service fabric version on an Azure cluster](#)
- [Upgrade Service Fabric version on a standalone windows server cluster](#)