**WorkshopPLUS**

**Microsoft Azure Service Fabric for Developers**

Service Fabric Health Reporting and Events

Student Lab Manual

Instructor Edition (Book Title Hidden Style)

v1.6, March 15, 2017

**Conditions and Terms of Use**

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

© 2015 Microsoft Corporation. All rights reserved.

**Copyright and Trademarks**

© 2015 Microsoft Corporation. All rights reserved.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at  
<http://www.microsoft.com/en-us/legal/intellectualproperty/Permissions/default.aspx>

Azure, Microsoft, Internet Explorer, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Contents

[Service Fabric Health Reporting and Events 4](#_Toc477340852)

[Objective 4](#_Toc477340853)

[Prerequisites 4](#_Toc477340854)

[Exercise 1: Health reporting 5](#_Toc477340855)

[Task 1 – Application Deployment 5](#_Toc477340856)

[Task 2 - Check the health by using PowerShell. 8](#_Toc477340857)

[Task 3 – Add custom health events to your service code 9](#_Toc477340858)

[Exercise 2: Implementing trace events 12](#_Toc477340859)

[Task 1 - View Service Fabric system events in Visual Studio 13](#_Toc477340860)

[Task 2 - Add your own custom traces to the application code 14](#_Toc477340861)

# Service Fabric Health Reporting and Events

Estimated time to complete this lab

30 minutes

## Objective

This lab walks you through an example of adding a health report to a service, and shows how the health status can be checked using the tools that Service Fabric provides. This lab is intended to be a quick introduction to the health monitoring capabilities in Service Fabric.

## Prerequisites

The following is required to complete this hands-on lab:

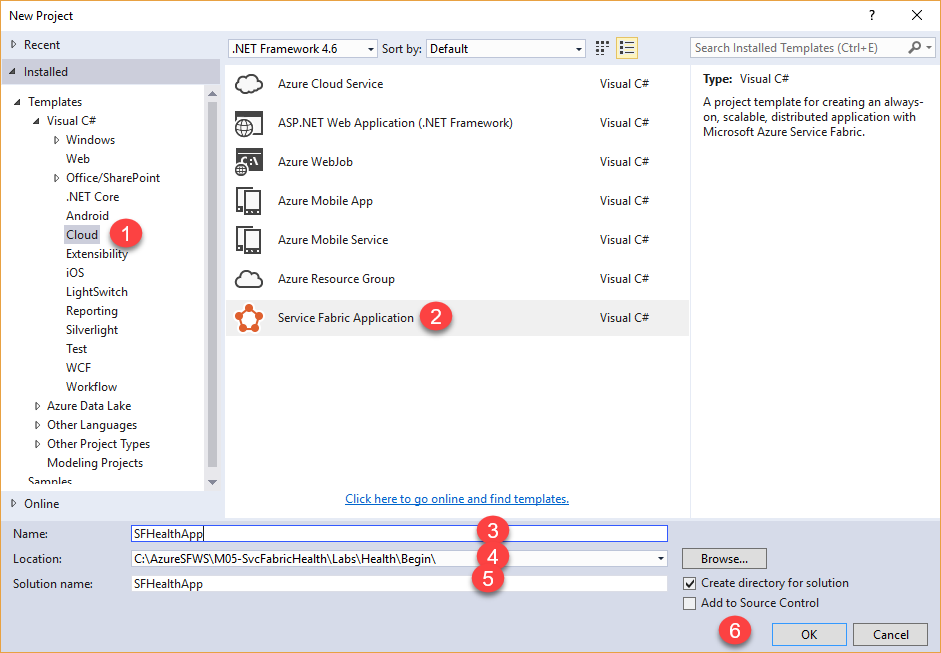
* Microsoft V[isual Studio 2017 Professional or Enterprise edition](http://www.microsoft.com/visualstudio/)
* [Microsoft Azure SDK for .NET for Visual Studio 2017](http://www.microsoft.com/windowsazure/sdk/)
* Microsoft Azure Service Fabric SDK – 2.4.164
* Microsoft Azure PowerShell
* A Microsoft Azure subscription

# Exercise 1: Health reporting

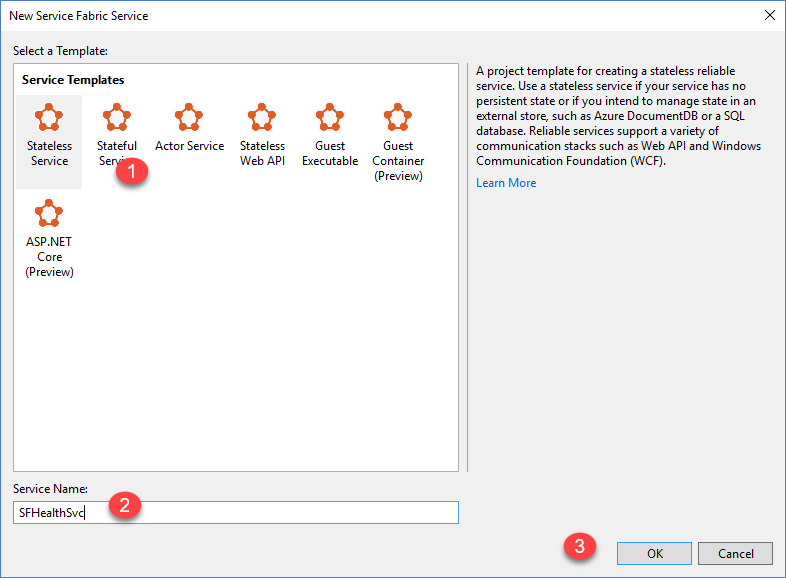
## Task 1 – Application Deployment

To deploy an application and check its health, follow these steps:

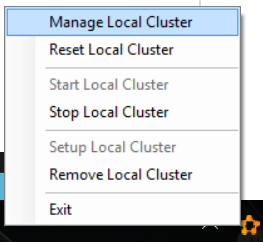
1. Launch Visual Studio as an administrator.
2. Select **File | New | Project |** **Visual** **C# | Cloud | Service Fabric Application** and create a service fabric application. Give the application the name **SFHealthApp** (or any name you choose) and place it in the **.\Labs\Health\Begin** folder. Select the **OK** button.



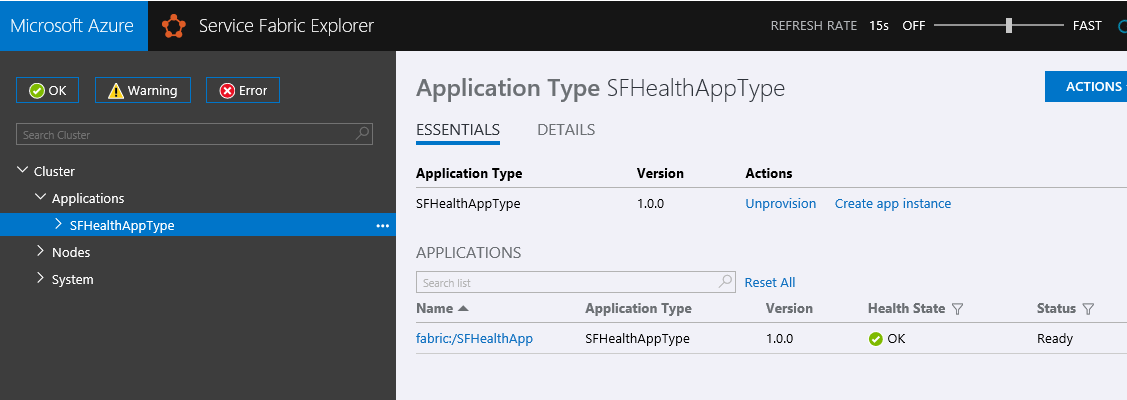
1. Choose the **Stateful Service** icon and give the service the name **SFHealthSvc**. Select the **OK** button.



1. Press **F5** to run the application in debug mode. The application will be deployed to the local cluster.
2. Once the application is running, launch Service Fabric Explorer by right-clicking the **Local Cluster Manager** in the notification area, and choose **Manage Local Cluster** from the context menu.



1. The application health should be displayed as in the image below. At this time, the application should be healthy with no errors. Leave the application running in Visual Studio.



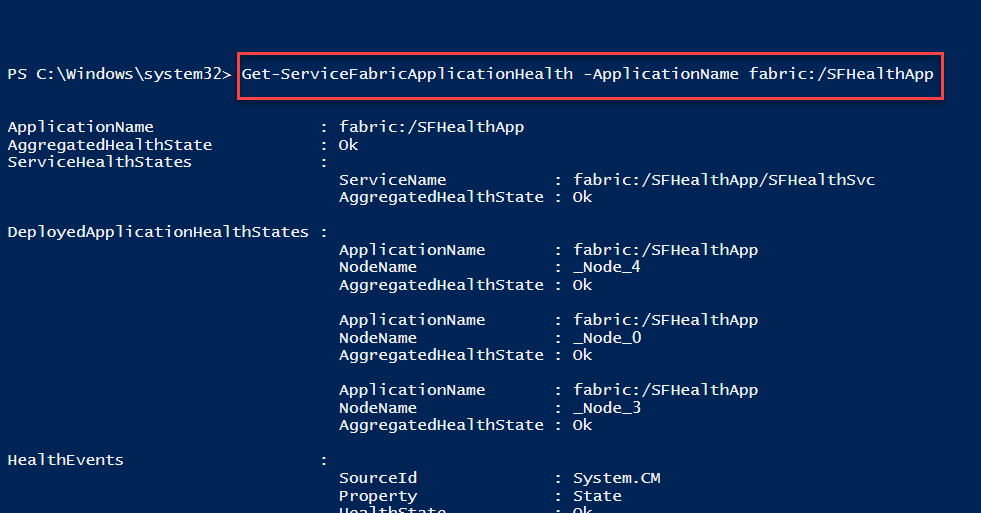
## Task 2 - Check the health by using PowerShell.

1. Open a Microsoft Azure PowerShell command prompt or use the PowerShell command prompt inside of Windows PowerShell ISE.
2. First connect to the local cluster with the Connect-ServiceFabricCluster commandlet.
3. Then check the application's health using **Get-ServiceFabricApplicationHealth** with the *-ApplicationName* parameter. It would look something like this:  
     
   Get-ServiceFabricApplicationHealth -ApplicationName fabric:/SFHealthApp

You can also check the health of a specific service by using **Get-ServiceFabricServiceHealth**. That command would look like this:

Get-ServiceFabricServiceHealth -ServiceName fabric:/SFHealthApp/SFHealthSvc

The health report for the application in PowerShell looks like this.



1. Shut down your running Visual Studio project.

## Task 3 – Add custom health events to your service code

The Service Fabric Visual Studio project templates contain sample code. The steps below show how you can report custom health events from your service code. Such reports will automatically show up in the standard tools for health monitoring that Service Fabric provides, such as Service Fabric Explorer, Azure portal health view, and PowerShell.

The name of the .cs files below will depend on what you named your service at the beginning of the project.

1. Open the **SFHealthSvc.cs** file in Visual Studio. Find the declaration for **var myDictionary** in the RunAsync method and add the following code below the line of code that declares the *myDictionary* variable. The **fabricClient** object created here will be used later to report health.

var fabricClient = new FabricClient(new FabricClientSettings() { HealthReportSendInterval = TimeSpan.FromSeconds(0) });

1. Next, look up the call **myDictionary.TryGetValueAsync** in the *RunAsync* method. You can see this returns a **result** that holds the current value of the counter, since the key logic in this application is to keep a count running. If this was a real application, and if the lack of result represented a failure, then you would want to report that to the health manager.
2. To report a health event for the lack of result representing a failure, add the code shown in steps 4 and 5 below after the **myDictionary.TryGetValueAsync** call. The event is reported as a **StatefulServiceReplicaHealthReport** since it's being reported from a stateful service. The PartitionId and ReplicaId that are passed in to the report event will help identify the source of this report when you see it in one of the health monitoring tools. That's important because a deployed stateful service can have multiple partitions and each partition can have multiple replicas. The **HealthInformation** parameter stores information about the health issue being reported.
3. Add this namespace to the **SFHealthSvc.cs** file.

using System.Fabric.Health;

1. 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);

}

1. Let's simulate this failure and view it being displayed in the health monitoring tools. To simulate the failure, comment out the first line in the health reporting code you added above. After you comment out the first line, the code will look as shown below. This will now trigger this health report each time RunAsync executes. After making the change run the application using **F5**.

//if(!result.HasValue)

{

var replicaHealthReport = new StatefulServiceReplicaHealthReport(

this.Context.PartitionId,

this.Context.ReplicaId,

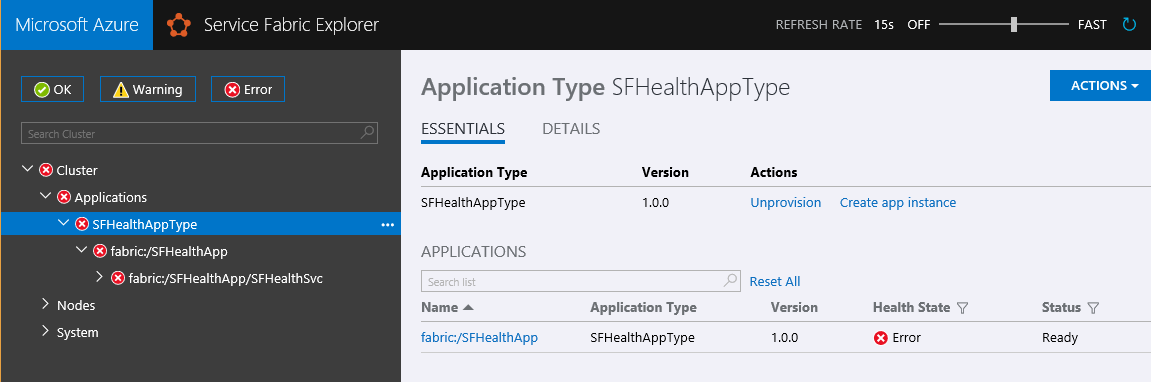
new HealthInformation("ServiceCode", "StateDictionary", HealthState.Error));

var fabricClient = new FabricClient(new FabricClientSettings() { HealthReportSendInterval = TimeSpan.FromSeconds(0) });

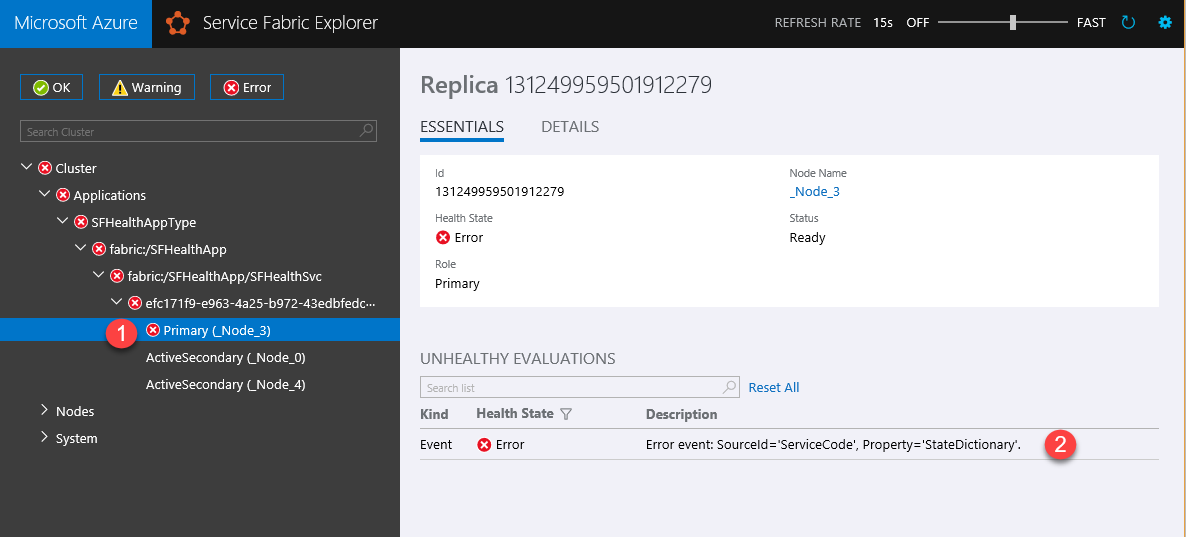
fabricClient.HealthManager.ReportHealth(replicaHealthReport);

}

1. Once the application is running, open Service Fabric Explorer to check health of the application. This time, the Service Fabric Explorer will show the application to be unhealthy. This is because of the error that was reported from the code that we added above.



1. If you select the primary replica in the tree view of Service Fabric Explorer, you will see that it shows the health in an error state. It also displays the health report details that were added to the HealthInformation parameter in the code. You can see the same health reports in Azure PowerShell also as well as the Azure portal.



This report will remain in the health manager until it is replaced by another report or this replica is deleted. Since we did not set a **TimeToLive** for this health report in the HealthInformation object, it will never expire.

1. Shut down the running Visual Studio application and uncomment the line of code you had commented out before.

# Exercise 2: Implementing trace events

Monitoring, detecting, diagnosing, and troubleshooting allow for services to continue operation with minimal disruptions. While monitoring and diagnostics are critical in an actual deployed production environment, the efficacy 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.

**The benefits of Event Tracing for Windows**

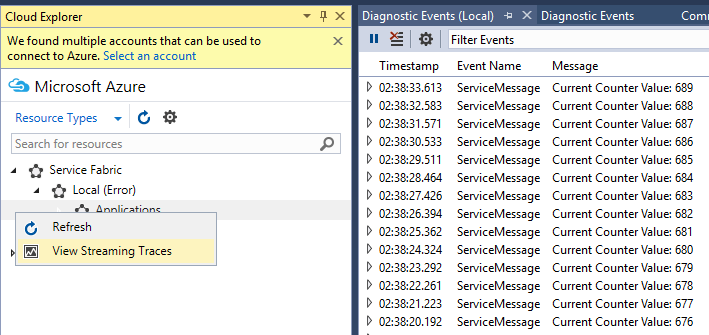
[Event Tracing for Windows](https://msdn.microsoft.com/library/windows/desktop/bb968803.aspx) (ETW) is the recommended technology for tracing messages in Service Fabric. Reasons for this are:

* **ETW is fast -** It was built as a tracing technology that has minimal impact on code execution times.
* **ETW tracing works 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 inter-relationships 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.

## Task 1 - View Service Fabric system events in Visual Studio

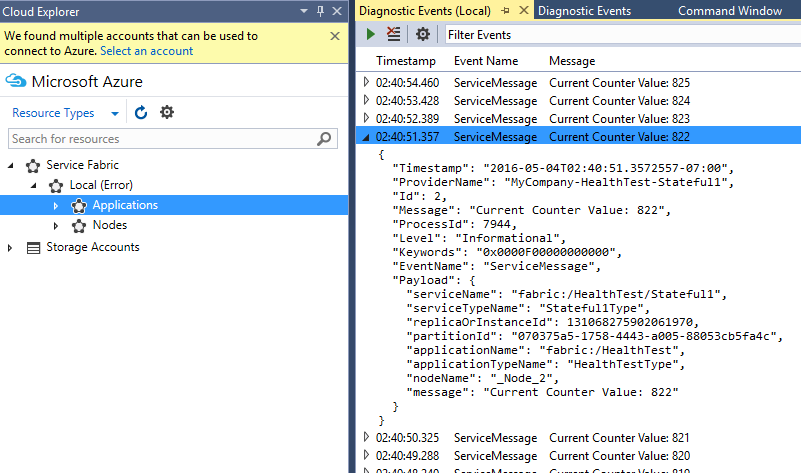
Service Fabric emits ETW events to help application developers understand what's happening in the platform. Use the application you created in Exercise 1 for the following steps.

1. If the diagnostics events window does not automatically show, go to the **Cloud Explorer** tab in Visual Studio, right-click ‘Applications’ at **Service Fabric | Local | Applications,** and choose **View Streaming Traces** in the context menu.



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.

Notice that the health reporting instrumentation you created in the first exercise does not show up in the application trace data. **Application** **trace data** is designed to aid developers in identifying the root cause and performance problems. **Health check data** allows everyone involved in the running of the cluster or application to have an indication of its health. These two mechanisms can be used together to allow for surfacing of the data and to identify the root cause and fix problems.



## Task 2 - 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.

In the sample project you created, you can view the line of code (provided by the template) which is reporting the current counter value. The code is in the RunAsync function and looks like this:

ServiceEventSource.Current.ServiceMessage(this, "Current Counter Value: {0}",

result.HasValue ? result.Value.ToString() : "Value does not exist.");

1. Right-click on **ServiceMessage** and choose the “*Go To Definition*” option. Notice that the ServiceMessage function is defined in the template code within the **ServiceEventSource.cs** file.

Service message takes three parameters, the last two are a string, and an array of arguments that together are passed to the format() function to allow the insertion of multiple dynamic values into the output.

In the **ServiceEventSource.cs** file, you will also find an overload for the **ServiceEventSource.ServiceMessage** method that should be used for high-frequency events due to performance reasons:

void ServiceMessage(

string serviceName,

string serviceTypeName,

long replicaOrInstanceId,

Guid partitionId,

string applicationName,

string applicationTypeName,

string nodeName,

string message)

{

This overload allows for more static state look up to happen initially and be reused for each call.

Also notice where the code for the class begins, there is a declaration / attribute defining the “EventSource” name (at the very top of the class):

[EventSource(Name = "MyCompany-SFHealthApp-SFHealthSvc”)]

internal sealed class ServiceEventSource : EventSource

1. Change the EventSource attribute described above to be prefixed by a company name of your choice.
2. Now add another event underneath the custom event defined in **SFHealthSvc.cs RunAsync**():

ServiceEventSource.Current.ServiceMessage(this.Context, "The current time is: {0}", DateTime.Now.ToString());

You can choose to report whatever you want here.

Switching back to **ServiceEventSource.cs**, notice the attributes defined above each event reporting function. Some are defined as [NonEvent], while others report back error conditions, such as:

private const int ServiceHostInitializationFailedEventId = 4;

**[Event(ServiceHostInitializationFailedEventId, Level = EventLevel.Error, Message = "Service host initialization failed", Keywords = Keywords.ServiceInitialization)]**

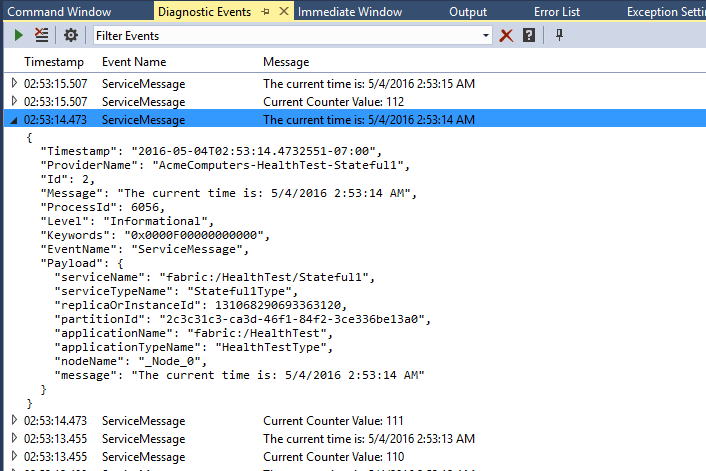
public void ServiceHostInitializationFailed(string exception)

{

WriteEvent(ServiceHostInitializationFailedEventId, exception);

}

1. Build, deploy, and run the application again to see your event(s) in the Visual Studio Diagnostic Events Viewer. If you debug the application with **F5**, the Diagnostic Events Viewer will open automatically.



In a real application these events could be aggregated using the diagnostics extension and then viewed and analyzed through Operational Insights or other tools as described [here](https://azure.microsoft.com/en-us/documentation/articles/service-fabric-diagnostics-how-to-setup-wad-operational-insights/).