

Contents

Azure Kubernetes Service (AKS)

Overview

[About AKS](#)

[Solution architectures](#)

Quickstarts

[Deploy an AKS Cluster](#)

[Use the Azure CLI](#)

[Use Azure PowerShell](#)

[Use the Azure portal](#)

[Use ARM template](#)

[Develop applications](#)

[Develop with Helm](#)

Tutorials

[1 - Prepare application for AKS](#)

[2 - Create container registry](#)

[3 - Create Kubernetes cluster](#)

[4 - Run application](#)

[5 - Scale application](#)

[6 - Update application](#)

[7 - Upgrade cluster](#)

Concepts

[Clusters and workloads](#)

[Access and identity](#)

[Security](#)

[Security Baseline](#)

[Security for applications and clusters](#)

[Security controls by Azure Policy](#)

[Networking](#)

[Storage](#)

Scale

Node auto-repair

Best practices

Overview

Baseline architecture for an AKS cluster

For cluster operators

Multi-tenancy and cluster isolation

Basic scheduler features

Advanced scheduler features

Authentication and authorization

Cluster security

Container image management

Networking

Storage

Business continuity (BC) and disaster recovery (DR)

For application developers

Resource management

Pod security

Quotas and regional limits

Migrate to AKS

Plan and execute a migration

Spring Boot

Tomcat

Wildfly

WebLogic

WebSphere

JBoss EAP

Supported Kubernetes version

Security Hardening in host OS

Azure Kubernetes Service Diagnostics overview

Sustainable software engineering

How-to guides

Cluster operations

- [Scale an AKS cluster](#)
- [Stop an AKS cluster](#)
- [Use planned maintenance \(preview\)](#)
- [Upgrade an AKS cluster](#)
- [Use Uptime SLA](#)
- [Use proximity placement groups](#)
- [Upgrade the node image](#)
- [Upgrade the node image automatically with GitHub Actions](#)
- [Process node OS updates](#)
- [Configure an AKS cluster](#)
- [Custom node configuration \(preview\)](#)
- [Integrate ACR with an AKS cluster](#)
- [Create virtual nodes](#)
 - [Use virtual nodes](#)
 - [Use the Azure CLI](#)
 - [Use the Azure portal](#)
- [Use Cluster Autoscaler](#)
- [Use Availability Zones](#)
- [Use node pools](#)
 - [Use multiple node pools](#)
 - [Use spot node pools](#)
 - [Use system node pools](#)
- [Deploy AKS with Terraform](#)
- [Azure portal Kubernetes resource view](#)
- [Use the Kubernetes dashboard](#)
- [Configure data volumes](#)
 - [Azure Disk - Dynamic](#)
 - [Azure Disk - Static](#)
 - [Azure Files - Dynamic](#)
 - [Azure Files - Static](#)
 - [NFS Server - Static](#)
 - [Azure NetApp Files](#)

[Use Azure Ultra Disks \(preview\)](#)

[CSI Storage Drivers](#)

[Enable CSI Storage Drivers \(preview\)](#)

[Azure Disk CSI drivers \(preview\)](#)

[Azure Files CSI drivers \(preview\)](#)

[Configure networking](#)

[Create or use existing virtual network](#)

[Use kubenet](#)

[Use Azure-CNI](#)

[Create an internal load balancer](#)

[Use a Standard Load Balancer](#)

[Use a static IP address](#)

[Ingress](#)

[Create a basic controller](#)

[Use HTTP application routing](#)

[Use internal network](#)

[Enable the AGIC add-on for an existing AKS cluster](#)

[Use TLS with your own certificates](#)

[Use TLS with Let's Encrypt](#)

[Use a dynamic public IP address](#)

[Use a static public IP address](#)

[Egress](#)

[Restrict and control cluster egress traffic](#)

[Use a user defined route for egress](#)

[Use a basic load balancer and static IP address](#)

[Customize CoreDNS](#)

[Security and authentication](#)

[Create service principal](#)

[Use managed identities](#)

[Use AAD pod identity \(preview\)](#)

[Limit access to cluster configuration file](#)

[Secure pod traffic with network policies](#)

[Use Azure Policy](#)

[Use pod security policies \(preview\)](#)

[Define API server authorized IP ranges](#)

[Control deployments with Azure Policy](#)

[Update cluster credentials](#)

[Enable Azure Active Directory integration](#)

[AKS-managed Azure AD](#)

[Azure AD integration \(legacy\)](#)

[Use Azure RBAC for Kubernetes Authorization](#)

[Use Kubernetes RBAC with Azure AD integration](#)

[Rotate certificates](#)

[Create a private cluster](#)

[BYOK for disks](#)

[Enable host-based encryption](#)

[CSI Secrets Store driver \(preview\)](#)

[Monitoring and logging](#)

[View cluster metrics](#)

[Azure Monitor for containers](#)

[View the control plane component logs](#)

[View the kubelet logs](#)

[View container data real-time](#)

[Use Windows Server containers](#)

[Create an AKS cluster](#)

[Create an AKS cluster with PowerShell](#)

[Connect remotely](#)

[Windows Server containers FAQ](#)

[Use the Kubernetes dashboard](#)

[Create Dockerfiles for Windows Server containers](#)

[Optimize Dockerfiles for Windows Server containers](#)

[Develop and run applications](#)

[Use Bridge to Kubernetes with Visual Studio Code](#)

[Use Bridge to Kubernetes with Visual Studio](#)

[Install existing applications with Helm](#)

[Use OpenFaaS](#)

[Run Spark jobs](#)

[Use GPUs](#)

[Build app with PostgreSQL](#)

[Build app with MySQL](#)

[Build app with Open Liberty or WebSphere Liberty](#)

[Use Azure API Management](#)

[Select and deploy a service mesh](#)

[About Service Meshes](#)

[Open Service Mesh AKS add-on \(preview\)](#)

[Use Istio](#)

[About Istio](#)

[Install and configure](#)

[Scenario - Intelligent routing and canary releases](#)

[Use Linkerd](#)

[About Linkerd](#)

[Install and configure](#)

[Use Consul](#)

[About Consul](#)

[Install and configure](#)

[DevOps](#)

[Use Ansible to create AKS clusters](#)

[Jenkins continuous deployment](#)

[Azure DevOps Project](#)

[Deployment Center Launcher](#)

[GitHub Actions for Kubernetes](#)

[Troubleshoot](#)

[Common issues](#)

[Checking for best practices](#)

[SSH node access](#)

[Linux performance tools](#)

[Check for Resource Health events \(preview\)](#)

Reference

[Azure CLI](#)

[REST](#)

[PowerShell](#)

[.NET](#)

[Python](#)

[Java](#)

[Node.js](#)

[Resource Manager template](#)

[Azure Policy built-ins](#)

Resources

[Build your skill with Microsoft Learn](#)

[Region availability](#)

[Pricing](#)

[Support policies](#)

[Azure Roadmap](#)

[AKS Roadmap](#)

[Stack Overflow](#)

[Videos](#)

[FAQ](#)

Azure Kubernetes Service

5/14/2021 • 5 minutes to read • [Edit Online](#)

Azure Kubernetes Service (AKS) simplifies deploying a managed Kubernetes cluster in Azure by offloading the operational overhead to Azure. As a hosted Kubernetes service, Azure handles critical tasks, like health monitoring and maintenance. Since Kubernetes masters are managed by Azure, you only manage and maintain the agent nodes. Thus, AKS is free; you only pay for the agent nodes within your clusters, not for the masters.

You can create an AKS cluster using:

- [The Azure CLI](#)
- [The Azure portal](#)
- [Azure PowerShell](#)
- Using template-driven deployment options, like [Azure Resource Manager templates](#) and Terraform

When you deploy an AKS cluster, the Kubernetes master and all nodes are deployed and configured for you. Advanced networking, Azure Active Directory (Azure AD) integration, monitoring, and other features can be configured during the deployment process.

For more information on Kubernetes basics, see [Kubernetes core concepts for AKS](#).

NOTE

This service supports [Azure Lighthouse](#), which lets service providers sign in to their own tenant to manage subscriptions and resource groups that customers have delegated.

AKS also supports Windows Server containers.

Access, security, and monitoring

For improved security and management, AKS lets you integrate with Azure AD to:

- Use Kubernetes role-based access control (Kubernetes RBAC).
- Monitor the health of your cluster and resources.

Identity and security management

Kubernetes RBAC

To limit access to cluster resources, AKS supports [Kubernetes RBAC](#). Kubernetes RBAC controls access and permissions to Kubernetes resources and namespaces.

Azure AD

You can configure an AKS cluster to integrate with Azure AD. With Azure AD integration, you can set up Kubernetes access based on existing identity and group membership. Your existing Azure AD users and groups can be provided with an integrated sign-on experience and access to AKS resources.

For more information on identity, see [Access and identity options for AKS](#).

To secure your AKS clusters, see [Integrate Azure Active Directory with AKS](#).

Integrated logging and monitoring

Azure Monitor for Container Health collects memory and processor performance metrics from containers, nodes, and controllers within your AKS cluster and deployed applications. You can review both container logs

and [the Kubernetes master logs](#), which are:

- Stored in an Azure Log Analytics workspace.
- Available through the Azure portal, Azure CLI, or a REST endpoint.

For more information, see [Monitor Azure Kubernetes Service container health](#).

Clusters and nodes

AKS nodes run on Azure virtual machines (VMs). With AKS nodes, you can connect storage to nodes and pods, upgrade cluster components, and use GPUs. AKS supports Kubernetes clusters that run multiple node pools to support mixed operating systems and Windows Server containers.

For more information about Kubernetes cluster, node, and node pool capabilities, see [Kubernetes core concepts for AKS](#).

Cluster node and pod scaling

As demand for resources change, the number of cluster nodes or pods that run your services automatically scales up or down. You can adjust both the horizontal pod autoscaler or the cluster autoscaler to adjust to demands and only run necessary resources.

For more information, see [Scale an Azure Kubernetes Service \(AKS\) cluster](#).

Cluster node upgrades

AKS offers multiple Kubernetes versions. As new versions become available in AKS, you can upgrade your cluster using the Azure portal or Azure CLI. During the upgrade process, nodes are carefully cordoned and drained to minimize disruption to running applications.

To learn more about lifecycle versions, see [Supported Kubernetes versions in AKS](#). For steps on how to upgrade, see [Upgrade an Azure Kubernetes Service \(AKS\) cluster](#).

GPU-enabled nodes

AKS supports the creation of GPU-enabled node pools. Azure currently provides single or multiple GPU-enabled VMs. GPU-enabled VMs are designed for compute-intensive, graphics-intensive, and visualization workloads.

For more information, see [Using GPUs on AKS](#).

Confidential computing nodes (public preview)

AKS supports the creation of Intel SGX-based, confidential computing node pools (DCSv2 VMs). Confidential computing nodes allow containers to run in a hardware-based, trusted execution environment (enclaves). Isolation between containers, combined with code integrity through attestation, can help with your defense-in-depth container security strategy. Confidential computing nodes support both confidential containers (existing Docker apps) and enclave-aware containers.

For more information, see [Confidential computing nodes on AKS](#).

Storage volume support

To support application workloads, you can mount static or dynamic storage volumes for persistent data. Depending on the number of connected pods expected to share the storage volumes, you can use storage backed by either:

- Azure Disks for single pod access, or
- Azure Files for multiple, concurrent pod access.

For more information, see [Storage options for applications in AKS](#).

Get started with dynamic persistent volumes using [Azure Disks](#) or [Azure Files](#).

Virtual networks and ingress

An AKS cluster can be deployed into an existing virtual network. In this configuration, every pod in the cluster is assigned an IP address in the virtual network, and can directly communicate with:

- Other pods in the cluster
- Other nodes in the virtual network.

Pods can also connect to other services in a peered virtual network and to on-premises networks over ExpressRoute or site-to-site (S2S) VPN connections.

For more information, see the [Network concepts for applications in AKS](#).

Ingress with HTTP application routing

The HTTP application routing add-on helps you easily access applications deployed to your AKS cluster. When enabled, the HTTP application routing solution configures an ingress controller in your AKS cluster.

As applications are deployed, publicly accessible DNS names are autoconfigured. The HTTP application routing sets up a DNS zone and integrates it with the AKS cluster. You can then deploy Kubernetes ingress resources as normal.

To get started with ingress traffic, see [HTTP application routing](#).

Development tooling integration

Kubernetes has a rich ecosystem of development and management tools that work seamlessly with AKS. These tools include Helm and the Kubernetes extension for Visual Studio Code.

Azure provides several tools that help streamline Kubernetes, such as DevOps Starter.

DevOps Starter

DevOps Starter provides a simple solution for bringing existing code and Git repositories into Azure. DevOps Starter automatically:

- Creates Azure resources (such as AKS);
- Configures a release pipeline in Azure DevOps Services that includes a build pipeline for CI;
- Sets up a release pipeline for CD; and,
- Generates an Azure Application Insights resource for monitoring.

For more information, see [DevOps Starter](#).

Docker image support and private container registry

AKS supports the Docker image format. For private storage of your Docker images, you can integrate AKS with Azure Container Registry (ACR).

To create a private image store, see [Azure Container Registry](#).

Kubernetes certification

AKS has been CNCF-certified as Kubernetes conformant.

Regulatory compliance

AKS is compliant with SOC, ISO, PCI DSS, and HIPAA. For more information, see [Overview of Microsoft Azure compliance](#).

Next steps

Learn more about deploying and managing AKS with the Azure CLI Quickstart.

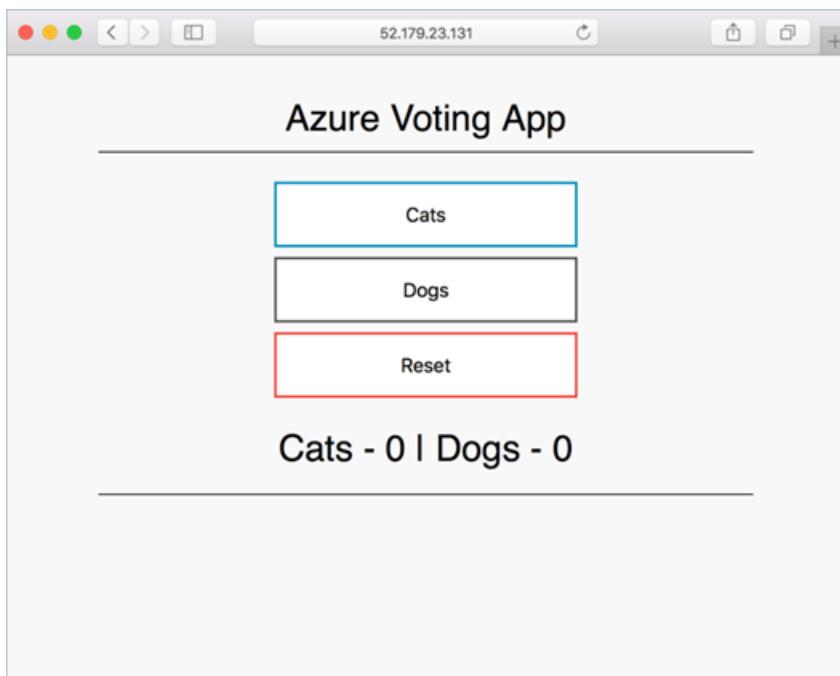
[Deploy an AKS Cluster using Azure CLI](#)

Quickstart: Deploy an Azure Kubernetes Service cluster using the Azure CLI

3/31/2021 • 6 minutes to read • [Edit Online](#)

Azure Kubernetes Service (AKS) is a managed Kubernetes service that lets you quickly deploy and manage clusters. In this quickstart, you will:

- Deploy an AKS cluster using the Azure CLI.
- Run a multi-container application with a web front-end and a Redis instance in the cluster.
- Monitor the health of the cluster and pods that run your application.



This quickstart assumes a basic understanding of Kubernetes concepts. For more information, see [Kubernetes core concepts for Azure Kubernetes Service \(AKS\)](#).

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

To learn more about creating a Windows Server node pool, see [Create an AKS cluster that supports Windows Server containers](#).

Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#).

 [Launch Cloud Shell](#)

- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
 - If you're using a local installation, sign in to the Azure CLI by using the `az login` command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
 - When you're prompted, install Azure CLI extensions on first use. For more information about

extensions, see [Use extensions with the Azure CLI](#).

- Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.
- This article requires version 2.0.64 or greater of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

NOTE

Run the commands as administrator if you plan to run the commands in this quickstart locally instead of in Azure Cloud Shell.

Create a resource group

An [Azure resource group](#) is a logical group in which Azure resources are deployed and managed. When you create a resource group, you will be prompted to specify a location. This location is:

- The storage location of your resource group metadata.
- Where your resources will run in Azure if you don't specify another region during resource creation.

The following example creates a resource group named *myResourceGroup* in the *eastus* location.

Create a resource group using the `az group create` command.

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

Output for successfully created resource group:

```
{
  "id": "/subscriptions/<guid>/resourceGroups/myResourceGroup",
  "location": "eastus",
  "managedBy": null,
  "name": "myResourceGroup",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null
}
```

Enable cluster monitoring

1. Verify *Microsoft.OperationsManagement* and *Microsoft OperationalInsights* are registered on your subscription. To check the registration status:

```
az provider show -n Microsoft.OperationsManagement -o table
az provider show -n Microsoft.OperationalInsights -o table
```

If they are not registered, register *Microsoft.OperationsManagement* and *Microsoft OperationalInsights* using:

```
az provider register --namespace Microsoft.OperationsManagement
az provider register --namespace Microsoft.OperationalInsights
```

2. Enable Azure Monitor for containers using the `--enable-addons monitoring` parameter.

Create AKS cluster

Create an AKS cluster using the `az aks create` command. The following example creates a cluster named `myAKSCluster` with one node:

```
az aks create --resource-group myResourceGroup --name myAKSCluster --node-count 1 --enable-addons monitoring  
--generate-ssh-keys
```

After a few minutes, the command completes and returns JSON-formatted information about the cluster.

NOTE

When you create an AKS cluster, a second resource group is automatically created to store the AKS resources. For more information, see [Why are two resource groups created with AKS?](#)

Connect to the cluster

To manage a Kubernetes cluster, use the Kubernetes command-line client, `kubectl`. `kubectl` is already installed if you use Azure Cloud Shell.

1. Install `kubectl` locally using the `az aks install-cli` command:

```
az aks install-cli
```

2. Configure `kubectl` to connect to your Kubernetes cluster using the `az aks get-credentials` command. The following command:

- Downloads credentials and configures the Kubernetes CLI to use them.
- Uses `~/.kube/config`, the default location for the [Kubernetes configuration file](#). Specify a different location for your Kubernetes configuration file using `--file`.

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```

3. Verify the connection to your cluster using the `kubectl get` command. This command returns a list of the cluster nodes.

```
kubectl get nodes
```

Output shows the single node created in the previous steps. Make sure the node status is *Ready*:

NAME	STATUS	ROLES	AGE	VERSION
aks-nodepool1-31718369-0	Ready	agent	6m44s	v1.12.8

Run the application

A [Kubernetes manifest file](#) defines a cluster's desired state, such as which container images to run.

In this quickstart, you will use a manifest to create all objects needed to run the [Azure Vote application](#). This manifest includes two [Kubernetes deployments](#):

- The sample Azure Vote Python applications.
- A Redis instance.

Two [Kubernetes Services](#) are also created:

- An internal service for the Redis instance.
- An external service to access the Azure Vote application from the internet.

1. Create a file named `azure-vote.yaml`.

- If you use the Azure Cloud Shell, this file can be created using `code`, `vi`, or `nano` as if working on a virtual or physical system

2. Copy in the following YAML definition:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
  template:
    metadata:
      labels:
        app: azure-vote-back
    spec:
      nodeSelector:
        "beta.kubernetes.io/os": linux
      containers:
        - name: azure-vote-back
          image: mcr.microsoft.com/oss/bitnami/redis:6.0.8
          env:
            - name: ALLOW_EMPTY_PASSWORD
              value: "yes"
          resources:
            requests:
              cpu: 100m
              memory: 128Mi
            limits:
              cpu: 250m
              memory: 256Mi
          ports:
            - containerPort: 6379
              name: redis
      ---  

apiVersion: v1
kind: Service
metadata:
  name: azure-vote-back
spec:
  ports:
    - port: 6379
  selector:
    app: azure-vote-back
  ---  

apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-front
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-front

```

```

app: azure-vote-front
template:
  metadata:
    labels:
      app: azure-vote-front
  spec:
    nodeSelector:
      "beta.kubernetes.io/os": linux
    containers:
      - name: azure-vote-front
        image: mcr.microsoft.com/azuredocs/azure-vote-front:v1
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
          - containerPort: 80
        env:
          - name: REDIS
            value: "azure-vote-back"
    ---
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-front
spec:
  type: LoadBalancer
  ports:
    - port: 80
  selector:
    app: azure-vote-front

```

3. Deploy the application using the [kubectl apply](#) command and specify the name of your YAML manifest:

```
kubectl apply -f azure-vote.yaml
```

Output shows the successfully created deployments and services:

```

deployment "azure-vote-back" created
service "azure-vote-back" created
deployment "azure-vote-front" created
service "azure-vote-front" created

```

Test the application

When the application runs, a Kubernetes service exposes the application front end to the internet. This process can take a few minutes to complete.

Monitor progress using the [kubectl get service](#) command with the `--watch` argument.

```
kubectl get service azure-vote-front --watch
```

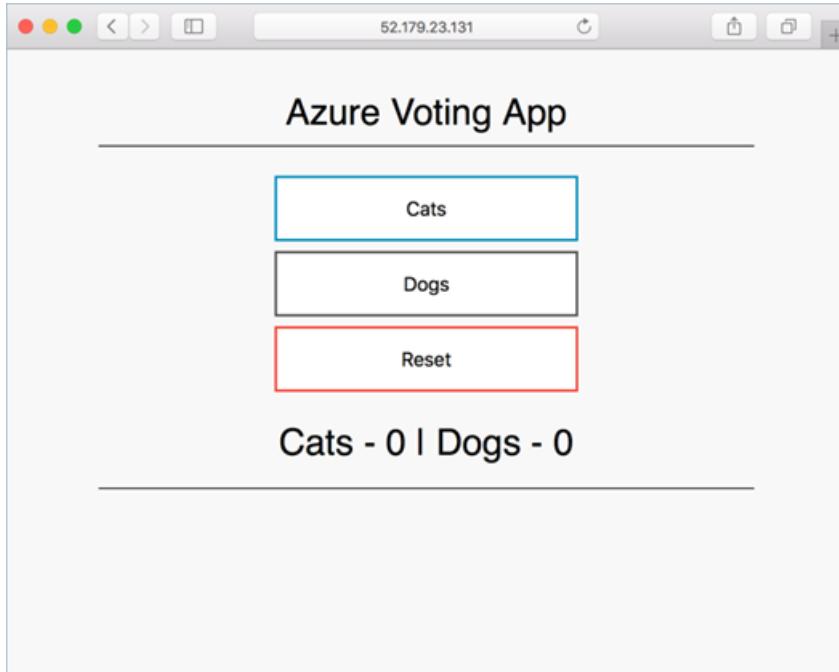
The EXTERNAL-IP output for the `azure-vote-front` service will initially show as *pending*.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
azure-vote-front	LoadBalancer	10.0.37.27	<pending>	80:30572/TCP	6s

Once the EXTERNAL-IP address changes from *pending* to an actual public IP address, use `CTRL-C` to stop the `kubectl` watch process. The following example output shows a valid public IP address assigned to the service:

```
azure-vote-front  LoadBalancer  10.0.37.27  52.179.23.131  80:30572/TCP  2m
```

To see the Azure Vote app in action, open a web browser to the external IP address of your service.



View the cluster nodes' and pods' health metrics captured by [Azure Monitor for containers](#) in the Azure portal.

Delete the cluster

To avoid Azure charges, clean up your unnecessary resources. Use the `az group delete` command to remove the resource group, container service, and all related resources.

```
az group delete --name myResourceGroup --yes --no-wait
```

NOTE

When you delete the cluster, the Azure Active Directory service principal used by the AKS cluster is not removed. For steps on how to remove the service principal, see [AKS service principal considerations and deletion](#).

If you used a managed identity, the identity is managed by the platform and does not require removal.

Get the code

Pre-existing container images were used in this quickstart to create a Kubernetes deployment. The related application code, Dockerfile, and Kubernetes manifest file are [available on GitHub](#).

Next steps

In this quickstart, you deployed a Kubernetes cluster and then deployed a multi-container application to it. [Access the Kubernetes web dashboard](#) for your AKS cluster.

To learn more about AKS, and walk through a complete code to deployment example, continue to the Kubernetes cluster tutorial.

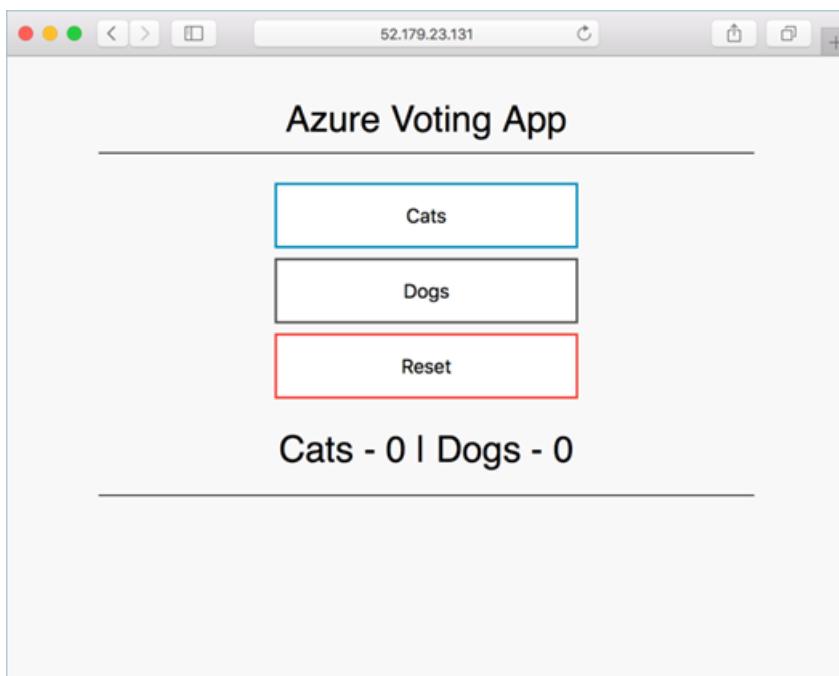
Quickstart: Deploy an Azure Kubernetes Service cluster using PowerShell

3/17/2021 • 6 minutes to read • [Edit Online](#)

Azure Kubernetes Service (AKS) is a managed Kubernetes service that lets you quickly deploy and manage clusters. In this quickstart, you will:

- Deploy an AKS cluster using PowerShell.
- Run a multi-container application with a web front-end and a Redis instance in the cluster.
- Monitor the health of the cluster and pods that run your application.

To learn more about creating a Windows Server node pool, see [Create an AKS cluster that supports Windows Server containers](#).



This quickstart assumes a basic understanding of Kubernetes concepts. For more information, see [Kubernetes core concepts for Azure Kubernetes Service \(AKS\)](#).

Prerequisites

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

If you're running PowerShell locally, install the Az PowerShell module and connect to your Azure account using the [Connect-AzAccount](#) cmdlet. For more information about installing the Az PowerShell module, see [Install Azure PowerShell](#).

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu bar at the upper right in the Azure portal .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

If you have multiple Azure subscriptions, select the appropriate subscription ID in which the resources should be billed using the [Set-AzContext](#) cmdlet.

```
Set-AzContext -SubscriptionId 00000000-0000-0000-0000-000000000000
```

Create a resource group

An [Azure resource group](#) is a logical group in which Azure resources are deployed and managed. When you create a resource group, you will be prompted to specify a location. This location is:

- The storage location of your resource group metadata.
- Where your resources will run in Azure if you don't specify another region during resource creation.

The following example creates a resource group named **myResourceGroup** in the **eastus** region.

Create a resource group using the [New-AzResourceGroup](#) cmdlet.

```
New-AzResourceGroup -Name myResourceGroup -Location eastus
```

Output for successfully created resource group:

```
ResourceGroupName : myResourceGroup
Location        : eastus
ProvisioningState : Succeeded
Tags            :
ResourceId      : /subscriptions/00000000-0000-0000-0000-000000000000/resourceGroups/myResourceGroup
```

Create AKS cluster

1. Generate an SSH key pair using the `ssh-keygen` command-line utility.

- For more details, see [Quick steps: Create and use an SSH public-private key pair for Linux VMs in Azure](#).
2. Create an AKS cluster using the `New-AzAks` cmdlet. Azure Monitor for containers is enabled by default.

The following example creates a cluster named `myAKSCluster` with one node.

```
New-AzAksCluster -ResourceGroupName myResourceGroup -Name myAKSCluster -NodeCount 1
```

After a few minutes, the command completes and returns information about the cluster.

NOTE

When you create an AKS cluster, a second resource group is automatically created to store the AKS resources. For more information, see [Why are two resource groups created with AKS?](#)

Connect to the cluster

To manage a Kubernetes cluster, use the Kubernetes command-line client, `kubectl`. `kubectl` is already installed if you use Azure Cloud Shell.

1. Install `kubectl` locally using the `Install-AzAksKubectl` cmdlet:

```
Install-AzAksKubectl
```

2. Configure `kubectl` to connect to your Kubernetes cluster using the `Import-AzAksCredential` cmdlet. The following cmdlet downloads credentials and configures the Kubernetes CLI to use them.

```
Import-AzAksCredential -ResourceGroupName myResourceGroup -Name myAKSCluster
```

3. Verify the connection to your cluster using the `kubectl get` command. This command returns a list of the cluster nodes.

```
.\kubectl get nodes
```

Output shows the single node created in the previous steps. Make sure the node status is *Ready*:

NAME	STATUS	ROLES	AGE	VERSION
aks-nodepool1-31718369-0	Ready	agent	6m44s	v1.15.10

Run the application

A [Kubernetes manifest file](#) defines a cluster's desired state, such as which container images to run.

In this quickstart, you will use a manifest to create all objects needed to run the [Azure Vote application](#). This manifest includes two [Kubernetes deployments](#):

- The sample Azure Vote Python applications.
- A Redis instance.

Two [Kubernetes Services](#) are also created:

- An internal service for the Redis instance.

- An external service to access the Azure Vote application from the internet.

1. Create a file named `azure-vote.yaml`.

- If you use the Azure Cloud Shell, this file can be created using `vi` or `nano` as if working on a virtual or physical system

2. Copy in the following YAML definition:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
  template:
    metadata:
      labels:
        app: azure-vote-back
    spec:
      nodeSelector:
        "beta.kubernetes.io/os": linux
      containers:
        - name: azure-vote-back
          image: mcr.microsoft.com/oss/bitnami/redis:6.0.8
          env:
            - name: ALLOW_EMPTY_PASSWORD
              value: "yes"
          resources:
            requests:
              cpu: 100m
              memory: 128Mi
            limits:
              cpu: 250m
              memory: 256Mi
          ports:
            - containerPort: 6379
              name: redis
      ---
      apiVersion: v1
      kind: Service
      metadata:
        name: azure-vote-back
      spec:
        ports:
          - port: 6379
        selector:
          app: azure-vote-back
      ---
      apiVersion: apps/v1
      kind: Deployment
      metadata:
        name: azure-vote-front
      spec:
        replicas: 1
        selector:
          matchLabels:
            app: azure-vote-front
        template:
          metadata:
            labels:
              app: azure-vote-front
        spec:
          nodeSelector:
            "beta.kubernetes.io/os": linux

```

```

  b6ca.kubernetes.io/ os : linux
  containers:
    - name: azure-vote-front
      image: mcr.microsoft.com/azuredocs/azure-vote-front:v1
      resources:
        requests:
          cpu: 100m
          memory: 128Mi
        limits:
          cpu: 250m
          memory: 256Mi
      ports:
        - containerPort: 80
      env:
        - name: REDIS
          value: "azure-vote-back"
    ---
  apiVersion: v1
  kind: Service
  metadata:
    name: azure-vote-front
  spec:
    type: LoadBalancer
    ports:
      - port: 80
    selector:
      app: azure-vote-front

```

3. Deploy the application using the [kubectl apply](#) command and specify the name of your YAML manifest:

```
.\kubectl apply -f azure-vote.yaml
```

Output shows the successfully created deployments and services:

```

deployment.apps/azure-vote-back created
service/azure-vote-back created
deployment.apps/azure-vote-front created
service/azure-vote-front created

```

Test the application

When the application runs, a Kubernetes service exposes the application front end to the internet. This process can take a few minutes to complete.

Monitor progress using the [kubectl get service](#) command with the `--watch` argument.

```
.\kubectl get service azure-vote-front --watch
```

The EXTERNAL-IP output for the `azure-vote-front` service will initially show as *pending*.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
azure-vote-front	LoadBalancer	10.0.37.27	<pending>	80:30572/TCP	6s

Once the EXTERNAL-IP address changes from *pending* to an actual public IP address, use `CTRL-C` to stop the [kubectl](#) watch process. The following example output shows a valid public IP address assigned to the service:

```
azure-vote-front  LoadBalancer  10.0.37.27  52.179.23.131  80:30572/TCP  2m
```

To see the Azure Vote app in action, open a web browser to the external IP address of your service.



View the cluster nodes' and pods' health metrics captured by Azure Monitor for containers in the Azure portal.

Delete the cluster

To avoid Azure charges, clean up your unnecessary resources. Use the [Remove-AzResourceGroup](#) cmdlet to remove the resource group, container service, and all related resources.

```
Remove-AzResourceGroup -Name myResourceGroup
```

NOTE

When you delete the cluster, the Azure Active Directory service principal used by the AKS cluster is not removed. For steps on how to remove the service principal, see [AKS service principal considerations and deletion](#).

If you used a managed identity, the identity is managed by the platform and does not require removal.

Get the code

Pre-existing container images were used in this quickstart to create a Kubernetes deployment. The related application code, Dockerfile, and Kubernetes manifest file are [available on GitHub](#).

Next steps

In this quickstart, you deployed a Kubernetes cluster and then deployed a multi-container application to it. [Access the Kubernetes web dashboard](#) for your AKS cluster.

To learn more about AKS, and walk through a complete code to deployment example, continue to the Kubernetes cluster tutorial.

[AKS tutorial](#)

Quickstart: Deploy an Azure Kubernetes Service (AKS) cluster using the Azure portal

4/21/2021 • 6 minutes to read • [Edit Online](#)

Azure Kubernetes Service (AKS) is a managed Kubernetes service that lets you quickly deploy and manage clusters. In this quickstart, you will:

- Deploy an AKS cluster using the Azure portal.
- Run a multi-container application with a web front-end and a Redis instance in the cluster.
- Monitor the health of the cluster and pods that run your application.



This quickstart assumes a basic understanding of Kubernetes concepts. For more information, see [Kubernetes core concepts for Azure Kubernetes Service \(AKS\)](#).

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

Prerequisites

Sign in to the Azure portal at <https://portal.azure.com>.

Create an AKS cluster

1. On the Azure portal menu or from the Home page, select **Create a resource**.
2. Select **Containers > Kubernetes Service**.
3. On the **Basics** page, configure the following options:
 - **Project details:**
 - Select an Azure Subscription.
 - Select or create an Azure Resource group, such as *myResourceGroup*.
 - **Cluster details:**

- Enter a **Kubernetes cluster name**, such as *myAKSCluster*.
- Select a **Region** and **Kubernetes version** for the AKS cluster.
- **Primary node pool:**
 - Select a **VM Node size** for the AKS nodes. The VM size *cannot* be changed once an AKS cluster has been deployed.
 - Select the number of nodes to deploy into the cluster. For this quickstart, set **Node count** to **1**. Node count *can* be adjusted after the cluster has been deployed.

Create Kubernetes cluster

Basics Node pools Authentication Networking Integrations Tags Review + create

Azure Kubernetes Service (AKS) manages your hosted Kubernetes environment, making it quick and easy to deploy and manage containerized applications without container orchestration expertise. It also eliminates the burden of ongoing operations and maintenance by provisioning, upgrading, and scaling resources on demand, without taking your applications offline. [Learn more about Azure Kubernetes Service](#)

Project details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

Resource group * ⓘ (New) myResourceGroup

Cluster details

Kubernetes cluster name * ⓘ myAKSCluster

Region * ⓘ (US) East US

Availability zones ⓘ Zones 1,2,3

Kubernetes version * ⓘ 1.17.11 (default)

Primary node pool

The number and size of nodes in the primary node pool in your cluster. For production workloads, at least 3 nodes are recommended for resiliency. For development or test workloads, only one node is required. If you would like to add additional node pools or to see additional configuration options for this node pool, go to the 'Node pools' tab above. You will be able to add additional node pools after creating your cluster. [Learn more about node pools in Azure Kubernetes Service](#)

Node size * ⓘ Standard DS2 v2

Node count * ⓘ 1

Review + create < Previous **Next : Node pools >**

4. Select **Next: Node pools** when complete.

5. Keep the default **Node pools** options. At the bottom of the screen, click **Next: Authentication**.

Caution

Newly created Azure AD service principals may take several minutes to propagate and become available, causing "service principal not found" errors and validation failures in Azure portal. If you hit this bump, please visit [our troubleshooting article](#) for mitigation.

6. On the **Authentication** page, configure the following options:

- Create a new cluster identity by either:
 - Leaving the **Authentication** field with **System-assinged managed identity**, or
 - Choosing **Service Principal** to use a service principal.
 - Select **(new) default service principal** to create a default service principal, or

- Select *Configure service principal* to use an existing one. You will need to provide the existing principal's SPN client ID and secret.
- Enable the Kubernetes role-based access control (Kubernetes RBAC) option to provide more fine-grained control over access to the Kubernetes resources deployed in your AKS cluster.

By default, *Basic* networking is used, and Azure Monitor for containers is enabled.

7. Click **Review + create** and then **Create** when validation completes.
8. It takes a few minutes to create the AKS cluster. When your deployment is complete, navigate to your resource by either:
 - Clicking **Go to resource**, or
 - Browsing to the AKS cluster resource group and selecting the AKS resource.
 - For example cluster dashboard below: browsing for *myResourceGroup* and selecting *myAKSCluster* resource.

Setting	Value
Resource group (change)	myResourceGroup
Kubernetes version	1.13.11
Status	Succeeded
Location	East US
API server address	myakscluster-dns-429509e5.hcp.eastus.azmk8s.io
HTTP application routing domain	N/A
Subscription (change)	My Subscription Name
Subscription ID	00000000-0000-0000-0000-000000000000
Node pools	1 node pools
Tags (change)	Click here to add tags

Monitor containers
Get health and performance insights
[Go to Azure Monitor insights](#)

View logs
Search and analyze logs using ad-hoc queries
[Go to Azure Monitor logs](#)

Connect to the cluster

To manage a Kubernetes cluster, use the Kubernetes command-line client, `kubectl`. `kubectl` is already installed if you use Azure Cloud Shell.

1. Open Cloud Shell using the  button on the top of the Azure portal.

The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with icons for Home, myAKSCluster, and other account settings. A search bar is also present. The main content area displays the details of the 'myAKSCluster' Kubernetes service, including its status as 'Succeeded', location as 'East US', and node pool information. Below this, there are sections for monitoring containers and viewing logs. At the bottom, a Cloud Shell terminal window is open, showing a successful connection attempt and a welcome message.

myAKSCluster
Kubernetes service

Search (Ctrl+/
)

Move Delete Refresh

Resource group (change)
myResourceGroup

Kubernetes version
1.13.11

Status
Succeeded

Location
East US

API server address
myakscluster-dns-429509e5...

Subscription (change)
My Subscription Name

HTTP application routing do...

Subscription ID
00000000-0000-0000-0000-000000000000

Tags (change)
Click here to add tags

Node pools
1 node pools

Settings

Node pools

Upgrade

Scale

Networking

Dev Spaces

Deployment center (preview)

Policies (preview)

Properties

Locks

Export template

Bash

Requesting a Cloud Shell. **Succeeded.**
Connecting terminal...

Welcome to Azure Cloud Shell

Type "az" to use Azure CLI
Type "help" to learn about Cloud Shell

user@Azure:~\$

NOTE

To perform these operations in a local shell installation:

1. Verify Azure CLI is installed.
 2. Connect to Azure via the `az login` command.

- Configure `kubectl` to connect to your Kubernetes cluster using the `az aks get-credentials` command. The following command downloads credentials and configures the Kubernetes CLI to use them.

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```

3. Verify the connection to your cluster using `kubectl get` to return a list of the cluster nodes.

```
kubectl get nodes
```

Output shows the single node created in the previous steps. Make sure the node status is *Ready*.

NAME	STATUS	ROLES	AGE	VERSION
aks-agentpool-14693408-0	Ready	agent	15m	v1.11.5

Run the application

A Kubernetes manifest file defines a cluster's desired state, like which container images to run.

In this quickstart, you will use a manifest to create all objects needed to run the Azure Vote application. This manifest includes two Kubernetes deployments:

- The sample Azure Vote Python applications.
- A Redis instance.

Two Kubernetes Services are also created:

- An internal service for the Redis instance.
- An external service to access the Azure Vote application from the internet.

1. In the Cloud Shell, use an editor to create a file named `azure-vote.yaml`, such as:

- `code azure-vote.yaml`
- `nano azure-vote.yaml`, or
- `vi azure-vote.yaml`.

2. Copy in the following YAML definition:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
  template:
    metadata:
      labels:
        app: azure-vote-back
    spec:
      nodeSelector:
        "beta.kubernetes.io/os": linux
      containers:
        - name: azure-vote-back
          image: mcr.microsoft.com/oss/bitnami/redis:6.0.8
          env:
            - name: ALLOW_EMPTY_PASSWORD
              value: "yes"
          resources:
            requests:
              cpu: 100m
              memory: 128Mi
            limits:
              cpu: 250m
              memory: 256Mi
          ports:
            - containerPort: 6379
              name: redis
      ---
```

```

apiVersion: v1
kind: Service
metadata:
  name: azure-vote-back
spec:
  selector:
    app: azure-vote-back
  ports:
    - port: 80
      targetPort: 6379
  type: LoadBalancer
  ---
```

```

name: azure-vote-back
spec:
  ports:
    - port: 6379
  selector:
    app: azure-vote-back
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-front
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-front
  template:
    metadata:
      labels:
        app: azure-vote-front
    spec:
      nodeSelector:
        "beta.kubernetes.io/os": linux
      containers:
        - name: azure-vote-front
          image: mcr.microsoft.com/azuredocs/azure-vote-front:v1
          resources:
            requests:
              cpu: 100m
              memory: 128Mi
            limits:
              cpu: 250m
              memory: 256Mi
          ports:
            - containerPort: 80
          env:
            - name: REDIS
              value: "azure-vote-back"
---
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-front
spec:
  type: LoadBalancer
  ports:
    - port: 80
  selector:
    app: azure-vote-front

```

3. Deploy the application using the `kubectl apply` command and specify the name of your YAML manifest:

```
kubectl apply -f azure-vote.yaml
```

Output shows the successfully created deployments and services:

```

deployment "azure-vote-back" created
service "azure-vote-back" created
deployment "azure-vote-front" created
service "azure-vote-front" created

```

Test the application

When the application runs, a Kubernetes service exposes the application front end to the internet. This process can take a few minutes to complete.

To monitor progress, use the `kubectl get service` command with the `--watch` argument.

```
kubectl get service azure-vote-front --watch
```

The EXTERNAL-IP output for the `azure-vote-front` service will initially show as *pending*.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
azure-vote-front	LoadBalancer	10.0.37.27	<pending>	80:30572/TCP	6s

Once the EXTERNAL-IP address changes from *pending* to an actual public IP address, use `CTRL-C` to stop the `kubectl` watch process. The following example output shows a valid public IP address assigned to the service:

```
azure-vote-front LoadBalancer 10.0.37.27 52.179.23.131 80:30572/TCP 2m
```

To see the Azure Vote app in action, open a web browser to the external IP address of your service.



Monitor health and logs

When you created the cluster, Azure Monitor for containers was enabled. Azure Monitor for containers provides health metrics for both the AKS cluster and pods running on the cluster.

Metric data takes a few minutes to populate in the Azure portal. To see current health status, uptime, and resource usage for the Azure Vote pods:

1. Browse back to the AKS resource in the Azure portal.
2. Under **Monitoring** on the left-hand side, choose **Insights**.
3. Across the top, choose to **+ Add Filter**.
4. Select **Namespace** as the property, then choose `<All but kube-system>`.
5. Select **Containers** to view them.

The `azure-vote-back` and `azure-vote-front` containers will display, as shown in the following example:

To view logs for the `azure-vote-front` pod, select **View container logs** from the containers list drop-down. These logs include the `stdout` and `stderr` streams from the container.

LogEntrySource	LogEntry	TimeGenerated [UTC]	Computer	Image
> stdout	1:M 18 Dec 2018 18:54:08.774 # Server initialized	2018-12-18T18:54:08.774	aks-nodepool1-31718369-0	redis:latest
> stdout	1:M 18 Dec 2018 18:54:08.774 * Ready to accept connections	2018-12-18T18:54:08.774	aks-nodepool1-31718369-0	redis:latest
> stdout	1:M 18 Dec 2018 18:54:08.774 # WARNING you have Transparent Hu...	2018-12-18T18:54:08.774	aks-nodepool1-31718369-0	redis:latest
> stdout	1:M 18 Dec 2018 18:54:08.774 # WARNING: The TCP backlog setting ...	2018-12-18T18:54:08.774	aks-nodepool1-31718369-0	redis:latest
> stdout	1:M 18 Dec 2018 18:54:08.774 * Running mode=standalone, port=63...	2018-12-18T18:54:08.774	aks-nodepool1-31718369-0	redis:latest
> stdout	1:C 18 Dec 2018 18:54:08.773 # Warning: no config file specified, us...	2018-12-18T18:54:08.773	aks-nodepool1-31718369-0	redis:latest
> stdout	1:C 18 Dec 2018 18:54:08.773 # Redis version=5.0.3, bits=64, comm...	2018-12-18T18:54:08.773	aks-nodepool1-31718369-0	redis:latest
> stdout	1:C 18 Dec 2018 18:54:08.773 # c000c000c000e Redis is starting o...	2018-12-18T18:54:08.773	aks-nodepool1-31718369-0	redis:latest

Delete cluster

To avoid Azure charges, clean up your unnecessary resources. Select the **Delete** button on the AKS cluster dashboard. You can also use the `az aks delete` command in the Cloud Shell:

```
az aks delete --resource-group myResourceGroup --name myAKSCluster --no-wait
```

NOTE

When you delete the cluster, the Azure Active Directory service principal used by the AKS cluster is not removed. For steps on how to remove the service principal, see [AKS service principal considerations and deletion](#).

If you used a managed identity, the identity is managed by the platform and does not require removal.

Get the code

Pre-existing container images were used in this quickstart to create a Kubernetes deployment. The related application code, Dockerfile, and Kubernetes manifest file are [available on GitHub](#).

Next steps

In this quickstart, you deployed a Kubernetes cluster and then deployed a multi-container application to it. Access the Kubernetes web dashboard for your AKS cluster.

To learn more about AKS by walking through a complete example, including building an application, deploying from Azure Container Registry, updating a running application, and scaling and upgrading your cluster, continue to the Kubernetes cluster tutorial.

[AKS tutorial](#)

Quickstart: Deploy an Azure Kubernetes Service (AKS) cluster using an ARM template

4/29/2021 • 8 minutes to read • [Edit Online](#)

Azure Kubernetes Service (AKS) is a managed Kubernetes service that lets you quickly deploy and manage clusters. In this quickstart, you will:

- Deploy an AKS cluster using an Azure Resource Manager template.
- Run a multi-container application with a web front-end and a Redis instance in the cluster.



An [ARM template](#) is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

This quickstart assumes a basic understanding of Kubernetes concepts. For more information, see [Kubernetes core concepts for Azure Kubernetes Service \(AKS\)](#).

If your environment meets the prerequisites and you're familiar with using ARM templates, select the **Deploy to Azure** button. The template will open in the Azure portal.

 [Deploy to Azure](#)

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

Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#).

 [Launch Cloud Shell](#)
- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
 - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish

the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).

- When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
- Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.
- This article requires version 2.0.61 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.
- To create an AKS cluster using a Resource Manager template, you provide an SSH public key. If you need this resource, see the following section; otherwise skip to the [Review the template](#) section.

Create an SSH key pair

To access AKS nodes, you connect using an SSH key pair (public and private), which you generate using the `ssh-keygen` command. By default, these files are created in the `~/.ssh` directory. Running the `ssh-keygen` command will overwrite any SSH key pair with the same name already existing in the given location.

1. Go to <https://shell.azure.com> to open Cloud Shell in your browser.
2. Run the `ssh-keygen` command. The following example creates an SSH key pair using RSA encryption and a bit length of 2048:

```
ssh-keygen -t rsa -b 2048
```

For more information about creating SSH keys, see [Create and manage SSH keys for authentication in Azure](#).

Review the template

The template used in this quickstart is from [Azure Quickstart templates](#).

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.1",
  "parameters": {
    "clusterName": {
      "type": "string",
      "defaultValue": "aks101cluster",
      "metadata": {
        "description": "The name of the Managed Cluster resource."
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "The location of the Managed Cluster resource."
      }
    },
    "dnsPrefix": {
      "type": "string",
      "metadata": {
        "description": "Optional DNS prefix to use with hosted Kubernetes API server FQDN."
      }
    },
    "osDiskSizeGB": {
      "type": "int",
      "defaultValue": 0,
      "minValue": 0,
```

```

    "maxLength": 1023,
    "metadata": {
        "description": "Disk size (in GB) to provision for each of the agent pool nodes. This value ranges from 0 to 1023. Specifying 0 will apply the default disk size for that agentVMSize."
    }
},
"agentCount": {
    "type": "int",
    "defaultValue": 3,
    "minValue": 1,
    "maxValue": 50,
    "metadata": {
        "description": "The number of nodes for the cluster."
    }
},
"agentVMSize": {
    "type": "string",
    "defaultValue": "Standard_DS2_v2",
    "metadata": {
        "description": "The size of the Virtual Machine."
    }
},
"linuxAdminUsername": {
    "type": "string",
    "metadata": {
        "description": "User name for the Linux Virtual Machines."
    }
},
"sshRSAPublicKey": {
    "type": "string",
    "metadata": {
        "description": "Configure all linux machines with the SSH RSA public key string. Your key should include three parts, for example 'ssh-rsa AAAAB...snip...UcyupgH azureuser@linuxvm'"
    }
},
"osType": {
    "type": "string",
    "defaultValue": "Linux",
    "allowedValues": [
        "Linux"
    ],
    "metadata": {
        "description": "The type of operating system."
    }
},
"resources": [
{
    "type": "Microsoft.ContainerService/managedClusters",
    "apiVersion": "2020-03-01",
    "name": "[parameters('clusterName')]",
    "location": "[parameters('location')]",
    "properties": {
        "dnsPrefix": "[parameters('dnsPrefix')]",
        "agentPoolProfiles": [
            {
                "name": "agentpool",
                "osDiskSizeGB": "[parameters('osDiskSizeGB')]",
                "count": "[parameters('agentCount')]",
                "vmSize": "[parameters('agentVMSize')]",
                "osType": "[parameters('osType')]",
                "storageProfile": "ManagedDisks"
            }
        ],
        "linuxProfile": {
            "adminUsername": "[parameters('linuxAdminUsername')]",
            "ssh": {
                "publicKeys": [
                    {

```

```
        "keyData": "[parameters('sshRSAPublicKey')]"
    }
]
}
},
"identity": {
    "type": "SystemAssigned"
}
],
"outputs": {
    "controlPlaneFQDN": {
        "type": "string",
        "value": "[reference(parameters('clusterName')).fqdn]"
    }
}
}
```

For more AKS samples, see the [AKS quickstart templates](#) site.

Deploy the template

1. Select the following button to sign in to Azure and open a template.



2. Select or enter the following values.

For this quickstart, leave the default values for the *OS Disk Size GB*, *Agent Count*, *Agent VM Size*, *OS Type*, and *Kubernetes Version*. Provide your own values for the following template parameters:

- **Subscription:** Select an Azure subscription.
- **Resource group:** Select **Create new**. Enter a unique name for the resource group, such as *myResourceGroup*, then choose **OK**.
- **Location:** Select a location, such as **East US**.
- **Cluster name:** Enter a unique name for the AKS cluster, such as *myAKSCluster*.
- **DNS prefix:** Enter a unique DNS prefix for your cluster, such as *myakscluster*.
- **Linux Admin Username:** Enter a username to connect using SSH, such as *azureuser*.
- **SSH RSA Public Key:** Copy and paste the *public* part of your SSH key pair (by default, the contents of *~/.ssh/id_rsa.pub*).



3. Select **Review + Create**.

It takes a few minutes to create the AKS cluster. Wait for the cluster to be successfully deployed before you move on to the next step.

Validate the deployment

Connect to the cluster

To manage a Kubernetes cluster, use the Kubernetes command-line client, `kubectl`. `kubectl` is already installed if you use Azure Cloud Shell.

1. Install `kubectl` locally using the `az aks install-cli` command:

```
az aks install-cli
```

2. Configure `kubectl` to connect to your Kubernetes cluster using the `az aks get-credentials` command. This command downloads credentials and configures the Kubernetes CLI to use them.

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```

- Verify the connection to your cluster using the [kubectl get](#) command. This command returns a list of the cluster nodes.

```
kubectl get nodes
```

Output shows the nodes created in the previous steps. Make sure that the status for all the nodes is *Ready*.

NAME	STATUS	ROLES	AGE	VERSION
aks-agentpool-41324942-0	Ready	agent	6m44s	v1.12.6
aks-agentpool-41324942-1	Ready	agent	6m46s	v1.12.6
aks-agentpool-41324942-2	Ready	agent	6m45s	v1.12.6

Run the application

A [Kubernetes manifest file](#) defines a cluster's desired state, such as which container images to run.

In this quickstart, you will use a manifest to create all objects needed to run the [Azure Vote application](#). This manifest includes two [Kubernetes deployments](#):

- The sample Azure Vote Python applications.
- A Redis instance.

Two [Kubernetes Services](#) are also created:

- An internal service for the Redis instance.
- An external service to access the Azure Vote application from the internet.

- Create a file named `azure-vote.yaml`.

- If you use the Azure Cloud Shell, this file can be created using `vi` or `nano` as if working on a virtual or physical system

- Copy in the following YAML definition:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
  template:
    metadata:
      labels:
        app: azure-vote-back
    spec:
      nodeSelector:
        "beta.kubernetes.io/os": linux
      containers:
        - name: azure-vote-back
          image: mcr.microsoft.com/oss/bitnami/redis:6.0.8
          env:
            - name: ALLOW_EMPTY_PASSWORD
              value: "yes"
          resources:
            requests:
              cpu: 100m
              memory: 128Mi
            limits:
              cpu: 200m
              memory: 256Mi
```

```

        cpu: 250m
        memory: 256Mi
      ports:
      - containerPort: 6379
        name: redis
    ---
  apiVersion: v1
  kind: Service
  metadata:
    name: azure-vote-back
  spec:
    ports:
    - port: 6379
    selector:
      app: azure-vote-back
  ---
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: azure-vote-front
  spec:
    replicas: 1
    selector:
      matchLabels:
        app: azure-vote-front
    template:
      metadata:
        labels:
          app: azure-vote-front
      spec:
        nodeSelector:
          "beta.kubernetes.io/os": linux
        containers:
        - name: azure-vote-front
          image: mcr.microsoft.com/azuredocs/azure-vote-front:v1
          resources:
            requests:
              cpu: 100m
              memory: 128Mi
            limits:
              cpu: 250m
              memory: 256Mi
          ports:
          - containerPort: 80
          env:
          - name: REDIS
            value: "azure-vote-back"
    ---
  apiVersion: v1
  kind: Service
  metadata:
    name: azure-vote-front
  spec:
    type: LoadBalancer
    ports:
    - port: 80
    selector:
      app: azure-vote-front

```

3. Deploy the application using the [kubectl apply](#) command and specify the name of your YAML manifest:

```
kubectl apply -f azure-vote.yaml
```

Output shows the successfully created deployments and services:

```
deployment "azure-vote-back" created
service "azure-vote-back" created
deployment "azure-vote-front" created
service "azure-vote-front" created
```

Test the application

When the application runs, a Kubernetes service exposes the application front end to the internet. This process can take a few minutes to complete.

Monitor progress using the `kubectl get service` command with the `--watch` argument.

```
kubectl get service azure-vote-front --watch
```

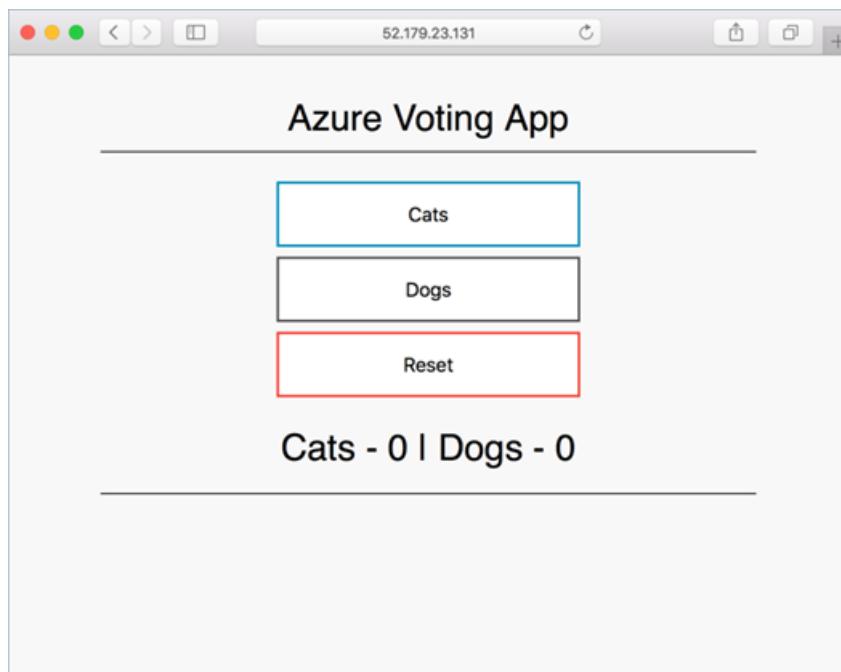
The EXTERNAL-IP output for the `azure-vote-front` service will initially show as *pending*.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
azure-vote-front	LoadBalancer	10.0.37.27	<pending>	80:30572/TCP	6s

Once the EXTERNAL-IP address changes from *pending* to an actual public IP address, use `CTRL-C` to stop the `kubectl` watch process. The following example output shows a valid public IP address assigned to the service:

```
azure-vote-front LoadBalancer 10.0.37.27 52.179.23.131 80:30572/TCP 2m
```

To see the Azure Vote app in action, open a web browser to the external IP address of your service.



Clean up resources

To avoid Azure charges, clean up your unnecessary resources. Use the `az group delete` command to remove the resource group, container service, and all related resources.

```
az group delete --name myResourceGroup --yes --no-wait
```

NOTE

When you delete the cluster, the Azure Active Directory service principal used by the AKS cluster is not removed. For steps on how to remove the service principal, see [AKS service principal considerations and deletion](#).

If you used a managed identity, the identity is managed by the platform and does not require removal.

Get the code

Pre-existing container images were used in this quickstart to create a Kubernetes deployment. The related application code, Dockerfile, and Kubernetes manifest file are [available on GitHub](#).

Next steps

In this quickstart, you deployed a Kubernetes cluster and then deployed a multi-container application to it. [Access the Kubernetes web dashboard](#) for your AKS cluster.

To learn more about AKS, and walk through a complete code to deployment example, continue to the Kubernetes cluster tutorial.

[AKS tutorial](#)

Quickstart: Develop on Azure Kubernetes Service (AKS) with Helm

5/14/2021 • 4 minutes to read • [Edit Online](#)

[Helm](#) is an open-source packaging tool that helps you install and manage the lifecycle of Kubernetes applications. Similar to Linux package managers like *APT* and *Yum*, Helm manages Kubernetes charts, which are packages of pre-configured Kubernetes resources.

In this quickstart, you'll use Helm to package and run an application on AKS. For more details on installing an existing application using Helm, see the [Install existing applications with Helm in AKS](#) how-to guide.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, you can create a [free account](#).
- [Azure CLI installed](#).
- [Helm v3 installed](#).

Create an Azure Container Registry

You'll need to store your container images in an Azure Container Registry (ACR) to run your application in your AKS cluster using Helm. Provide your own registry name unique within Azure and containing 5-50 alphanumeric characters. The *Basic* SKU is a cost-optimized entry point for development purposes that provides a balance of storage and throughput.

The below example uses [az acr create](#) to create an ACR named *MyHelmACR* in *MyResourceGroup* with the *Basic* SKU.

```
az group create --name MyResourceGroup --location eastus
az acr create --resource-group MyResourceGroup --name MyHelmACR --sku Basic
```

Output will be similar to the following example. Take note of your *loginServer* value for your ACR since you'll use it in a later step. In the below example, *myhelmacr.azurecr.io* is the *loginServer* for *MyHelmACR*.

```
{  
    "adminUserEnabled": false,  
    "creationDate": "2019-06-11T13:35:17.998425+00:00",  
    "id":  
        "/subscriptions/<ID>/resourceGroups/MyResourceGroup/providers/Microsoft.ContainerRegistry/registries/MyHelmACR",  
    "location": "eastus",  
    "loginServer": "myhelmacr.azurecr.io",  
    "name": "MyHelmACR",  
    "networkRuleSet": null,  
    "provisioningState": "Succeeded",  
    "resourceGroup": "MyResourceGroup",  
    "sku": {  
        "name": "Basic",  
        "tier": "Basic"  
    },  
    "status": null,  
    "storageAccount": null,  
    "tags": {},  
    "type": "Microsoft.ContainerRegistry/registries"  
}
```

Create an AKS cluster

Your new AKS cluster needs access to your ACR to pull the container images and run them. Use the following command to:

- Create an AKS cluster called *MyAKS* and attach *MyHelmACR*.
- Grant the *MyAKS* cluster access to your *MyHelmACR* ACR.

```
az aks create -g MyResourceGroup -n MyAKS --location eastus --attach-acr MyHelmACR --generate-ssh-keys
```

Connect to your AKS cluster

To connect a Kubernetes cluster locally, use the Kubernetes command-line client, `kubectl`. `kubectl` is already installed if you use Azure Cloud Shell.

1. Install `kubectl` locally using the `az aks install-cli` command:

```
az aks install-cli
```

2. Configure `kubectl` to connect to your Kubernetes cluster using the `az aks get-credentials` command.

The following command example gets credentials for the AKS cluster named *MyAKS* in the *MyResourceGroup*:

```
az aks get-credentials --resource-group MyResourceGroup --name MyAKS
```

Download the sample application

This quickstart uses [an example Node.js application](#). Clone the application from GitHub and navigate to the `dev-spaces/samples/nodejs/getting-started/webfrontend` directory.

```
git clone https://github.com/Azure/dev-spaces  
cd dev-spaces/samples/nodejs/getting-started/webfrontend
```

Create a Dockerfile

Create a new *Dockerfile* file using the following commands:

```
FROM node:latest  
  
WORKDIR /webfrontend  
  
COPY package.json ./  
  
RUN npm install  
  
COPY . .  
  
EXPOSE 80  
CMD ["node", "server.js"]
```

Build and push the sample application to the ACR

Using the preceding Dockerfile, run the `az acr build` command to build and push an image to the registry. The `.` at the end of the command sets the location of the Dockerfile (in this case, the current directory).

```
az acr build --image webfrontend:v1 \  
--registry MyHelmACR \  
--file Dockerfile .
```

Create your Helm chart

Generate your Helm chart using the `helm create` command.

```
helm create webfrontend
```

Update *webfrontend/values.yaml*:

- Replace the `loginServer` of your registry that you noted in an earlier step, such as `myhelmacr.azurecr.io`.
- Change `image.repository` to `<loginServer>/webfrontend`
- Change `service.type` to `LoadBalancer`

For example:

```
# Default values for webfrontend.  
# This is a YAML-formatted file.  
# Declare variables to be passed into your templates.  
  
replicaCount: 1  
  
image:  
  repository: myhelmacr.azurecr.io/webfrontend  
  pullPolicy: IfNotPresent  
...  
service:  
  type: LoadBalancer  
  port: 80  
...
```

Update `appVersion` to `v1` in `webfrontend/Chart.yaml`. For example

```
apiVersion: v2  
name: webfrontend  
...  
# This is the version number of the application being deployed. This version number should be  
# incremented each time you make changes to the application.  
appVersion: v1
```

Run your Helm chart

Install your application using your Helm chart using the `helm install` command.

```
helm install webfrontend webfrontend/
```

It takes a few minutes for the service to return a public IP address. Monitor progress using the `kubectl get service` command with the `--watch` argument.

```
$ kubectl get service --watch  
  
NAME          TYPE      CLUSTER-IP   EXTERNAL-IP  PORT(S)        AGE  
webfrontend   LoadBalancer  10.0.141.72 <pending>    80:32150/TCP  2m  
...  
webfrontend   LoadBalancer  10.0.141.72 <EXTERNAL-IP>  80:32150/TCP  7m
```

Navigate to your application's load balancer in a browser using the `<EXTERNAL-IP>` to see the sample application.

Delete the cluster

Use the `az group delete` command to remove the resource group, the AKS cluster, the container registry, the container images stored in the ACR, and all related resources.

```
az group delete --name MyResourceGroup --yes --no-wait
```

NOTE

When you delete the cluster, the Azure Active Directory service principal used by the AKS cluster is not removed. For steps on how to remove the service principal, see [AKS service principal considerations and deletion](#).

If you used a managed identity, the identity is managed by the platform and does not require removal.

Next steps

For more information about using Helm, see the Helm documentation.

[Helm documentation](#)

Tutorial: Prepare an application for Azure Kubernetes Service (AKS)

3/5/2021 • 3 minutes to read • [Edit Online](#)

In this tutorial, part one of seven, a multi-container application is prepared for use in Kubernetes. Existing development tools such as Docker Compose are used to locally build and test an application. You learn how to:

- Clone a sample application source from GitHub
- Create a container image from the sample application source
- Test the multi-container application in a local Docker environment

Once completed, the following application runs in your local development environment:



In later tutorials, the container image is uploaded to an Azure Container Registry, and then deployed into an AKS cluster.

Before you begin

This tutorial assumes a basic understanding of core Docker concepts such as containers, container images, and `docker` commands. For a primer on container basics, see [Get started with Docker](#).

To complete this tutorial, you need a local Docker development environment running Linux containers. Docker provides packages that configure Docker on a [Mac](#), [Windows](#), or [Linux](#) system.

NOTE

Azure Cloud Shell does not include the Docker components required to complete every step in these tutorials. Therefore, we recommend using a full Docker development environment.

Get application code

The [sample application](#) used in this tutorial is a basic voting app consisting of a front-end web component and a back-end Redis instance. The web component is packaged into a custom container image. The Redis instance uses an unmodified image from Docker Hub.

Use [git](#) to clone the sample application to your development environment:

```
git clone https://github.com/Azure-Samples/azure-voting-app-redis.git
```

Change into the cloned directory.

```
cd azure-voting-app-redis
```

Inside the directory is the application source code, a pre-created Docker compose file, and a Kubernetes manifest file. These files are used throughout the tutorial set. The contents and structure of the directory are as follows:

```
azure-voting-app-redis
|   azure-vote-all-in-one-redis.yaml
|   docker-compose.yaml
|   LICENSE
|   README.md

|-- azure-vote
|   |   app_init.supervisord.conf
|   |   Dockerfile
|   |   Dockerfile-for-app-service
|   |   sshd_config

|   |-- azure-vote
|       |       config_file.cfg
|       |       main.py

|       |       static
|           |           default.css

|       |       templates
|           |           index.html

|-- jenkins-tutorial
    |       config-jenkins.sh
    |       deploy-jenkins-vm.sh
```

Create container images

[Docker Compose](#) can be used to automate building container images and the deployment of multi-container applications.

Use the sample `docker-compose.yaml` file to create the container image, download the Redis image, and start the application:

```
docker-compose up -d
```

When completed, use the [docker images](#) command to see the created images. Three images have been downloaded or created. The *azure-vote-front* image contains the front-end application and uses the *nginx-flask* image as a base. The *redis* image is used to start a Redis instance.

```
$ docker images
```

REPOSITORY SIZE	TAG	IMAGE ID	CREATED
mcr.microsoft.com/azuredocs/azure-vote-front 944MB	v1	84b41c268ad9	9 seconds ago
mcr.microsoft.com/oss/bitnami/redis 103MB	6.0.8	3a54a920bb6c	2 days ago
tiangolo/uwsgi-nginx-flask 944MB	python3.6	a16ce562e863	6 weeks ago

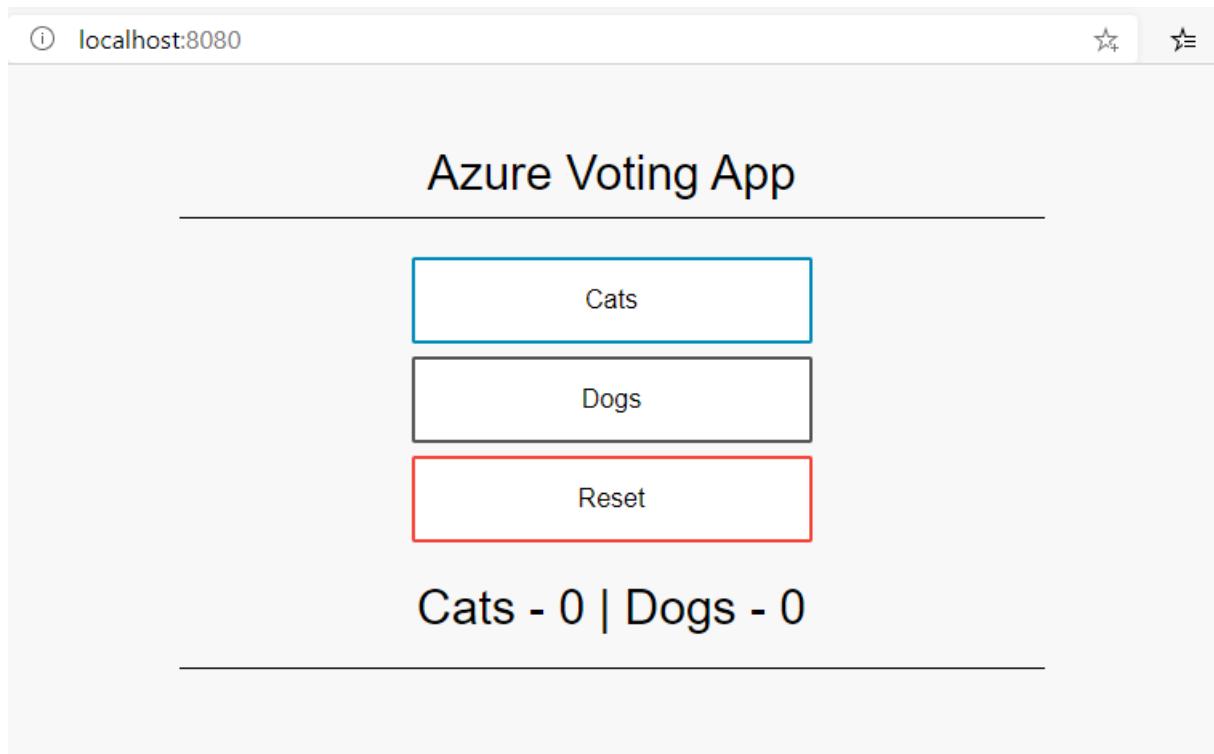
Run the [docker ps](#) command to see the running containers:

```
$ docker ps
```

CONTAINER ID STATUS	IMAGE PORTS	NAMES	COMMAND	CREATED
d10e5244f237 Up 3 minutes	mcr.microsoft.com/azuredocs/azure-vote-front:v1 443/tcp, 0.0.0.0:8080->80/tcp	azure-vote-front	"/entrypoint.sh /sta..."	3 minutes ago
21574cb38c1f Up 3 minutes	mcr.microsoft.com/oss/bitnami/redis:6.0.8 0.0.0.0:6379->6379/tcp	azure-vote-back	"/opt/bitnami/script..."	3 minutes ago

Test application locally

To see the running application, enter <http://localhost:8080> in a local web browser. The sample application loads, as shown in the following example:



Clean up resources

Now that the application's functionality has been validated, the running containers can be stopped and removed. ***Do not delete the container images*** - in the next tutorial, the *azure-vote-front* image is uploaded to an Azure Container Registry instance.

Stop and remove the container instances and resources with the [docker-compose down](#) command:

```
docker-compose down
```

When the local application has been removed, you have a Docker image that contains the Azure Vote application, *azure-vote-front*, for use with the next tutorial.

Next steps

In this tutorial, an application was tested and container images created for the application. You learned how to:

- Clone a sample application source from GitHub
- Create a container image from the sample application source
- Test the multi-container application in a local Docker environment

Advance to the next tutorial to learn how to store container images in Azure Container Registry.

[Push images to Azure Container Registry](#)

Tutorial: Deploy and use Azure Container Registry

4/21/2021 • 4 minutes to read • [Edit Online](#)

Azure Container Registry (ACR) is a private registry for container images. A private container registry lets you securely build and deploy your applications and custom code. In this tutorial, part two of seven, you deploy an ACR instance and push a container image to it. You learn how to:

- Create an Azure Container Registry (ACR) instance
- Tag a container image for ACR
- Upload the image to ACR
- View images in your registry

In later tutorials, this ACR instance is integrated with a Kubernetes cluster in AKS, and an application is deployed from the image.

Before you begin

In the [previous tutorial](#), a container image was created for a simple Azure Voting application. If you have not created the Azure Voting app image, return to [Tutorial 1 – Create container images](#).

This tutorial requires that you're running the Azure CLI version 2.0.53 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Create an Azure Container Registry

To create 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 the following example, a resource group named `myResourceGroup` is created in the `eastus` region:

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

Create an Azure Container Registry instance with the `az acr create` command and provide your own registry name. The registry name must be unique within Azure, and contain 5-50 alphanumeric characters. In the rest of this tutorial, `<acrName>` is used as a placeholder for the container registry name. Provide your own unique registry name. The *Basic* SKU is a cost-optimized entry point for development purposes that provides a balance of storage and throughput.

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

Log in to the container registry

To use the ACR instance, you must first log in. Use the `az acr login` command and provide the unique name given to the container registry in the previous step.

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

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

Tag a container image

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

```
docker images
```

The above command's output shows list of your current local images:

REPOSITORY	TAG	IMAGE ID	CREATED
mcr.microsoft.com/azuredocs/azure-vote-front	v1	84b41c268ad9	7 minutes ago
944MB			
mcr.microsoft.com/oss/bitnami/redis	6.0.8	3a54a920bb6c	2 days ago
103MB			
tiangolo/uwsgi-nginx-flask	python3.6	a16ce562e863	6 weeks ago
944MB			

To use the *azure-vote-front* container image with ACR, the image needs to be tagged with the login server address of your registry. This tag is used for routing when pushing container images to an image registry.

To get the login server address, use the [az acr list](#) command and query for the *loginServer* as follows:

```
az acr list --resource-group myResourceGroup --query "[].{acrLoginServer:loginServer}" --output table
```

Now, tag your local *azure-vote-front* image with the *acrLoginServer* address of the container registry. To indicate the image version, add *.v1* to the end of the image name:

```
docker tag mcr.microsoft.com/azuredocs/azure-vote-front:v1 <acrLoginServer>/azure-vote-front:v1
```

To verify the tags are applied, run [docker images](#) again.

```
docker images
```

An image is tagged with the ACR instance address and a version number.

REPOSITORY	TAG	IMAGE ID	CREATED
mcr.microsoft.com/azuredocs/azure-vote-front	v1	84b41c268ad9	16 minutes ago
944MB			
mycontainerregistry.azurecr.io/azure-vote-front	v1	84b41c268ad9	16 minutes ago
944MB			
mcr.microsoft.com/oss/bitnami/redis	6.0.8	3a54a920bb6c	2 days ago
103MB			
tiangolo/uwsgi-nginx-flask	python3.6	a16ce562e863	6 weeks ago
944MB			

Push images to registry

With your image built and tagged, push the *azure-vote-front* image to your ACR instance. Use [docker push](#) and provide your own *acrLoginServer* address for the image name as follows:

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

It may take a few minutes to complete the image push to ACR.

List images in registry

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

Provide your own <acrName> as follows:

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

The following example output lists the *azure-vote-front* image as available in the registry:

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

To see the tags for a specific image, use the [az acr repository show-tags](#) command as follows:

```
az acr repository show-tags --name <acrName> --repository azure-vote-front --output table
```

The following example output shows the *v1* image tagged in a previous step:

```
Result
-----
v1
```

You now have a container image that is stored in a private Azure Container Registry instance. This image is deployed from ACR to a Kubernetes cluster in the next tutorial.

Next steps

In this tutorial, you created an Azure Container Registry and pushed an image for use in an AKS cluster. You learned how to:

- Create an Azure Container Registry (ACR) instance
- Tag a container image for ACR
- Upload the image to ACR
- View images in your registry

Advance to the next tutorial to learn how to deploy a Kubernetes cluster in Azure.

[Deploy Kubernetes cluster](#)

Tutorial: Deploy an Azure Kubernetes Service (AKS) cluster

4/21/2021 • 3 minutes to read • [Edit Online](#)

Kubernetes provides a distributed platform for containerized applications. With AKS, you can quickly create a production ready Kubernetes cluster. In this tutorial, part three of seven, a Kubernetes cluster is deployed in AKS. You learn how to:

- Deploy a Kubernetes AKS cluster that can authenticate to an Azure container registry
- Install the Kubernetes CLI (kubectl)
- Configure kubectl to connect to your AKS cluster

In later tutorials, the Azure Vote application is deployed to the cluster, scaled, and updated.

Before you begin

In previous tutorials, a container image was created and uploaded to an Azure Container Registry instance. If you haven't done these steps, and would like to follow along, start at [Tutorial 1 – Create container images](#).

This tutorial requires that you're running the Azure CLI version 2.0.53 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Create a Kubernetes cluster

AKS clusters can use Kubernetes role-based access control (Kubernetes RBAC). These controls let you define access to resources based on roles assigned to users. Permissions are combined if a user is assigned multiple roles, and permissions can be scoped to either a single namespace or across the whole cluster. By default, the Azure CLI automatically enables Kubernetes RBAC when you create an AKS cluster.

Create an AKS cluster using `az aks create`. The following example creates a cluster named *myAKSCluster* in the resource group named *myResourceGroup*. This resource group was created in the [previous tutorial](#) in the *eastus* region. The following example does not specify a region so the AKS cluster is also created in the *eastus* region. For more information, see [Quotas, virtual machine size restrictions, and region availability in Azure Kubernetes Service \(AKS\)](#) for more information about resource limits and region availability for AKS.

To allow an AKS cluster to interact with other Azure resources, a cluster identity is automatically created, since you did not specify one. Here, this cluster identity is [granted the right to pull images](#) from the Azure Container Registry (ACR) instance you created in the previous tutorial. To execute the command successfully, you're required to have an **Owner** or **Azure account administrator** role on the Azure subscription.

```
az aks create \
    --resource-group myResourceGroup \
    --name myAKSCluster \
    --node-count 2 \
    --generate-ssh-keys \
    --attach-acr <acrName>
```

To avoid needing an **Owner** or **Azure account administrator** role, you can also manually configure a service principal to pull images from ACR. For more information, see [ACR authentication with service principals](#) or [Authenticate from Kubernetes with a pull secret](#). Alternatively, you can use a [managed identity](#) instead of a service principal for easier management.

After a few minutes, the deployment completes, and returns JSON-formatted information about the AKS deployment.

NOTE

To ensure your cluster to operate reliably, you should run at least 2 (two) nodes.

Install the Kubernetes CLI

To connect to the Kubernetes cluster from your local computer, you use `kubectl`, the Kubernetes command-line client.

If you use the Azure Cloud Shell, `kubectl` is already installed. You can also install it locally using the [az aks install-cli](#) command:

```
az aks install-cli
```

Connect to cluster using kubectl

To configure `kubectl` to connect to your Kubernetes cluster, use the [az aks get-credentials](#) command. The following example gets credentials for the AKS cluster named *myAKSCluster* in the *myResourceGroup*.

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```

To verify the connection to your cluster, run the `kubectl get nodes` command to return a list of the cluster nodes:

```
$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
aks-nodepool1-37463671-vmss000000	Ready	agent	2m37s	v1.18.10
aks-nodepool1-37463671-vmss000001	Ready	agent	2m28s	v1.18.10

Next steps

In this tutorial, a Kubernetes cluster was deployed in AKS, and you configured `kubectl` to connect to it. You learned how to:

- Deploy a Kubernetes AKS cluster that can authenticate to an Azure container registry
- Install the Kubernetes CLI (`kubectl`)
- Configure `kubectl` to connect to your AKS cluster

Advance to the next tutorial to learn how to deploy an application to the cluster.

[Deploy application in Kubernetes](#)

Tutorial: Run applications in Azure Kubernetes Service (AKS)

3/5/2021 • 3 minutes to read • [Edit Online](#)

Kubernetes provides a distributed platform for containerized applications. You build and deploy your own applications and services into a Kubernetes cluster, and let the cluster manage the availability and connectivity. In this tutorial, part four of seven, a sample application is deployed into a Kubernetes cluster. You learn how to:

- Update a Kubernetes manifest file
- Run an application in Kubernetes
- Test the application

In later tutorials, this application is scaled out and updated.

This quickstart assumes a basic understanding of Kubernetes concepts. For more information, see [Kubernetes core concepts for Azure Kubernetes Service \(AKS\)](#).

Before you begin

In previous tutorials, an application was packaged into a container image, this image was uploaded to Azure Container Registry, and a Kubernetes cluster was created.

To complete this tutorial, you need the pre-created `azure-vote-all-in-one-redis.yaml` Kubernetes manifest file. This file was downloaded with the application source code in a previous tutorial. Verify that you've cloned the repo, and that you have changed directories into the cloned repo. If you haven't done these steps, and would like to follow along, start with [Tutorial 1 – Create container images](#).

This tutorial requires that you're running the Azure CLI version 2.0.53 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Update the manifest file

In these tutorials, an Azure Container Registry (ACR) instance stores the container image for the sample application. To deploy the application, you must update the image name in the Kubernetes manifest file to include the ACR login server name.

Get the ACR login server name using the `az acr list` command as follows:

```
az acr list --resource-group myResourceGroup --query "[].{acrLoginServer:loginServer}" --output table
```

The sample manifest file from the git repo cloned in the first tutorial uses the login server name of *microsoft*. Make sure that you're in the cloned *azure-voting-app-redis* directory, then open the manifest file with a text editor, such as `vi`:

```
vi azure-vote-all-in-one-redis.yaml
```

Replace *microsoft* with your ACR login server name. The image name is found on line 60 of the manifest file. The following example shows the default image name:

```
containers:
- name: azure-vote-front
  image: mcr.microsoft.com/azuredocs/azure-vote-front:v1
```

Provide your own ACR login server name so that your manifest file looks like the following example:

```
containers:
- name: azure-vote-front
  image: <acrName>.azurecr.io/azure-vote-front:v1
```

Save and close the file. In `vi`, use `:wq`.

Deploy the application

To deploy your application, use the [kubectl apply](#) command. This command parses the manifest file and creates the defined Kubernetes objects. Specify the sample manifest file, as shown in the following example:

```
kubectl apply -f azure-vote-all-in-one-redis.yaml
```

The following example output shows the resources successfully created in the AKS cluster:

```
$ kubectl apply -f azure-vote-all-in-one-redis.yaml

deployment "azure-vote-back" created
service "azure-vote-back" created
deployment "azure-vote-front" created
service "azure-vote-front" created
```

Test the application

When the application runs, a Kubernetes service exposes the application front end to the internet. This process can take a few minutes to complete.

To monitor progress, use the [kubectl get service](#) command with the `--watch` argument.

```
kubectl get service azure-vote-front --watch
```

Initially the *EXTERNAL-IP* for the *azure-vote-front* service is shown as *pending*.

```
azure-vote-front  LoadBalancer  10.0.34.242  <pending>      80:30676/TCP  5s
```

When the *EXTERNAL-IP* address changes from *pending* to an actual public IP address, use `CTRL-C` to stop the `kubectl` watch process. The following example output shows a valid public IP address assigned to the service:

```
azure-vote-front  LoadBalancer  10.0.34.242  52.179.23.131  80:30676/TCP  67s
```

To see the application in action, open a web browser to the external IP address of your service:



If the application didn't load, it might be due to an authorization problem with your image registry. To view the status of your containers, use the `kubectl get pods` command. If the container images can't be pulled, see [Authenticate with Azure Container Registry from Azure Kubernetes Service](#).

Next steps

In this tutorial, a sample Azure vote application was deployed to a Kubernetes cluster in AKS. You learned how to:

- Update a Kubernetes manifest files
- Run an application in Kubernetes
- Test the application

Advance to the next tutorial to learn how to scale a Kubernetes application and the underlying Kubernetes infrastructure.

[Scale Kubernetes application and infrastructure](#)

Tutorial: Scale applications in Azure Kubernetes Service (AKS)

4/21/2021 • 4 minutes to read • [Edit Online](#)

If you've followed the tutorials, you have a working Kubernetes cluster in AKS and you deployed the sample Azure Voting app. In this tutorial, part five of seven, you scale out the pods in the app and try pod autoscaling. You also learn how to scale the number of Azure VM nodes to change the cluster's capacity for hosting workloads. You learn how to:

- Scale the Kubernetes nodes
- Manually scale Kubernetes pods that run your application
- Configure autoscaling pods that run the app front-end

In later tutorials, the Azure Vote application is updated to a new version.

Before you begin

In previous tutorials, an application was packaged into a container image. This image was uploaded to Azure Container Registry, and you created an AKS cluster. The application was then deployed to the AKS cluster. If you haven't done these steps, and would like to follow along, start with [Tutorial 1 – Create container images](#).

This tutorial requires that you're running the Azure CLI version 2.0.53 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Manually scale pods

When the Azure Vote front-end and Redis instance were deployed in previous tutorials, a single replica was created. To see the number and state of pods in your cluster, use the `kubectl get` command as follows:

```
kubectl get pods
```

The following example output shows one front-end pod and one back-end pod:

NAME	READY	STATUS	RESTARTS	AGE
azure-vote-back-2549686872-4d2r5	1/1	Running	0	31m
azure-vote-front-848767080-tf34m	1/1	Running	0	31m

To manually change the number of pods in the `azure-vote-front` deployment, use the `kubectl scale` command. The following example increases the number of front-end pods to 5:

```
kubectl scale --replicas=5 deployment/azure-vote-front
```

Run `kubectl get pods` again to verify that AKS successfully creates the additional pods. After a minute or so, the pods are available in your cluster:

```
kubectl get pods
```

	READY	STATUS	RESTARTS	AGE
azure-vote-back-2606967446-nmpcf	1/1	Running	0	15m
azure-vote-front-3309479140-2hfh0	1/1	Running	0	3m
azure-vote-front-3309479140-bzt05	1/1	Running	0	3m
azure-vote-front-3309479140-fvcvm	1/1	Running	0	3m
azure-vote-front-3309479140-hrbf2	1/1	Running	0	15m
azure-vote-front-3309479140-qphz8	1/1	Running	0	3m

Autoscale pods

Kubernetes supports [horizontal pod autoscaling](#) to adjust the number of pods in a deployment depending on CPU utilization or other select metrics. The [Metrics Server](#) is used to provide resource utilization to Kubernetes, and is automatically deployed in AKS clusters versions 1.10 and higher. To see the version of your AKS cluster, use the `az aks show` command, as shown in the following example:

```
az aks show --resource-group myResourceGroup --name myAKSCluster --query kubernetesVersion --output table
```

NOTE

If your AKS cluster is less than 1.10, the Metrics Server is not automatically installed. Metrics Server installation manifests are available as a `components.yaml` asset on Metrics Server releases, which means you can install them via a url. To learn more about these YAML definitions, see the [Deployment](#) section of the readme.

Example installation:

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/download/v0.3.6/components.yaml
```

To use the autoscaler, all containers in your pods and your pods must have CPU requests and limits defined. In the `azure-vote-front` deployment, the front-end container already requests 0.25 CPU, with a limit of 0.5 CPU. These resource requests and limits are defined as shown in the following example snippet:

```
resources:
  requests:
    cpu: 250m
  limits:
    cpu: 500m
```

The following example uses the `kubectl autoscale` command to autoscale the number of pods in the `azure-vote-front` deployment. If average CPU utilization across all pods exceeds 50% of their requested usage, the autoscaler increases the pods up to a maximum of 10 instances. A minimum of 3 instances is then defined for the deployment:

```
kubectl autoscale deployment azure-vote-front --cpu-percent=50 --min=3 --max=10
```

Alternatively, you can create a manifest file to define the autoscaler behavior and resource limits. The following is an example of a manifest file named `azure-vote-hpa.yaml`.

```

apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: azure-vote-back-hpa
spec:
  maxReplicas: 10 # define max replica count
  minReplicas: 3 # define min replica count
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: azure-vote-back
  targetCPUUtilizationPercentage: 50 # target CPU utilization

---

apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: azure-vote-front-hpa
spec:
  maxReplicas: 10 # define max replica count
  minReplicas: 3 # define min replica count
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: azure-vote-front
  targetCPUUtilizationPercentage: 50 # target CPU utilization

```

Use `kubectl apply` to apply the autoscaler defined in the `azure-vote-hpa.yaml` manifest file.

```
kubectl apply -f azure-vote-hpa.yaml
```

To see the status of the autoscaler, use the `kubectl get hpa` command as follows:

```
kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
azure-vote-front	Deployment/azure-vote-front	0% / 50%	3	10	3	2m

After a few minutes, with minimal load on the Azure Vote app, the number of pod replicas decreases automatically to three. You can use `kubectl get pods` again to see the unneeded pods being removed.

Manually scale AKS nodes

If you created your Kubernetes cluster using the commands in the previous tutorial, it has two nodes. You can adjust the number of nodes manually if you plan more or fewer container workloads on your cluster.

The following example increases the number of nodes to three in the Kubernetes cluster named `myAKSCluster`. The command takes a couple of minutes to complete.

```
az aks scale --resource-group myResourceGroup --name myAKSCluster --node-count 3
```

When the cluster has successfully scaled, the output is similar to following example:

```
"agentPoolProfiles": [
  {
    "count": 3,
    "dnsPrefix": null,
    "fqdn": null,
    "name": "myAKSCluster",
    "osDiskSizeGb": null,
    "osType": "Linux",
    "ports": null,
    "storageProfile": "ManagedDisks",
    "vmSize": "Standard_D2_v2",
    "vnetSubnetId": null
  }
]
```

Next steps

In this tutorial, you used different scaling features in your Kubernetes cluster. You learned how to:

- Manually scale Kubernetes pods that run your application
- Configure autoscaling pods that run the app front-end
- Manually scale the Kubernetes nodes

Advance to the next tutorial to learn how to update application in Kubernetes.

[Update an application in Kubernetes](#)

Tutorial: Update an application in Azure Kubernetes Service (AKS)

3/5/2021 • 4 minutes to read • [Edit Online](#)

After an application has been deployed in Kubernetes, it can be updated by specifying a new container image or image version. An update is staged so that only a portion of the deployment is updated at the same time. This staged update enables the application to keep running during the update. It also provides a rollback mechanism if a deployment failure occurs.

In this tutorial, part six of seven, the sample Azure Vote app is updated. You learn how to:

- Update the front-end application code
- Create an updated container image
- Push the container image to Azure Container Registry
- Deploy the updated container image

Before you begin

In previous tutorials, an application was packaged into a container image. This image was uploaded to Azure Container Registry, and you created an AKS cluster. The application was then deployed to the AKS cluster.

An application repository was also cloned that includes the application source code, and a pre-created Docker Compose file used in this tutorial. Verify that you've created a clone of the repo, and have changed directories into the cloned directory. If you haven't completed these steps, and want to follow along, start with [Tutorial 1 – Create container images](#).

This tutorial requires that you're running the Azure CLI version 2.0.53 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Update an application

Let's make a change to the sample application, then update the version already deployed to your AKS cluster. Make sure that you're in the cloned `azure-voting-app-redis` directory. The sample application source code can then be found inside the `azure-vote` directory. Open the `config_file.cfg` file with an editor, such as `vi`:

```
vi azure-vote/azure-vote/config_file.cfg
```

Change the values for `VOTE1VALUE` and `VOTE2VALUE` to different values, such as colors. The following example shows the updated values:

```
# UI Configurations
TITLE = 'Azure Voting App'
VOTE1VALUE = 'Blue'
VOTE2VALUE = 'Purple'
SHOWHOST = 'false'
```

Save and close the file. In `vi`, use `:wq`.

Update the container image

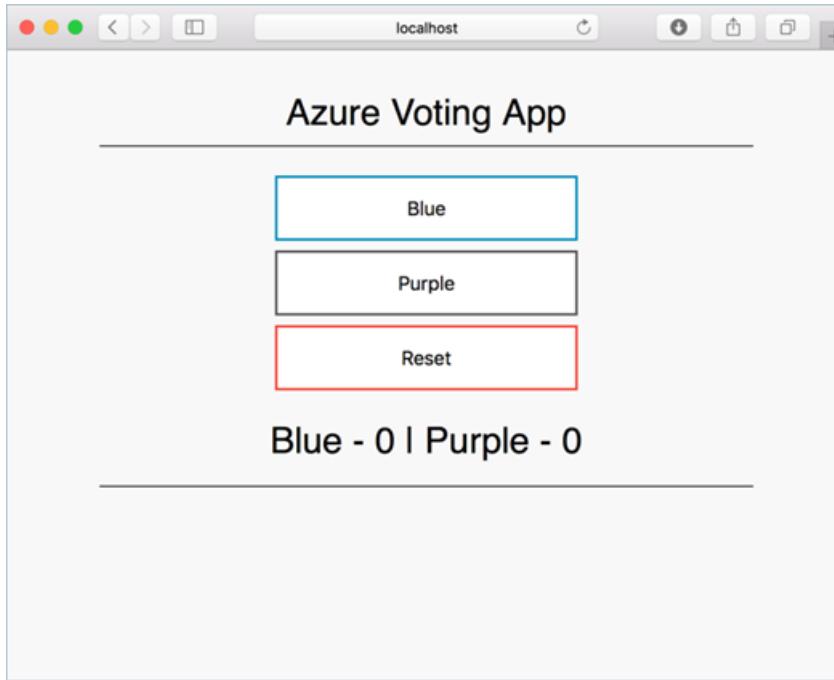
To re-create the front-end image and test the updated application, use `docker-compose`. The `--build` argument is used to instruct Docker Compose to re-create the application image:

```
docker-compose up --build -d
```

Test the application locally

To verify that the updated container image shows your changes, open a local web browser to

```
http://localhost:8080 .
```



The updated values provided in the `config_file.cfg` file are displayed in your running application.

Tag and push the image

To correctly use the updated image, tag the `azure-vote-front` image with the login server name of your ACR registry. Get the login server name with the `az acr list` command:

```
az acr list --resource-group myResourceGroup --query "[].{acrLoginServer:loginServer}" --output table
```

Use `docker tag` to tag the image. Replace `<acrLoginServer>` with your ACR login server name or public registry hostname, and update the image version to `:v2` as follows:

```
docker tag mcr.microsoft.com/azuredocs/azure-vote-front:v1 <acrLoginServer>/azure-vote-front:v2
```

Now use `docker push` to upload the image to your registry. Replace `<acrLoginServer>` with your ACR login server name.

NOTE

If you experience issues pushing to your ACR registry, make sure that you are still logged in. Run the `az acr login` command using the name of your Azure Container Registry that you created in the [Create an Azure Container Registry](#) step. For example, `az acr login --name <azure container registry name>`.

```
docker push <acrLoginServer>/azure-vote-front:v2
```

Deploy the updated application

To provide maximum uptime, multiple instances of the application pod must be running. Verify the number of running front-end instances with the [kubectl get pods](#) command:

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
azure-vote-back-217588096-5w632	1/1	Running	0	10m
azure-vote-front-233282510-b5pkz	1/1	Running	0	10m
azure-vote-front-233282510-dhrtr	1/1	Running	0	10m
azure-vote-front-233282510-pqbfk	1/1	Running	0	10m

If you don't have multiple front-end pods, scale the *azure-vote-front* deployment as follows:

```
kubectl scale --replicas=3 deployment/azure-vote-front
```

To update the application, use the [kubectl set](#) command. Update `<acrLoginServer>` with the login server or host name of your container registry, and specify the *v2* application version:

```
kubectl set image deployment azure-vote-front azure-vote-front=<acrLoginServer>/azure-vote-front:v2
```

To monitor the deployment, use the [kubectl get pod](#) command. As the updated application is deployed, your pods are terminated and re-created with the new container image.

```
kubectl get pods
```

The following example output shows pods terminating and new instances running as the deployment progresses:

```
$ kubectl get pods
```

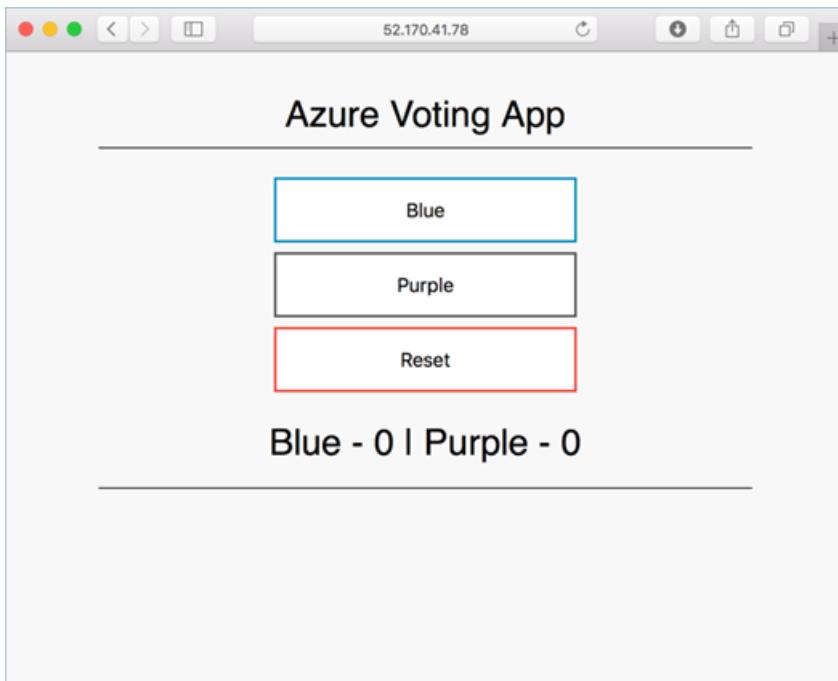
NAME	READY	STATUS	RESTARTS	AGE
azure-vote-back-2978095810-gq9g0	1/1	Running	0	5m
azure-vote-front-1297194256-tpjlg	1/1	Running	0	1m
azure-vote-front-1297194256-tptnx	1/1	Running	0	5m
azure-vote-front-1297194256-zktw9	1/1	Terminating	0	1m

Test the updated application

To view the update application, first get the external IP address of the `azure-vote-front` service:

```
kubectl get service azure-vote-front
```

Now open a web browser to the IP address of your service:



Next steps

In this tutorial, you updated an application and rolled out this update to your AKS cluster. You learned how to:

- Update the front-end application code
- Create an updated container image
- Push the container image to Azure Container Registry
- Deploy the updated container image

Advance to the next tutorial to learn how to upgrade an AKS cluster to a new version of Kubernetes.

[Upgrade Kubernetes](#)

Tutorial: Upgrade Kubernetes in Azure Kubernetes Service (AKS)

4/21/2021 • 3 minutes to read • [Edit Online](#)

As part of the application and cluster lifecycle, you may wish to upgrade to the latest available version of Kubernetes and use new features. An Azure Kubernetes Service (AKS) cluster can be upgraded using the Azure CLI.

In this tutorial, part seven of seven, a Kubernetes cluster is upgraded. You learn how to:

- Identify current and available Kubernetes versions
- Upgrade the Kubernetes nodes
- Validate a successful upgrade

Before you begin

In previous tutorials, an application was packaged into a container image. This image was uploaded to Azure Container Registry, and you created an AKS cluster. The application was then deployed to the AKS cluster. If you have not done these steps, and would like to follow along, start with [Tutorial 1 – Create container images](#).

This tutorial requires that you are running the Azure CLI version 2.0.53 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Get available cluster versions

Before you upgrade a cluster, use the `az aks get-upgrades` command to check which Kubernetes releases are available for upgrade:

```
az aks get-upgrades --resource-group myResourceGroup --name myAKSCluster
```

In the following example, the current version is *1.18.10*, and the available versions are shown under *upgrades*.

```
{
  "agentPoolProfiles": null,
  "controlPlaneProfile": {
    "kubernetesVersion": "1.18.10",
    ...
  },
  "upgrades": [
    {
      "isPreview": null,
      "kubernetesVersion": "1.19.1"
    },
    {
      "isPreview": null,
      "kubernetesVersion": "1.19.3"
    }
  ],
  ...
}
```

Upgrade a cluster

To minimize disruption to running applications, AKS nodes are carefully cordoned and drained. In this process, the following steps are performed:

1. The Kubernetes scheduler prevents additional pods being scheduled on a node that is to be upgraded.
2. Running pods on the node are scheduled on other nodes in the cluster.
3. A node is created that runs the latest Kubernetes components.
4. When the new node is ready and joined to the cluster, the Kubernetes scheduler begins to run pods on it.
5. The old node is deleted, and the next node in the cluster begins the cordon and drain process.

Use the [az aks upgrade](#) command to upgrade the AKS cluster.

```
az aks upgrade \
--resource-group myResourceGroup \
--name myAKSCluster \
--kubernetes-version KUBERNETES_VERSION
```

NOTE

You can only upgrade one minor version at a time. For example, you can upgrade from *1.14.x* to *1.15.x*, but cannot upgrade from *1.14.x* to *1.16.x* directly. To upgrade from *1.14.x* to *1.16.x*, first upgrade from *1.14.x* to *1.15.x*, then perform another upgrade from *1.15.x* to *1.16.x*.

The following condensed example output shows the result of upgrading to *1.19.1*. Notice the *kubernetesVersion* now reports *1.19.1*:

```
{
  "agentPoolProfiles": [
    {
      "count": 3,
      "maxPods": 110,
      "name": "nodepool1",
      "osType": "Linux",
      "storageProfile": "ManagedDisks",
      "vmSize": "Standard_DS1_v2",
    }
  ],
  "dnsPrefix": "myAKSclust-myResourceGroup-19da35",
  "enableRbac": false,
  "fqdn": "myaksclust-myresourcegroup-19da35-bd54a4be.hcp.eastus.azmk8s.io",
  "id": "/subscriptions/<Subscription
ID>/resourcegroups/myResourceGroup/providers/Microsoft.ContainerService/managedClusters/myAKSCluster",
  "kubernetesVersion": "1.19.1",
  "location": "eastus",
  "name": "myAKSCluster",
  "type": "Microsoft.ContainerService/ManagedClusters"
}
```

Validate an upgrade

Confirm that the upgrade was successful using the [az aks show](#) command as follows:

```
az aks show --resource-group myResourceGroup --name myAKSCluster --output table
```

The following example output shows the AKS cluster runs *KubernetesVersion 1.19.1*:

Name	Location	ResourceGroup	KubernetesVersion	ProvisioningState	Fqdn
myAKSCluster	eastus	myResourceGroup	1.19.1	Succeeded	myaksclust-
			myresourcegroup-19da35-bd54a4be.hcp.eastus.azmk8s.io		

Delete the cluster

As this tutorial is the last part of the series, you may want to delete the AKS cluster. As the Kubernetes nodes run on Azure virtual machines (VMs), they continue to incur charges even if you don't use the cluster. Use the [az group delete](#) command to remove the resource group, container service, and all related resources.

```
az group delete --name myResourceGroup --yes --no-wait
```

NOTE

When you delete the cluster, the Azure Active Directory service principal used by the AKS cluster is not removed. For steps on how to remove the service principal, see [AKS service principal considerations and deletion](#). If you used a managed identity, the identity is managed by the platform and does not require you to provision or rotate any secrets.

Next steps

In this tutorial, you upgraded Kubernetes in an AKS cluster. You learned how to:

- Identify current and available Kubernetes versions
- Upgrade the Kubernetes nodes
- Validate a successful upgrade

For more information on AKS, see [AKS overview](#). For guidance on creating full solutions with AKS, see [AKS solution guidance](#).

Kubernetes core concepts for Azure Kubernetes Service (AKS)

4/8/2021 • 14 minutes to read • [Edit Online](#)

Application development continues to move toward a container-based approach, increasing our need to orchestrate and manage resources. As the leading platform, Kubernetes provides reliable scheduling of fault-tolerant application workloads. Azure Kubernetes Service (AKS), a managed Kubernetes offering, further simplifies container-based application deployment and management.

This article introduces:

- Core Kubernetes infrastructure components:
 - *control plane*
 - *nodes*
 - *node pools*
- Workload resources:
 - *pods*
 - *deployments*
 - *sets*
- How to group resources into *namespaces*.

What is Kubernetes?

Kubernetes is a rapidly evolving platform that manages container-based applications and their associated networking and storage components. Kubernetes focuses on the application workloads, not the underlying infrastructure components. Kubernetes provides a declarative approach to deployments, backed by a robust set of APIs for management operations.

You can build and run modern, portable, microservices-based applications, using Kubernetes to orchestrate and manage the availability of the application components. Kubernetes supports both stateless and stateful applications as teams progress through the adoption of microservices-based applications.

As an open platform, Kubernetes allows you to build your applications with your preferred programming language, OS, libraries, or messaging bus. Existing continuous integration and continuous delivery (CI/CD) tools can integrate with Kubernetes to schedule and deploy releases.

AKS provides a managed Kubernetes service that reduces the complexity of deployment and core management tasks, like upgrade coordination. The Azure platform manages the AKS control plane, and you only pay for the AKS nodes that run your applications. AKS is built on top of the open-source Azure Kubernetes Service Engine: [aks-engine](#).

Kubernetes cluster architecture

A Kubernetes cluster is divided into two components:

- *Control plane*: provides the core Kubernetes services and orchestration of application workloads.
- *Nodes*: run your application workloads.



Control plane

When you create an AKS cluster, a control plane is automatically created and configured. This control plane is provided at no cost as a managed Azure resource abstracted from the user. You only pay for the nodes attached to the AKS cluster. The control plane and its resources reside only on the region where you created the cluster.

The control plane includes the following core Kubernetes components:

COMPONENT	DESCRIPTION
<i>kube-apiserver</i>	The API server is how the underlying Kubernetes APIs are exposed. This component provides the interaction for management tools, such as <code>kubectl</code> or the Kubernetes dashboard.
<i>etcd</i>	To maintain the state of your Kubernetes cluster and configuration, the highly available <i>etcd</i> is a key value store within Kubernetes.
<i>kube-scheduler</i>	When you create or scale applications, the Scheduler determines what nodes can run the workload and starts them.
<i>kube-controller-manager</i>	The Controller Manager oversees a number of smaller Controllers that perform actions such as replicating pods and handling node operations.

AKS provides a single-tenant control plane, with a dedicated API server, scheduler, etc. You define the number and size of the nodes, and the Azure platform configures the secure communication between the control plane and nodes. Interaction with the control plane occurs through Kubernetes APIs, such as `kubectl` or the Kubernetes dashboard.

While you don't need to configure components (like a highly available *etcd* store) with this managed control plane, you can't access the control plane directly. Kubernetes control plane and node upgrades are orchestrated through the Azure CLI or Azure portal. To troubleshoot possible issues, you can review the control plane logs through Azure Monitor logs.

To configure or directly access a control plane, deploy your own Kubernetes cluster using [aks-engine](#).

For associated best practices, see [Best practices for cluster security and upgrades in AKS](#).

Nodes and node pools

To run your applications and supporting services, you need a Kubernetes *node*. An AKS cluster has at least one node, an Azure virtual machine (VM) that runs the Kubernetes node components and container runtime.

COMPONENT	DESCRIPTION
<code>kubelet</code>	The Kubernetes agent that processes the orchestration requests from the control plane and scheduling of running the requested containers.
<code>kube-proxy</code>	Handles virtual networking on each node. The proxy routes network traffic and manages IP addressing for services and pods.
<code>container runtime</code>	Allows containerized applications to run and interact with additional resources, such as the virtual network and storage. AKS clusters using Kubernetes version 1.19+ node pools use <code>containerd</code> as their container runtime. AKS clusters using Kubernetes prior to node pool version 1.19 for node pools use <code>Moby</code> (upstream docker) as their container runtime.



The Azure VM size for your nodes defines the storage CPUs, memory, size, and type available (such as high-performance SSD or regular HDD). Plan the node size around whether your applications may require large amounts of CPU and memory or high-performance storage. Scale out the number of nodes in your AKS cluster to meet demand.

In AKS, the VM image for your cluster's nodes is based on Ubuntu Linux or Windows Server 2019. When you create an AKS cluster or scale out the number of nodes, the Azure platform automatically creates and configures the requested number of VMs. Agent nodes are billed as standard VMs, so any VM size discounts (including [Azure reservations](#)) are automatically applied.

Deploy your own Kubernetes cluster with `aks-engine` if using a different host OS, container runtime, or including different custom packages. The upstream `aks-engine` releases features and provides configuration options ahead of support in AKS clusters. So, if you wish to use a container runtime other than `containerd` or `Moby`, you can run `aks-engine` to configure and deploy a Kubernetes cluster that meets your current needs.

Resource reservations

AKS uses node resources to help the node function as part of your cluster. This usage can create a discrepancy between your node's total resources and the allocatable resources in AKS. Remember this information when setting requests and limits for user deployed pods.

To find a node's allocatable resources, run:

```
kubectl describe node [NODE_NAME]
```

To maintain node performance and functionality, AKS reserves resources on each node. As a node grows larger in resources, the resource reservation grows due to a higher need for management of user-deployed pods.

NOTE

Using AKS add-ons such as Container Insights (OMS) will consume additional node resources.

Two types of resources are reserved:

- **CPU**

Reserved CPU is dependent on node type and cluster configuration, which may cause less allocatable CPU due to running additional features.

CPU CORES ON HOST	1	2	4	8	16	32	64
Kube-reserved (millicores)	60	100	140	180	260	420	740

- **Memory**

Memory utilized by AKS includes the sum of two values.

1. **kubelet daemon**

The `kubelet` daemon is installed on all Kubernetes agent nodes to manage container creation and termination.

By default on AKS, `kubelet` daemon has the `memory.available<750Mi` eviction rule, ensuring a node must always have at least 750 Mi allocatable at all times. When a host is below that available memory threshold, the `kubelet` will trigger to terminate one of the running pods and free up memory on the host machine.

2. **A regressive rate of memory reservations** for the kubelet daemon to properly function (*kube-reserved*).

- 25% of the first 4 GB of memory
- 20% of the next 4 GB of memory (up to 8 GB)
- 10% of the next 8 GB of memory (up to 16 GB)
- 6% of the next 112 GB of memory (up to 128 GB)
- 2% of any memory above 128 GB

Memory and CPU allocation rules:

- Keep agent nodes healthy, including some hosting system pods critical to cluster health.
- Cause the node to report less allocatable memory and CPU than it would if it were not part of a Kubernetes cluster.

The above resource reservations can't be changed.

For example, if a node offers 7 GB, it will report 34% of memory not allocatable including the 750Mi hard eviction threshold.

$$0.75 + (0.25*4) + (0.20*3) = 0.75\text{GB} + 1\text{GB} + 0.6\text{GB} = 2.35\text{GB} / 7\text{GB} = 33.57\% \text{ reserved}$$

In addition to reservations for Kubernetes itself, the underlying node OS also reserves an amount of CPU and memory resources to maintain OS functions.

For associated best practices, see [Best practices for basic scheduler features in AKS](#).

Node pools

Nodes of the same configuration are grouped together into *node pools*. A Kubernetes cluster contains at least one node pool. The initial number of nodes and size are defined when you create an AKS cluster, which creates a *default node pool*. This default node pool in AKS contains the underlying VMs that run your agent nodes.

NOTE

To ensure your cluster operates reliably, you should run at least two (2) nodes in the default node pool.

You scale or upgrade an AKS cluster against the default node pool. You can choose to scale or upgrade a specific node pool. For upgrade operations, running containers are scheduled on other nodes in the node pool until all the nodes are successfully upgraded.

For more information about how to use multiple node pools in AKS, see [Create and manage multiple node pools for a cluster in AKS](#).

Node selectors

In an AKS cluster with multiple node pools, you may need to tell the Kubernetes Scheduler which node pool to use for a given resource. For example, ingress controllers shouldn't run on Windows Server nodes.

Node selectors let you define various parameters, like node OS, to control where a pod should be scheduled.

The following basic example schedules an NGINX instance on a Linux node using the node selector `"beta.kubernetes.io/os": "linux"`.

```
kind: Pod
apiVersion: v1
metadata:
  name: nginx
spec:
  containers:
    - name: myfrontend
      image: mcr.microsoft.com/oss/nginx/nginx:1.15.12-alpine
  nodeSelector:
    "beta.kubernetes.io/os": linux
```

For more information on how to control where pods are scheduled, see [Best practices for advanced scheduler features in AKS](#).

Pods

Kubernetes uses *pods* to run an instance of your application. A pod represents a single instance of your application.

Pods typically have a 1:1 mapping with a container. In advanced scenarios, a pod may contain multiple containers. Multi-container pods are scheduled together on the same node, and allow containers to share related resources.

When you create a pod, you can define *resource requests* to request a certain amount of CPU or memory resources. The Kubernetes Scheduler tries meet the request by scheduling the pods to run on a node with available resources. You can also specify maximum resource limits to prevent a pod from consuming too much compute resource from the underlying node. Best practice is to include resource limits for all pods to help the Kubernetes Scheduler identify necessary, permitted resources.

For more information, see [Kubernetes pods](#) and [Kubernetes pod lifecycle](#).

A pod is a logical resource, but application workloads run on the containers. Pods are typically ephemeral,

disposable resources. Individually scheduled pods miss some of the high availability and redundancy Kubernetes features. Instead, pods are deployed and managed by Kubernetes *Controllers*, such as the Deployment Controller.

Deployments and YAML manifests

A *deployment* represents identical pods managed by the Kubernetes Deployment Controller. A deployment defines the number of pod *replicas* to create. The Kubernetes Scheduler ensures that additional pods are scheduled on healthy nodes if pods or nodes encounter problems.

You can update deployments to change the configuration of pods, container image used, or attached storage. The Deployment Controller:

- Drains and terminates a given number of replicas.
- Creates replicas from the new deployment definition.
- Continues the process until all replicas in the deployment are updated.

Most stateless applications in AKS should use the deployment model rather than scheduling individual pods. Kubernetes can monitor deployment health and status to ensure that the required number of replicas run within the cluster. When scheduled individually, pods aren't restarted if they encounter a problem, and aren't rescheduled on healthy nodes if their current node encounters a problem.

You don't want to disrupt management decisions with an update process if your application requires a minimum number of available instances. *Pod Disruption Budgets* define how many replicas in a deployment can be taken down during an update or node upgrade. For example, if you have *five (5)* replicas in your deployment, you can define a pod disruption of *4 (four)* to only allow one replica to be deleted or rescheduled at a time. As with pod resource limits, best practice is to define pod disruption budgets on applications that require a minimum number of replicas to always be present.

Deployments are typically created and managed with `kubectl create` or `kubectl apply`. Create a deployment by defining a manifest file in the YAML format.

The following example creates a basic deployment of the NGINX web server. The deployment specifies *three (3)* replicas to be created, and requires port *80* to be open on the container. Resource requests and limits are also defined for CPU and memory.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: mcr.microsoft.com/oss/nginx/nginx:1.15.2-alpine
          ports:
            - containerPort: 80
          resources:
            requests:
              cpu: 250m
              memory: 64Mi
            limits:
              cpu: 500m
              memory: 256Mi
```

More complex applications can be created by including services (such as load balancers) within the YAML manifest.

For more information, see [Kubernetes deployments](#).

Package management with Helm

[Helm](#) is commonly used to manage applications in Kubernetes. You can deploy resources by building and using existing public Helm *charts* that contain a packaged version of application code and Kubernetes YAML manifests. You can store Helm charts either locally or in a remote repository, such as an [Azure Container Registry Helm chart repo](#).

To use Helm, install the Helm client on your computer, or use the Helm client in the [Azure Cloud Shell](#). Search for or create Helm charts, and then install them to your Kubernetes cluster. For more information, see [Install existing applications with Helm in AKS](#).

StatefulSets and DaemonSets

Using the Kubernetes Scheduler, the Deployment Controller runs replicas on any available node with available resources. While this approach may be sufficient for stateless applications, The Deployment Controller is not ideal for applications that require:

- A persistent naming convention or storage.
- A replica to exist on each select node within a cluster.

Two Kubernetes resources, however, let you manage these types of applications:

- *StatefulSets* maintain the state of applications beyond an individual pod lifecycle, such as storage.
- *DaemonSets* ensure a running instance on each node, early in the Kubernetes bootstrap process.

StatefulSets

Modern application development often aims for stateless applications. For stateful applications, like those that include database components, you can use *StatefulSets*. Like deployments, a StatefulSet creates and manages at least one identical pod. Replicas in a StatefulSet follow a graceful, sequential approach to deployment, scale,

upgrade, and termination. The naming convention, network names, and storage persist as replicas are rescheduled with a StatefulSet.

Define the application in YAML format using `kind: StatefulSet`. From there, the StatefulSet Controller handles the deployment and management of the required replicas. Data is written to persistent storage, provided by Azure Managed Disks or Azure Files. With StatefulSets, the underlying persistent storage remains, even when the StatefulSet is deleted.

For more information, see [Kubernetes StatefulSets](#).

Replicas in a StatefulSet are scheduled and run across any available node in an AKS cluster. To ensure at least one pod in your set runs on a node, you use a DaemonSet instead.

DaemonSets

For specific log collection or monitoring, you may need to run a pod on all, or selected, nodes. You can use `DaemonSet` deploy one or more identical pods, but the DaemonSet Controller ensures that each node specified runs an instance of the pod.

The DaemonSet Controller can schedule pods on nodes early in the cluster boot process, before the default Kubernetes scheduler has started. This ability ensures that the pods in a DaemonSet are started before traditional pods in a Deployment or StatefulSet are scheduled.

Like StatefulSets, a DaemonSet is defined as part of a YAML definition using `kind: DaemonSet`.

For more information, see [Kubernetes DaemonSets](#).

NOTE

If using the [Virtual Nodes add-on](#), DaemonSets will not create pods on the virtual node.

Namespaces

Kubernetes resources, such as pods and deployments, are logically grouped into a *namespace* to divide an AKS cluster and restrict create, view, or manage access to resources. For example, you can create namespaces to separate business groups. Users can only interact with resources within their assigned namespaces.



When you create an AKS cluster, the following namespaces are available:

NAMESPACE	DESCRIPTION
-----------	-------------

NAMESPACE	DESCRIPTION
<i>default</i>	Where pods and deployments are created by default when none is provided. In smaller environments, you can deploy applications directly into the default namespace without creating additional logical separations. When you interact with the Kubernetes API, such as with <code>kubectl get pods</code> , the default namespace is used when none is specified.
<i>kube-system</i>	Where core resources exist, such as network features like DNS and proxy, or the Kubernetes dashboard. You typically don't deploy your own applications into this namespace.
<i>kube-public</i>	Typically not used, but can be used for resources to be visible across the whole cluster, and can be viewed by any user.

For more information, see [Kubernetes namespaces](#).

Next steps

This article covers some of the core Kubernetes components and how they apply to AKS clusters. For more information on core Kubernetes and AKS concepts, see the following articles:

- [Kubernetes / AKS access and identity](#)
- [Kubernetes / AKS security](#)
- [Kubernetes / AKS virtual networks](#)
- [Kubernetes / AKS storage](#)
- [Kubernetes / AKS scale](#)

Access and identity options for Azure Kubernetes Service (AKS)

5/12/2021 • 14 minutes to read • [Edit Online](#)

You can authenticate, authorize, secure, and control access to Kubernetes clusters in a variety of ways.

- Using Kubernetes role-based access control (Kubernetes RBAC), you can grant users, groups, and service accounts access to only the resources they need.
- With Azure Kubernetes Service (AKS), you can further enhance the security and permissions structure via Azure Active Directory and Azure RBAC.

Kubernetes RBAC and AKS help you secure your cluster access and provide only the minimum required permissions to developers and operators.

This article introduces the core concepts that help you authenticate and assign permissions in AKS.

AKS service permissions

When creating a cluster, AKS generates or modifies resources it needs (like VMs and NICs) to create and run the cluster on behalf of the user. This identity is distinct from the cluster's identity permission, which is created during cluster creation.

Identity creating and operating the cluster permissions

The following permissions are needed by the identity creating and operating the cluster.

PERMISSION	REASON
<code>Microsoft.Compute/diskEncryptionSets/read</code>	Required to read disk encryption set ID.
<code>Microsoft.Compute/proximityPlacementGroups/write</code>	Required for updating proximity placement groups.
<code>Microsoft.Network/applicationGateways/read</code> <code>Microsoft.Network/applicationGateways/write</code> <code>Microsoft.Network/virtualNetworks/subnets/join/action</code>	Required to configure application gateways and join the subnet.
<code>Microsoft.Network/virtualNetworks/subnets/join/action</code>	Required to configure the Network Security Group for the subnet when using a custom VNET.
<code>Microsoft.Network/publicIPAddresses/join/action</code> <code>Microsoft.Network/publicIPPrefixes/join/action</code>	Required to configure the outbound public IPs on the Standard Load Balancer.
<code>Microsoft.OperationalInsights/workspaces/sharedkeys/read</code> <code>Microsoft.OperationalInsights/workspaces/read</code> <code>Microsoft.OperationsManagement/solutions/write</code> <code>Microsoft.OperationsManagement/solutions/read</code> <code>Microsoft.ManagedIdentity/userAssignedIdentities/assign/action</code>	Required to create and update Log Analytics workspaces and Azure monitoring for containers.

AKS cluster identity permissions

The following permissions are used by the AKS cluster identity, which is created and associated with the AKS cluster. Each permission is used for the reasons below:

PERMISSION	REASON
Microsoft.ContainerService/managedClusters/*	Required for creating users and operating the cluster
Microsoft.Network/loadBalancers/delete Microsoft.Network/loadBalancers/read Microsoft.Network/loadBalancers/write	Required to configure the load balancer for a LoadBalancer service.
Microsoft.Network/publicIPAddresses/delete Microsoft.Network/publicIPAddresses/read Microsoft.Network/publicIPAddresses/write	Required to find and configure public IPs for a LoadBalancer service.
Microsoft.Network/publicIPAddresses/join/action	Required for configuring public IPs for a LoadBalancer service.
Microsoft.Network/networkSecurityGroups/read Microsoft.Network/networkSecurityGroups/write	Required to create or delete security rules for a LoadBalancer service.
Microsoft.Compute/disks/delete Microsoft.Compute/disks/read Microsoft.Compute/disks/write Microsoft.Compute/locations/DiskOperations/read	Required to configure AzureDisks.
Microsoft.Storage/storageAccounts/delete Microsoft.Storage/storageAccounts/listKeys/action Microsoft.Storage/storageAccounts/read Microsoft.Storage/storageAccounts/write Microsoft.Storage/operations/read	Required to configure storage accounts for AzureFile or AzureDisk.
Microsoft.Network/routeTables/read Microsoft.Network/routeTables/routes/delete Microsoft.Network/routeTables/routes/read Microsoft.Network/routeTables/routes/write Microsoft.Network/routeTables/write	Required to configure route tables and routes for nodes.
Microsoft.Compute/virtualMachines/read	Required to find information for virtual machines in a VMAS, such as zones, fault domain, size, and data disks.
Microsoft.Compute/virtualMachines/write	Required to attach AzureDisks to a virtual machine in a VMAS.
Microsoft.Compute/virtualMachineScaleSets/read Microsoft.Compute/virtualMachineScaleSets/virtualMachines/machine Microsoft.Compute/virtualMachineScaleSets/virtualmachines/instanceView/read	Required to find information for virtual machines in a virtual machine scale set, such as zones, fault domain, size, and data disks.
Microsoft.Network/networkInterfaces/write	Required to add a virtual machine in a VMAS to a load balancer backend address pool.
Microsoft.Compute/virtualMachineScaleSets/write	Required to add a virtual machine scale set to a load balancer backend address pools and scale out nodes in a virtual machine scale set.
Microsoft.Compute/virtualMachineScaleSets/virtualmachines/required	Required to attach AzureDisks and add a virtual machine from a virtual machine scale set to the load balancer.

PERMISSION	REASON
Microsoft.Network/networkInterfaces/read	Required to search internal IPs and load balancer backend address pools for virtual machines in a VMAS.
Microsoft.Compute/virtualMachineScaleSets/virtualMachines/read	Required to search internal IPs and load balancer backend address pools for a virtual machine in a virtual machine scale set.
Microsoft.Compute/virtualMachineScaleSets/virtualMachines/read Microsoft.Network/publicIPs/read Microsoft.Network/publicIPs/write	Required to find public IP for configuration machine in a virtual machine scale set.
Microsoft.Network/virtualNetworks/read Microsoft.Network/virtualNetworks/subnets/read	Required to verify if a subnet exists for the internal load balancer in another resource group.
Microsoft.Compute/snapshots/delete Microsoft.Compute/snapshots/read Microsoft.Compute/snapshots/write	Required to configure snapshots for AzureDisk.
Microsoft.Compute/locations/vmSizes/read Microsoft.Compute/locations/operations/read	Required to find virtual machine sizes for finding AzureDisk volume limits.

Additional cluster identity permissions

When creating a cluster with specific attributes, you will need the following additional permissions for the cluster identity. Since these permissions are not automatically assigned, you must add them to the cluster identity after it's created.

PERMISSION	REASON
Microsoft.Network/networkSecurityGroups/write Microsoft.Network/networkSecurityGroups/read	Required if using a network security group in another resource group. Required to configure security rules for a LoadBalancer service.
Microsoft.Network/virtualNetworks/subnets/read Microsoft.Network/virtualNetworks/subnets/join/action	Required if using a subnet in another resource group such as a custom VNET.
Microsoft.Network/routeTables/routes/read Microsoft.Network/routeTables/routes/write	Required if using a subnet associated with a route table in another resource group such as a custom VNET with a custom route table. Required to verify if a subnet already exists for the subnet in the other resource group.
Microsoft.Network/virtualNetworks/subnets/read	Required if using an internal load balancer in another resource group. Required to verify if a subnet already exists for the internal load balancer in the resource group.
Microsoft.Network/privateDnszones/*	Required if using a private DNS zone in another resource group such as a custom privateDNSZone.

Kubernetes RBAC

Kubernetes RBAC provides granular filtering of user actions. With this control mechanism:

- You assign users or user groups permission to create and modify resources or view logs from running application workloads.
- You can scope permissions to a single namespace or across the entire AKS cluster.

- You create *roles* to define permissions, and then assign those roles to users with *role bindings*.

For more information, see [Using Kubernetes RBAC authorization](#).

Roles and ClusterRoles

Roles

Before assigning permissions to users with Kubernetes RBAC, you'll define user permissions as a *Role*. Grant permissions within a namespace using roles.

NOTE

Kubernetes roles *grant* permissions; they don't *deny* permissions.

To grant permissions across the entire cluster or to cluster resources outside a given namespace, you can instead use *ClusterRoles*.

ClusterRoles

A ClusterRole grants and applies permissions to resources across the entire cluster, not a specific namespace.

RoleBindings and ClusterRoleBindings

Once you've defined roles to grant permissions to resources, you assign those Kubernetes RBAC permissions with a *RoleBinding*. If your AKS cluster [integrates with Azure Active Directory \(Azure AD\)](#), RoleBindings grant permissions to Azure AD users to perform actions within the cluster. See how in [Control access to cluster resources using Kubernetes role-based access control and Azure Active Directory identities](#).

RoleBindings

Assign roles to users for a given namespace using RoleBindings. With RoleBindings, you can logically segregate a single AKS cluster, only enabling users to access the application resources in their assigned namespace.

To bind roles across the entire cluster, or to cluster resources outside a given namespace, you instead use *ClusterRoleBindings*.

ClusterRoleBinding

With a ClusterRoleBinding, you bind roles to users and apply to resources across the entire cluster, not a specific namespace. This approach lets you grant administrators or support engineers access to all resources in the AKS cluster.

NOTE

Microsoft/AKS performs any cluster actions with user consent under a built-in Kubernetes role `aks-service` and built-in role binding `aks-service-rolebinding`.

This role enables AKS to troubleshoot and diagnose cluster issues, but can't modify permissions nor create roles or role bindings, or other high privilege actions. Role access is only enabled under active support tickets with just-in-time (JIT) access. Read more about [AKS support policies](#).

Kubernetes service accounts

Service accounts are one of the primary user types in Kubernetes. The Kubernetes API holds and manages service accounts. Service account credentials are stored as Kubernetes secrets, allowing them to be used by authorized pods to communicate with the API Server. Most API requests provide an authentication token for a service account or a normal user account.

Normal user accounts allow more traditional access for human administrators or developers, not just services and processes. While Kubernetes doesn't provide an identity management solution to store regular user accounts and passwords, you can integrate external identity solutions into Kubernetes. For AKS clusters, this integrated identity solution is Azure AD.

For more information on the identity options in Kubernetes, see [Kubernetes authentication](#).

Azure AD integration

Enhance your AKS cluster security with Azure AD integration. Built on decades of enterprise identity management, Azure AD is a multi-tenant, cloud-based directory and identity management service that combines core directory services, application access management, and identity protection. With Azure AD, you can integrate on-premises identities into AKS clusters to provide a single source for account management and security.



With Azure AD-integrated AKS clusters, you can grant users or groups access to Kubernetes resources within a namespace or across the cluster.

1. To obtain a `kubectl` configuration context, a user runs the `az aks get-credentials` command.
2. When a user interacts with the AKS cluster with `kubectl`, they're prompted to sign in with their Azure AD credentials.

This approach provides a single source for user account management and password credentials. The user can only access the resources as defined by the cluster administrator.

Azure AD authentication is provided to AKS clusters with OpenID Connect. OpenID Connect is an identity layer built on top of the OAuth 2.0 protocol. For more information on OpenID Connect, see the [Open ID connect documentation](#). From inside of the Kubernetes cluster, [Webhook Token Authentication](#) is used to verify authentication tokens. Webhook token authentication is configured and managed as part of the AKS cluster.

Webhook and API server



As shown in the graphic above, the API server calls the AKS webhook server and performs the following steps:

1. `kubectl` uses the Azure AD client application to sign in users with [OAuth 2.0 device authorization grant flow](#).
2. Azure AD provides an `access_token`, `id_token`, and a `refresh_token`.
3. The user makes a request to `kubectl` with an `access_token` from `kubeconfig`.
4. `kubectl` sends the `access_token` to API Server.
5. The API Server is configured with the Auth WebHook Server to perform validation.
6. The authentication webhook server confirms the JSON Web Token signature is valid by checking the Azure AD public signing key.
7. The server application uses user-provided credentials to query group memberships of the logged-in user from the MS Graph API.
8. A response is sent to the API Server with user information such as the user principal name (UPN) claim of the access token, and the group membership of the user based on the object ID.
9. The API performs an authorization decision based on the Kubernetes Role/RoleBinding.
10. Once authorized, the API server returns a response to `kubectl`.
11. `kubectl` provides feedback to the user.

Learn how to integrate AKS with Azure AD with our [AKS-managed Azure AD integration how-to guide](#).

Azure role-based access control

Azure role-based access control (RBAC) is an authorization system built on [Azure Resource Manager](#) that provides fine-grained access management of Azure resources.

RBAC SYSTEM	DESCRIPTION
Kubernetes RBAC	Designed to work on Kubernetes resources within your AKS cluster.
Azure RBAC	Designed to work on resources within your Azure subscription.

With Azure RBAC, you create a *role definition* that outlines the permissions to be applied. You then assign a user or group this role definition via a *role assignment* for a particular *scope*. The scope can be an individual resource, a resource group, or across the subscription.

For more information, see [What is Azure role-based access control \(Azure RBAC\)?](#)

There are two levels of access needed to fully operate an AKS cluster:

- [Access the AKS resource in your Azure subscription.](#)
 - Control scaling or upgrading your cluster using the AKS APIs.
 - Pull your `kubeconfig`.
- Access to the Kubernetes API. This access is controlled by either:
 - [Kubernetes RBAC](#) (traditionally).
 - [Integrating Azure RBAC with AKS for Kubernetes authorization.](#)

Azure RBAC to authorize access to the AKS resource

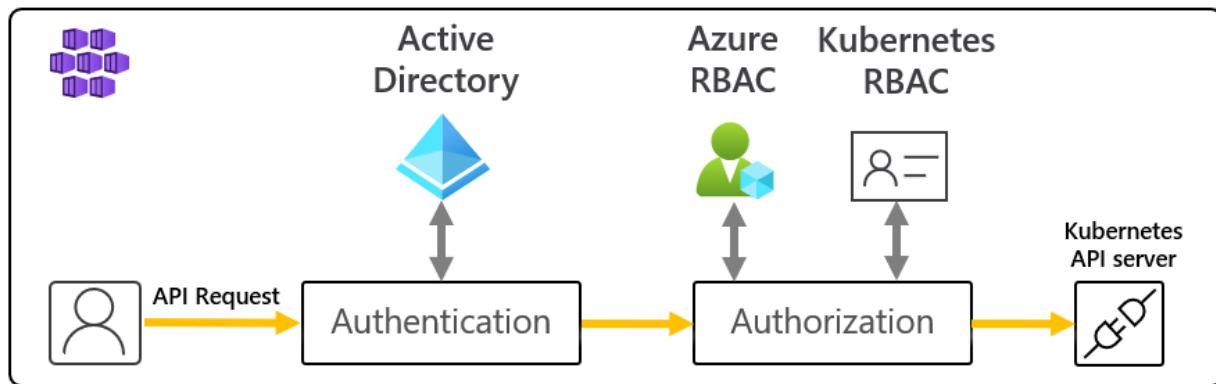
With Azure RBAC, you can provide your users (or identities) with granular access to AKS resources across one or more subscriptions. For example, you could use the [Azure Kubernetes Service Contributor role](#) to scale and upgrade your cluster. Meanwhile, another user with the [Azure Kubernetes Service Cluster Admin role](#) only has permission to pull the Admin `kubeconfig`.

Alternatively, you could give your user the general [Contributor](#) role. With the general Contributor role, users can perform the above permissions and every action possible on the AKS resource, except managing permissions.

[Use Azure RBAC to define access to the Kubernetes configuration file in AKS.](#)

Azure RBAC for Kubernetes Authorization

With the Azure RBAC integration, AKS will use a Kubernetes Authorization webhook server so you can manage Azure AD-integrated Kubernetes cluster resource permissions and assignments using Azure role definition and role assignments.



As shown in the above diagram, when using the Azure RBAC integration, all requests to the Kubernetes API will follow the same authentication flow as explained on the [Azure Active Directory integration section](#).

If the identity making the request exists in Azure AD, Azure will team with Kubernetes RBAC to authorize the request. If the identity exists outside of Azure AD (i.e., a Kubernetes service account), authorization will defer to the normal Kubernetes RBAC.

In this scenario, you use Azure RBAC mechanisms and APIs to assign users built-in roles or create custom roles, just as you would with Kubernetes roles.

With this feature, you not only give users permissions to the AKS resource across subscriptions, but you also configure the role and permissions for inside each of those clusters controlling Kubernetes API access. For example, you can grant the `Azure Kubernetes Service RBAC Viewer` role on the subscription scope. The role recipient will be able to list and get all Kubernetes objects from all clusters without modifying them.

IMPORTANT

You need to enable Azure RBAC for Kubernetes authorization before using this feature. For more details and step by step guidance, follow our [Use Azure RBAC for Kubernetes Authorization](#) how-to guide.

Built-in roles

AKS provides the following four built-in roles. They are similar to the [Kubernetes built-in roles](#) with a few differences, like supporting CRDs. See the full list of actions allowed by each [Azure built-in role](#).

ROLE	DESCRIPTION
Azure Kubernetes Service RBAC Viewer	Allows read-only access to see most objects in a namespace. Doesn't allow viewing roles or role bindings. Doesn't allow viewing <code>Secrets</code> . Reading the <code>Secrets</code> contents enables access to <code>ServiceAccount</code> credentials in the namespace, which would allow API access as any <code>ServiceAccount</code> in the namespace (a form of privilege escalation).
Azure Kubernetes Service RBAC Writer	Allows read/write access to most objects in a namespace. Doesn't allow viewing or modifying roles, or role bindings. Allows accessing <code>Secrets</code> and running pods as any <code>ServiceAccount</code> in the namespace, so it can be used to gain the API access levels of any <code>ServiceAccount</code> in the namespace.
Azure Kubernetes Service RBAC Admin	Allows admin access, intended to be granted within a namespace. Allows read/write access to most resources in a namespace (or cluster scope), including the ability to create roles and role bindings within the namespace. Doesn't allow write access to resource quota or to the namespace itself.
Azure Kubernetes Service RBAC Cluster Admin	Allows super-user access to perform any action on any resource. Gives full control over every resource in the cluster and in all namespaces.

Summary

View the table for a quick summary of how users can authenticate to Kubernetes when Azure AD integration is enabled. In all cases, the user's sequence of commands is:

1. Run `az login` to authenticate to Azure.
2. Run `az aks get-credentials` to download credentials for the cluster into `.kube/config`.
3. Run `kubectl` commands.
 - The first command may trigger browser-based authentication to authenticate to the cluster, as described in the following table.

In the Azure portal, you can find:

- The *Role Grant* (Azure RBAC role grant) referred to in the second column is shown on the **Access Control** tab.
- The Cluster Admin Azure AD Group is shown on the **Configuration** tab.
 - Also found with parameter name `--aad-admin-group-object-ids` in the Azure CLI.

DESCRIPTION	ROLE GRANT REQUIRED	CLUSTER ADMIN AZURE AD GROUP(S)	WHEN TO USE
Legacy admin login using client certificate	Azure Kubernetes Admin Role. This role allows <code>az aks get-credentials</code> to be used with the <code>--admin</code> flag, which downloads a legacy (non-Azure AD) cluster admin certificate into the user's <code>.kube/config</code> . This is the only purpose of "Azure Kubernetes Admin Role".	n/a	If you're permanently blocked by not having access to a valid Azure AD group with access to your cluster.
Azure AD with manual (Cluster)RoleBindings	Azure Kubernetes User Role. The "User" role allows <code>az aks get-credentials</code> to be used without the <code>--admin</code> flag. (This is the only purpose of "Azure Kubernetes User Role".) The result, on an Azure AD-enabled cluster, is the download of an empty entry into <code>.kube/config</code> , which triggers browser-based authentication when it's first used by <code>kubectl</code> .	User is not in any of these groups. Because the user is not in any Cluster Admin groups, their rights will be controlled entirely by any RoleBindings or ClusterRoleBindings that have been set up by cluster admins. The (Cluster)RoleBindings nominate Azure AD users or Azure AD groups as their subjects. If no such bindings have been set up, the user will not be able to execute any <code>kubectl</code> commands.	If you want fine-grained access control, and you're not using Azure RBAC for Kubernetes Authorization. Note that the user who sets up the bindings must log in by one of the other methods listed in this table.
Azure AD by member of admin group	Same as above	User is a member of one of the groups listed here. AKS automatically generates a ClusterRoleBinding that binds all of the listed groups to the <code>cluster-admin</code> Kubernetes role. So users in these groups can run all <code>kubectl</code> commands as <code>cluster-admin</code> .	If you want to conveniently grant users full admin rights, and are <i>not</i> using Azure RBAC for Kubernetes authorization.
Azure AD with Azure RBAC for Kubernetes Authorization	Two roles: First, Azure Kubernetes User Role (as above). Second, one of the "Azure Kubernetes Service RBAC..." roles listed above, or your own custom alternative.	The admin roles field on the Configuration tab is irrelevant when Azure RBAC for Kubernetes Authorization is enabled.	You are using Azure RBAC for Kubernetes authorization. This approach gives you fine-grained control, without the need to set up RoleBindings or ClusterRoleBindings.

Next steps

- To get started with Azure AD and Kubernetes RBAC, see [Integrate Azure Active Directory with AKS](#).
- For associated best practices, see [Best practices for authentication and authorization in AKS](#).
- To get started with Azure RBAC for Kubernetes Authorization, see [Use Azure RBAC to authorize access within the Azure Kubernetes Service \(AKS\) Cluster](#).

- To get started securing your `kubeconfig` file, see [Limit access to cluster configuration file](#)

For more information on core Kubernetes and AKS concepts, see the following articles:

- [Kubernetes / AKS clusters and workloads](#)
- [Kubernetes / AKS security](#)
- [Kubernetes / AKS virtual networks](#)
- [Kubernetes / AKS storage](#)
- [Kubernetes / AKS scale](#)

Security concepts for applications and clusters in Azure Kubernetes Service (AKS)

4/8/2021 • 6 minutes to read • [Edit Online](#)

Cluster security protects your customer data as you run application workloads in Azure Kubernetes Service (AKS).

Kubernetes includes security components, such as *network policies* and *Secrets*. Meanwhile, Azure includes components like network security groups and orchestrated cluster upgrades. AKS combines these security components to:

- Keep your AKS cluster running the latest OS security updates and Kubernetes releases.
- Provide secure pod traffic and access to sensitive credentials.

This article introduces the core concepts that secure your applications in AKS:

- [Security concepts for applications and clusters in Azure Kubernetes Service \(AKS\)](#)
 - [Master security](#)
 - [Node security](#)
 - [Compute isolation](#)
 - [Cluster upgrades](#)
 - [Cordon and drain](#)
 - [Network security](#)
 - [Azure network security groups](#)
 - [Kubernetes Secrets](#)
 - [Next steps](#)

Master security

In AKS, the Kubernetes master components are part of the managed service provided, managed, and maintained by Microsoft. Each AKS cluster has its own single-tenanted, dedicated Kubernetes master to provide the API Server, Scheduler, etc.

By default, the Kubernetes API server uses a public IP address and a fully qualified domain name (FQDN). You can limit access to the API server endpoint using [authorized IP ranges](#). You can also create a fully [private cluster](#) to limit API server access to your virtual network.

You can control access to the API server using Kubernetes role-based access control (Kubernetes RBAC) and Azure RBAC. For more information, see [Azure AD integration with AKS](#).

Node security

AKS nodes are Azure virtual machines (VMs) that you manage and maintain.

- Linux nodes run an optimized Ubuntu distribution using the `containerd` or Moby container runtime.
- Windows Server nodes run an optimized Windows Server 2019 release using the `containerd` or Moby container runtime.

When an AKS cluster is created or scaled up, the nodes are automatically deployed with the latest OS security updates and configurations.

NOTE

AKS clusters using:

- Kubernetes version 1.19 node pools and greater use `containerd` as its container runtime.
- Kubernetes prior to v1.19 node pools use [Moby](#) (upstream docker) as its container runtime.

Node security patches

Linux nodes

The Azure platform automatically applies OS security patches to Linux nodes on a nightly basis. If a Linux OS security update requires a host reboot, it won't automatically reboot. You can either:

- Manually reboot the Linux nodes.
- Use [Kured](#), an open-source reboot daemon for Kubernetes. Kured runs as a [DaemonSet](#) and monitors each node for a file indicating that a reboot is required.

Reboots are managed across the cluster using the same [cordon and drain process](#) as a cluster upgrade.

Windows Server nodes

For Windows Server nodes, Windows Update doesn't automatically run and apply the latest updates. Schedule Windows Server node pool upgrades in your AKS cluster around the regular Windows Update release cycle and your own validation process. This upgrade process creates nodes that run the latest Windows Server image and patches, then removes the older nodes. For more information on this process, see [Upgrade a node pool in AKS](#).

Node deployment

Nodes are deployed into a private virtual network subnet, with no public IP addresses assigned. For troubleshooting and management purposes, SSH is enabled by default and only accessible using the internal IP address.

Node storage

To provide storage, the nodes use Azure Managed Disks. For most VM node sizes, Azure Managed Disks are Premium disks backed by high-performance SSDs. The data stored on managed disks is automatically encrypted at rest within the Azure platform. To improve redundancy, Azure Managed Disks are securely replicated within the Azure datacenter.

Hostile multi-tenant workloads

Currently, Kubernetes environments aren't safe for hostile multi-tenant usage. Extra security features, like *Pod Security Policies* or Kubernetes RBAC for nodes, efficiently block exploits. For true security when running hostile multi-tenant workloads, only trust a hypervisor. The security domain for Kubernetes becomes the entire cluster, not an individual node.

For these types of hostile multi-tenant workloads, you should use physically isolated clusters. For more information on ways to isolate workloads, see [Best practices for cluster isolation in AKS](#).

Compute isolation

Because of compliance or regulatory requirements, certain workloads may require a high degree of isolation from other customer workloads. For these workloads, Azure provides [isolated VMs](#) to use as the agent nodes in an AKS cluster. These VMs are isolated to a specific hardware type and dedicated to a single customer.

Select one of the [isolated VMs sizes](#) as the **node size** when creating an AKS cluster or adding a node pool.

Cluster upgrades

Azure provides upgrade orchestration tools to upgrade of an AKS cluster and components, maintain security and compliance, and access the latest features. This upgrade orchestration includes both the Kubernetes master

and agent components.

To start the upgrade process, specify one of the [listed available Kubernetes versions](#). Azure then safely cordons and drains each AKS node and upgrades.

Cordon and drain

During the upgrade process, AKS nodes are individually cordoned from the cluster to prevent new pods from being scheduled on them. The nodes are then drained and upgraded as follows:

1. A new node is deployed into the node pool.
 - This node runs the latest OS image and patches.
2. One of the existing nodes is identified for upgrade.
3. Pods on the identified node are gracefully terminated and scheduled on the other nodes in the node pool.
4. The emptied node is deleted from the AKS cluster.
5. Steps 1-4 are repeated until all nodes are successfully replaced as part of the upgrade process.

For more information, see [Upgrade an AKS cluster](#).

Network security

For connectivity and security with on-premises networks, you can deploy your AKS cluster into existing Azure virtual network subnets. These virtual networks connect back to your on-premises network using Azure Site-to-Site VPN or Express Route. Define Kubernetes ingress controllers with private, internal IP addresses to limit services access to the internal network connection.

Azure network security groups

To filter virtual network traffic flow, Azure uses network security group rules. These rules define the source and destination IP ranges, ports, and protocols allowed or denied access to resources. Default rules are created to allow TLS traffic to the Kubernetes API server. You create services with load balancers, port mappings, or ingress routes. AKS automatically modifies the network security group for traffic flow.

If you provide your own subnet for your AKS cluster, **do not** modify the subnet-level network security group managed by AKS. Instead, create more subnet-level network security groups to modify the flow of traffic. Make sure they don't interfere with necessary traffic managing the cluster, such as load balancer access, communication with the control plane, and [egress](#).

Kubernetes network policy

To limit network traffic between pods in your cluster, AKS offers support for [Kubernetes network policies](#). With network policies, you can allow or deny specific network paths within the cluster based on namespaces and label selectors.

Kubernetes Secrets

With a Kubernetes *Secret*, you inject sensitive data into pods, such as access credentials or keys.

1. Create a Secret using the Kubernetes API.
2. Define your pod or deployment and request a specific Secret.
 - Secrets are only provided to nodes with a scheduled pod that requires them.
 - The Secret is stored in *tmpfs*, not written to disk.
3. When you delete the last pod on a node requiring a Secret, the Secret is deleted from the node's *tmpfs*.
 - Secrets are stored within a given namespace and can only be accessed by pods within the same namespace.

Using Secrets reduces the sensitive information defined in the pod or service YAML manifest. Instead, you

request the Secret stored in Kubernetes API Server as part of your YAML manifest. This approach only provides the specific pod access to the Secret.

NOTE

The raw secret manifest files contain the secret data in base64 format (see the [official documentation](#) for more details). Treat these files as sensitive information, and never commit them to source control.

Kubernetes secrets are stored in etcd, a distributed key-value store. Etcd store is fully managed by AKS and [data is encrypted at rest within the Azure platform](#).

Next steps

To get started with securing your AKS clusters, see [Upgrade an AKS cluster](#).

For associated best practices, see [Best practices for cluster security and upgrades in AKS](#) and [Best practices for pod security in AKS](#).

For more information on core Kubernetes and AKS concepts, see:

- [Kubernetes / AKS clusters and workloads](#)
- [Kubernetes / AKS identity](#)
- [Kubernetes / AKS virtual networks](#)
- [Kubernetes / AKS storage](#)
- [Kubernetes / AKS scale](#)

Azure Policy Regulatory Compliance controls for Azure Kubernetes Service (AKS)

5/14/2021 • 8 minutes to read • [Edit Online](#)

Regulatory Compliance in Azure Policy provides initiative definitions (*built-ins*) created and managed by Microsoft, for the compliance domains and security controls related to different compliance standards. This page lists the Azure Kubernetes Service (AKS) compliance domains and security controls.

You can assign the built-ins for a **security control** individually to help make your Azure resources compliant with the specific standard.

The title of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Policy Version** column to view the source on the [Azure Policy GitHub repo](#).

IMPORTANT

Each control below is associated with one or more Azure Policy definitions. These policies may help you [assess compliance](#) with the control; however, there often is not a one-to-one or complete match between a control and one or more policies. As such, **Compliant** in Azure Policy refers only to the policies themselves; this doesn't ensure you're fully compliant with all requirements of a control. In addition, the compliance standard includes controls that aren't addressed by any Azure Policy definitions at this time. Therefore, compliance in Azure Policy is only a partial view of your overall compliance status. The associations between controls and Azure Policy Regulatory Compliance definitions for these compliance standards may change over time.

Azure Security Benchmark

The [Azure Security Benchmark](#) provides recommendations on how you can secure your cloud solutions on Azure. To see how this service completely maps to the Azure Security Benchmark, see the [Azure Security Benchmark mapping files](#).

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - Azure Security Benchmark](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Network Security	NS-1	Implement security for internal traffic	Authorized IP ranges should be defined on Kubernetes Services	2.0.1
Network Security	NS-4	Protect applications and services from external network attacks	Authorized IP ranges should be defined on Kubernetes Services	2.0.1
Privileged Access	PA-7	Follow just enough administration (least privilege principle)	Role-Based Access Control (RBAC) should be used on Kubernetes Services	1.0.2

Domain	Control ID	Control Title	Policy	Policy Version
Data Protection	DP-4	Encrypt sensitive information in transit	Kubernetes clusters should be accessible only over HTTPS	6.0.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	Azure Policy Add-on for Kubernetes service (AKS) should be installed and enabled on your clusters	1.0.2
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	Kubernetes cluster containers CPU and memory resource limits should not exceed the specified limits	6.0.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	Kubernetes cluster containers should not share host process ID or host IPC namespace	3.0.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	Kubernetes cluster containers should only listen on allowed ports	6.1.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	Kubernetes cluster containers should only use allowed AppArmor profiles	3.0.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	Kubernetes cluster containers should only use allowed capabilities	3.0.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	Kubernetes cluster containers should only use allowed images	6.1.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	Kubernetes cluster containers should run with a read only root file system	3.0.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	Kubernetes cluster pod hostPath volumes should only use allowed host paths	3.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	Kubernetes cluster pods and containers should only run with approved user and group IDs	3.0.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	Kubernetes cluster pods should only use approved host network and port range	3.0.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	Kubernetes cluster services should listen only on allowed ports	6.1.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	Kubernetes cluster should not allow privileged containers	7.0.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	Kubernetes clusters should not allow container privilege escalation	3.0.0

Azure Security Benchmark v1

The [Azure Security Benchmark](#) provides recommendations on how you can secure your cloud solutions on Azure. To see how this service completely maps to the Azure Security Benchmark, see the [Azure Security Benchmark mapping files](#).

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - Azure Security Benchmark](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Network Security	1.1	Protect resources using Network Security Groups or Azure Firewall on your Virtual Network	Authorized IP ranges should be defined on Kubernetes Services	2.0.1
Data Protection	4.6	Use Azure RBAC to control access to resources	Role-Based Access Control (RBAC) should be used on Kubernetes Services	1.0.2
Vulnerability Management	5.3	Deploy automated third-party software patch management solution	Kubernetes Services should be upgraded to a non-vulnerable Kubernetes version	1.0.2

CIS Microsoft Azure Foundations Benchmark 1.1.0

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - CIS Microsoft Azure Foundations Benchmark 1.1.0](#). For more information about this compliance standard, see [CIS Microsoft Azure Foundations Benchmark](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Other Security Considerations	8.5	Enable role-based access control (RBAC) within Azure Kubernetes Services	Role-Based Access Control (RBAC) should be used on Kubernetes Services	1.0.2

CIS Microsoft Azure Foundations Benchmark 1.3.0

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - CIS Microsoft Azure Foundations Benchmark 1.3.0](#). For more information about this compliance standard, see [CIS Microsoft Azure Foundations Benchmark](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Other Security Considerations	8.5	Enable role-based access control (RBAC) within Azure Kubernetes Services	Role-Based Access Control (RBAC) should be used on Kubernetes Services	1.0.2

CMMC Level 3

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - CMMC Level 3](#). For more information about this compliance standard, see [Cybersecurity Maturity Model Certification \(CMMC\)](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC.1.001	Limit information system access to authorized users, processes acting on behalf of authorized users, and devices (including other information systems).	Kubernetes cluster pods should only use approved host network and port range	3.0.0
Access Control	AC.1.001	Limit information system access to authorized users, processes acting on behalf of authorized users, and devices (including other information systems).	Role-Based Access Control (RBAC) should be used on Kubernetes Services	1.0.2

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Access Control	AC.1.002	Limit information system access to the types of transactions and functions that authorized users are permitted to execute.	Kubernetes cluster pods should only use approved host network and port range	3.0.0
Access Control	AC.1.002	Limit information system access to the types of transactions and functions that authorized users are permitted to execute.	Role-Based Access Control (RBAC) should be used on Kubernetes Services	1.0.2
Access Control	AC.2.007	Employ the principle of least privilege, including for specific security functions and privileged accounts.	Role-Based Access Control (RBAC) should be used on Kubernetes Services	1.0.2
Access Control	AC.2.016	Control the flow of CUI in accordance with approved authorizations.	Role-Based Access Control (RBAC) should be used on Kubernetes Services	1.0.2
Configuration Management	CM.2.062	Employ the principle of least functionality by configuring organizational systems to provide only essential capabilities.	Role-Based Access Control (RBAC) should be used on Kubernetes Services	1.0.2
Configuration Management	CM.3.068	Restrict, disable, or prevent the use of nonessential programs, functions, ports, protocols, and services.	Kubernetes cluster pods should only use approved host network and port range	3.0.0
Risk Assessment	RM.2.143	Remediate vulnerabilities in accordance with risk assessments.	Kubernetes Services should be upgraded to a non-vulnerable Kubernetes version	1.0.2

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	SC.1.175	Monitor, control, and protect communications (i.e., information transmitted or received by organizational systems) at the external boundaries and key internal boundaries of organizational systems.	Kubernetes cluster pods should only use approved host network and port range	3.0.0
System and Communications Protection	SC.3.177	Employ FIPS-validated cryptography when used to protect the confidentiality of CUI.	Both operating systems and data disks in Azure Kubernetes Service clusters should be encrypted by customer-managed keys	1.0.0
System and Communications Protection	SC.3.183	Deny network communications traffic by default and allow network communications traffic by exception (i.e., deny all, permit by exception).	Kubernetes cluster pods should only use approved host network and port range	3.0.0
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	Kubernetes Services should be upgraded to a non-vulnerable Kubernetes version	1.0.2

HIPAA HITRUST 9.2

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - HIPAA HITRUST 9.2](#). For more information about this compliance standard, see [HIPAA HITRUST 9.2](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
--------	------------	---------------	--------------------------	----------------------------

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Privilege Management	1149.01c2System.9 - 01.c	The organization facilitates information sharing by enabling authorized users to determine a business partner's access when discretion is allowed as defined by the organization and by employing manual processes or automated mechanisms to assist users in making information sharing/collaboration decisions.	Role-Based Access Control (RBAC) should be used on Kubernetes Services	1.0.2
Privilege Management	1153.01c3System.35 - 01.c	All file system access not explicitly required is disabled, and only authorized users are permitted access to only that which is expressly required for the performance of the users' job duties.	Role-Based Access Control (RBAC) should be used on Kubernetes Services	1.0.2
Segregation of Duties	1229.09c1Organizational.1 - 09.c	Separation of duties is used to limit the risk of unauthorized or unintentional modification of information and systems.	Role-Based Access Control (RBAC) should be used on Kubernetes Services	1.0.2

NIST SP 800-171 R2

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - NIST SP 800-171 R2](#). For more information about this compliance standard, see [NIST SP 800-171 R2](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
System and Information Integrity	3.14.1	Identify, report, and correct system flaws in a timely manner.	Kubernetes Services should be upgraded to a non-vulnerable Kubernetes version	1.0.2

Next steps

- Learn more about [Azure Policy Regulatory Compliance](#).
- See the built-ins on the [Azure Policy GitHub repo](#).

Network concepts for applications in Azure Kubernetes Service (AKS)

4/8/2021 • 9 minutes to read • [Edit Online](#)

In a container-based, microservices approach to application development, application components work together to process their tasks. Kubernetes provides various resources enabling this cooperation:

- You can connect to and expose applications internally or externally.
- You can build highly available applications by load balancing your applications.
- For your more complex applications, you can configure ingress traffic for SSL/TLS termination or routing of multiple components.
- For security reasons, you can restrict the flow of network traffic into or between pods and nodes.

This article introduces the core concepts that provide networking to your applications in AKS:

- [Services](#)
- [Azure virtual networks](#)
- [Ingress controllers](#)
- [Network policies](#)

Kubernetes basics

To allow access to your applications or between application components, Kubernetes provides an abstraction layer to virtual networking. Kubernetes nodes connect to a virtual network, providing inbound and outbound connectivity for pods. The *kube-proxy* component runs on each node to provide these network features.

In Kubernetes:

- *Services* logically group pods to allow for direct access on a specific port via an IP address or DNS name.
- You can distribute traffic using a *load balancer*.
- More complex routing of application traffic can also be achieved with *Ingress Controllers*.
- Security and filtering of the network traffic for pods is possible with Kubernetes *network policies*.

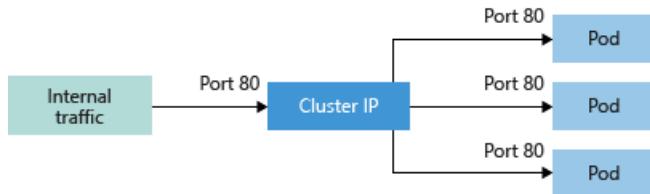
The Azure platform also simplifies virtual networking for AKS clusters. When you create a Kubernetes load balancer, you also create and configure the underlying Azure load balancer resource. As you open network ports to pods, the corresponding Azure network security group rules are configured. For HTTP application routing, Azure can also configure *external DNS* as new ingress routes are configured.

Services

To simplify the network configuration for application workloads, Kubernetes uses *Services* to logically group a set of pods together and provide network connectivity. The following Service types are available:

- **Cluster IP**

Creates an internal IP address for use within the AKS cluster. Good for internal-only applications that support other workloads within the cluster.



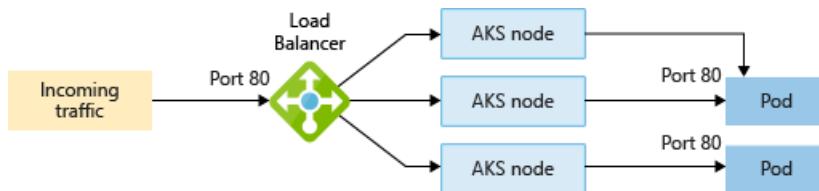
- **NodePort**

Creates a port mapping on the underlying node that allows the application to be accessed directly with the node IP address and port.



- **LoadBalancer**

Creates an Azure load balancer resource, configures an external IP address, and connects the requested pods to the load balancer backend pool. To allow customers' traffic to reach the application, load balancing rules are created on the desired ports.



For extra control and routing of the inbound traffic, you may instead use an [Ingress controller](#).

- **ExternalName**

Creates a specific DNS entry for easier application access.

Either the load balancers and services IP address can be dynamically assigned, or you can specify an existing static IP address. You can assign both internal and external static IP addresses. Existing static IP addresses are often tied to a DNS entry.

You can create both *internal* and *external* load balancers. Internal load balancers are only assigned a private IP address, so they can't be accessed from the Internet.

Azure virtual networks

In AKS, you can deploy a cluster that uses one of the following two network models:

- *Kubenet* networking

The network resources are typically created and configured as the AKS cluster is deployed.

- *Azure Container Networking Interface (CNI)* networking

The AKS cluster is connected to existing virtual network resources and configurations.

Kubenet (basic) networking

The *kubenet* networking option is the default configuration for AKS cluster creation. With *kubenet*

1. Nodes receive an IP address from the Azure virtual network subnet.
2. Pods receive an IP address from a logically different address space than the nodes' Azure virtual network

subnet.

3. Network address translation (NAT) is then configured so that the pods can reach resources on the Azure virtual network.
4. The source IP address of the traffic is translated to the node's primary IP address.

Nodes use the [kubenet](#) Kubernetes plugin. You can:

- Let the Azure platform create and configure the virtual networks for you, or
- Choose to deploy your AKS cluster into an existing virtual network subnet.

Remember, only the nodes receive a routable IP address. The pods use NAT to communicate with other resources outside the AKS cluster. This approach reduces the number of IP addresses you need to reserve in your network space for pods to use.

For more information, see [Configure kubenet networking for an AKS cluster](#).

Azure CNI (advanced) networking

With Azure CNI, every pod gets an IP address from the subnet and can be accessed directly. These IP addresses must be planned in advance and unique across your network space. Each node has a configuration parameter for the maximum number of pods it supports. The equivalent number of IP addresses per node are then reserved up front. Without planning, this approach can lead to IP address exhaustion or the need to rebuild clusters in a larger subnet as your application demands grow.

Unlike kubenet, traffic to endpoints in the same virtual network isn't NAT'd to the node's primary IP. The source address for traffic inside the virtual network is the pod IP. Traffic that's external to the virtual network still NATs to the node's primary IP.

Nodes use the [Azure CNI](#) Kubernetes plugin.



For more information, see [Configure Azure CNI for an AKS cluster](#).

Compare network models

Both kubenet and Azure CNI provide network connectivity for your AKS clusters. However, there are advantages and disadvantages to each. At a high level, the following considerations apply:

- **kubenet**
 - Conserves IP address space.
 - Uses Kubernetes internal or external load balancer to reach pods from outside of the cluster.
 - You manually manage and maintain user-defined routes (UDRs).
 - Maximum of 400 nodes per cluster.
- **Azure CNI**
 - Pods get full virtual network connectivity and can be directly reached via their private IP address from connected networks.
 - Requires more IP address space.

The following behavior differences exist between kubenet and Azure CNI:

Capability	Kubenet	Azure CNI
Deploy cluster in existing or new virtual network	Supported - UDRs manually applied	Supported
Pod-pod connectivity	Supported	Supported
Pod-VM connectivity; VM in the same virtual network	Works when initiated by pod	Works both ways
Pod-VM connectivity; VM in peered virtual network	Works when initiated by pod	Works both ways
On-premises access using VPN or Express Route	Works when initiated by pod	Works both ways
Access to resources secured by service endpoints	Supported	Supported
Expose Kubernetes services using a load balancer service, App Gateway, or ingress controller	Supported	Supported
Default Azure DNS and Private Zones	Supported	Supported

Regarding DNS, with both kubenet and Azure CNI plugins DNS are offered by CoreDNS, a deployment running in AKS with its own autoscaler. For more information on CoreDNS on Kubernetes, see [Customizing DNS Service](#). CoreDNS by default is configured to forward unknown domains to the DNS functionality of the Azure Virtual Network where the AKS cluster is deployed. Hence, Azure DNS and Private Zones will work for pods running in AKS.

Support scope between network models

Whatever network model you use, both kubenet and Azure CNI can be deployed in one of the following ways:

- The Azure platform can automatically create and configure the virtual network resources when you create an AKS cluster.
- You can manually create and configure the virtual network resources and attach to those resources when you create your AKS cluster.

Although capabilities like service endpoints or UDRs are supported with both kubenet and Azure CNI, the [support policies for AKS](#) define what changes you can make. For example:

- If you manually create the virtual network resources for an AKS cluster, you're supported when configuring your own UDRs or service endpoints.
- If the Azure platform automatically creates the virtual network resources for your AKS cluster, you can't manually change those AKS-managed resources to configure your own UDRs or service endpoints.

Ingress controllers

When you create a LoadBalancer-type Service, you also create an underlying Azure load balancer resource. The load balancer is configured to distribute traffic to the pods in your Service on a given port.

The LoadBalancer only works at layer 4. At layer 4, the Service is unaware of the actual applications, and can't make any more routing considerations.

Ingress controllers work at layer 7, and can use more intelligent rules to distribute application traffic. Ingress controllers typically route HTTP traffic to different applications based on the inbound URL.



Create an ingress resource

In AKS, you can create an Ingress resource using NGINX, a similar tool, or the AKS HTTP application routing feature. When you enable HTTP application routing for an AKS cluster, the Azure platform creates the Ingress controller and an *External-DNS* controller. As new Ingress resources are created in Kubernetes, the required DNS A records are created in a cluster-specific DNS zone.

For more information, see [Deploy HTTP application routing](#).

Application Gateway Ingress Controller (AGIC)

With the Application Gateway Ingress Controller (AGIC) add-on, AKS customers leverage Azure's native Application Gateway level 7 load-balancer to expose cloud software to the Internet. AGIC monitors the host Kubernetes cluster and continuously updates an Application Gateway, exposing selected services to the Internet.

To learn more about the AGIC add-on for AKS, see [What is Application Gateway Ingress Controller?](#).

SSL/TLS termination

SSL/TLS termination is another common feature of Ingress. On large web applications accessed via HTTPS, the Ingress resource handles the TLS termination rather than within the application itself. To provide automatic TLS certification generation and configuration, you can configure the Ingress resource to use providers such as "Let's Encrypt".

For more information on configuring an NGINX Ingress controller with Let's Encrypt, see [Ingress and TLS](#).

Client source IP preservation

Configure your ingress controller to preserve the client source IP on requests to containers in your AKS cluster. When your ingress controller routes a client's request to a container in your AKS cluster, the original source IP of that request is unavailable to the target container. When you enable *client source IP preservation*, the source IP for the client is available in the request header under *X-Forwarded-For*.

If you're using client source IP preservation on your ingress controller, you can't use TLS pass-through. Client source IP preservation and TLS pass-through can be used with other services, such as the *LoadBalancer* type.

Network security groups

A network security group filters traffic for VMs like the AKS nodes. As you create Services, such as a LoadBalancer, the Azure platform automatically configures any necessary network security group rules.

You don't need to manually configure network security group rules to filter traffic for pods in an AKS cluster. Simply define any required ports and forwarding as part of your Kubernetes Service manifests. Let the Azure platform create or update the appropriate rules.

You can also use network policies to automatically apply traffic filter rules to pods.

Network policies

By default, all pods in an AKS cluster can send and receive traffic without limitations. For improved security,

define rules that control the flow of traffic, like:

- Backend applications are only exposed to required frontend services.
- Database components are only accessible to the application tiers that connect to them.

Network policy is a Kubernetes feature available in AKS that lets you control the traffic flow between pods. You allow or deny traffic to the pod based on settings such as assigned labels, namespace, or traffic port. While network security groups are better for AKS nodes, network policies are a more suited, cloud-native way to control the flow of traffic for pods. As pods are dynamically created in an AKS cluster, required network policies can be automatically applied.

For more information, see [Secure traffic between pods using network policies in Azure Kubernetes Service \(AKS\)](#).

Next steps

To get started with AKS networking, create and configure an AKS cluster with your own IP address ranges using [kubenet](#) or [Azure CNI](#).

For associated best practices, see [Best practices for network connectivity and security in AKS](#).

For more information on core Kubernetes and AKS concepts, see the following articles:

- [Kubernetes / AKS clusters and workloads](#)
- [Kubernetes / AKS access and identity](#)
- [Kubernetes / AKS security](#)
- [Kubernetes / AKS storage](#)
- [Kubernetes / AKS scale](#)

Storage options for applications in Azure Kubernetes Service (AKS)

4/8/2021 • 7 minutes to read • [Edit Online](#)

Applications running in Azure Kubernetes Service (AKS) may need to store and retrieve data. While some application workloads can use local, fast storage on unneeded, emptied nodes, others require storage that persists on more regular data volumes within the Azure platform.

Multiple pods may need to:

- Share the same data volumes.
- Reattach data volumes if the pod is rescheduled on a different node.

Finally, you may need to inject sensitive data or application configuration information into pods.

This article introduces the core concepts that provide storage to your applications in AKS:

- [Volumes](#)
- [Persistent volumes](#)
- [Storage classes](#)
- [Persistent volume claims](#)



Volumes

Kubernetes typically treats individual pods as ephemeral, disposable resources. Applications have different approaches available to them for using and persisting data. A *volume* represents a way to store, retrieve, and persist data across pods and through the application lifecycle.

Traditional volumes are created as Kubernetes resources backed by Azure Storage. You can manually create data volumes to be assigned to pods directly, or have Kubernetes automatically create them. Data volumes can use Azure Disks or Azure Files.

Azure Disks

Use *Azure Disks* to create a Kubernetes *DataDisk* resource. Disks can use:

- Azure Premium storage, backed by high-performance SSDs, or
- Azure Standard storage, backed by regular HDDs.

TIP

For most production and development workloads, use Premium storage.

Since Azure Disks are mounted as *ReadWriteOnce*, they're only available to a single pod. For storage volumes that can be accessed by multiple pods simultaneously, use Azure Files.

Azure Files

Use *Azure Files* to mount an SMB 3.0 share backed by an Azure Storage account to pods. Files let you share data across multiple nodes and pods and can use:

- Azure Premium storage, backed by high-performance SSDs, or
- Azure Standard storage backed by regular HDDs.

Volume types

Kubernetes volumes represent more than just a traditional disk for storing and retrieving information.

Kubernetes volumes can also be used as a way to inject data into a pod for use by the containers.

Common volume types in Kubernetes include:

emptyDir

Commonly used as temporary space for a pod. All containers within a pod can access the data on the volume. Data written to this volume type persists only for the lifespan of the pod. Once you delete the pod, the volume is deleted. This volume typically uses the underlying local node disk storage, though it can also exist only in the node's memory.

secret

You can use *secret* volumes to inject sensitive data into pods, such as passwords.

1. Create a Secret using the Kubernetes API.
2. Define your pod or deployment and request a specific Secret.
 - Secrets are only provided to nodes with a scheduled pod that requires them.
 - The Secret is stored in *tmpfs*, not written to disk.
3. When you delete the last pod on a node requiring a Secret, the Secret is deleted from the node's *tmpfs*.
 - Secrets are stored within a given namespace and can only be accessed by pods within the same namespace.

configMap

You can use *configMap* to inject key-value pair properties into pods, such as application configuration information. Define application configuration information as a Kubernetes resource, easily updated and applied to new instances of pods as they're deployed.

Like using a Secret:

1. Create a ConfigMap using the Kubernetes API.
2. Request the ConfigMap when you define a pod or deployment.
 - ConfigMaps are stored within a given namespace and can only be accessed by pods within the same namespace.

Persistent volumes

Volumes defined and created as part of the pod lifecycle only exist until you delete the pod. Pods often expect their storage to remain if a pod is rescheduled on a different host during a maintenance event, especially in StatefulSets. A *persistent volume* (PV) is a storage resource created and managed by the Kubernetes API that can exist beyond the lifetime of an individual pod.

You can use Azure Disks or Files to provide the PersistentVolume. As noted in the [Volumes](#) section, the choice of Disks or Files is often determined by the need for concurrent access to the data or the performance tier.



A PersistentVolume can be *statically* created by a cluster administrator, or *dynamically* created by the Kubernetes API server. If a pod is scheduled and requests currently unavailable storage, Kubernetes can create the underlying Azure Disk or Files storage and attach it to the pod. Dynamic provisioning uses a *StorageClass* to identify what type of Azure storage needs to be created.

Storage classes

To define different tiers of storage, such as Premium and Standard, you can create a *StorageClass*.

The StorageClass also defines the *reclaimPolicy*. When you delete the pod and the persistent volume is no longer required, the reclaimPolicy controls the behavior of the underlying Azure storage resource. The underlying storage resource can either be deleted or kept for use with a future pod.

In AKS, four initial `StorageClasses` are created for cluster using the in-tree storage plugins:

PERMISSION	REASON
<code>default</code>	Uses Azure StandardSSD storage to create a Managed Disk. The reclaim policy ensures that the underlying Azure Disk is deleted when the persistent volume that used it is deleted.
<code>managed-premium</code>	Uses Azure Premium storage to create a Managed Disk. The reclaim policy again ensures that the underlying Azure Disk is deleted when the persistent volume that used it is deleted.
<code>azurefile</code>	Uses Azure Standard storage to create an Azure File Share. The reclaim policy ensures that the underlying Azure File Share is deleted when the persistent volume that used it is deleted.
<code>azurefile-premium</code>	Uses Azure Premium storage to create an Azure File Share. The reclaim policy ensures that the underlying Azure File Share is deleted when the persistent volume that used it is deleted.

For clusters using the new Container Storage Interface (CSI) external plugins (preview) the following extra `StorageClasses` are created:

PERMISSION	REASON
<code>managed-csi</code>	Uses Azure StandardSSD locally redundant storage (LRS) to create a Managed Disk. The reclaim policy ensures that the underlying Azure Disk is deleted when the persistent volume that used it is deleted. The storage class also configures the persistent volumes to be expandable, you just need to edit the persistent volume claim with the new size.

PERMISSION	REASON
managed-csi-premium	Uses Azure Premium locally redundant storage (LRS) to create a Managed Disk. The reclaim policy again ensures that the underlying Azure Disk is deleted when the persistent volume that used it is deleted. Similarly, this storage class allows for persistent volumes to be expanded.
azurefile-csi	Uses Azure Standard storage to create an Azure File Share. The reclaim policy ensures that the underlying Azure File Share is deleted when the persistent volume that used it is deleted.
azurefile-csi-premium	Uses Azure Premium storage to create an Azure File Share. The reclaim policy ensures that the underlying Azure File Share is deleted when the persistent volume that used it is deleted.

Unless you specify a StorageClass for a persistent volume, the default StorageClass will be used. Ensure volumes use the appropriate storage you need when requesting persistent volumes.

You can create a StorageClass for additional needs using `kubectl`. The following example uses Premium Managed Disks and specifies that the underlying Azure Disk should be *retain*ed when you delete the pod:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: managed-premium-retain
provisioner: kubernetes.io/azure-disk
reclaimPolicy: Retain
parameters:
  storageaccounttype: Premium_LRS
  kind: Managed
```

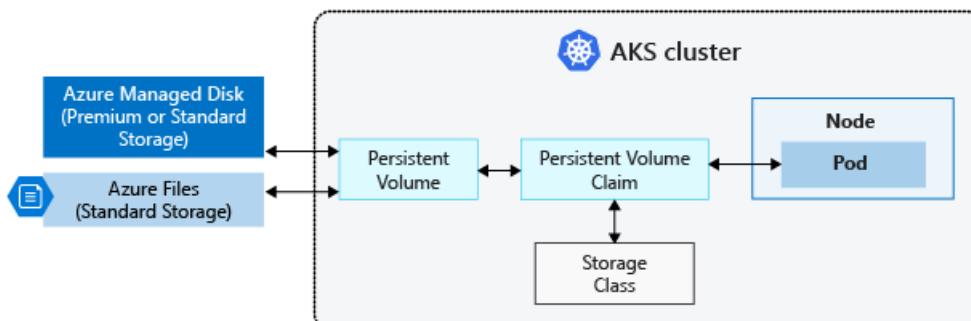
NOTE

AKS reconciles the default storage classes and will overwrite any changes you make to those storage classes.

Persistent volume claims

A PersistentVolumeClaim requests either Disk or File storage of a particular StorageClass, access mode, and size. The Kubernetes API server can dynamically provision the underlying Azure storage resource if no existing resource can fulfill the claim based on the defined StorageClass.

The pod definition includes the volume mount once the volume has been connected to the pod.



Once an available storage resource has been assigned to the pod requesting storage, PersistentVolume is *bound*

to a PersistentVolumeClaim. Persistent volumes are 1:1 mapped to claims.

The following example YAML manifest shows a persistent volume claim that uses the *managed-premium* StorageClass and requests a Disk *5Gi* in size:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: azure-managed-disk
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: managed-premium
  resources:
    requests:
      storage: 5Gi
```

When you create a pod definition, you also specify:

- The persistent volume claim to request the desired storage.
- The *volumeMount* for your applications to read and write data.

The following example YAML manifest shows how the previous persistent volume claim can be used to mount a volume at */mnt/azure*.

```
kind: Pod
apiVersion: v1
metadata:
  name: nginx
spec:
  containers:
    - name: myfrontend
      image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
      volumeMounts:
        - mountPath: "/mnt/azure"
          name: volume
  volumes:
    - name: volume
      persistentVolumeClaim:
        claimName: azure-managed-disk
```

For mounting a volume in a Windows container, specify the drive letter and path. For example:

```
...
  volumeMounts:
    - mountPath: "d:"
      name: volume
    - mountPath: "c:\k"
      name: k-dir
...

```

Next steps

For associated best practices, see [Best practices for storage and backups in AKS](#).

To see how to create dynamic and static volumes that use Azure Disks or Azure Files, see the following how-to articles:

- [Create a static volume using Azure Disks](#)
- [Create a static volume using Azure Files](#)

- [Create a dynamic volume using Azure Disks](#)
- [Create a dynamic volume using Azure Files](#)

For more information on core Kubernetes and AKS concepts, see the following articles:

- [Kubernetes / AKS clusters and workloads](#)
- [Kubernetes / AKS identity](#)
- [Kubernetes / AKS security](#)
- [Kubernetes / AKS virtual networks](#)
- [Kubernetes / AKS scale](#)

Scaling options for applications in Azure Kubernetes Service (AKS)

4/10/2021 • 6 minutes to read • [Edit Online](#)

As you run applications in Azure Kubernetes Service (AKS), you may need to increase or decrease the amount of compute resources. As the number of application instances you need change, the number of underlying Kubernetes nodes may also need to change. You also might need to quickly provision a large number of additional application instances.

This article introduces the core concepts that help you scale applications in AKS:

- [Manually scale](#)
- [Horizontal pod autoscaler \(HPA\)](#)
- [Cluster autoscaler](#)
- [Azure Container Instance \(ACI\) integration with AKS](#)

Manually scale pods or nodes

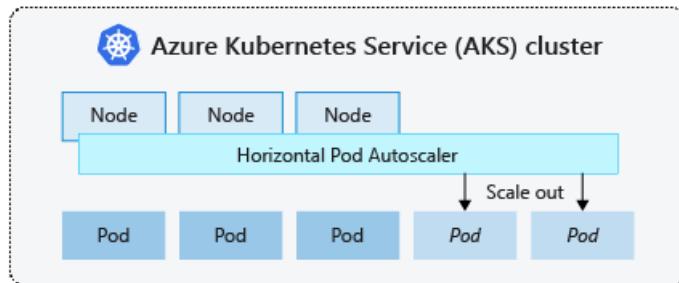
You can manually scale replicas (pods) and nodes to test how your application responds to a change in available resources and state. Manually scaling resources also lets you define a set amount of resources to use to maintain a fixed cost, such as the number of nodes. To manually scale, you define the replica or node count. The Kubernetes API then schedules creating additional pods or draining nodes based on that replica or node count.

When scaling down nodes, the Kubernetes API calls the relevant Azure Compute API tied to the compute type used by your cluster. For example, for clusters built on VM Scale Sets the logic for selecting which nodes to remove is determined by the VM Scale Sets API. To learn more about how nodes are selected for removal on scale down, see the [VMSS FAQ](#).

To get started with manually scaling pods and nodes see [Scale applications in AKS](#).

Horizontal pod autoscaler

Kubernetes uses the horizontal pod autoscaler (HPA) to monitor the resource demand and automatically scale the number of replicas. By default, the horizontal pod autoscaler checks the Metrics API every 30 seconds for any required changes in replica count. When changes are required, the number of replicas is increased or decreased accordingly. Horizontal pod autoscaler works with AKS clusters that have deployed the Metrics Server for Kubernetes 1.8+.



When you configure the horizontal pod autoscaler for a given deployment, you define the minimum and maximum number of replicas that can run. You also define the metric to monitor and base any scaling decisions on, such as CPU usage.

To get started with the horizontal pod autoscaler in AKS, see [Autoscale pods in AKS](#).

Cooldown of scaling events

As the horizontal pod autoscaler checks the Metrics API every 30 seconds, previous scale events may not have successfully completed before another check is made. This behavior could cause the horizontal pod autoscaler to change the number of replicas before the previous scale event could receive application workload and the resource demands to adjust accordingly.

To minimize race events, a delay value is set. This value defines how long the horizontal pod autoscaler must wait after a scale event before another scale event can be triggered. This behavior allows the new replica count to take effect and the Metrics API to reflect the distributed workload. There is [no delay for scale-up events as of Kubernetes 1.12](#), however the delay on scale down events is defaulted to 5 minutes.

Currently, you can't tune these cooldown values from the default.

Cluster autoscaler

To respond to changing pod demands, Kubernetes has a cluster autoscaler, that adjusts the number of nodes based on the requested compute resources in the node pool. By default, the cluster autoscaler checks the Metrics API server every 10 seconds for any required changes in node count. If the cluster autoscale determines that a change is required, the number of nodes in your AKS cluster is increased or decreased accordingly. The cluster autoscaler works with Kubernetes RBAC-enabled AKS clusters that run Kubernetes 1.10.x or higher.



Cluster autoscaler is typically used alongside the horizontal pod autoscaler. When combined, the horizontal pod autoscaler increases or decreases the number of pods based on application demand, and the cluster autoscaler adjusts the number of nodes as needed to run those additional pods accordingly.

To get started with the cluster autoscaler in AKS, see [Cluster Autoscaler on AKS](#).

Scale out events

If a node doesn't have sufficient compute resources to run a requested pod, that pod can't progress through the scheduling process. The pod can't start unless additional compute resources are available within the node pool.

When the cluster autoscaler notices pods that can't be scheduled because of node pool resource constraints, the number of nodes within the node pool is increased to provide the additional compute resources. When those additional nodes are successfully deployed and available for use within the node pool, the pods are then scheduled to run on them.

If your application needs to scale rapidly, some pods may remain in a state waiting to be scheduled until the additional nodes deployed by the cluster autoscaler can accept the scheduled pods. For applications that have high burst demands, you can scale with virtual nodes and Azure Container Instances.

Scale in events

The cluster autoscaler also monitors the pod scheduling status for nodes that haven't recently received new scheduling requests. This scenario indicates the node pool has more compute resources than are required, and the number of nodes can be decreased.

A node that passes a threshold for no longer being needed for 10 minutes by default is scheduled for deletion. When this situation occurs, pods are scheduled to run on other nodes within the node pool, and the cluster autoscaler decreases the number of nodes.

Your applications may experience some disruption as pods are scheduled on different nodes when the cluster autoscaler decreases the number of nodes. To minimize disruption, avoid applications that use a single pod instance.

Burst to Azure Container Instances

To rapidly scale your AKS cluster, you can integrate with Azure Container Instances (ACI). Kubernetes has built-in components to scale the replica and node count. However, if your application needs to rapidly scale, the horizontal pod autoscaler may schedule more pods than can be provided by the existing compute resources in the node pool. If configured, this scenario would then trigger the cluster autoscaler to deploy additional nodes in the node pool, but it may take a few minutes for those nodes to successfully provision and allow the Kubernetes scheduler to run pods on them.



ACI lets you quickly deploy container instances without additional infrastructure overhead. When you connect with AKS, ACI becomes a secured, logical extension of your AKS cluster. The [virtual nodes](#) component, which is based on [Virtual Kubelet](#), is installed in your AKS cluster that presents ACI as a virtual Kubernetes node.

Kubernetes can then schedule pods that run as ACI instances through virtual nodes, not as pods on VM nodes directly in your AKS cluster.

Your application requires no modification to use virtual nodes. Deployments can scale across AKS and ACI and with no delay as cluster autoscaler deploys new nodes in your AKS cluster.

Virtual nodes are deployed to an additional subnet in the same virtual network as your AKS cluster. This virtual network configuration allows the traffic between ACI and AKS to be secured. Like an AKS cluster, an ACI instance is a secure, logical compute resource that is isolated from other users.

Next steps

To get started with scaling applications, first follow the [quickstart to create an AKS cluster with the Azure CLI](#). You can then start to manually or automatically scale applications in your AKS cluster:

- Manually scale [pods](#) or [nodes](#)
- Use the [horizontal pod autoscaler](#)
- Use the [cluster autoscaler](#)

For more information on core Kubernetes and AKS concepts, see the following articles:

- [Kubernetes / AKS clusters and workloads](#)
- [Kubernetes / AKS access and identity](#)
- [Kubernetes / AKS security](#)
- [Kubernetes / AKS virtual networks](#)

- Kubernetes / AKS storage

Azure Kubernetes Service (AKS) node auto-repair

5/14/2021 • 2 minutes to read • [Edit Online](#)

AKS continuously monitors the health state of worker nodes and performs automatic node repair if they become unhealthy. The Azure virtual machine (VM) platform [performs maintenance on VMs](#) experiencing issues.

AKS and Azure VMs work together to minimize service disruptions for clusters.

In this document, you'll learn how automatic node repair functionality behaves for both Windows and Linux nodes.

How AKS checks for unhealthy nodes

AKS uses the following rules to determine if a node is unhealthy and needs repair:

- The node reports **NotReady** status on consecutive checks within a 10-minute timeframe.
- The node doesn't report any status within 10 minutes.

You can manually check the health state of your nodes with kubectl.

```
kubectl get nodes
```

How automatic repair works

NOTE

AKS initiates repair operations with the user account **aks-remediator**.

If AKS identifies an unhealthy node that remains unhealthy for 10 minutes, AKS takes the following actions:

1. Reboot the node.
2. If the reboot is unsuccessful, reimagine the node.
3. If the reimagine is unsuccessful, create and reimagine a new node.

Alternative remediations are investigated by AKS engineers if auto-repair is unsuccessful.

If AKS finds multiple unhealthy nodes during a health check, each node is repaired individually before another repair begins.

Limitations

In many cases, AKS can determine if a node is unhealthy and attempt to repair the issue, but there are cases where AKS either can't repair the issue or can't detect that there is an issue. For example, AKS can't detect issues if a node status is not being reported due to error in network configuration.

Next steps

Use [Availability Zones](#) to increase high availability with your AKS cluster workloads.

Cluster operator and developer best practices to build and manage applications on Azure Kubernetes Service (AKS)

4/8/2021 • 2 minutes to read • [Edit Online](#)

Building and running applications successfully in Azure Kubernetes Service (AKS) require understanding and implementation of some key considerations, including:

- Multi-tenancy and scheduler features.
- Cluster and pod security.
- Business continuity and disaster recovery.

The AKS product group, engineering teams, and field teams (including global black belts [GBBs]) contributed to, wrote, and grouped the following best practices and conceptual articles. Their purpose is to help cluster operators and developers understand the considerations above and implement the appropriate features.

Cluster operator best practices

As a cluster operator, work together with application owners and developers to understand their needs. You can then use the following best practices to configure your AKS clusters as needed.

Multi-tenancy

- [Best practices for cluster isolation](#)
 - Includes multi-tenancy core components and logical isolation with namespaces.
- [Best practices for basic scheduler features](#)
 - Includes using resource quotas and pod disruption budgets.
- [Best practices for advanced scheduler features](#)
 - Includes using taints and tolerations, node selectors and affinity, and inter-pod affinity and anti-affinity.
- [Best practices for authentication and authorization](#)
 - Includes integration with Azure Active Directory, using Kubernetes role-based access control (Kubernetes RBAC), using Azure RBAC, and pod identities.

Security

- [Best practices for cluster security and upgrades](#)
 - Includes securing access to the API server, limiting container access, and managing upgrades and node reboots.
- [Best practices for container image management and security](#)
 - Includes securing the image and runtimes and automated builds on base image updates.
- [Best practices for pod security](#)
 - Includes securing access to resources, limiting credential exposure, and using pod identities and digital key vaults.

Network and storage

- [Best practices for network connectivity](#)
 - Includes different network models, using ingress and web application firewalls (WAF), and securing

node SSH access.

- [Best practices for storage and backups](#)
 - Includes choosing the appropriate storage type and node size, dynamically provisioning volumes, and data backups.

Running enterprise-ready workloads

- [Best practices for business continuity and disaster recovery](#)
 - Includes using region pairs, multiple clusters with Azure Traffic Manager, and geo-replication of container images.

Developer best practices

As a developer or application owner, you can simplify your development experience and define requirements for application performance needs.

- [Best practices for application developers to manage resources](#)
 - Includes defining pod resource requests and limits, configuring development tools, and checking for application issues.
- [Best practices for pod security](#)
 - Includes securing access to resources, limiting credential exposure, and using pod identities and digital key vaults.

Kubernetes / AKS concepts

To help understand some of the features and components of these best practices, you can also see the following conceptual articles for clusters in Azure Kubernetes Service (AKS):

- [Kubernetes core concepts](#)
- [Access and identity](#)
- [Security concepts](#)
- [Network concepts](#)
- [Storage options](#)
- [Scaling options](#)

Next steps

If you need to get started with AKS, follow one of the quickstarts to deploy an Azure Kubernetes Service (AKS) cluster using the [Azure CLI](#) or [Azure portal](#).

Best practices for cluster isolation in Azure Kubernetes Service (AKS)

4/8/2021 • 3 minutes to read • [Edit Online](#)

As you manage clusters in Azure Kubernetes Service (AKS), you often need to isolate teams and workloads. AKS provides flexibility in how you can run multi-tenant clusters and isolate resources. To maximize your investment in Kubernetes, first understand and implement AKS multi-tenancy and isolation features.

This best practices article focuses on isolation for cluster operators. In this article, you learn how to:

- Plan for multi-tenant clusters and separation of resources
- Use logical or physical isolation in your AKS clusters

Design clusters for multi-tenancy

Kubernetes lets you logically isolate teams and workloads in the same cluster. The goal is to provide the least number of privileges, scoped to the resources each team needs. A Kubernetes [Namespace](#) creates a logical isolation boundary. Additional Kubernetes features and considerations for isolation and multi-tenancy include the following areas:

Scheduling

Scheduling uses basic features such as resource quotas and pod disruption budgets. For more information about these features, see [Best practices for basic scheduler features in AKS](#).

More advanced scheduler features include:

- Taints and tolerations
- Node selectors
- Node and pod affinity or anti-affinity.

For more information about these features, see [Best practices for advanced scheduler features in AKS](#).

Networking

Networking uses network policies to control the flow of traffic in and out of pods.

Authentication and authorization

Authentication and authorization uses:

- Role-based access control (RBAC)
- Azure Active Directory (AD) integration
- Pod identities
- Secrets in Azure Key Vault

For more information about these features, see [Best practices for authentication and authorization in AKS](#).

Containers

Containers include:

- The Azure Policy Add-on for AKS to enforce pod security.
- The use of pod security contexts.
- Scanning both images and the runtime for vulnerabilities.

- Using App Armor or Seccomp (Secure Computing) to restrict container access to the underlying node.

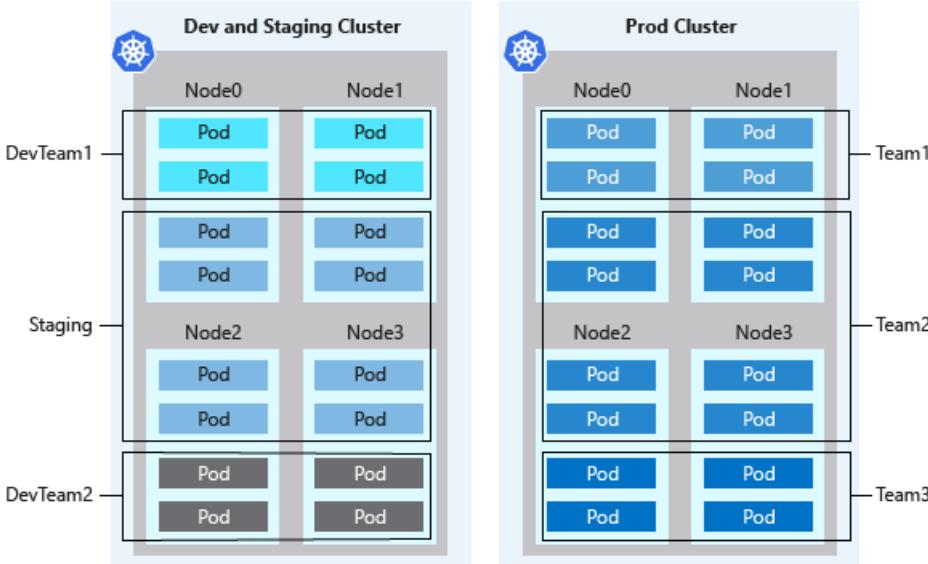
Logically isolate clusters

Best practice guidance

Separate teams and projects using *logical isolation*. Minimize the number of physical AKS clusters you deploy to isolate teams or applications.

With logical isolation, a single AKS cluster can be used for multiple workloads, teams, or environments.

Kubernetes [Namespaces](#) form the logical isolation boundary for workloads and resources.



Logical separation of clusters usually provides a higher pod density than physically isolated clusters, with less excess compute capacity sitting idle in the cluster. When combined with the Kubernetes cluster autoscaler, you can scale the number of nodes up or down to meet demands. This best practice approach to autoscaling minimizes costs by running only the number of nodes required.

Currently, Kubernetes environments aren't completely safe for hostile multi-tenant usage. In a multi-tenant environment, multiple tenants are working on a common, shared infrastructure. If all tenants cannot be trusted, you will need extra planning to prevent tenants from impacting the security and service of others.

Additional security features, like *Pod Security Policies* or Kubernetes RBAC for nodes, efficiently block exploits. For true security when running hostile multi-tenant workloads, you should only trust a hypervisor. The security domain for Kubernetes becomes the entire cluster, not an individual node.

For these types of hostile multi-tenant workloads, you should use physically isolated clusters.

Physically isolate clusters

Best practice guidance

Minimize the use of physical isolation for each separate team or application deployment. Instead, use *logical isolation*, as discussed in the previous section.

Physically separating AKS clusters is a common approach to cluster isolation. In this isolation model, teams or workloads are assigned their own AKS cluster. While physical isolation might look like the easiest way to isolate workloads or teams, it adds management and financial overhead. Now, you must maintain these multiple clusters and individually provide access and assign permissions. You'll also be billed for each the individual node.



Physically separate clusters usually have a low pod density. Since each team or workload has their own AKS cluster, the cluster is often over-provisioned with compute resources. Often, a small number of pods are scheduled on those nodes. Unclaimed node capacity can't be used for applications or services in development by other teams. These excess resources contribute to the additional costs in physically separate clusters.

Next steps

This article focused on cluster isolation. For more information about cluster operations in AKS, see the following best practices:

- [Basic Kubernetes scheduler features](#)
- [Advanced Kubernetes scheduler features](#)
- [Authentication and authorization](#)

Best practices for basic scheduler features in Azure Kubernetes Service (AKS)

4/8/2021 • 5 minutes to read • [Edit Online](#)

As you manage clusters in Azure Kubernetes Service (AKS), you often need to isolate teams and workloads. The Kubernetes scheduler lets you control the distribution of compute resources, or limit the impact of maintenance events.

This best practices article focuses on basic Kubernetes scheduling features for cluster operators. In this article, you learn how to:

- Use resource quotas to provide a fixed amount of resources to teams or workloads
- Limit the impact of scheduled maintenance using pod disruption budgets
- Check for missing pod resource requests and limits using the `kube-advisor` tool

Enforce resource quotas

Best practice guidance

Plan and apply resource quotas at the namespace level. If pods don't define resource requests and limits, reject the deployment. Monitor resource usage and adjust quotas as needed.

Resource requests and limits are placed in the pod specification. Limits are used by the Kubernetes scheduler at deployment time to find an available node in the cluster. Limits and requests work at the individual pod level. For more information about how to define these values, see [Define pod resource requests and limits](#)

To provide a way to reserve and limit resources across a development team or project, you should use *resource quotas*. These quotas are defined on a namespace, and can be used to set quotas on the following basis:

- **Compute resources**, such as CPU and memory, or GPUs.
- **Storage resources**, including the total number of volumes or amount of disk space for a given storage class.
- **Object count**, such as maximum number of secrets, services, or jobs can be created.

Kubernetes doesn't overcommit resources. Once your cumulative resource request total passes the assigned quota, all further deployments will be unsuccessful.

When you define resource quotas, all pods created in the namespace must provide limits or requests in their pod specifications. If they don't provide these values, you can reject the deployment. Instead, you can [configure default requests and limits for a namespace](#).

The following example YAML manifest named `dev-app-team-quotas.yaml` sets a hard limit of a total of 10 CPUs, 20Gi of memory, and 10 pods:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: dev-app-team
spec:
  hard:
    cpu: "10"
    memory: 20Gi
    pods: "10"
```

This resource quota can be applied by specifying the namespace, such as *dev-apps*:

```
kubectl apply -f dev-app-team-quotas.yaml --namespace dev-apps
```

Work with your application developers and owners to understand their needs and apply the appropriate resource quotas.

For more information about available resource objects, scopes, and priorities, see [Resource quotas in Kubernetes](#).

Plan for availability using pod disruption budgets

Best practice guidance

To maintain the availability of applications, define Pod Disruption Budgets (PDBs) to make sure that a minimum number of pods are available in the cluster.

There are two disruptive events that cause pods to be removed:

Involuntary disruptions

Involuntary disruptions are events beyond the typical control of the cluster operator or application owner. Include:

- Hardware failure on the physical machine
- Kernel panic
- Deletion of a node VM

Involuntary disruptions can be mitigated by:

- Using multiple replicas of your pods in a deployment.
- Running multiple nodes in the AKS cluster.

Voluntary disruptions

Voluntary disruptions are events requested by the cluster operator or application owner. Include:

- Cluster upgrades
- Updated deployment template
- Accidentally deleting a pod

Kubernetes provides *pod disruption budgets* for voluntary disruptions, letting you plan for how deployments or replica sets respond when a voluntary disruption event occurs. Using pod disruption budgets, cluster operators can define a minimum available or maximum unavailable resource count.

If you upgrade a cluster or update a deployment template, the Kubernetes scheduler will schedule extra pods on other nodes before allowing voluntary disruption events to continue. The scheduler waits to reboot a node until

the defined number of pods are successfully scheduled on other nodes in the cluster.

Let's look at an example of a replica set with five pods that run NGINX. The pods in the replica set are assigned the label `app: nginx-frontend`. During a voluntary disruption event, such as a cluster upgrade, you want to make sure at least three pods continue to run. The following YAML manifest for a `PodDisruptionBudget` object defines these requirements:

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: nginx-pdb
spec:
  minAvailable: 3
  selector:
    matchLabels:
      app: nginx-frontend
```

You can also define a percentage, such as `60%`, which allows you to automatically compensate for the replica set scaling up the number of pods.

You can define a maximum number of unavailable instances in a replica set. Again, a percentage for the maximum unavailable pods can also be defined. The following pod disruption budget YAML manifest defines that no more than two pods in the replica set be unavailable:

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: nginx-pdb
spec:
  maxUnavailable: 2
  selector:
    matchLabels:
      app: nginx-frontend
```

Once your pod disruption budget is defined, you create it in your AKS cluster as with any other Kubernetes object:

```
kubectl apply -f nginx-pdb.yaml
```

Work with your application developers and owners to understand their needs and apply the appropriate pod disruption budgets.

For more information about using pod disruption budgets, see [Specify a disruption budget for your application](#).

Regularly check for cluster issues with kube-advisor

Best practice guidance

Regularly run the latest version of `kube-advisor` open source tool to detect issues in your cluster. If you apply resource quotas on an existing AKS cluster, run `kube-advisor` first to find pods that don't have resource requests and limits defined.

The `kube-advisor` tool is an associated AKS open source project that scans a Kubernetes cluster and reports identified issues. `kube-advisor` proves useful in identifying pods without resource requests and limits in place.

While the `kube-advisor` tool can report on resource request and limits missing in PodSpecs for Windows and

Linux applications, the tool itself must be scheduled on a Linux pod. Schedule a pod to run on a node pool with a specific OS using a [node selector](#) in the pod's configuration.

Tracking pods without set resource requests and limits in an AKS cluster hosting multiple development teams and applications can be difficult. As a best practice, regularly run `kube-advisor` on your AKS clusters, especially if you don't assign resource quotas to namespaces.

Next steps

This article focused on basic Kubernetes scheduler features. For more information about cluster operations in AKS, see the following best practices:

- [Multi-tenancy and cluster isolation](#)
- [Advanced Kubernetes scheduler features](#)
- [Authentication and authorization](#)

Best practices for advanced scheduler features in Azure Kubernetes Service (AKS)

4/22/2021 • 7 minutes to read • [Edit Online](#)

As you manage clusters in Azure Kubernetes Service (AKS), you often need to isolate teams and workloads. Advanced features provided by the Kubernetes scheduler let you control:

- Which pods can be scheduled on certain nodes.
- How multi-pod applications can be appropriately distributed across the cluster.

This best practices article focuses on advanced Kubernetes scheduling features for cluster operators. In this article, you learn how to:

- Use taints and tolerations to limit what pods can be scheduled on nodes.
- Give preference to pods to run on certain nodes with node selectors or node affinity.
- Split apart or group together pods with inter-pod affinity or anti-affinity.

Provide dedicated nodes using taints and tolerations

Best practice guidance:

Limit access for resource-intensive applications, such as ingress controllers, to specific nodes. Keep node resources available for workloads that require them, and don't allow scheduling of other workloads on the nodes.

When you create your AKS cluster, you can deploy nodes with GPU support or a large number of powerful CPUs. You can use these nodes for large data processing workloads such as machine learning (ML) or artificial intelligence (AI).

Since this node resource hardware is typically expensive to deploy, limit the workloads that can be scheduled on these nodes. Instead, you'd dedicate some nodes in the cluster to run ingress services and prevent other workloads.

This support for different nodes is provided by using multiple node pools. An AKS cluster provides one or more node pools.

The Kubernetes scheduler uses taints and tolerations to restrict what workloads can run on nodes.

- Apply a **taint** to a node to indicate only specific pods can be scheduled on them.
- Then apply a **toleration** to a pod, allowing them to *tolerate* a node's taint.

When you deploy a pod to an AKS cluster, Kubernetes only schedules pods on nodes whose taint aligns with the toleration. For example, assume you added a node pool in your AKS cluster for nodes with GPU support. You define name, such as *gpu*, then a value for scheduling. Setting this value to *NoSchedule* restricts the Kubernetes scheduler from scheduling pods with undefined toleration on the node.

```
az aks nodepool add \
    --resource-group myResourceGroup \
    --cluster-name myAKSCluster \
    --name taintnp \
    --node-taints sku=gpu:NoSchedule \
    --no-wait
```

With a taint applied to nodes in the node pool, you'll define a toleration in the pod specification that allows scheduling on the nodes. The following example defines the `sku: gpu` and `effect: NoSchedule` to tolerate the taint applied to the node pool in the previous step:

```
kind: Pod
apiVersion: v1
metadata:
  name: tf-mnist
spec:
  containers:
  - name: tf-mnist
    image: mcr.microsoft.com/azuredocs/samples-tf-mnist-demo:gpu
    resources:
      requests:
        cpu: 0.5
        memory: 2Gi
      limits:
        cpu: 4.0
        memory: 16Gi
    tolerations:
    - key: "sku"
      operator: "Equal"
      value: "gpu"
      effect: "NoSchedule"
```

When this pod is deployed using `kubectl apply -f gpu-toleration.yaml`, Kubernetes can successfully schedule the pod on the nodes with the taint applied. This logical isolation lets you control access to resources within a cluster.

When you apply taints, work with your application developers and owners to allow them to define the required tolerations in their deployments.

For more information about how to use multiple node pools in AKS, see [Create and manage multiple node pools for a cluster in AKS](#).

Behavior of taints and tolerations in AKS

When you upgrade a node pool in AKS, taints and tolerations follow a set pattern as they're applied to new nodes:

Default clusters that use VM scale sets

You can [taint a node pool](#) from the AKS API to have newly scaled out nodes receive API specified node taints.

Let's assume:

1. You begin with a two-node cluster: `node1` and `node2`.
2. You upgrade the node pool.
3. Two additional nodes are created: `node3` and `node4`.
4. The taints are passed on respectively.
5. The original `node1` and `node2` are deleted.

Clusters without VM scale set support

Again, let's assume:

1. You have a two-node cluster: *node1* and *node2*.
2. You upgrade then node pool.
3. An additional node is created: *node3*.
4. The taints from *node1* are applied to *node3*.
5. *node1* is deleted.
6. A new *node1* is created to replace to original *node1*.
7. The *node2* taints are applied to the new *node1*.
8. *node2* is deleted.

In essence *node1* becomes *node3*, and *node2* becomes the new *node1*.

When you scale a node pool in AKS, taints and tolerations do not carry over by design.

Control pod scheduling using node selectors and affinity

Best practice guidance

Control the scheduling of pods on nodes using node selectors, node affinity, or inter-pod affinity. These settings allow the Kubernetes scheduler to logically isolate workloads, such as by hardware in the node.

Taints and tolerations logically isolate resources with a hard cut-off. If the pod doesn't tolerate a node's taint, it isn't scheduled on the node.

Alternatively, you can use node selectors. For example, you label nodes to indicate locally attached SSD storage or a large amount of memory, and then define in the pod specification a node selector. Kubernetes schedules those pods on a matching node.

Unlike tolerations, pods without a matching node selector can still be scheduled on labeled nodes. This behavior allows unused resources on the nodes to consume, but prioritizes pods that define the matching node selector.

Let's look at an example of nodes with a high amount of memory. These nodes prioritize pods that request a high amount of memory. To ensure the resources don't sit idle, they also allow other pods to run. The follow example command adds a node pool with the label *hardware=highmem* to the *myAKSCluster* in the *myResourceGroup*. All nodes in that node pool will have this label.

```
az aks nodepool add \
    --resource-group myResourceGroup \
    --cluster-name myAKSCluster \
    --name labelnp \
    --node-count 1 \
    --labels hardware=highmem \
    --no-wait
```

A pod specification then adds the `nodeSelector` property to define a node selector that matches the label set on a node:

```

kind: Pod
apiVersion: v1
metadata:
  name: tf-mnist
spec:
  containers:
    - name: tf-mnist
      image: mcr.microsoft.com/azuredocs/samples-tf-mnist-demo:gpu
      resources:
        requests:
          cpu: 0.5
          memory: 2Gi
        limits:
          cpu: 4.0
          memory: 16Gi
  nodeSelector:
    hardware: highmem

```

When you use these scheduler options, work with your application developers and owners to allow them to correctly define their pod specifications.

For more information about using node selectors, see [Assigning Pods to Nodes](#).

Node affinity

A node selector is a basic solution for assigning pods to a given node. *Node affinity* provides more flexibility, allowing you to define what happens if the pod can't be matched with a node. You can:

- *Require* that Kubernetes scheduler matches a pod with a labeled host. Or,
- *Prefer* a match but allow the pod to be scheduled on a different host if no match is available.

The following example sets the node affinity to *requiredDuringSchedulingIgnoredDuringExecution*. This affinity requires the Kubernetes schedule to use a node with a matching label. If no node is available, the pod has to wait for scheduling to continue. To allow the pod to be scheduled on a different node, you can instead set the value to *preferredDuringSchedulingIgnoreDuringExecution*.

```

kind: Pod
apiVersion: v1
metadata:
  name: tf-mnist
spec:
  containers:
    - name: tf-mnist
      image: mcr.microsoft.com/azuredocs/samples-tf-mnist-demo:gpu
      resources:
        requests:
          cpu: 0.5
          memory: 2Gi
        limits:
          cpu: 4.0
          memory: 16Gi
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: hardware
                operator: In
                values: highmem

```

The *IgnoredDuringExecution* part of the setting indicates that the pod shouldn't be evicted from the node if the node labels change. The Kubernetes scheduler only uses the updated node labels for new pods being scheduled,

not pods already scheduled on the nodes.

For more information, see [Affinity and anti-affinity](#).

Inter-pod affinity and anti-affinity

One final approach for the Kubernetes scheduler to logically isolate workloads is using inter-pod affinity or anti-affinity. These settings define that pods either *shouldn't* or *should* be scheduled on a node that has an existing matching pod. By default, the Kubernetes scheduler tries to schedule multiple pods in a replica set across nodes. You can define more specific rules around this behavior.

For example, you have a web application that also uses an Azure Cache for Redis.

1. You use pod anti-affinity rules to request that the Kubernetes scheduler distributes replicas across nodes.
2. You use affinity rules to ensure each web app component is scheduled on the same host as a corresponding cache.

The distribution of pods across nodes looks like the following example:

NODE 1	NODE 2	NODE 3
webapp-1	webapp-2	webapp-3
cache-1	cache-2	cache-3

Inter-pod affinity and anti-affinity provide a more complex deployment than node selectors or node affinity. With the deployment, you logically isolate resources and control how Kubernetes schedules pods on nodes.

For a complete example of this web application with Azure Cache for Redis example, see [Co-locate pods on the same node](#).

Next steps

This article focused on advanced Kubernetes scheduler features. For more information about cluster operations in AKS, see the following best practices:

- [Multi-tenancy and cluster isolation](#)
- [Basic Kubernetes scheduler features](#)
- [Authentication and authorization](#)

Best practices for authentication and authorization in Azure Kubernetes Service (AKS)

4/8/2021 • 6 minutes to read • [Edit Online](#)

As you deploy and maintain clusters in Azure Kubernetes Service (AKS), you implement ways to manage access to resources and services. Without these controls:

- Accounts could have access to unnecessary resources and services.
- Tracking which set of credentials were used to make changes could be difficult.

This best practices article focuses on how a cluster operator can manage access and identity for AKS clusters. In this article, you learn how to:

- Authenticate AKS cluster users with Azure Active Directory.
- Control access to resources with Kubernetes role-based access control (Kubernetes RBAC).
- Use Azure RBAC to granularly control access to the AKS resource, the Kubernetes API at scale, and the `kubeconfig`.
- Use a managed identity to authenticate pods themselves with other services.

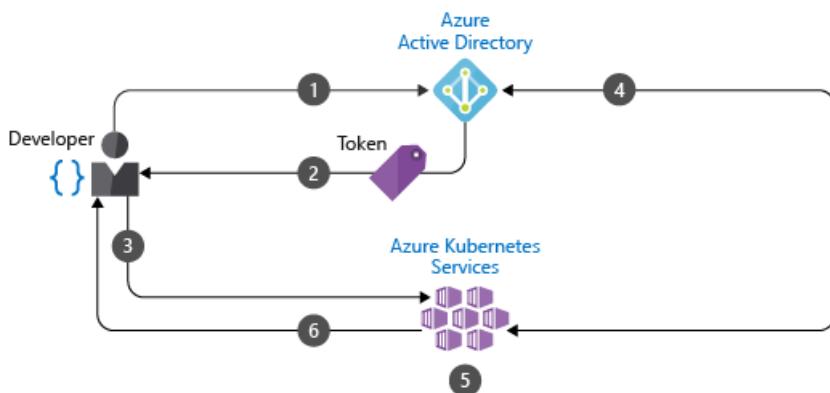
Use Azure Active Directory (Azure AD)

Best practice guidance

Deploy AKS clusters with Azure AD integration. Using Azure AD centralizes the identity management component. Any change in user account or group status is automatically updated in access to the AKS cluster. Scope users or groups to the minimum permissions amount using [Roles](#), [ClusterRoles](#), or [Bindings](#).

Your Kubernetes cluster developers and application owners need access to different resources. Kubernetes lacks an identity management solution for you to control the resources with which users can interact. Instead, you typically integrate your cluster with an existing identity solution. Enter Azure AD: an enterprise-ready identity management solution that integrates with AKS clusters.

With Azure AD-integrated clusters in AKS, you create *Roles* or *ClusterRoles* defining access permissions to resources. You then *bind* the roles to users or groups from Azure AD. Learn more about these Kubernetes RBAC in [the next section](#). Azure AD integration and how you control access to resources can be seen in the following diagram:



1. Developer authenticates with Azure AD.
2. The Azure AD token issuance endpoint issues the access token.

3. The developer performs an action using the Azure AD token, such as `kubectl create pod`.
4. Kubernetes validates the token with Azure AD and fetches the developer's group memberships.
5. Kubernetes RBAC and cluster policies are applied.
6. Developer's request is successful based on previous validation of Azure AD group membership and Kubernetes RBAC and policies.

To create an AKS cluster that uses Azure AD, see [Integrate Azure Active Directory with AKS](#).

Use Kubernetes role-based access control (Kubernetes RBAC)

Best practice guidance

Define user or group permissions to cluster resources with Kubernetes RBAC. Create roles and bindings that assign the least amount of permissions required. Integrate with Azure AD to automatically update any user status or group membership change and keep access to cluster resources current.

In Kubernetes, you provide granular access control to cluster resources. You define permissions at the cluster level, or to specific namespaces. You determine what resources can be managed and with what permissions. You then apply these roles to users or groups with a binding. For more information about *Roles*, *ClusterRoles*, and *Bindings*, see [Access and identity options for Azure Kubernetes Service \(AKS\)](#).

As an example, you create a role with full access to resources in the namespace named *finance-app*, as shown in the following example YAML manifest:

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: finance-app-full-access-role
  namespace: finance-app
rules:
- apiGroups: []
  resources: ["*"]
  verbs: ["*"]
```

You then create a RoleBinding and bind the Azure AD user *developer1@contoso.com* to the RoleBinding, as shown in the following YAML manifest:

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: finance-app-full-access-role-binding
  namespace: finance-app
subjects:
- kind: User
  name: developer1@contoso.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: finance-app-full-access-role
  apiGroup: rbac.authorization.k8s.io
```

When *developer1@contoso.com* is authenticated against the AKS cluster, they have full permissions to resources in the *finance-app* namespace. In this way, you logically separate and control access to resources. Use Kubernetes RBAC in conjunction with Azure AD-integration.

To see how to use Azure AD groups to control access to Kubernetes resources using Kubernetes RBAC, see [Control access to cluster resources using role-based access control and Azure Active Directory identities in AKS](#).

Use Azure RBAC

Best practice guidance

Use Azure RBAC to define the minimum required user and group permissions to AKS resources in one or more subscriptions.

There are two levels of access needed to fully operate an AKS cluster:

1. Access the AKS resource on your Azure subscription.

This access level allows you to:

- Control scaling or upgrading your cluster using the AKS APIs
- Pull your `kubeconfig`.

To see how to control access to the AKS resource and the `kubeconfig`, see [Limit access to cluster configuration file](#).

2. Access to the Kubernetes API.

This access level is controlled either by:

- [Kubernetes RBAC](#) (traditionally) or
- By integrating Azure RBAC with AKS for Kubernetes authorization.

To see how to granularly give permissions to the Kubernetes API using Azure RBAC, see [Use Azure RBAC for Kubernetes authorization](#).

Use Pod-managed Identities

Best practice guidance

Don't use fixed credentials within pods or container images, as they are at risk of exposure or abuse. Instead, use *pod identities* to automatically request access using a central Azure AD identity solution.

NOTE

Pod identities are intended for use with Linux pods and container images only. Pod-managed identities support for Windows containers is coming soon.

To access other Azure services, like Cosmos DB, Key Vault, or Blob Storage, the pod needs access credentials. You could define access credentials with the container image or inject them as a Kubernetes secret. Either way, you would need to manually create and assign them. Usually, these credentials are reused across pods and aren't regularly rotated.

With pod-managed identities for Azure resources, you automatically request access to services through Azure AD. Pod-managed identities is now currently in preview for AKS. Please refer to the [Use Azure Active Directory pod-managed identities in Azure Kubernetes Service (Preview)](<https://docs.microsoft.com/azure/aks/use-azure-ad-pod-identity>) documentation to get started.

Instead of manually defining credentials for pods, pod-managed identities request an access token in real time, using it to access only their assigned services. In AKS, there are two components that handle the operations to allow pods to use managed identities:

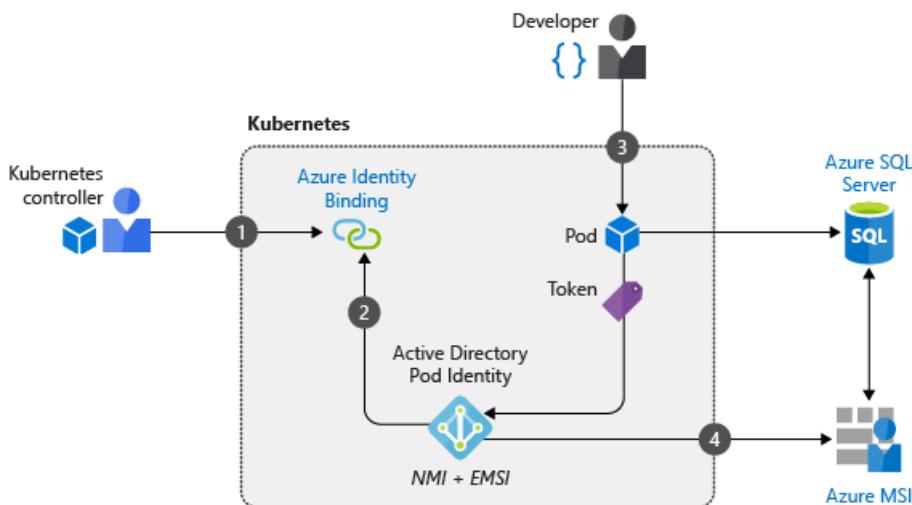
- **The Node Management Identity (NMI) server** is a pod that runs as a DaemonSet on each node in the AKS cluster. The NMI server listens for pod requests to Azure services.

- The Azure Resource Provider queries the Kubernetes API server and checks for an Azure identity mapping that corresponds to a pod.

When pods request access to an Azure service, network rules redirect the traffic to the NMI server.

- The NMI server:
 - Identifies pods requesting access to Azure services based on their remote address.
 - Queries the Azure Resource Provider.
- The Azure Resource Provider checks for Azure identity mappings in the AKS cluster.
- The NMI server requests an access token from Azure AD based on the pod's identity mapping.
- Azure AD provides access to the NMI server, which is returned to the pod.
 - This access token can be used by the pod to then request access to services in Azure.

In the following example, a developer creates a pod that uses a managed identity to request access to Azure SQL Database:



- Cluster operator creates a service account to map identities when pods request access to services.
- The NMI server is deployed to relay any pod requests, along with the Azure Resource Provider, for access tokens to Azure AD.
- A developer deploys a pod with a managed identity that requests an access token through the NMI server.
- The token is returned to the pod and used to access Azure SQL Database

NOTE

Pod-managed identities is currently in preview status.

To use Pod-managed identities, see [Use Azure Active Directory pod-managed identities in Azure Kubernetes Service \(Preview\)](#).

Next steps

This best practices article focused on authentication and authorization for your cluster and resources. To implement some of these best practices, see the following articles:

- [Integrate Azure Active Directory with AKS](#)
- [Use Azure Active Directory pod-managed identities in Azure Kubernetes Service \(Preview\)](#)

For more information about cluster operations in AKS, see the following best practices:

- [Multi-tenancy and cluster isolation](#)

- Basic Kubernetes scheduler features
- Advanced Kubernetes scheduler features

Best practices for cluster security and upgrades in Azure Kubernetes Service (AKS)

4/21/2021 • 9 minutes to read • [Edit Online](#)

As you manage clusters in Azure Kubernetes Service (AKS), workload and data security is a key consideration. When you run multi-tenant clusters using logical isolation, you especially need to secure resource and workload access. Minimize the risk of attack by applying the latest Kubernetes and node OS security updates.

This article focuses on how to secure your AKS cluster. You learn how to:

- Use Azure Active Directory and Kubernetes role-based access control (Kubernetes RBAC) to secure API server access.
- Secure container access to node resources.
- Upgrade an AKS cluster to the latest Kubernetes version.
- Keep nodes up to date and automatically apply security patches.

You can also read the best practices for [container image management](#) and for [pod security](#).

You can also use [Azure Kubernetes Services integration with Security Center](#) to help detect threats and view recommendations for securing your AKS clusters.

Secure access to the API server and cluster nodes

Best practice guidance

One of the most important ways to secure your cluster is to secure access to the Kubernetes API server. To control access to the API server, integrate Kubernetes RBAC with Azure Active Directory (Azure AD). With these controls, you secure AKS the same way that you secure access to your Azure subscriptions.

The Kubernetes API server provides a single connection point for requests to perform actions within a cluster. To secure and audit access to the API server, limit access and provide the lowest possible permission levels. While this approach isn't unique to Kubernetes, it's especially important when you've logically isolated your AKS cluster for multi-tenant use.

Azure AD provides an enterprise-ready identity management solution that integrates with AKS clusters. Since Kubernetes doesn't provide an identity management solution, you may be hard-pressed to granularly restrict access to the API server. With Azure AD-integrated clusters in AKS, you use your existing user and group accounts to authenticate users to the API server.



Using Kubernetes RBAC and Azure AD-integration, you can secure the API server and provide the minimum permissions required to a scoped resource set, like a single namespace. You can grant different Azure AD users or groups different Kubernetes roles. With granular permissions, you can restrict access to the API server and provide a clear audit trail of actions performed.

The recommended best practice is to use *groups* to provide access to files and folders instead of individual identities. For example, use an Azure AD *group* membership to bind users to Kubernetes roles rather than individual *users*. As a user's group membership changes, their access permissions on the AKS cluster change accordingly.

Meanwhile, let's say you bind the individual user directly to a role and their job function changes. While the Azure AD group memberships update, their permissions on the AKS cluster would not. In this scenario, the user ends up with more permissions than they require.

For more information about Azure AD integration, Kubernetes RBAC, and Azure RBAC, see [Best practices for authentication and authorization in AKS](#).

Secure container access to resources

Best practice guidance

Limit access to actions that containers can perform. Provide the least number of permissions, and avoid the use of root access or privileged escalation.

In the same way that you should grant users or groups the minimum privileges required, you should also limit containers to only necessary actions and processes. To minimize the risk of attack, avoid configuring applications and containers that require escalated privileges or root access.

For example, set `allowPrivilegeEscalation: false` in the pod manifest. These built-in Kubernetes *pod security contexts* let you define additional permissions, such as the user or group to run as, or the Linux capabilities to expose. For more best practices, see [Secure pod access to resources](#).

For even more granular control of container actions, you can also use built-in Linux security features such as *AppArmor* and *seccomp*.

1. Define Linux security features at the node level.
2. Implement features through a pod manifest.

Built-in Linux security features are only available on Linux nodes and pods.

NOTE

Currently, Kubernetes environments aren't completely safe for hostile multi-tenant usage. Additional security features, like *AppArmor*, *seccomp*, *Pod Security Policies*, or Kubernetes RBAC for nodes, efficiently block exploits.

For true security when running hostile multi-tenant workloads, only trust a hypervisor. The security domain for Kubernetes becomes the entire cluster, not an individual node.

For these types of hostile multi-tenant workloads, you should use physically isolated clusters.

App Armor

To limit container actions, you can use the [AppArmor](#) Linux kernel security module. AppArmor is available as part of the underlying AKS node OS, and is enabled by default. You create AppArmor profiles that restrict read, write, or execute actions, or system functions like mounting filesystems. Default AppArmor profiles restrict access to various `/proc` and `/sys` locations, and provide a means to logically isolate containers from the underlying node. AppArmor works for any application that runs on Linux, not just Kubernetes pods.



To see AppArmor in action, the following example creates a profile that prevents writing to files.

1. [SSH](#) to an AKS node.
2. Create a file named `deny-write.profile`.
3. Paste the following content:

```
#include <tunables/global>
profile k8s-apparmor-example-deny-write flags=(attach_disconnected) {
    #include <abstractions/base>

    file,
    # Deny all file writes.
    deny /** w,
}
```

AppArmor profiles are added using the `apparmor_parser` command.

1. Add the profile to AppArmor.
2. Specify the name of the profile created in the previous step:

```
sudo apparmor_parser deny-write.profile
```

If the profile is correctly parsed and applied to AppArmor, you won't see any output and you'll be returned to the command prompt.

3. From your local machine, create a pod manifest named `aks-apparmor.yaml`. This manifest:
 - Defines an annotation for `container.apparmor.security.beta.kubernetes.io`.
 - References the `deny-write` profile created in the previous steps.

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-apparmor
  annotations:
    container.apparmor.security.beta.kubernetes.io/hello: localhost/k8s-apparmor-example-deny-write
spec:
  containers:
  - name: hello
    image: mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11
    command: [ "sh", "-c", "echo 'Hello AppArmor!' && sleep 1h" ]
```

4. With the pod deployed, use verify the `hello-apparmor` pod shows as *blocked*.

```
$ kubectl get pods

NAME        READY   STATUS    RESTARTS   AGE
aks-ssh     1/1     Running   0          4m2s
hello-apparmor 0/1     Blocked   0          50s
```

For more information about AppArmor, see [AppArmor profiles in Kubernetes](#).

Secure computing

While AppArmor works for any Linux application, [seccomp \(secure computing\)](#) works at the process level. Seccomp is also a Linux kernel security module, and is natively supported by the Docker runtime used by AKS nodes. With seccomp, you can limit container process calls. Align to the best practice of granting the container minimal permission only to run by:

- Defining with filters what actions to allow or deny.
- Annotating within a pod YAML manifest to associate with the seccomp filter.

To see seccomp in action, create a filter that prevents changing permissions on a file.

1. [SSH](#) to an AKS node.
2. Create a seccomp filter named `/var/lib/kubelet/seccomp/prevent-chmod`.
3. Paste the following content:

```
{
  "defaultAction": "SCMP_ACT_ALLOW",
  "syscalls": [
    {
      "name": "chmod",
      "action": "SCMP_ACT_ERRNO"
    },
    {
      "name": "fchmodat",
      "action": "SCMP_ACT_ERRNO"
    },
    {
      "name": "chmodat",
      "action": "SCMP_ACT_ERRNO"
    }
  ]
}
```

In version 1.19 and later, you need to configure the following:

```
{
  "defaultAction": "SCMP_ACT_ALLOW",
  "syscalls": [
    {
      "names": ["chmod","fchmodat","chmodat"],
      "action": "SCMP_ACT_ERRNO"
    }
  ]
}
```

4. From your local machine, create a pod manifest named `aks-seccomp.yaml` and paste the following content. This manifest:

- Defines an annotation for `seccomp.security.alpha.kubernetes.io`.

- References the `prevent-chmod` filter created in the previous step.

```
apiVersion: v1
kind: Pod
metadata:
  name: chmod-prevented
  annotations:
    seccomp.security.alpha.kubernetes.io/pod: localhost/prevent-chmod
spec:
  containers:
  - name: chmod
    image: mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11
    command:
      - "chmod"
    args:
      - "777"
      - /etc/hostname
  restartPolicy: Never
```

In version 1.19 and later, you need to configure the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: chmod-prevented
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: prevent-chmod
  containers:
  - name: chmod
    image: mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11
    command:
      - "chmod"
    args:
      - "777"
      - /etc/hostname
  restartPolicy: Never
```

5. Deploy the sample pod using the [kubectl apply](#) command:

```
kubectl apply -f ./aks-seccomp.yaml
```

6. View pod status using the [kubectl get pods](#) command.

- The pod reports an error.
- The `chmod` command is prevented from running by the seccomp filter, as shown in the following example output:

```
$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
chmod-prevented  0/1     Error     0          7s
```

For more information about available filters, see [Seccomp security profiles for Docker](#).

Regularly update to the latest version of Kubernetes

Best practice guidance

To stay current on new features and bug fixes, regularly upgrade the Kubernetes version in your AKS cluster.

Kubernetes releases new features at a quicker pace than more traditional infrastructure platforms. Kubernetes updates include:

- New features
- Bug or security fixes

New features typically move through *alpha* and *beta* status before they become *stable*. Once stable, are generally available and recommended for production use. Kubernetes new feature release cycle allows you to update Kubernetes without regularly encountering breaking changes or adjusting your deployments and templates.

AKS supports three minor versions of Kubernetes. Once a new minor patch version is introduced, the oldest minor version and patch releases supported are retired. Minor Kubernetes updates happen on a periodic basis. To stay within support, ensure you have a governance process to check for necessary upgrades. For more information, see [Supported Kubernetes versions AKS](#).

To check the versions that are available for your cluster, use the [az aks get-upgrades](#) command as shown in the following example:

```
az aks get-upgrades --resource-group myResourceGroup --name myAKSCluster
```

You can then upgrade your AKS cluster using the [az aks upgrade](#) command. The upgrade process safely:

- Cordons and drains one node at a time.
- Schedules pods on remaining nodes.
- Deploys a new node running the latest OS and Kubernetes versions.

IMPORTANT

Test new minor versions in a dev test environment and validate that your workload remains healthy with the new Kubernetes version.

Kubernetes may deprecate APIs (like in version 1.16) that your workloads rely on. When bringing new versions into production, consider using [multiple node pools on separate versions](#) and upgrade individual pools one at a time to progressively roll the update across a cluster. If running multiple clusters, upgrade one cluster at a time to progressively monitor for impact or changes.

```
az aks upgrade --resource-group myResourceGroup --name myAKSCluster --kubernetes-version  
KUBERNETES_VERSION
```

For more information about upgrades in AKS, see [Supported Kubernetes versions in AKS](#) and [Upgrade an AKS cluster](#).

Process Linux node updates and reboots using kured

Best practice guidance

While AKS automatically downloads and installs security fixes on each Linux node, it does not automatically reboot.

1. Use `kured` to watch for pending reboots.

2. Safely cordon and drain the node to allow the node to reboot.
3. Apply the updates.
4. Be as secure as possible with respect to the OS.

For Windows Server nodes, regularly perform an AKS upgrade operation to safely cordon and drain pods and deploy updated nodes.

Each evening, Linux nodes in AKS get security patches through their distro update channel. This behavior is automatically configured as the nodes are deployed in an AKS cluster. To minimize disruption and potential impact to running workloads, nodes are not automatically rebooted if a security patch or kernel update requires it.

The open-source [kured \(KUBernetes REboot Daemon\)](#) project by Weaveworks watches for pending node reboots. When a Linux node applies updates that require a reboot, the node is safely cordoned and drained to move and schedule the pods on other nodes in the cluster. Once the node is rebooted, it is added back into the cluster and Kubernetes resumes pod scheduling. To minimize disruption, only one node at a time is permitted to be rebooted by `kured`.



If you want even closer control over reboots, `kured` can integrate with Prometheus to prevent reboots if there are other maintenance events or cluster issues in progress. This integration reduces complication by rebooting nodes while you are actively troubleshooting other issues.

For more information about how to handle node reboots, see [Apply security and kernel updates to nodes in AKS](#).

Next steps

This article focused on how to secure your AKS cluster. To implement some of these areas, see the following articles:

- [Integrate Azure Active Directory with AKS](#)
- [Upgrade an AKS cluster to the latest version of Kubernetes](#)
- [Process security updates and node reboots with kured](#)

Best practices for container image management and security in Azure Kubernetes Service (AKS)

4/8/2021 • 2 minutes to read • [Edit Online](#)

Container and container image security is a major priority while you develop and run applications in Azure Kubernetes Service (AKS). Containers with outdated base images or unpatched application runtimes introduce a security risk and possible attack vector.

Minimize risks by integrating and running scan and remediation tools in your containers at build and runtime. The earlier you catch the vulnerability or outdated base image, the more secure your cluster.

In this article, “*containers*” means both:

- The container images stored in a container registry.
- The running containers.

This article focuses on how to secure your containers in AKS. You learn how to:

- Scan for and remediate image vulnerabilities.
- Automatically trigger and redeploy container images when a base image is updated.

You can also read the best practices for [cluster security](#) and for [pod security](#).

You can also use [Container security in Security Center](#) to help scan your containers for vulnerabilities. [Azure Container Registry integration](#) with Security Center helps protect your images and registry from vulnerabilities.

Secure the images and run time

Best practice guidance

Scan your container images for vulnerabilities. Only deploy validated images. Regularly update the base images and application runtime. Redeploy workloads in the AKS cluster.

When adopting container-based workloads, you'll want to verify the security of images and runtime used to build your own applications. How do you avoid introducing security vulnerabilities into your deployments?

- Include in your deployment workflow a process to scan container images using tools such as [Twistlock](#) or [Aqua](#).
- Only allow verified images to be deployed.



For example, you can use a continuous integration and continuous deployment (CI/CD) pipeline to automate the image scans, verification, and deployments. Azure Container Registry includes these vulnerabilities scanning capabilities.

Automatically build new images on base image update

Best practice guidance

As you use base images for application images, use automation to build new images when the base image is updated. Since updated base images typically include security fixes, update any downstream application container images.

Each time a base image is updated, you should also update any downstream container images. Integrate this build process into validation and deployment pipelines such as [Azure Pipelines](#) or Jenkins. These pipelines make sure that your applications continue to run on the updated based images. Once your application container images are validated, the AKS deployments can then be updated to run the latest, secure images.

Azure Container Registry Tasks can also automatically update container images when the base image is updated. With this feature, you build a few base images and keep them updated with bug and security fixes.

For more information about base image updates, see [Automate image builds on base image update with Azure Container Registry Tasks](#).

Next steps

This article focused on how to secure your containers. To implement some of these areas, see the following articles:

- [Automate image builds on base image update with Azure Container Registry Tasks](#)

Best practices for network connectivity and security in Azure Kubernetes Service (AKS)

4/8/2021 • 10 minutes to read • [Edit Online](#)

As you create and manage clusters in Azure Kubernetes Service (AKS), you provide network connectivity for your nodes and applications. These network resources include IP address ranges, load balancers, and ingress controllers. To maintain a high quality of service for your applications, you need to strategize and configure these resources.

This best practices article focuses on network connectivity and security for cluster operators. In this article, you learn how to:

- Compare the kubenet and Azure Container Networking Interface (CNI) network modes in AKS.
- Plan for required IP addressing and connectivity.
- Distribute traffic using load balancers, ingress controllers, or a web application firewall (WAF).
- Securely connect to cluster nodes.

Choose the appropriate network model

Best practice guidance

Use Azure CNI networking in AKS for integration with existing virtual networks or on-premises networks. This network model allows greater separation of resources and controls in an enterprise environment.

Virtual networks provide the basic connectivity for AKS nodes and customers to access your applications. There are two different ways to deploy AKS clusters into virtual networks:

- **Azure CNI networking**

Deploys into a virtual network and uses the [Azure CNI](#) Kubernetes plugin. Pods receive individual IPs that can route to other network services or on-premises resources.

- **Kubenet networking**

Azure manages the virtual network resources as the cluster is deployed and uses the [kubenet](#) Kubernetes plugin.

For production deployments, both kubenet and Azure CNI are valid options.

CNI Networking

Azure CNI is a vendor-neutral protocol that lets the container runtime make requests to a network provider. It assigns IP addresses to pods and nodes, and provides IP address management (IPAM) features as you connect to existing Azure virtual networks. Each node and pod resource receives an IP address in the Azure virtual network - no need for extra routing to communicate with other resources or services.



Notably, Azure CNI networking for production allows for separation of control and management of resources. From a security perspective, you often want different teams to manage and secure those resources. With Azure CNI networking, you connect to existing Azure resources, on-premises resources, or other services directly via IP addresses assigned to each pod.

When you use Azure CNI networking, the virtual network resource is in a separate resource group to the AKS cluster. Delegate permissions for the AKS cluster identity to access and manage these resources. The cluster identity used by the AKS cluster must have at least [Network Contributor](#) permissions on the subnet within your virtual network.

If you wish to define a [custom role](#) instead of using the built-in Network Contributor role, the following permissions are required:

- `Microsoft.Network/virtualNetworks/subnets/join/action`
- `Microsoft.Network/virtualNetworks/subnets/read`

By default, AKS uses a managed identity for its cluster identity. However, you are able to use a service principal instead. For more information about:

- AKS service principal delegation, see [Delegate access to other Azure resources](#).
- Managed identities, see [Use managed identities](#).

As each node and pod receives its own IP address, plan out the address ranges for the AKS subnets. Keep in mind:

- The subnet must be large enough to provide IP addresses for every node, pods, and network resource that you deploy.
 - With both kubenet and Azure CNI networking, each node running has default limits to the number of pods.
- Each AKS cluster must be placed in its own subnet.
- Avoid using IP address ranges that overlap with existing network resources.
 - Necessary to allow connectivity to on-premises or peered networks in Azure.
- To handle scale out events or cluster upgrades, you need extra IP addresses available in the assigned subnet.
 - This extra address space is especially important if you use Windows Server containers, as those node pools require an upgrade to apply the latest security patches. For more information on Windows Server nodes, see [Upgrade a node pool in AKS](#).

To calculate the IP address required, see [Configure Azure CNI networking in AKS](#).

When creating a cluster with Azure CNI networking, you specify other address ranges for the cluster, such as the Docker bridge address, DNS service IP, and service address range. In general, make sure these address ranges:

- Don't overlap each other.
- Don't overlap with any networks associated with the cluster, including any virtual networks, subnets, on-premises and peered networks.

For the specific details around limits and sizing for these address ranges, see [Configure Azure CNI networking in AKS](#).

Kubenet networking

Although kubenet doesn't require you to set up the virtual networks before the cluster is deployed, there are disadvantages to waiting:

- Since nodes and pods are placed on different IP subnets, User Defined Routing (UDR) and IP forwarding routes traffic between pods and nodes. This extra routing may reduce network performance.
- Connections to existing on-premises networks or peering to other Azure virtual networks can be complex.

Since you don't create the virtual network and subnets separately from the AKS cluster, Kubenet is ideal for:

- Small development or test workloads.
- Simple websites with low traffic.
- Lifting and shifting workloads into containers.

For most production deployments, you should plan for and use Azure CNI networking.

You can also [configure your own IP address ranges and virtual networks using kubenet](#). Like Azure CNI networking, these address ranges shouldn't overlap each other and shouldn't overlap with any networks associated with the cluster (virtual networks, subnets, on-premises and peered networks).

For the specific details around limits and sizing for these address ranges, see [Use kubenet networking with your own IP address ranges in AKS](#).

Distribute ingress traffic

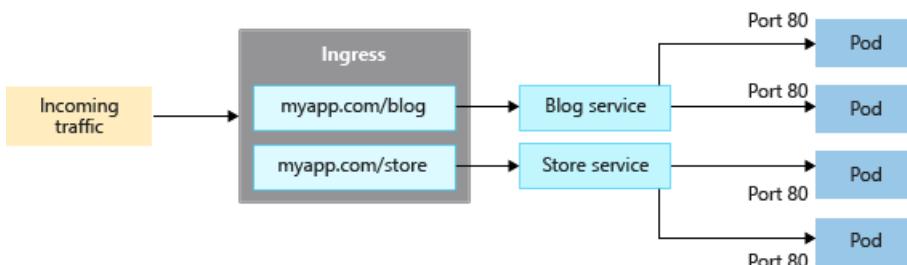
Best practice guidance

To distribute HTTP or HTTPS traffic to your applications, use ingress resources and controllers. Compared to an Azure load balancer, ingress controllers provide extra features and can be managed as native Kubernetes resources.

While an Azure load balancer can distribute customer traffic to applications in your AKS cluster, it's limited in understanding that traffic. A load balancer resource works at layer 4, and distributes traffic based on protocol or ports.

Most web applications using HTTP or HTTPS should use Kubernetes ingress resources and controllers, which work at layer 7. Ingress can distribute traffic based on the URL of the application and handle TLS/SSL termination. Ingress also reduces the number of IP addresses you expose and map.

With a load balancer, each application typically needs a public IP address assigned and mapped to the service in the AKS cluster. With an ingress resource, a single IP address can distribute traffic to multiple applications.



There are two components for ingress:

- An ingress *resource*
- An ingress *controller*

Ingress resource

The *ingress resource* is a YAML manifest of `kind: Ingress`. It defines the host, certificates, and rules to route traffic to services running in your AKS cluster.

The following example YAML manifest would distribute traffic for *myapp.com* to one of two services, *blogservice* or *storeservice*. The customer is directed to one service or the other based on the URL they access.

```
kind: Ingress
metadata:
  name: myapp-ingress
  annotations: kubernetes.io/ingress.class: "PublicIngress"
spec:
  tls:
    - hosts:
        - myapp.com
      secretName: myapp-secret
  rules:
    - host: myapp.com
      http:
        paths:
          - path: /blog
            backend:
              serviceName: blogservice
              servicePort: 80
          - path: /store
            backend:
              serviceName: storeservice
              servicePort: 80
```

Ingress controller

An *ingress controller* is a daemon that runs on an AKS node and watches for incoming requests. Traffic is then distributed based on the rules defined in the ingress resource. While the most common ingress controller is based on [NGINX](#), AKS doesn't restrict you to a specific controller. You can use [Contour](#), [HAProxy](#), [Traefik](#), etc.

Ingress controllers must be scheduled on a Linux node. Indicate that the resource should run on a Linux-based node using a node selector in your YAML manifest or Helm chart deployment. For more information, see [Use node selectors to control where pods are scheduled in AKS](#).

NOTE

Windows Server nodes shouldn't run the ingress controller.

There are many scenarios for ingress, including the following how-to guides:

- [Create a basic ingress controller with external network connectivity](#)
- [Create an ingress controller that uses an internal, private network and IP address](#)
- [Create an ingress controller that uses your own TLS certificates](#)
- [Create an ingress controller that uses Let's Encrypt to automatically generate TLS certificates with a dynamic public IP address or with a static public IP address](#)

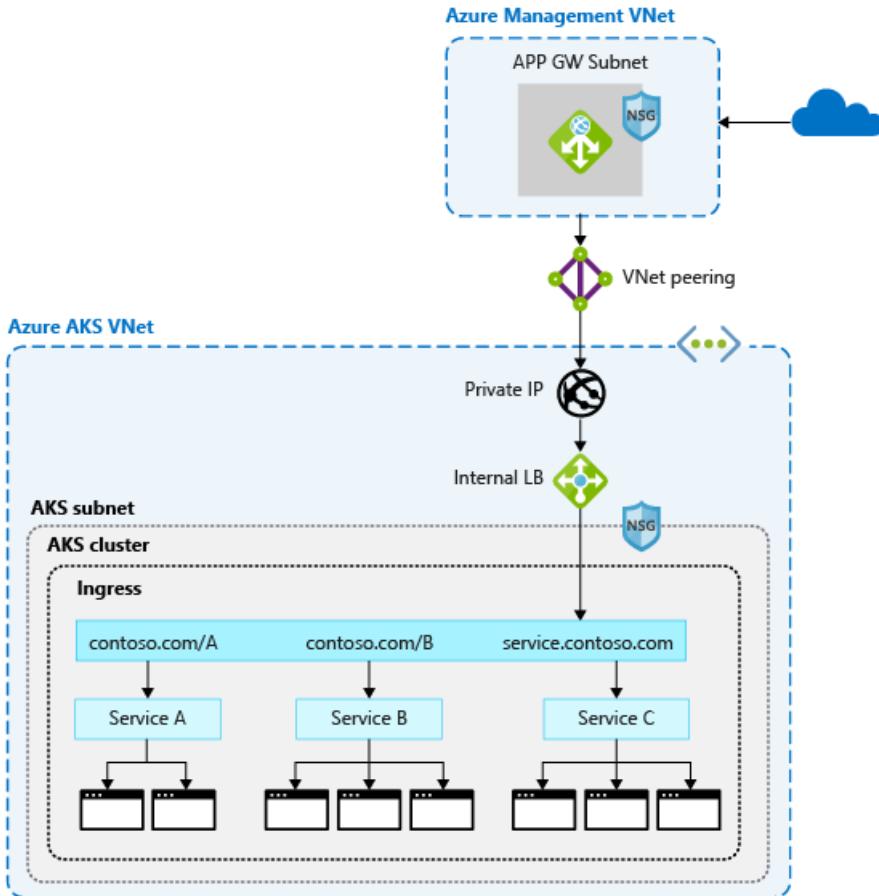
Secure traffic with a web application firewall (WAF)

Best practice guidance

To scan incoming traffic for potential attacks, use a web application firewall (WAF) such as [Barracuda WAF for Azure](#) or Azure Application Gateway. These more advanced network resources can also route traffic beyond just HTTP and HTTPS connections or basic TLS termination.

Typically, an ingress controller is a Kubernetes resource in your AKS cluster that distributes traffic to services and applications. The controller runs as a daemon on an AKS node, and consumes some of the node's resources, like CPU, memory, and network bandwidth. In larger environments, you'll want to:

- Offload some of this traffic routing or TLS termination to a network resource outside of the AKS cluster.
- Scan incoming traffic for potential attacks.



For that extra layer of security, a web application firewall (WAF) filters the incoming traffic. With a set of rules, the Open Web Application Security Project (OWASP) watches for attacks like cross-site scripting or cookie poisoning. [Azure Application Gateway](#) (currently in preview in AKS) is a WAF that integrates with AKS clusters, locking in these security features before the traffic reaches your AKS cluster and applications.

Since other third-party solutions also perform these functions, you can continue to use existing investments or expertise in your preferred product.

Load balancer or ingress resources continually run in your AKS cluster and refine the traffic distribution. App Gateway can be centrally managed as an ingress controller with a resource definition. To get started, [create an Application Gateway Ingress controller](#).

Control traffic flow with network policies

Best practice guidance

Use network policies to allow or deny traffic to pods. By default, all traffic is allowed between pods within a cluster. For improved security, define rules that limit pod communication.

Network policy is a Kubernetes feature available in AKS that lets you control the traffic flow between pods. You allow or deny traffic to the pod based on settings such as assigned labels, namespace, or traffic port. Network policies are a cloud-native way to control the flow of traffic for pods. As pods are dynamically created in an AKS cluster, required network policies can be automatically applied.

To use network policy, enable the feature when you create a new AKS cluster. You can't enable network policy on an existing AKS cluster. Plan ahead to enable network policy on the necessary clusters.

NOTE

Network policy should only be used for Linux-based nodes and pods in AKS.

You create a network policy as a Kubernetes resource using a YAML manifest. Policies are applied to defined pods, with ingress or egress rules defining traffic flow.

The following example applies a network policy to pods with the *app: backend* label applied to them. The ingress rule only allows traffic from pods with the *app: frontend* label:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: backend-policy
spec:
  podSelector:
    matchLabels:
      app: backend
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
```

To get started with policies, see [Secure traffic between pods using network policies in Azure Kubernetes Service \(AKS\)](#).

Securely connect to nodes through a bastion host

Best practice guidance

Don't expose remote connectivity to your AKS nodes. Create a bastion host, or jump box, in a management virtual network. Use the bastion host to securely route traffic into your AKS cluster to remote management tasks.

You can complete most operations in AKS using the Azure management tools or through the Kubernetes API server. AKS nodes are only available on a private network and aren't connected to the public internet. To connect to nodes and provide maintenance and support, route your connections through a bastion host, or jump box. Verify this host lives in a separate, securely-peered management virtual network to the AKS cluster virtual network.



The management network for the bastion host should be secured, too. Use an [Azure ExpressRoute](#) or [VPN gateway](#) to connect to an on-premises network, and control access using network security groups.

Next steps

This article focused on network connectivity and security. For more information about network basics in Kubernetes, see [Network concepts for applications in Azure Kubernetes Service \(AKS\)](#)

Best practices for storage and backups in Azure Kubernetes Service (AKS)

4/8/2021 • 5 minutes to read • [Edit Online](#)

As you create and manage clusters in Azure Kubernetes Service (AKS), your applications often need storage. Make sure you understand pod performance needs and access methods so that you can select the best storage for your application. The AKS node size may impact your storage choices. Plan for ways to back up and test the restore process for attached storage.

This best practices article focuses on storage considerations for cluster operators. In this article, you learn:

- What types of storage are available.
- How to correctly size AKS nodes for storage performance.
- Differences between dynamic and static provisioning of volumes.
- Ways to back up and secure your data volumes.

Choose the appropriate storage type

Best practice guidance

Understand the needs of your application to pick the right storage. Use high performance, SSD-backed storage for production workloads. Plan for network-based storage when you need multiple concurrent connections.

Applications often require different types and speeds of storage. Determine the most appropriate storage type by asking the following questions.

- Do your applications need storage that connects to individual pods?
- Do your applications need storage shared across multiple pods?
- Is the storage for read-only access to data?
- Will the storage be used to write large amounts of structured data?

The following table outlines the available storage types and their capabilities:

USE CASE	VOLUME PLUGIN	READ/WRITE ONCE	READ-ONLY MANY	READ/WRITE MANY	WINDOWS SERVER CONTAINER SUPPORT
Shared configuration	Azure Files	Yes	Yes	Yes	Yes
Structured app data	Azure Disks	Yes	No	No	Yes
Unstructured data, file system operations	BlobFuse	Yes	Yes	Yes	No

AKS provides two primary types of secure storage for volumes backed by Azure Disks or Azure Files. Both use the default Azure Storage Service Encryption (SSE) that encrypts data at rest. Disks cannot be encrypted using

Azure Disk Encryption at the AKS node level.

Both Azure Files and Azure Disks are available in Standard and Premium performance tiers:

- *Premium* disks
 - Backed by high-performance solid-state disks (SSDs).
 - Recommended for all production workloads.
- *Standard* disks
 - Backed by regular spinning disks (HDDs).
 - Good for archival or infrequently accessed data.

Understand the application performance needs and access patterns to choose the appropriate storage tier. For more information about Managed Disks sizes and performance tiers, see [Azure Managed Disks overview](#)

Create and use storage classes to define application needs

Define the type of storage you want using Kubernetes *storage classes*. The storage class is then referenced in the pod or deployment specification. Storage class definitions work together to create the appropriate storage and connect it to pods.

For more information, see [Storage classes in AKS](#).

Size the nodes for storage needs

Best practice guidance

Each node size supports a maximum number of disks. Different node sizes also provide different amounts of local storage and network bandwidth. Plan appropriately for your application demands to deploy the right size of nodes.

AKS nodes run as various Azure VM types and sizes. Each VM size provides:

- A different amount of core resources such as CPU and memory.
- A maximum number of disks that can be attached.

Storage performance also varies between VM sizes for the maximum local and attached disk IOPS (input/output operations per second).

If your applications require Azure Disks as their storage solution, strategize an appropriate node VM size.

Storage capabilities and CPU and memory amounts play a major role when deciding on a VM size.

For example, while both the *Standard_B2ms* and *Standard_DS2_v2* VM sizes include a similar amount of CPU and memory resources, their potential storage performance is different:

NODE TYPE AND SIZE	VCPU	MEMORY (GIB)	MAX DATA DISKS	MAX UNCACHED DISK IOPS	MAX UNCACHED THROUGHPUT (MBPS)
Standard_B2ms	2	8	4	1,920	22.5
Standard_DS2_v2	2	7	8	6,400	96

In this example, the *Standard_DS2_v2* offers twice as many attached disks, and three to four times the amount of IOPS and disk throughput. If you only compared core compute resources and compared costs, you might have chosen the *Standard_B2ms* VM size with poor storage performance and limitations.

Work with your application development team to understand their storage capacity and performance needs.

Choose the appropriate VM size for the AKS nodes to meet or exceed their performance needs. Regularly baseline applications to adjust VM size as needed.

For more information about available VM sizes, see [Sizes for Linux virtual machines in Azure](#).

Dynamically provision volumes

Best practice guidance

To reduce management overhead and enable scaling, avoid statically create and assign persistent volumes.

Use dynamic provisioning. In your storage classes, define the appropriate reclaim policy to minimize unneeded storage costs once pods are deleted.

To attach storage to pods, use persistent volumes. Persistent volumes can be created manually or dynamically. Creating persistent volumes manually adds management overhead and limits your ability to scale. Instead, provision persistent volume dynamically to simplify storage management and allow your applications to grow and scale as needed.



A persistent volume claim (PVC) lets you dynamically create storage as needed. Underlying Azure disks are created as pods request them. In the pod definition, request a volume to be created and attached to a designated mount path.

For the concepts on how to dynamically create and use volumes, see [Persistent Volumes Claims](#).

To see these volumes in action, see how to dynamically create and use a persistent volume with [Azure Disks](#) or [Azure Files](#).

As part of your storage class definitions, set the appropriate *reclaimPolicy*. This reclaimPolicy controls the behavior of the underlying Azure storage resource when the pod is deleted. The underlying storage resource can either be deleted or retained for future pod use. Set the reclaimPolicy to *retain* or *delete*.

Understand your application needs, and implement regular checks for retained storage to minimize the amount of unused and billed storage.

For more information about storage class options, see [storage reclaim policies](#).

Secure and back up your data

Best practice guidance

Back up your data using an appropriate tool for your storage type, such as Velero or Azure Backup. Verify the integrity and security of those backups.

When your applications store and consume data persisted on disks or in files, you need to take regular backups or snapshots of that data. Azure Disks can use built-in snapshot technologies. Your applications may need to flush writes-to-disk before you perform the snapshot operation. [Velero](#) can back up persistent volumes along

with additional cluster resources and configurations. If you can't [remove state from your applications](#), back up the data from persistent volumes and regularly test the restore operations to verify data integrity and the processes required.

Understand the limitations of the different approaches to data backups and if you need to quiesce your data prior to snapshot. Data backups don't necessarily let you restore your application environment or cluster deployment. For more information about those scenarios, see [Best practices for business continuity and disaster recovery in AKS](#).

Next steps

This article focused on storage best practices in AKS. For more information about storage basics in Kubernetes, see [Storage concepts for applications in AKS](#).

Best practices for business continuity and disaster recovery in Azure Kubernetes Service (AKS)

4/8/2021 • 7 minutes to read • [Edit Online](#)

As you manage clusters in Azure Kubernetes Service (AKS), application uptime becomes important. By default, AKS provides high availability by using multiple nodes in a [Virtual Machine Scale Set \(VMSS\)](#). But these multiple nodes don't protect your system from a region failure. To maximize your uptime, plan ahead to maintain business continuity and prepare for disaster recovery.

This article focuses on how to plan for business continuity and disaster recovery in AKS. You learn how to:

- Plan for AKS clusters in multiple regions.
- Route traffic across multiple clusters by using Azure Traffic Manager.
- Use geo-replication for your container image registries.
- Plan for application state across multiple clusters.
- Replicate storage across multiple regions.

Plan for multiregion deployment

Best practice

When you deploy multiple AKS clusters, choose regions where AKS is available. Use paired regions.

An AKS cluster is deployed into a single region. To protect your system from region failure, deploy your application into multiple AKS clusters across different regions. When planning where to deploy your AKS cluster, consider:

- [AKS region availability](#)
 - Choose regions close to your users.
 - AKS continually expands into new regions.
- [Azure paired regions](#)
 - For your geographic area, choose two regions paired together.
 - Paired regions coordinate platform updates and prioritize recovery efforts where needed.
- [Service availability](#)
 - Decide whether your paired regions should be hot/hot, hot/warm, or hot/cold.
 - Do you want to run both regions at the same time, with one region *ready* to start serving traffic? Or,
 - Do you want to give one region time to get ready to serve traffic?

AKS region availability and paired regions are a joint consideration. Deploy your AKS clusters into paired regions designed to manage region disaster recovery together. For example, AKS is available in East US and West US. These regions are paired. Choose these two regions when you're creating an AKS BC/DR strategy.

When you deploy your application, add another step to your CI/CD pipeline to deploy to these multiple AKS clusters. Updating your deployment pipelines prevents applications from deploying into only one of your regions and AKS clusters. In that scenario, customer traffic directed to a secondary region won't receive the latest code updates.

Use Azure Traffic Manager to route traffic

Best practice

Azure Traffic Manager can direct you to your closest AKS cluster and application instance. For the best performance and redundancy, direct all application traffic through Traffic Manager before it goes to your AKS cluster.

If you have multiple AKS clusters in different regions, use Traffic Manager to control traffic flow to the applications running in each cluster. [Azure Traffic Manager](#) is a DNS-based traffic load balancer that can distribute network traffic across regions. Use Traffic Manager to route users based on cluster response time or based on geography.



If you have a single AKS cluster, you typically connect to the service IP or DNS name of a given application. In a multi-cluster deployment, you should connect to a Traffic Manager DNS name that points to the services on each AKS cluster. Define these services by using Traffic Manager endpoints. Each endpoint is the *service load balancer IP*. Use this configuration to direct network traffic from the Traffic Manager endpoint in one region to the endpoint in a different region.



Traffic Manager performs DNS lookups and returns your most appropriate endpoint. Nested profiles can

prioritize a primary location. For example, you should connect to their closest geographic region. If that region has a problem, Traffic Manager directs you to a secondary region. This approach ensures that you can connect to an application instance even if your closest geographic region is unavailable.

For information on how to set up endpoints and routing, see [Configure the geographic traffic routing method by using Traffic Manager](#).

Application routing with Azure Front Door Service

Using split TCP-based anycast protocol, [Azure Front Door Service](#) promptly connects your end users to the nearest Front Door POP (Point of Presence). More features of Azure Front Door Service:

- TLS termination
- Custom domain
- Web application firewall
- URL Rewrite
- Session affinity

Review the needs of your application traffic to understand which solution is the most suitable.

Interconnect regions with global virtual network peering

Connect both virtual networks to each other through [virtual network peering](#) to enable communication between clusters. Virtual network peering interconnects virtual networks, providing high bandwidth across Microsoft's backbone network - even across different geographic regions.

Before peering virtual networks with running AKS clusters, use the standard Load Balancer in your AKS cluster. This prerequisite makes Kubernetes services reachable across the virtual network peering.

Enable geo-replication for container images

Best practice

Store your container images in Azure Container Registry and geo-replicate the registry to each AKS region.

To deploy and run your applications in AKS, you need a way to store and pull the container images. Container Registry integrates with AKS, so it can securely store your container images or Helm charts. Container Registry supports multimaster geo-replication to automatically replicate your images to Azure regions around the world.

To improve performance and availability:

1. Use Container Registry geo-replication to create a registry in each region where you have an AKS cluster.
2. Each AKS cluster then pulls container images from the local container registry in the same region:



When you use Container Registry geo-replication to pull images from the same region, the results are:

- **Faster:** Pull images from high-speed, low-latency network connections within the same Azure region.

- **More reliable:** If a region is unavailable, your AKS cluster pulls the images from an available container registry.
- **Cheaper:** No network egress charge between datacenters.

Geo-replication is a *Premium* SKU container registry feature. For information on how to configure geo-replication, see [Container Registry geo-replication](#).

Remove service state from inside containers

Best practice

Avoid storing service state inside the container. Instead, use an Azure platform as a service (PaaS) that supports multi-region replication.

Service state refers to the in-memory or on-disk data required by a service to function. State includes the data structures and member variables that the service reads and writes. Depending on how the service is architected, the state might also include files or other resources stored on the disk. For example, the state might include the files a database uses to store data and transaction logs.

State can be either externalized or co-located with the code that manipulates the state. Typically, you externalize state by using a database or other data store that runs on different machines over the network or that runs out of process on the same machine.

Containers and microservices are most resilient when the processes that run inside them don't retain state. Since applications almost always contain some state, use a PaaS solution, such as:

- Azure Cosmos DB
- Azure Database for PostgreSQL
- Azure Database for MySQL
- Azure SQL Database

To build portable applications, see the following guidelines:

- [The 12-factor app methodology](#)
- [Run a web application in multiple Azure regions](#)

Create a storage migration plan

Best practice

If you use Azure Storage, prepare and test how to migrate your storage from the primary region to the backup region.

Your applications might use Azure Storage for their data. If so, your applications are spread across multiple AKS clusters in different regions. You need to keep the storage synchronized. Here are two common ways to replicate storage:

- Infrastructure-based asynchronous replication
- Application-based asynchronous replication

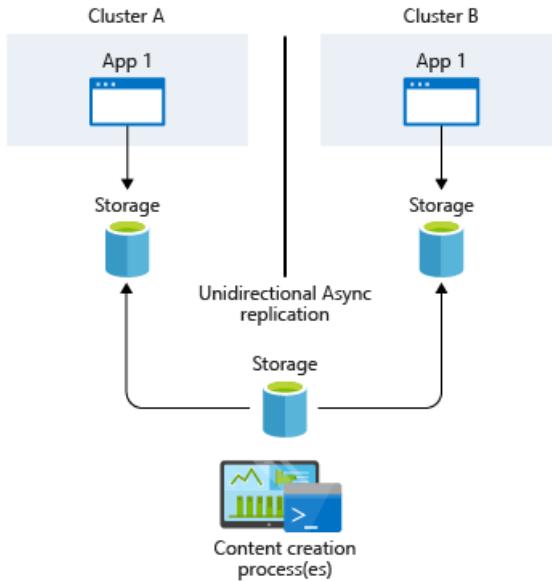
Infrastructure-based asynchronous replication

Your applications might require persistent storage even after a pod is deleted. In Kubernetes, you can use persistent volumes to persist data storage. Persistent volumes are mounted to a node VM and then exposed to the pods. Persistent volumes follow pods even if the pods are moved to a different node inside the same cluster.

The replication strategy you use depends on your storage solution. The following common storage solutions provide their own guidance about disaster recovery and replication:

- [Gluster](#)
- [Ceph](#)
- [Rook](#)
- [Portworx](#)

Typically, you provide a common storage point where applications write their data. This data is then replicated across regions and accessed locally.



If you use Azure Managed Disks, you can use [Velero on Azure](#) and [Kasten](#) to handle replication and disaster recovery. These options are back up solutions native to but unsupported by Kubernetes.

Application-based asynchronous replication

Kubernetes currently provides no native implementation for application-based asynchronous replication. Since containers and Kubernetes are loosely coupled, any traditional application or language approach should work. Typically, the applications themselves replicate the storage requests, which are then written to each cluster's underlying data storage.



Next steps

This article focuses on business continuity and disaster recovery considerations for AKS clusters. For more information about cluster operations in AKS, see these articles about best practices:

- [Multitenancy and cluster isolation](#)
- [Basic Kubernetes scheduler features](#)

Best practices for application developers to manage resources in Azure Kubernetes Service (AKS)

4/8/2021 • 5 minutes to read • [Edit Online](#)

As you develop and run applications in Azure Kubernetes Service (AKS), there are a few key areas to consider. How you manage your application deployments can negatively impact the end-user experience of services that you provide. To succeed, keep in mind some best practices you can follow as you develop and run applications in AKS.

This article focuses on running your cluster and workloads from an application developer perspective. For information about administrative best practices, see [Cluster operator best practices for isolation and resource management in Azure Kubernetes Service \(AKS\)](#). In this article, you learn:

- Pod resource requests and limits.
- Ways to develop and deploy applications with Bridge to Kubernetes and Visual Studio Code.
- How to use the `kube-advisor` tool to check for issues with deployments.

Define pod resource requests and limits

Best practice guidance

Set pod requests and limits on all pods in your YAML manifests. If the AKS cluster uses *resource quotas* and you don't define these values, your deployment may be rejected.

Use pod requests and limits to manage the compute resources within an AKS cluster. Pod requests and limits inform the Kubernetes scheduler which compute resources to assign to a pod.

Pod CPU/Memory requests

Pod requests define a set amount of CPU and memory that the pod needs regularly.

In your pod specifications, it's **best practice and very important** to define these requests and limits based on the above information. If you don't include these values, the Kubernetes scheduler cannot take into account the resources your applications require to aid in scheduling decisions.

Monitor the performance of your application to adjust pod requests.

- If you underestimate pod requests, your application may receive degraded performance due to overscheduling a node.
- If requests are overestimated, your application may have increased difficulty getting scheduled.

Pod CPU/Memory limits**

Pod limits set the maximum amount of CPU and memory that a pod can use.

- *Memory limits* define which pods should be killed when nodes are unstable due to insufficient resources. Without proper limits set, pods will be killed until resource pressure is lifted.
- While a pod may exceed the *CPU limit* periodically, the pod will not be killed for exceeding the CPU limit.

Pod limits define when a pod has lost control of resource consumption. When it exceeds the limit, the pod is marked for killing. This behavior maintains node health and minimizes impact to pods sharing the node. Not setting a pod limit defaults it to the highest available value on a given node.

Avoid setting a pod limit higher than your nodes can support. Each AKS node reserves a set amount of CPU and memory for the core Kubernetes components. Your application may try to consume too many resources on the node for other pods to successfully run.

Monitor the performance of your application at different times during the day or week. Determine peak demand times and align the pod limits to the resources required to meet maximum needs.

IMPORTANT

In your pod specifications, define these requests and limits based on the above information. Failing to include these values prevents the Kubernetes scheduler from accounting for resources your applications require to aid in scheduling decisions.

If the scheduler places a pod on a node with insufficient resources, application performance will be degraded. Cluster administrators **must** set *resource quotas* on a namespace that requires you to set resource requests and limits. For more information, see [resource quotas on AKS clusters](#).

When you define a CPU request or limit, the value is measured in CPU units.

- 1.0 CPU equates to one underlying virtual CPU core on the node.
 - The same measurement is used for GPUs.
- You can define fractions measured in millicores. For example, 100m is 0.1 of an underlying vCPU core.

In the following basic example for a single NGINX pod, the pod requests 100m of CPU time, and 128Mi of memory. The resource limits for the pod are set to 250m CPU and 256Mi memory:

```
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
    - name: mypod
      image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
      resources:
        requests:
          cpu: 100m
          memory: 128Mi
        limits:
          cpu: 250m
          memory: 256Mi
```

For more information about resource measurements and assignments, see [Managing compute resources for containers](#).

Develop and debug applications against an AKS cluster

Best practice guidance

Development teams should deploy and debug against an AKS cluster using Bridge to Kubernetes.

With Bridge to Kubernetes, you can develop, debug, and test applications directly against an AKS cluster. Developers within a team collaborate to build and test throughout the application lifecycle. You can continue to use existing tools such as Visual Studio or Visual Studio Code with the Bridge to Kubernetes extension.

Using integrated development and test process with Bridge to Kubernetes reduces the need for local test environments like [minikube](#). Instead, you develop and test against an AKS cluster, even secured and isolated clusters.

NOTE

Bridge to Kubernetes is intended for use with applications that run on Linux pods and nodes.

Use the Visual Studio Code (VS Code) extension for Kubernetes

Best practice guidance

Install and use the VS Code extension for Kubernetes when you write YAML manifests. You can also use the extension for integrated deployment solution, which may help application owners that infrequently interact with the AKS cluster.

The [Visual Studio Code extension for Kubernetes](#) helps you develop and deploy applications to AKS. The extension provides:

- Intellisense for Kubernetes resources, Helm charts, and templates.
- Browse, deploy, and edit capabilities for Kubernetes resources from within VS Code.
- An intellisense check for resource requests or limits being set in the pod specifications:

```
1 kind: Pod
2 apiVersion: v1
3 metadata:
4   name: mypod
5 spec:
6   containers:
7     - name: mypod
8       image: nginx:1.15.5
9       resources:
10         requests:
11           cpu: 100m
12           No memory limit specified for this container - this could starve other processes
13
14           limits:
15             cpu: 250m
```

Regularly check for application issues with kube-advisor

Best practice guidance

Regularly run the latest version of `kube-advisor` open-source tool to detect issues in your cluster. Run `kube-advisor` before applying resource quotas on an existing AKS cluster to find pods that don't have resource requests and limits defined.

The [kube-advisor](#) tool is an associated AKS open-source project that scans a Kubernetes cluster and reports on identified issues. One useful check is to identify pods without resource requests and limits in place.

While the `kube-advisor` tool can report on resource requests and limits missing in PodSpecs for Windows and Linux applications, `kube-advisor` itself must be scheduled on a Linux pod. Use a `node selector` in the pod's configuration to schedule a pod to run on a node pool with a specific OS.

In an AKS cluster that hosts many development teams and applications, you'll find it easier to track pods using resource requests and limits. As a best practice, regularly run `kube-advisor` on your AKS clusters.

Next steps

This article focused on how to run your cluster and workloads from a cluster operator perspective. For information about administrative best practices, see [Cluster operator best practices for isolation and resource management](#).

management in Azure Kubernetes Service (AKS).

To implement some of these best practices, see the following articles:

- [Develop with Bridge to Kubernetes](#)
- [Check for issues with kube-advisor](#)

Best practices for pod security in Azure Kubernetes Service (AKS)

11/2/2020 • 6 minutes to read • [Edit Online](#)

As you develop and run applications in Azure Kubernetes Service (AKS), the security of your pods is a key consideration. Your applications should be designed for the principle of least number of privileges required. Keeping private data secure is top of mind for customers. You don't want credentials like database connection strings, keys, or secrets and certificates exposed to the outside world where an attacker could take advantage of those secrets for malicious purposes. Don't add them to your code or embed them in your container images. This approach would create a risk for exposure and limit the ability to rotate those credentials as the container images will need to be rebuilt.

This best practices article focuses on how to secure pods in AKS. You learn how to:

- Use pod security context to limit access to processes and services or privilege escalation
- Authenticate with other Azure resources using pod managed identities
- Request and retrieve credentials from a digital vault such as Azure Key Vault

You can also read the best practices for [cluster security](#) and for [container image management](#).

Secure pod access to resources

Best practice guidance - To run as a different user or group and limit access to the underlying node processes and services, define pod security context settings. Assign the least number of privileges required.

For your applications to run correctly, pods should run as a defined user or group and not as *root*. The `securityContext` for a pod or container lets you define settings such as `runAsUser` or `fsGroup` to assume the appropriate permissions. Only assign the required user or group permissions, and don't use the security context as a means to assume additional permissions. The `runAsUser`, privilege escalation, and other Linux capabilities settings are only available on Linux nodes and pods.

When you run as a non-root user, containers cannot bind to the privileged ports under 1024. In this scenario, Kubernetes Services can be used to disguise the fact that an app is running on a particular port.

A pod security context can also define additional capabilities or permissions for accessing processes and services. The following common security context definitions can be set:

- **allowPrivilegeEscalation** defines if the pod can assume *root* privileges. Design your applications so this setting is always set to *false*.
- **Linux capabilities** let the pod access underlying node processes. Take care with assigning these capabilities. Assign the least number of privileges needed. For more information, see [Linux capabilities](#).
- **SELinux labels** is a Linux kernel security module that lets you define access policies for services, processes, and filesystem access. Again, assign the least number of privileges needed. For more information, see [SELinux options in Kubernetes](#)

The following example pod YAML manifest sets security context settings to define:

- Pod runs as user ID *1000* and part of group ID *2000*
- Can't escalate privileges to use `root`
- Allows Linux capabilities to access network interfaces and the host's real-time (hardware) clock

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    fsGroup: 2000
  containers:
    - name: security-context-demo
      image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
      securityContext:
        runAsUser: 1000
        allowPrivilegeEscalation: false
        capabilities:
          add: ["NET_ADMIN", "SYS_TIME"]
```

Work with your cluster operator to determine what security context settings you need. Try to design your applications to minimize additional permissions and access the pod requires. There are additional security features to limit access using AppArmor and seccomp (secure computing) that can be implemented by cluster operators. For more information, see [Secure container access to resources](#).

Limit credential exposure

Best practice guidance - Don't define credentials in your application code. Use managed identities for Azure resources to let your pod request access to other resources. A digital vault, such as Azure Key Vault, should also be used to store and retrieve digital keys and credentials. Pod managed identities is intended for use with Linux pods and container images only.

To limit the risk of credentials being exposed in your application code, avoid the use of fixed or shared credentials. Credentials or keys shouldn't be included directly in your code. If these credentials are exposed, the application needs to be updated and redeployed. A better approach is to give pods their own identity and way to authenticate themselves, or automatically retrieve credentials from a digital vault.

Use Azure Container Compute Upstream projects

IMPORTANT

Associated AKS open source projects are not supported by Azure technical support. They are provided for users to self-install into clusters and gather feedback from our community.

The following [associated AKS open source projects](#) let you automatically authenticate pods or request credentials and keys from a digital vault. These projects are maintained by the Azure Container Compute Upstream team and are part of a [broader list of projects available for use](#).

- [Azure Active Directory Pod Identity](#)
- [Azure Key Vault Provider for Secrets Store CSI Driver](#)

Use pod managed identities

A managed identity for Azure resources lets a pod authenticate itself against Azure services that support it, such as Storage or SQL. The pod is assigned an Azure Identity that lets them authenticate to Azure Active Directory and receive a digital token. This digital token can be presented to other Azure services that check if the pod is authorized to access the service and perform the required actions. This approach means that no secrets are required for database connection strings, for example. The simplified workflow for pod managed identity is shown in the following diagram:

With a managed identity, your application code doesn't need to include credentials to access a service, such as Azure Storage. As each pod authenticates with its own identity, so you can audit and review access. If your application connects with other Azure services, use managed identities to limit credential reuse and risk of exposure.

For more information about pod identities, see [Configure an AKS cluster to use pod managed identities and with your applications](#)

Use Azure Key Vault with Secrets Store CSI Driver

Using the pod identity project enables authentication against supporting Azure services. For your own services or applications without managed identities for Azure resources, you can still authenticate using credentials or keys. A digital vault can be used to store these secret contents.

When applications need a credential, they communicate with the digital vault, retrieve the latest secret contents, and then connect to the required service. Azure Key Vault can be this digital vault. The simplified workflow for retrieving a credential from Azure Key Vault using pod managed identities is shown in the following diagram:

With Key Vault, you store and regularly rotate secrets such as credentials, storage account keys, or certificates. You can integrate Azure Key Vault with an AKS cluster using the [Azure Key Vault provider for the Secrets Store CSI Driver](#). The Secrets Store CSI driver enables the AKS cluster to natively retrieve secret contents from Key Vault and securely provide them only to the requesting pod. Work with your cluster operator to deploy the Secrets Store CSI Driver onto AKS worker nodes. You can use a pod managed identity to request access to Key Vault and retrieve the secret contents needed through the Secrets Store CSI Driver.

Azure Key Vault with Secrets Store CSI Driver can be used for Linux nodes and pods which require a Kubernetes version of 1.16 or greater. For Windows nodes and pods a Kubernetes version of 1.18 or greater is required.

Next steps

This article focused on how to secure your pods. To implement some of these areas, see the following articles:

- [Use managed identities for Azure resources with AKS](#)
- [Integrate Azure Key Vault with AKS](#)

Quotas, virtual machine size restrictions, and region availability in Azure Kubernetes Service (AKS)

4/7/2021 • 2 minutes to read • [Edit Online](#)

All Azure services set default limits and quotas for resources and features, including usage restrictions for certain virtual machine (VM) SKUs.

This article details the default resource limits for Azure Kubernetes Service (AKS) resources and the availability of AKS in Azure regions.

Service quotas and limits

RESOURCE	LIMIT
Maximum clusters per subscription	5000
Maximum nodes per cluster with Virtual Machine Availability Sets and Basic Load Balancer SKU	100
Maximum nodes per cluster with Virtual Machine Scale Sets and Standard Load Balancer SKU	1000 (across all node pools)
Maximum node pools per cluster	100
Maximum pods per node: Basic networking with Kubenet	Maximum: 250 Azure CLI default: 110 Azure Resource Manager template default: 110 Azure portal deployment default: 30
Maximum pods per node: Advanced networking with Azure Container Networking Interface	Maximum: 250 Default: 30
Open Service Mesh (OSM) AKS addon preview	Kubernetes Cluster Version: 1.19+ ¹ OSM controllers per cluster: 1 ¹ Pods per OSM controller: 500 ¹ Kubernetes service accounts managed by OSM: 50 ¹

¹The OSM add-on for AKS is in a preview state and will undergo additional enhancements before general availability (GA). During the preview phase, it's recommended to not surpass the limits shown.

Provisioned infrastructure

All other network, compute, and storage limitations apply to the provisioned infrastructure. For the relevant limits, see [Azure subscription and service limits](#).

IMPORTANT

When you upgrade an AKS cluster, extra resources are temporarily consumed. These resources include available IP addresses in a virtual network subnet or virtual machine vCPU quota.

For Windows Server containers, you can perform an upgrade operation to apply the latest node updates. If you don't have the available IP address space or vCPU quota to handle these temporary resources, the cluster upgrade process will fail. For more information on the Windows Server node upgrade process, see [Upgrade a node pool in AKS](#).

Restricted VM sizes

Each node in an AKS cluster contains a fixed amount of compute resources such as vCPU and memory. If an AKS node contains insufficient compute resources, pods might fail to run correctly. To ensure the required *kube-system* pods and your applications can be reliably scheduled, **don't use the following VM SKUs in AKS:**

- Standard_A0
- Standard_A1
- Standard_A1_v2
- Standard_B1s
- Standard_B1ms
- Standard_F1
- Standard_F1s

For more information on VM types and their compute resources, see [Sizes for virtual machines in Azure](#).

Region availability

For the latest list of where you can deploy and run clusters, see [AKS region availability](#).

Next steps

You can increase certain default limits and quotas. If your resource supports an increase, request the increase through an [Azure support request](#) (for **Issue type**, select **Quota**).

Migrate to Azure Kubernetes Service (AKS)

5/14/2021 • 7 minutes to read • [Edit Online](#)

To help you plan and execute a successful migration to Azure Kubernetes Service (AKS), this guide provides details for the current recommended AKS configuration. While this article doesn't cover every scenario, it contains links to more detailed information for planning a successful migration.

This document helps support the following scenarios:

- Containerizing certain applications and migrating them to AKS using [Azure Migrate](#).
- Migrating an AKS Cluster backed by [Availability Sets](#) to [Virtual Machine Scale Sets](#).
- Migrating an AKS cluster to use a [Standard SKU load balancer](#).
- Migrating from [Azure Container Service \(ACS\) - retiring January 31, 2020](#) to AKS.
- Migrating from [AKS engine](#) to AKS.
- Migrating from non-Azure based Kubernetes clusters to AKS.
- Moving existing resources to a different region.

When migrating, ensure your target Kubernetes version is within the supported window for AKS. Older versions may not be within the supported range and will require a version upgrade to be supported by AKS. For more information, see [AKS supported Kubernetes versions](#).

If you're migrating to a newer version of Kubernetes, review [Kubernetes version and version skew support policy](#).

Several open-source tools can help with your migration, depending on your scenario:

- [Velero](#) (Requires Kubernetes 1.7+)
- [Azure Kube CLI extension](#)
- [ReShifter](#)

In this article we will summarize migration details for:

- Containerizing applications through Azure Migrate
- AKS with Standard Load Balancer and Virtual Machine Scale Sets
- Existing attached Azure Services
- Ensure valid quotas
- High Availability and business continuity
- Considerations for stateless applications
- Considerations for stateful applications
- Deployment of your cluster configuration

Use Azure Migrate to migrate your applications to AKS

Azure Migrate offers a unified platform to assess and migrate to Azure on-premises servers, infrastructure, applications, and data. For AKS, you can use Azure Migrate for the following tasks:

- [Containerize ASP.NET applications and migrate to AKS](#)
- [Containerize Java web applications and migrate to AKS](#)

AKS with Standard Load Balancer and Virtual Machine Scale Sets

AKS is a managed service offering unique capabilities with lower management overhead. Since AKS is a managed service, you must select from a set of [regions](#) which AKS supports. You may need to modify your existing applications to keep them healthy on the AKS-managed control plane during the transition from your existing cluster to AKS.

We recommend using AKS clusters backed by [Virtual Machine Scale Sets](#) and the [Azure Standard Load Balancer](#) to ensure you get features such as:

- [Multiple node pools](#),
- [Availability Zones](#),
- [Authorized IP ranges](#),
- [Cluster Autoscaler](#),
- [Azure Policy for AKS](#), and
- Other new features as they are released.

AKS clusters backed by [Virtual Machine Availability Sets](#) lack support for many of these features.

The following example creates an AKS cluster with single node pool backed by a virtual machine (VM) scale set. The cluster:

- Uses a standard load balancer.
- Enables the cluster autoscaler on the node pool for the cluster.
- Sets a minimum of 1 and maximum of 3 nodes.

```
# First create a resource group
az group create --name myResourceGroup --location eastus

# Now create the AKS cluster and enable the cluster autoscaler
az aks create \
    --resource-group myResourceGroup \
    --name myAKSCluster \
    --node-count 1 \
    --vm-set-type VirtualMachineScaleSets \
    --load-balancer-sku standard \
    --enable-cluster-autoscaler \
    --min-count 1 \
    --max-count 3
```

Existing attached Azure Services

When migrating clusters, you may have attached external Azure services. While the following services don't require resource recreation, they will require updating connections from previous to new clusters to maintain functionality.

- Azure Container Registry
- Log Analytics
- Application Insights
- Traffic Manager
- Storage Account
- External Databases

Ensure valid quotas

Since other VMs will be deployed into your subscription during migration, you should verify that your quotas and limits are sufficient for these resources. If necessary, request an increase in [vCPU quota](#).

You may need to request an increase for [Network quotas](#) to ensure you don't exhaust IPs. For more information, see [networking and IP ranges for AKS](#).

For more information, see [Azure subscription and service limits](#). To check your current quotas, in the Azure portal, go to the [subscriptions blade](#), select your subscription, and then select [Usage + quotas](#).

High Availability and Business Continuity

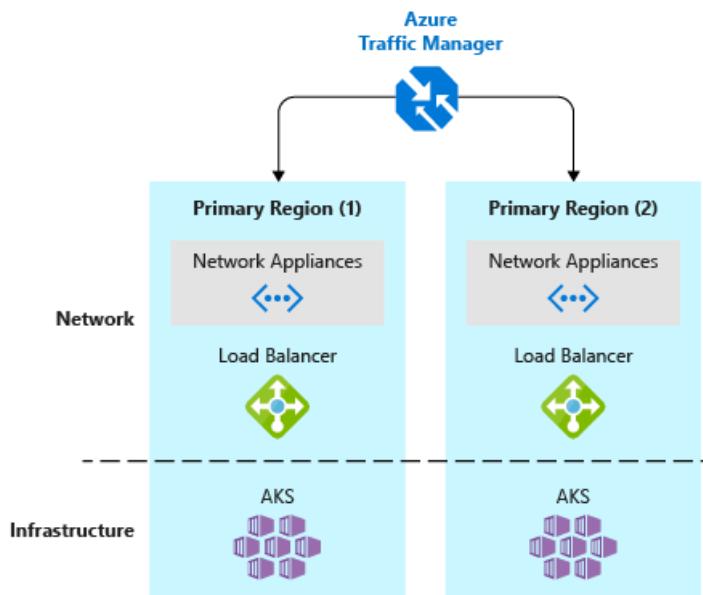
If your application can't handle downtime, you will need to follow best practices for high availability migration scenarios. Read more about [Best practices for complex business continuity planning, disaster recovery, and maximizing uptime in Azure Kubernetes Service \(AKS\)](#).

For complex applications, you'll typically migrate over time rather than all at once, meaning the old and new environments might need to communicate over the network. Applications previously using `ClusterIP` services to communicate might need to be exposed as type `LoadBalancer` and be secured appropriately.

To complete the migration, you'll want to point clients to the new services that are running on AKS. We recommend that you redirect traffic by updating DNS to point to the Load Balancer sitting in front of your AKS cluster.

[Azure Traffic Manager](#) can direct customers to the desired Kubernetes cluster and application instance. Traffic Manager is a DNS-based traffic load balancer that can distribute network traffic across regions. For the best performance and redundancy, direct all application traffic through Traffic Manager before it goes to your AKS cluster.

In a multi-cluster deployment, customers should connect to a Traffic Manager DNS name that points to the services on each AKS cluster. Define these services by using Traffic Manager endpoints. Each endpoint is the *service load balancer IP*. Use this configuration to direct network traffic from the Traffic Manager endpoint in one region to the endpoint in a different region.



[Azure Front Door Service](#) is another option for routing traffic for AKS clusters. With Azure Front Door Service, you can define, manage, and monitor the global routing for your web traffic by optimizing for best performance and instant global failover for high availability.

Considerations for stateless applications

Stateless application migration is the most straightforward case:

1. Apply your resource definitions (YAML or Helm) to the new cluster.
2. Ensure everything works as expected.
3. Redirect traffic to activate your new cluster.

Considerations for stateful applications

Carefully plan your migration of stateful applications to avoid data loss or unexpected downtime.

- If you use Azure Files, you can mount the file share as a volume into the new cluster. See [Mount Static Azure Files as a Volume](#).
- If you use Azure Managed Disks, you can only mount the disk if unattached to any VM. See [Mount Static Azure Disk as a Volume](#).
- If neither of those approaches work, you can use a backup and restore options. See [Velero on Azure](#).

Azure Files

Unlike disks, Azure Files can be mounted to multiple hosts concurrently. In your AKS cluster, Azure and Kubernetes don't prevent you from creating a pod that your AKS cluster still uses. To prevent data loss and unexpected behavior, ensure that the clusters don't write to the same files simultaneously.

If your application can host multiple replicas that point to the same file share, follow the stateless migration steps and deploy your YAML definitions to your new cluster.

If not, one possible migration approach involves the following steps:

1. Validate your application is working correctly.
2. Point your live traffic to your new AKS cluster.
3. Disconnect the old cluster.

If you want to start with an empty share and make a copy of the source data, you can use the [`az storage file copy`](#) commands to migrate your data.

Migrating persistent volumes

If you're migrating existing persistent volumes to AKS, you'll generally follow these steps:

1. Quiesce writes to the application.
 - This step is optional and requires downtime.
2. Take snapshots of the disks.
3. Create new managed disks from the snapshots.
4. Create persistent volumes in AKS.
5. Update pod specifications to [use existing volumes](#) rather than PersistentVolumeClaims (static provisioning).
6. Deploy your application to AKS.
7. Validate your application is working correctly.
8. Point your live traffic to your new AKS cluster.

IMPORTANT

If you choose not to quiesce writes, you'll need to replicate data to the new deployment. Otherwise you'll miss the data that was written after you took the disk snapshots.

Some open-source tools can help you create managed disks and migrate volumes between Kubernetes clusters:

- [Azure CLI Disk Copy extension](#) copies and converts disks across resource groups and Azure regions.
- [Azure Kube CLI extension](#) enumerates ACS Kubernetes volumes and migrates them to an AKS cluster.

Deployment of your cluster configuration

We recommend that you use your existing Continuous Integration (CI) and Continuous Deliver (CD) pipeline to deploy a known-good configuration to AKS. You can use Azure Pipelines to [build and deploy your applications to AKS](#). Clone your existing deployment tasks and ensure that `kubeconfig` points to the new AKS cluster.

If that's not possible, export resource definitions from your existing Kubernetes cluster and then apply them to AKS. You can use `kubectl` to export objects.

```
kubectl get deployment -o=yaml --export > deployments.yaml
```

Moving existing resources to another region

You may want to move your AKS cluster to a [different region supported by AKS](#). We recommend that you create a new cluster in the other region, then deploy your resources and applications to your new cluster.

In addition, if you have any services running on your AKS cluster, you will need to install and configure those services on your cluster in the new region.

In this article, we summarized migration details for:

- AKS with Standard Load Balancer and Virtual Machine Scale Sets
- Existing attached Azure Services
- Ensure valid quotas
- High Availability and business continuity
- Considerations for stateless applications
- Considerations for stateful applications
- Deployment of your cluster configuration

Supported Kubernetes versions in Azure Kubernetes Service (AKS)

4/28/2021 • 8 minutes to read • [Edit Online](#)

The Kubernetes community releases minor versions roughly every three months. Recently, the Kubernetes community has [increased the support window for each version from 9 months to 12 months](#), starting with version 1.19.

Minor version releases include new features and improvements. Patch releases are more frequent (sometimes weekly) and are intended for critical bug fixes within a minor version. Patch releases include fixes for security vulnerabilities or major bugs.

Kubernetes versions

Kubernetes uses the standard [Semantic Versioning](#) versioning scheme for each version:

[major].[minor].[patch]

Example:

1.17.7
1.17.8

Each number in the version indicates general compatibility with the previous version:

- **Major versions** change when incompatible API updates or backwards compatibility may be broken.
- **Minor versions** change when functionality updates are made that are backwards compatible to the other minor releases.
- **Patch versions** change when backwards-compatible bug fixes are made.

Aim to run the latest patch release of the minor version you're running. For example, your production cluster is on `1.17.7`. `1.17.8` is the latest available patch version available for the `1.17` series. You should upgrade to `1.17.8` as soon as possible to ensure your cluster is fully patched and supported.

Kubernetes version support policy

AKS defines a generally available version as a version enabled in all SLO or SLA measurements and available in all regions. AKS supports three GA minor versions of Kubernetes:

- The latest GA minor version that is released in AKS (which we'll refer to as N).
- Two previous minor versions.
 - Each supported minor version also supports a maximum of two (2) stable patches.

AKS may also support preview versions, which are explicitly labeled and subject to [Preview terms and conditions](#).

NOTE

AKS uses safe deployment practices which involve gradual region deployment. This means it may take up to 10 business days for a new release or a new version to be available in all regions.

The supported window of Kubernetes versions on AKS is known as "N-2": (N (Latest release) - 2 (minor versions)).

For example, if AKS introduces `1.17.a` today, support is provided for the following versions:

NEW MINOR VERSION	SUPPORTED VERSION LIST
<code>1.17.a</code>	<code>1.17.a, 1.17.b, 1.16.c, 1.16.d, 1.15.e, 1.15.f</code>

Where ".letter" is representative of patch versions.

When a new minor version is introduced, the oldest minor version and patch releases supported are deprecated and removed. For example, the current supported version list is:

```
1.17.a  
1.17.b  
1.16.c  
1.16.d  
1.15.e  
1.15.f
```

AKS releases `1.18.*`, removing all the `1.15.*` versions out of support in 30 days.

NOTE

If customers are running an unsupported Kubernetes version, they will be asked to upgrade when requesting support for the cluster. Clusters running unsupported Kubernetes releases are not covered by the [AKS support policies](#).

In addition to the above, AKS supports a maximum of two **patch** releases of a given minor version. So given the following supported versions:

```
Current Supported Version List  
-----  
1.17.8, 1.17.7, 1.16.10, 1.16.9
```

If AKS releases `1.17.9` and `1.16.11`, the oldest patch versions are deprecated and removed, and the supported version list becomes:

```
New Supported Version List  
-----  
1.17.*9*, 1.17.*8*, 1.16.*11*, 1.16.*10*
```

Supported `kubectl` versions

You can use one minor version older or newer of `kubectl` relative to your `kube-apiserver` version, consistent with the [Kubernetes support policy for kubectl](#).

For example, if your `kube-apiserver` is at `1.17`, then you can use versions `1.16` to `1.18` of `kubectl` with that `kube-apiserver`.

To install or update your version of `kubectl`, run `az aks install-cli`.

Release and deprecation process

You can reference upcoming version releases and deprecations on the [AKS Kubernetes Release Calendar](#).

For new **minor** versions of Kubernetes:

- AKS publishes a pre-announcement with the planned date of a new version release and respective old version deprecation on the [AKS Release notes](#) at least 30 days prior to removal.
- AKS uses [Azure Advisor](#) to alert users if a new version will cause issues in their cluster because of deprecated APIs. Azure Advisor is also used to alert the user if they are currently out of support.
- AKS publishes a [service health notification](#) available to all users with AKS and portal access, and sends an email to the subscription administrators with the planned version removal dates.

NOTE

To find out who is your subscription administrators or to change it, please refer to [manage Azure subscriptions](#).

- Users have **30 days** from version removal to upgrade to a supported minor version release to continue receiving support.

For new **patch** versions of Kubernetes:

- Because of the urgent nature of patch versions, they can be introduced into the service as they become available.
- In general, AKS does not broadly communicate the release of new patch versions. However, AKS constantly monitors and validates available CVE patches to support them in AKS in a timely manner. If a critical patch is found or user action is required, AKS will notify users to upgrade to the newly available patch.
- Users have **30 days** from a patch release's removal from AKS to upgrade into a supported patch and continue receiving support.

Supported versions policy exceptions

AKS reserves the right to add or remove new/existing versions with one or more critical production-impacting bugs or security issues without advance notice.

Specific patch releases may be skipped or rollout accelerated, depending on the severity of the bug or security issue.

Azure portal and CLI versions

When you deploy an AKS cluster in the portal or with the Azure CLI, the cluster defaults to the N-1 minor version and latest patch. For example, if AKS supports **1.17.a**, **1.17.b**, **1.16.c**, **1.16.d**, **1.15.e**, and **1.15.f**, the default version selected is **1.16.c**.

To find out what versions are currently available for your subscription and region, use the [az aks get-versions](#) command. The following example lists the available Kubernetes versions for the *EastUS* region:

```
az aks get-versions --location eastus --output table
```

AKS Kubernetes Release Calendar

For the past release history, see [Kubernetes](#).

K8S VERSION	UPSTREAM RELEASE	AKS PREVIEW	AKS GA	END OF LIFE
1.18	Mar-23-20	May 2020	Aug 2020	1.21 GA

K8S VERSION	UPSTREAM RELEASE	AKS PREVIEW	AKS GA	END OF LIFE
1.19	Aug-04-20	Sep 2020	Nov 2020	1.22 GA
1.20	Dec-08-20	Jan 2021	Mar 2021	1.23 GA
1.21	Apr-08-21	May 2021	Jun 2021	1.24 GA

FAQ

How does Microsoft notify me of new Kubernetes versions?

The AKS team publishes pre-announcements with planned dates of the new Kubernetes versions in our documentation, our [GitHub](#) as well as emails to subscription administrators who own clusters that are going to fall out of support. In addition to announcements, AKS also uses [Azure Advisor](#) to notify the customer inside the Azure Portal to alert users if they are out of support, as well as alerting them of deprecated APIs that will affect their application or development process.

How often should I expect to upgrade Kubernetes versions to stay in support?

Starting with Kubernetes 1.19, the [open source community has expanded support to 1 year](#). AKS commits to enabling patches and support matching the upstream commitments. For AKS clusters on 1.19 and greater, you will be able to upgrade at a minimum of once a year to stay on a supported version.

For versions on 1.18 or below, the window of support remains at 9 months, requiring an upgrade once every 9 months to stay on a supported version. Regularly test new versions and be prepared to upgrade to newer versions to capture the latest stable enhancements within Kubernetes.

What happens when a user upgrades a Kubernetes cluster with a minor version that isn't supported?

If you're on the *n-3* version or older, it means you're outside of support and will be asked to upgrade. When your upgrade from version n-3 to n-2 succeeds, you're back within our support policies. For example:

- If the oldest supported AKS version is *1.15.a* and you are on *1.14.b* or older, you're outside of support.
- When you successfully upgrade from *1.14.b* to *1.15.a* or higher, you're back within our support policies.

Downgrades are not supported.

What does 'Outside of Support' mean

'Outside of Support' means that:

- The version you're running is outside of the supported versions list.
- You'll be asked to upgrade the cluster to a supported version when requesting support, unless you're within the 30-day grace period after version deprecation.

Additionally, AKS doesn't make any runtime or other guarantees for clusters outside of the supported versions list.

What happens when a user scales a Kubernetes cluster with a minor version that isn't supported?

For minor versions not supported by AKS, scaling in or out should continue to work. Since there are no Quality of Service guarantees, we recommend upgrading to bring your cluster back into support.

Can a user stay on a Kubernetes version forever?

If a cluster has been out of support for more than three (3) minor versions and has been found to carry security

risks, Azure proactively contacts you to upgrade your cluster. If you do not take further action, Azure reserves the right to automatically upgrade your cluster on your behalf.

What version does the control plane support if the node pool is not in one of the supported AKS versions?

The control plane must be within a window of versions from all node pools. For details on upgrading the control plane or node pools, visit documentation on [upgrading node pools](#).

Can I skip multiple AKS versions during cluster upgrade?

When you upgrade a supported AKS cluster, Kubernetes minor versions cannot be skipped. For example, upgrades between:

- *1.12.x-> 1.13.x*: allowed.
- *1.13.x-> 1.14.x*: allowed.
- *1.12.x-> 1.14.x*: not allowed.

To upgrade from *1.12.x-> 1.14.x*:

1. Upgrade from *1.12.x-> 1.13.x*.
2. Upgrade from *1.13.x-> 1.14.x*.

Skipping multiple versions can only be done when upgrading from an unsupported version back into a supported version. For example, you can upgrade from an unsupported *1.10.x* to a supported *1.15.x*.

Can I create a new 1.xx.x cluster during its 30 day support window?

No. Once a version is deprecated/removed, you cannot create a cluster with that version. As the change rolls out, you will start to see the old version removed from your version list. This process may take up to two weeks from announcement, progressively by region.

I am on a freshly deprecated version, can I still add new node pools? Or will I have to upgrade?

No. You will not be allowed to add node pools of the deprecated version to your cluster. You can add node pools of a new version. However, this may require you to update the control plane first.

Next steps

For information on how to upgrade your cluster, see [Upgrade an Azure Kubernetes Service \(AKS\) cluster](#).

Security hardening for AKS agent node host OS

4/7/2021 • 2 minutes to read • [Edit Online](#)

As a secure service, Azure Kubernetes Service (AKS) complies with SOC, ISO, PCI DSS, and HIPAA standards. This article covers the security hardening applied to AKS virtual machine (VM) hosts. For more information about AKS security, see [Security concepts for applications and clusters in Azure Kubernetes Service \(AKS\)](#).

NOTE

This document is scoped to Linux agents in AKS only.

AKS clusters are deployed on host VMs, which run a security-optimized OS used for containers running on AKS. This host OS is based on an [Ubuntu 16.04.LTS](#) image with more [security hardening](#) and optimizations applied.

The goal of the security hardened host OS is to reduce the surface area of attack and optimize for the deployment of containers in a secure manner.

IMPORTANT

The security hardened OS is **not** CIS benchmarked. While it overlaps with CIS benchmarks, the goal is not to be CIS-compliant. The goal for host OS hardening is to converge on a level of security consistent with Microsoft's own internal host security standards.

Security hardening features

- AKS provides a security-optimized host OS by default, but no option to select an alternate operating system.
- Azure applies daily patches (including security patches) to AKS virtual machine hosts.
 - Some of these patches require a reboot, while others will not.
 - You're responsible for scheduling AKS VM host reboots as needed.
 - For guidance on how to automate AKS patching, see [patching AKS nodes](#).

What is configured

CIS	AUDIT DESCRIPTION
1.1.1.1	Ensure mounting of cramfs filesystems is disabled
1.1.1.2	Ensure mounting of freevxfs filesystems is disabled
1.1.1.3	Ensure mounting of jffs2 filesystems is disabled
1.1.1.4	Ensure mounting of HFS filesystems is disabled
1.1.1.5	Ensure mounting of HFS Plus filesystems is disabled
1.4.3	Ensure authentication required for single user mode

CIS	AUDIT DESCRIPTION
1.7.1.2	Ensure local login warning banner is configured properly
1.7.1.3	Ensure remote login warning banner is configured properly
1.7.1.5	Ensure permissions on /etc/issue are configured
1.7.1.6	Ensure permissions on /etc/issue.net are configured
2.1.5	Ensure that --streaming-connection-idle-timeout is not set to 0
3.1.2	Ensure packet redirect sending is disabled
3.2.1	Ensure source routed packages are not accepted
3.2.2	Ensure ICMP redirects are not accepted
3.2.3	Ensure secure ICMP redirects are not accepted
3.2.4	Ensure suspicious packets are logged
3.3.1	Ensure IPv6 router advertisements are not accepted
3.5.1	Ensure DCCP is disabled
3.5.2	Ensure SCTP is disabled
3.5.3	Ensure RDS is disabled
3.5.4	Ensure TIPC is disabled
4.2.1.2	Ensure logging is configured
5.1.2	Ensure permissions on /etc/crontab are configured
5.2.4	Ensure SSH X11 forwarding is disabled
5.2.5	Ensure SSH MaxAuthTries is set to 4 or less
5.2.8	Ensure SSH root login is disabled
5.2.10	Ensure SSH PermitUserEnvironment is disabled
5.2.11	Ensure only approved MAX algorithms are used
5.2.12	Ensure SSH Idle Timeout Interval is configured
5.2.13	Ensure SSH LoginGraceTime is set to one minute or less
5.2.15	Ensure SSH warning banner is configured

CIS	AUDIT DESCRIPTION
5.3.1	Ensure password creation requirements are configured
5.4.1.1	Ensure password expiration is 90 days or less
5.4.1.4	Ensure inactive password lock is 30 days or less
5.4.4	Ensure default user umask is 027 or more restrictive
5.6	Ensure access to the su command is restricted

Additional notes

- To further reduce the attack surface area, some unnecessary kernel module drivers have been disabled in the OS.
- The security hardened OS is built and maintained specifically for AKS and is **not** supported outside of the AKS platform.

Next steps

For more information about AKS security, see the following articles:

- [Azure Kubernetes Service \(AKS\)](#)
- [AKS security considerations](#)
- [AKS best practices](#)

Azure Kubernetes Service Diagnostics (preview) overview

4/7/2021 • 2 minutes to read • [Edit Online](#)

Troubleshooting Azure Kubernetes Service (AKS) cluster issues plays an important role in maintaining your cluster, especially if your cluster is running mission-critical workloads. AKS Diagnostics is an intelligent, self-diagnostic experience that:

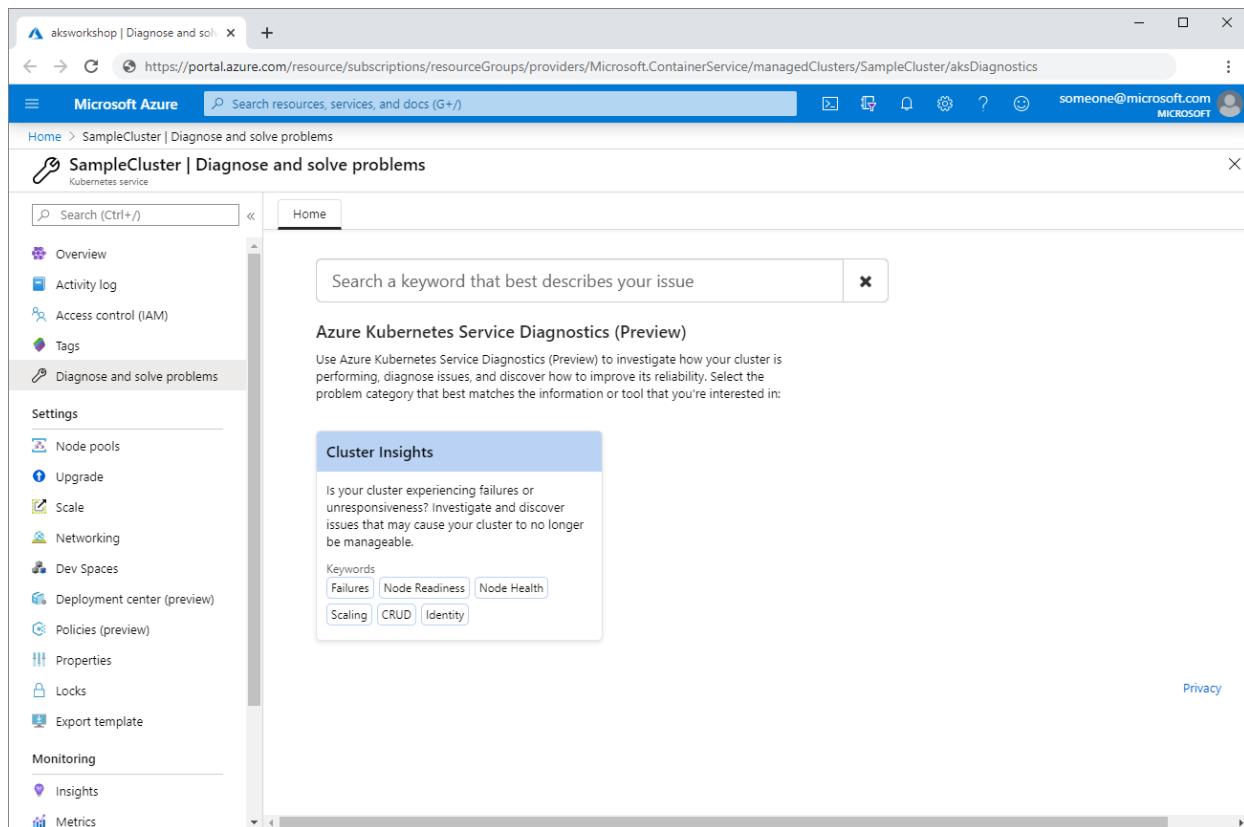
- Helps you identify and resolve problems in your cluster.
- Is cloud-native.
- Requires no extra configuration or billing cost.

This feature is now in public preview.

Open AKS Diagnostics

To access AKS Diagnostics:

1. Navigate to your Kubernetes cluster in the [Azure portal](#).
2. Click on **Diagnose and solve problems** in the left navigation, which opens AKS Diagnostics.
3. Choose a category that best describes the issue of your cluster, like *Cluster Node Issues*, by:
 - Using the keywords in the homepage tile.
 - Typing a keyword that best describes your issue in the search bar.



View a diagnostic report

After you click on a category, you can view a diagnostic report specific to your cluster. Diagnostic reports

intelligently call out any issues in your cluster with status icons. You can drill down on each topic by clicking **More Info** to see a detailed description of:

- Issues
- Recommended actions
- Links to helpful docs
- Related-metrics
- Logging data

Diagnostic reports generate based on the current state of your cluster after running various checks. They can be useful for pinpointing the problem of your cluster and understanding next steps to resolve the issue.

The screenshot shows the 'Cluster Insights' diagnostic report. At the top, there are time range buttons (1h, 6h, 1d), a date range (2019-10-21 23:23 to 2019-10-22 23:07), a UTC switch, and a refresh icon. Below this is a section titled 'Cluster Insights' with a subtitle 'Identifies scenarios that may cause a cluster to no longer be manageable.' It includes 'Send Feedback' and 'Copy Report' buttons. The main content is divided into two sections: 'Observations' and 'Successful Checks'. The 'Observations' section has a magnifying glass icon and lists 'Cluster Node Issues' with a red exclamation mark icon. The 'Description' column states 'Node Issues Detected' and 'One or more Node based issues have been detected.', with a 'More Info' link. The 'Successful Checks' section has a green checkmark icon and lists 'Identity and Security Management' and 'Create, Read, Update & Delete Operations' both with green checkmark icons. Each has a 'Description' column and a 'More Info' link. A red box highlights the 'More Info' link for the 'Cluster Node Issues' row.

This screenshot shows the detailed view for 'Cluster Node Issues'. It starts with a header 'Cluster Node Issues' and a 'Back to Observations' link. Below is a section for 'Node Issues Detected' with a red exclamation mark icon. The 'Description' column states 'One or more Node based issues have been detected.' with a like/dislike icon. The next section is 'Node Insufficient Resources Detected' with a red exclamation mark icon. The 'Description' column states 'AKS monitoring has detected at least one node has insufficient resources. You can run the following to better understand your resource consumption: kubectl get no -o wide && kubectl top no'. The 'Recommended Action' column lists: 'It is recommended you attempt to either:' followed by '1) Horizontally Scale your Cluster [Click Here](#)', '2) Resize your VMs (temporary fix: [Click Here](#))', and '3) Migrate your workload to an AKS cluster with appropriate VM Sizes'. If the error still persists, it suggests engaging Microsoft Support. The 'Logging Data' section shows a table with two rows: 'NodeName' (aks-agentpool-00000000-0) and 'ConstrainedResource' (memoryPressure).

Cluster Insights

The following diagnostic checks are available in **Cluster Insights**.

Cluster Node Issues

Cluster Node Issues checks for node-related issues that cause your cluster to behave unexpectedly.

- Node readiness issues
- Node failures
- Insufficient resources
- Node missing IP configuration
- Node CNI failures
- Node not found
- Node power off
- Node authentication failure
- Node kube-proxy stale

Create, read, update & delete (CRUD) operations

CRUD Operations checks for any CRUD operations that cause issues in your cluster.

- In-use subnet delete operation error
- Network security group delete operation error
- In-use route table delete operation error
- Referenced resource provisioning error
- Public IP address delete operation error
- Deployment failure due to deployment quota
- Operation error due to organization policy
- Missing subscription registration
- VM extension provisioning error
- Subnet capacity
- Quota exceeded error

Identity and security management

Identity and Security Management detects authentication and authorization errors that prevent communication to your cluster.

- Node authorization failures
- 401 errors
- 403 errors

Next steps

- Collect logs to help you further troubleshoot your cluster issues by using [AKS Periscope](#).
- Read the [triage practices section](#) of the AKS day-2 operations guide.
- Post your questions or feedback at [UserVoice](#) by adding "[Diag]" in the title.

Sustainable software engineering principles in Azure Kubernetes Service (AKS)

4/7/2021 • 4 minutes to read • [Edit Online](#)

The sustainable software engineering principles are a set of competencies to help you define, build, and run sustainable applications. The overall goal is to reduce your carbon footprint of every aspect of your application. [The Principles of Sustainable Software Engineering](#) has an overview of the principles of sustainable software engineering.

Sustainable software engineering is a shift in priorities and focus. In many cases, the way most software is designed and run highlights fast performance and low latency. Meanwhile, sustainable software engineering focuses on reducing as much carbon emission as possible. Consider:

- Applying sustainable software engineering principles can give you faster performance or lower latency, such as by lowering total network travel.
- Reducing carbon emissions may cause slower performance or increased latency, such as delaying low-priority workloads.

Before applying sustainable software engineering principles to your application, review the priorities, needs, and trade-offs of your application.

Measure and optimize

To lower the carbon footprint of your AKS clusters, you need understand how your cluster's resources are being used. [Azure Monitor](#) provides details on your cluster's resource usage, such as memory and CPU usage. This data informs your decision to reduce the carbon footprint of your cluster and observes the effect of your changes.

You can also install the [Microsoft Sustainability Calculator](#) to see the carbon footprint of all your Azure resources.

Increase resource utilization

One approach to lowering your carbon footprint is to reduce your idle time. Reducing your idle time involves increasing the utilization of your compute resources. For example:

1. You had four nodes in your cluster, each running at 50% capacity. So, all four of your nodes have 50% unused capacity remaining idle.
2. You reduced your cluster to three nodes, each running at 67% capacity with the same workload. You would have successfully decreased your unused capacity to 33% on each node and increased your utilization.

IMPORTANT

When considering changing the resources in your cluster, verify your [system pools](#) have enough resources to maintain the stability of your cluster's core system components. **Never** reduce your cluster's resources to the point where your cluster may become unstable.

After reviewing your cluster's utilization, consider using the features offered by [multiple node pools](#):

- Node sizing

Use [node sizing](#) to define node pools with specific CPU and memory profiles, allowing you to tailor your nodes to your workload needs. By sizing your nodes to your workload needs, you can run a few nodes at higher utilization.

- Cluster scaling

Configure how your cluster [scales](#). Use the [horizontal pod autoscaler](#) and the [cluster autoscaler](#) to scale your cluster automatically based on your configuration. Control how your cluster scales to keep all your nodes running at a high utilization while staying in sync with changes to your cluster's workload.

- Spot pools

For cases where a workload is tolerant to sudden interruptions or terminations, you can use [spot pools](#). Spot pools take advantage of idle capacity within Azure. For example, spot pools may work well for batch jobs or development environments.

NOTE

Increasing utilization can also reduce excess nodes, which reduces the energy consumed by [resource reservations on each node](#).

Finally, review the CPU and memory *requests* and *limits* in the Kubernetes manifests of your applications.

- As you lower memory and CPU values, more memory and CPU are available to the cluster to run other workloads.
- As you run more workloads with lower CPU and memory, your cluster becomes more densely allocated, which increases your utilization.

When reducing the CPU and memory for your applications, your applications' behavior may become degraded or unstable if you set CPU and memory values too low. Before changing the CPU and memory *requests* and *limits*, run some benchmarking tests to verify if the values are set appropriately. Never reduce these values to the point of application instability.

Reduce network travel

By reducing requests and responses travel distance to and from your cluster, you can reduce carbon emissions and electricity consumption by networking devices. After reviewing your network traffic, consider creating clusters [in regions](#) closer to the source of your network traffic. You can use [Azure Traffic Manager](#) to route traffic to the closest cluster and [proximity placement groups](#) and reduce the distance between Azure resources.

IMPORTANT

When considering making changes to your cluster's networking, never reduce network travel at the cost of meeting workload requirements. For example, while using [availability zones](#) causes more network travel on your cluster, availability zones may be necessary to handle workload requirements.

Demand shaping

Where possible, consider shifting demand for your cluster's resources to times or regions where you can use excess capacity. For example, consider:

- Changing the time or region for a batch job to run.
- Using [spot pools](#).
- Refactoring your application to use a queue to defer running workloads that don't need immediate

processing.

Next steps

Learn more about the features of AKS mentioned in this article:

- [Multiple node pools](#)
- [Node sizing](#)
- [Scaling a cluster](#)
- [Horizontal pod autoscaler](#)
- [Cluster autoscaler](#)
- [Spot pools](#)
- [System pools](#)
- [Resource reservations](#)
- [Proximity placement groups](#)
- [Availability Zones](#)

Scale the node count in an Azure Kubernetes Service (AKS) cluster

4/21/2021 • 2 minutes to read • [Edit Online](#)

If the resource needs of your applications change, you can manually scale an AKS cluster to run a different number of nodes. When you scale down, nodes are carefully [cordoned and drained](#) to minimize disruption to running applications. When you scale up, AKS waits until nodes are marked `Ready` by the Kubernetes cluster before pods are scheduled on them.

Scale the cluster nodes

First, get the *name* of your node pool using the [az aks show](#) command. The following example gets the node pool name for the cluster named *myAKSCluster* in the *myResourceGroup* resource group:

```
az aks show --resource-group myResourceGroup --name myAKSCluster --query agentPoolProfiles
```

The following example output shows that the *name* is *nodepool1*:

```
[  
 {  
   "count": 1,  
   "maxPods": 110,  
   "name": "nodepool1",  
   "osDiskSizeGb": 30,  
   "osType": "Linux",  
   "storageProfile": "ManagedDisks",  
   "vmSize": "Standard_DS2_v2"  
 }  
]
```

Use the [az aks scale](#) command to scale the cluster nodes. The following example scales a cluster named *myAKSCluster* to a single node. Provide your own `--nodepool-name` from the previous command, such as *nodepool1*:

```
az aks scale --resource-group myResourceGroup --name myAKSCluster --node-count 1 --nodepool-name <your node  
pool name>
```

The following example output shows the cluster has successfully scaled to one node, as shown in the *agentPoolProfiles* section:

```
{  
  "aadProfile": null,  
  "addonProfiles": null,  
  "agentPoolProfiles": [  
    {  
      "count": 1,  
      "maxPods": 110,  
      "name": "nodepool1",  
      "osDiskSizeGb": 30,  
      "osType": "Linux",  
      "storageProfile": "ManagedDisks",  
      "vmSize": "Standard_DS2_v2",  
      "vnetSubnetId": null  
    }  
  ],  
  [...]  
}
```

Scale `User` node pools to 0

Unlike `System` node pools that always require running nodes, `User` node pools allow you to scale to 0. To learn more on the differences between system and user node pools, see [System and user node pools](#).

To scale a user pool to 0, you can use the `az aks nodepool scale` in alternative to the above `az aks scale` command, and set 0 as your node count.

```
az aks nodepool scale --name <your node pool name> --cluster-name myAKScluster --resource-group  
myResourceGroup --node-count 0
```

You can also autoscale `User` node pools to 0 nodes, by setting the `--min-count` parameter of the [Cluster Autoscaler](#) to 0.

Next steps

In this article, you manually scaled an AKS cluster to increase or decrease the number of nodes. You can also use the [cluster autoscaler](#) to automatically scale your cluster.

Stop and Start an Azure Kubernetes Service (AKS) cluster

4/21/2021 • 2 minutes to read • [Edit Online](#)

Your AKS workloads may not need to run continuously, for example a development cluster that is used only during business hours. This leads to times where your Azure Kubernetes Service (AKS) cluster might be idle, running no more than the system components. You can reduce the cluster footprint by scaling all the [User node pools to 0](#), but your [System pool](#) is still required to run the system components while the cluster is running. To optimize your costs further during these periods, you can completely turn off (stop) your cluster. This action will stop your control plane and agent nodes altogether, allowing you to save on all the compute costs, while maintaining all your objects and cluster state stored for when you start it again. You can then pick up right where you left off after a weekend or to have your cluster running only while you run your batch jobs.

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart [using the Azure CLI](#) or [using the Azure portal](#).

Limitations

When using the cluster start/stop feature, the following restrictions apply:

- This feature is only supported for Virtual Machine Scale Sets backed clusters.
- The cluster state of a stopped AKS cluster is preserved for up to 12 months. If your cluster is stopped for more than 12 months, the cluster state cannot be recovered. For more information, see the [AKS Support Policies](#).
- You can only start or delete a stopped AKS cluster. To perform any operation like scale or upgrade, start your cluster first.

Stop an AKS Cluster

You can use the `az aks stop` command to stop a running AKS cluster's nodes and control plane. The following example stops a cluster named *myAKSCluster*:

```
az aks stop --name myAKSCluster --resource-group myResourceGroup
```

You can verify when your cluster is stopped by using the `az aks show` command and confirming the [powerState](#) shows as [Stopped](#) as on the below output:

```
{
[...]
  "nodeResourceGroup": "MC_myResourceGroup_myAKSCluster_westus2",
  "powerState":{
    "code": "Stopped"
  },
  "privateFqdn": null,
  "provisioningState": "Succeeded",
  "resourceGroup": "myResourceGroup",
[...]
}
```

If the `provisioningState` shows `Stopping` that means your cluster hasn't fully stopped yet.

IMPORTANT

If you are using [Pod Disruption Budgets](#) the stop operation can take longer as the drain process will take more time to complete.

Start an AKS Cluster

You can use the `az aks start` command to start a stopped AKS cluster's nodes and control plane. The cluster is restarted with the previous control plane state and number of agent nodes.

The following example starts a cluster named *myAKSCluster*:

```
az aks start --name myAKSCluster --resource-group myResourceGroup
```

You can verify when your cluster has started by using the `az aks show` command and confirming the `powerState` shows `Running` as on the below output:

```
{
[...]
"nodeResourceGroup": "MC_myResourceGroup_myAKSCluster_westus2",
"powerState": {
    "code": "Running"
},
"privateFqdn": null,
"provisioningState": "Succeeded",
"resourceGroup": "myResourceGroup",
[...]
}
```

If the `provisioningState` shows `Starting` that means your cluster hasn't fully started yet.

Next steps

- To learn how to scale `user` pools to 0, see [Scale `User` pools to 0](#).
- To learn how to save costs using Spot instances, see [Add a spot node pool to AKS](#).
- To learn more about the AKS support policies, see [AKS support policies](#).

Use Planned Maintenance to schedule maintenance windows for your Azure Kubernetes Service (AKS) cluster (preview)

5/17/2021 • 4 minutes to read • [Edit Online](#)

Your AKS cluster has regular maintenance performed on it automatically. By default, this work can happen at any time. Planned Maintenance allows you to schedule weekly maintenance windows that will update your control plane as well as your kube-system Pods on a VMSS instance and minimize workload impact. Once scheduled, all your maintenance will occur during the window you selected. You can schedule one or more weekly windows on your cluster by specifying a day or time range on a specific day. Maintenance Windows are configured using the Azure CLI.

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart [using the Azure CLI](#) or [using the Azure portal](#).

IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

Limitations

When using Planned Maintenance, the following restrictions apply:

- AKS reserves the right to break these windows for unplanned/reactive maintenance operations that are urgent or critical.
- Currently, performing maintenance operations are considered *best-effort only* and are not guaranteed to occur within a specified window.
- Updates cannot be blocked for more than seven days.

Install `aks-preview` CLI extension

You also need the `aks-preview` Azure CLI extension version 0.5.4 or later. Install the `aks-preview` Azure CLI extension by using the [az extension add](#) command. Or install any available updates by using the [az extension update](#) command.

```
# Install the aks-preview extension
az extension add --name aks-preview

# Update the extension to make sure you have the latest version installed
az extension update --name aks-preview
```

Allow maintenance on every Monday at 1:00am to 2:00am

To add a maintenance window, you can use the `az aks maintenanceconfiguration add` command.

IMPORTANT

Planned Maintenance windows are specified in Coordinated Universal Time (UTC).

```
az aks maintenanceconfiguration add -g MyResourceGroup --cluster-name myAKSCluster --name default --weekday Monday --start-hour 1
```

The following example output shows the maintenance window from 1:00am to 2:00am every Monday.

```
{- Finished ..
  "id": "/subscriptions/<subscriptionID>/resourcegroups/MyResourceGroup/providers/Microsoft.ContainerService/managedClusters/myAKSCluster/maintenanceConfigurations/default",
  "name": "default",
  "notAllowedTime": null,
  "resourceGroup": "MyResourceGroup",
  "systemData": null,
  "timeInWeek": [
    {
      "day": "Monday",
      "hourSlots": [
        1
      ]
    }
  ],
  "type": null
}
```

To allow maintenance any time during a day, omit the *start-hour* parameter. For example, the following command sets the maintenance window for the full day every Monday:

```
az aks maintenanceconfiguration add -g MyResourceGroup --cluster-name myAKSCluster --name default --weekday Monday
```

Add a maintenance configuration with a JSON file

You can also use a JSON file to create a maintenance window instead of using parameters. Create a `test.json` file with the following contents:

```
{  
    "timeInWeek": [  
        {  
            "day": "Tuesday",  
            "hour_slots": [  
                1,  
                2  
            ]  
        },  
        {  
            "day": "Wednesday",  
            "hour_slots": [  
                1,  
                6  
            ]  
        }  
    ],  
    "notAllowedTime": [  
        {  
            "start": "2021-05-26T03:00:00Z",  
            "end": "2021-05-30T12:00:00Z"  
        }  
    ]  
}
```

The above JSON file specifies maintenance windows every Tuesday at 1:00am - 3:00am and every Wednesday at 1:00am - 2:00am and at 6:00am - 7:00am. There is also an exception from `2021-05-26T03:00:00Z` to `2021-05-30T12:00:00Z` where maintenance isn't allowed even if it overlaps with a maintenance window. The following command adds the maintenance windows from `test.json`.

```
az aks maintenanceconfiguration add -g MyResourceGroup --cluster-name myAKSCluster --name default --config-file ./test.json
```

Update an existing maintenance window

To update an existing maintenance configuration, use the `az aks maintenanceconfiguration update` command.

```
az aks maintenanceconfiguration update -g MyResourceGroup --cluster-name myAKSCluster --name default --weekday Monday --start-hour 1
```

List all maintenance windows in an existing cluster

To see all current maintenance configuration windows in your AKS Cluster, use the

```
az aks maintenanceconfiguration list
```

```
az aks maintenanceconfiguration list -g MyResourceGroup --cluster-name myAKSCluster
```

In the output below, you can see that there are two maintenance windows configured for myAKSCluster. One window is on Mondays at 1:00am and another window is on Friday at 4:00am.

```
[  
  {  
    "id":  
      "/subscriptions/<subscriptionID>/resourcegroups/MyResourceGroup/providers/Microsoft.ContainerService/managedClusters/myAKSCluster/maintenanceConfigurations/default",  
      "name": "default",  
      "notAllowedTime": null,  
      "resourceGroup": "MyResourceGroup",  
      "systemData": null,  
      "timeInWeek": [  
        {  
          "day": "Monday",  
          "hourSlots": [  
            1  
          ]  
        }  
      ],  
      "type": null  
    },  
    {  
      "id":  
        "/subscriptions/<subscriptionID>/resourcegroups/MyResourceGroup/providers/Microsoft.ContainerService/managedClusters/myAKSCluster/maintenanceConfigurations/testConfiguration",  
        "name": "testConfiguration",  
        "notAllowedTime": null,  
        "resourceGroup": "MyResourceGroup",  
        "systemData": null,  
        "timeInWeek": [  
          {  
            "day": "Friday",  
            "hourSlots": [  
              4  
            ]  
          }  
        ],  
        "type": null  
      }  
  ]
```

Show a specific maintenance configuration window in an AKS cluster

To see a specific maintenance configuration window in your AKS Cluster, use the

```
az aks maintenanceconfiguration show
```

```
az aks maintenanceconfiguration show -g MyResourceGroup --cluster-name myAKSCluster --name default
```

The following example output shows the maintenance window for *default*.

```
{  
  "id":  
    "/subscriptions/<subscriptionID>/resourcegroups/MyResourceGroup/providers/Microsoft.ContainerService/managed  
Clusters/myAKSCluster/maintenanceConfigurations/default",  
  "name": "default",  
  "notAllowedTime": null,  
  "resourceGroup": "MyResourceGroup",  
  "systemData": null,  
  "timeInWeek": [  
    {  
      "day": "Monday",  
      "hourSlots": [  
        1  
      ]  
    }  
  ],  
  "type": null  
}
```

Delete a certain maintenance configuration window in an existing AKS Cluster

To delete a certain maintenance configuration window in your AKS Cluster, use the

```
az aks maintenanceconfiguration delete
```

```
az aks maintenanceconfiguration delete -g MyResourceGroup --cluster-name myAKSCluster --name default
```

Using Planned Maintenance with Cluster Auto-Upgrade

Planned Maintenance will detect if you are using Cluster Auto-Upgrade and schedule your upgrades during your maintenance window automatically. For more details on about Cluster Auto-Upgrade, see [Upgrade an Azure Kubernetes Service \(AKS\) cluster](#).

Next steps

- To get started with upgrading your AKS cluster, see [Upgrade an AKS cluster](#)

Upgrade an Azure Kubernetes Service (AKS) cluster

5/17/2021 • 8 minutes to read • [Edit Online](#)

Part of the AKS cluster lifecycle involves performing periodic upgrades to the latest Kubernetes version. It is important you apply the latest security releases, or upgrade to get the latest features. This article shows you how to upgrade the master components or a single, default node pool in an AKS cluster.

For AKS clusters that use multiple node pools or Windows Server nodes, see [Upgrade a node pool in AKS](#).

Before you begin

This article requires that you are running the Azure CLI version 2.0.65 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

WARNING

An AKS cluster upgrade triggers a cordon and drain of your nodes. If you have a low compute quota available, the upgrade may fail. For more information, see [increase quotas](#)

Check for available AKS cluster upgrades

To check which Kubernetes releases are available for your cluster, use the `az aks get-upgrades` command. The following example checks for available upgrades to *myAKSCluster* in *myResourceGroup*.

```
az aks get-upgrades --resource-group myResourceGroup --name myAKSCluster --output table
```

NOTE

When you upgrade a supported AKS cluster, Kubernetes minor versions cannot be skipped. All upgrades must be performed sequentially by major version number. For example, upgrades between *1.14.x-> 1.15.x* or *1.15.x-> 1.16.x* are allowed, however *1.14.x-> 1.16.x* is not allowed.

Skipping multiple versions can only be done when upgrading from an *unsupported version* back to a *supported version*. For example, an upgrade from an unsupported *1.10.x-->* a supported *1.15.x* can be completed.

The following example output shows that the cluster can be upgraded to versions *1.19.1* and *1.19.3*:

Name	ResourceGroup	MasterVersion	Upgrades
default	myResourceGroup	1.18.10	1.19.1, 1.19.3

If no upgrade is available, you will get the message:

```
ERROR: Table output unavailable. Use the --query option to specify an appropriate query. Use --debug for more info.
```

Customize node surge upgrade

IMPORTANT

Node surges require subscription quota for the requested max surge count for each upgrade operation. For example, a cluster that has 5 node pools, each with a count of 4 nodes, has a total of 20 nodes. If each node pool has a max surge value of 50%, additional compute and IP quota of 10 nodes (2 nodes * 5 pools) is required to complete the upgrade.

If using Azure CNI, validate there are available IPs in the subnet as well to [satisfy IP requirements of Azure CNI](#).

By default, AKS configures upgrades to surge with one additional node. A default value of one for the max surge settings will enable AKS to minimize workload disruption by creating an additional node before the cordon/drain of existing applications to replace an older versioned node. The max surge value may be customized per node pool to enable a trade-off between upgrade speed and upgrade disruption. By increasing the max surge value, the upgrade process completes faster, but setting a large value for max surge may cause disruptions during the upgrade process.

For example, a max surge value of 100% provides the fastest possible upgrade process (doubling the node count) but also causes all nodes in the node pool to be drained simultaneously. You may wish to use a higher value such as this for testing environments. For production node pools, we recommend a max_surge setting of 33%.

AKS accepts both integer values and a percentage value for max surge. An integer such as "5" indicates five additional nodes to surge. A value of "50%" indicates a surge value of half the current node count in the pool. Max surge percent values can be a minimum of 1% and a maximum of 100%. A percent value is rounded up to the nearest node count. If the max surge value is lower than the current node count at the time of upgrade, the current node count is used for the max surge value.

During an upgrade, the max surge value can be a minimum of 1 and a maximum value equal to the number of nodes in your node pool. You can set larger values, but the maximum number of nodes used for max surge won't be higher than the number of nodes in the pool at the time of upgrade.

IMPORTANT

The max surge setting on a node pool is permanent. Subsequent Kubernetes upgrades or node version upgrades will use this setting. You may change the max surge value for your node pools at any time. For production node pools, we recommend a max-surge setting of 33%.

Use the following commands to set max surge values for new or existing node pools.

```
# Set max surge for a new node pool
az aks nodepool add -n mynodepool -g MyResourceGroup --cluster-name MyManagedCluster --max-surge 33%
```

```
# Update max surge for an existing node pool
az aks nodepool update -n mynodepool -g MyResourceGroup --cluster-name MyManagedCluster --max-surge 5
```

Upgrade an AKS cluster

With a list of available versions for your AKS cluster, use the [az aks upgrade](#) command to upgrade. During the upgrade process, AKS will:

- add a new buffer node (or as many nodes as configured in [max surge](#)) to the cluster that runs the specified Kubernetes version.
- [cordon and drain](#) one of the old nodes to minimize disruption to running applications (if you're using max surge it will [cordon and drain](#) as many nodes at the same time as the number of buffer nodes specified).

- When the old node is fully drained, it will be reimaged to receive the new version and it will become the buffer node for the following node to be upgraded.
- This process repeats until all nodes in the cluster have been upgraded.
- At the end of the process, the last buffer node will be deleted, maintaining the existing agent node count and zone balance.

```
az aks upgrade \
--resource-group myResourceGroup \
--name myAKScluster \
--kubernetes-version KUBERNETES_VERSION
```

It takes a few minutes to upgrade the cluster, depending on how many nodes you have.

IMPORTANT

Ensure that any `PodDisruptionBudgets` (PDBs) allow for at least 1 pod replica to be moved at a time otherwise the drain/evict operation will fail. If the drain operation fails, the upgrade operation will fail by design to ensure that the applications are not disrupted. Please correct what caused the operation to stop (incorrect PDBs, lack of quota, and so on) and re-try the operation.

To confirm that the upgrade was successful, use the `az aks show` command:

```
az aks show --resource-group myResourceGroup --name myAKScluster --output table
```

The following example output shows that the cluster now runs `1.18.10`:

Name	Location	ResourceGroup	KubernetesVersion	ProvisioningState	Fqdn
myAKScluster	eastus	myResourceGroup	1.18.10	Succeeded	myakscluster-dns-379cbbb9.hcp.eastus.azmk8s.io

Set auto-upgrade channel

In addition to manually upgrading a cluster, you can set an auto-upgrade channel on your cluster. The following upgrade channels are available:

CHANNEL	ACTION	EXAMPLE
<code>none</code>	disables auto-upgrades and keeps the cluster at its current version of Kubernetes	Default setting if left unchanged
<code>patch</code>	automatically upgrade the cluster to the latest supported patch version when it becomes available while keeping the minor version the same.	For example, if a cluster is running version <code>1.17.7</code> and versions <code>1.17.9</code> , <code>1.18.4</code> , <code>1.18.6</code> , and <code>1.19.1</code> are available, your cluster is upgraded to <code>1.17.9</code>
<code>stable</code>	automatically upgrade the cluster to the latest supported patch release on minor version $N-1$, where N is the latest supported minor version.	For example, if a cluster is running version <code>1.17.7</code> and versions <code>1.17.9</code> , <code>1.18.4</code> , <code>1.18.6</code> , and <code>1.19.1</code> are available, your cluster is upgraded to <code>1.18.6</code> .

CHANNEL	ACTION	EXAMPLE
rapid	automatically upgrade the cluster to the latest supported patch release on the latest supported minor version.	In cases where the cluster is at a version of Kubernetes that is at an $N-2$ minor version where N is the latest supported minor version, the cluster first upgrades to the latest supported patch version on $N-1$ minor version. For example, if a cluster is running version 1.17.7 and versions 1.17.9, 1.18.4, 1.18.6, and 1.19.1 are available, your cluster first is upgraded to 1.18.6, then is upgraded to 1.19.1.
node-image	automatically upgrade the node image to the latest version available.	Microsoft provides patches and new images for image nodes frequently (usually weekly), but your running nodes won't get the new images unless you do a node image upgrade. Turning on the node-image channel will automatically update your node images whenever a new version is available.

NOTE

Cluster auto-upgrade only updates to GA versions of Kubernetes and will not update to preview versions.

Automatically upgrading a cluster follows the same process as manually upgrading a cluster. For more details, see [Upgrade an AKS cluster](#).

The cluster auto-upgrade for AKS clusters is a preview feature.

IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

Register the `AutoUpgradePreview` feature flag by using the `az feature register` command, as shown in the following example:

```
az feature register --namespace Microsoft.ContainerService -n AutoUpgradePreview
```

It takes a few minutes for the status to show *Registered*. Verify the registration status by using the `az feature list` command:

```
az feature list -o table --query "[?contains(name, 'Microsoft.ContainerService/AutoUpgradePreview')].{Name:name,State:properties.state}"
```

When ready, refresh the registration of the *Microsoft.ContainerService* resource provider by using the `az`

[provider register](#) command:

```
az provider register --namespace Microsoft.ContainerService
```

To set the auto-upgrade channel when creating a cluster, use the *auto-upgrade-channel* parameter, similar to the following example.

```
az aks create --resource-group myResourceGroup --name myAKSCluster --auto-upgrade-channel stable --generate-ssh-keys
```

To set the auto-upgrade channel on existing cluster, update the *auto-upgrade-channel* parameter, similar to the following example.

```
az aks update --resource-group myResourceGroup --name myAKSCluster --auto-upgrade-channel stable
```

Using Cluster Auto-Upgrade with Planned Maintenance

If you are using Planned Maintenance as well as Auto-Upgrade, your upgrade will start during your specified maintenance window. For more details on Planned Maintenance, see [Use Planned Maintenance to schedule maintenance windows for your Azure Kubernetes Service \(AKS\) cluster \(preview\)](#).

Next steps

This article showed you how to upgrade an existing AKS cluster. To learn more about deploying and managing AKS clusters, see the set of tutorials.

[AKS tutorials](#)

Azure Kubernetes Service (AKS) Uptime SLA

4/28/2021 • 3 minutes to read • [Edit Online](#)

Uptime SLA is an optional feature to enable a financially backed, higher SLA for a cluster. Uptime SLA guarantees 99.95% availability of the Kubernetes API server endpoint for clusters that use [Availability Zones](#) and 99.9% of availability for clusters that don't use Availability Zones. AKS uses master node replicas across update and fault domains to ensure SLA requirements are met.

Customers needing an SLA to meet compliance requirements or require extending an SLA to their end users should enable this feature. Customers with critical workloads that will benefit from a higher uptime SLA may also benefit. Using the Uptime SLA feature with Availability Zones enables a higher availability for the uptime of the Kubernetes API server.

Customers can still create unlimited free clusters with a service level objective (SLO) of 99.5% and opt for the preferred SLO or SLA Uptime as needed.

IMPORTANT

For clusters with egress lockdown, see [limit egress traffic](#) to open appropriate ports.

Region availability

- Uptime SLA is available in public regions and Azure Government regions where [AKS is supported](#).
- Uptime SLA is available for [private AKS clusters](#) in all public regions where AKS is supported.

SLA terms and conditions

Uptime SLA is a paid feature and enabled per cluster. Uptime SLA pricing is determined by the number of discrete clusters, and not by the size of the individual clusters. You can view [Uptime SLA pricing details](#) for more information.

Before you begin

- Install the [Azure CLI](#) version 2.8.0 or later

Creating a new cluster with Uptime SLA

To create a new cluster with the Uptime SLA, you use the Azure CLI.

The following example creates a resource group named *myResourceGroup* in the *eastus* location:

```
# Create a resource group
az group create --name myResourceGroup --location eastus
```

Use the [`az aks create`](#) command to create an AKS cluster. The following example creates a cluster named *myAKSCluster* with one node. This operation takes several minutes to complete:

```
# Create an AKS cluster with uptime SLA
az aks create --resource-group myResourceGroup --name myAKSCluster --uptime-sla --node-count 1
```

After a few minutes, the command completes and returns JSON-formatted information about the cluster. The following JSON snippet shows the paid tier for the SKU, indicating your cluster is enabled with Uptime SLA:

```
},
"sku": {
  "name": "Basic",
  "tier": "Paid"
},
```

Modify an existing cluster to use Uptime SLA

You can optionally update your existing clusters to use Uptime SLA.

If you created an AKS cluster with the previous steps, delete the resource group:

```
# Delete the existing cluster by deleting the resource group
az group delete --name myResourceGroup --yes --no-wait
```

Create a new resource group:

```
# Create a resource group
az group create --name myResourceGroup --location eastus
```

Create a new cluster, and don't use Uptime SLA:

```
# Create a new cluster without uptime SLA
az aks create --resource-group myResourceGroup --name myAKSCluster --node-count 1
```

Use the `az aks update` command to update the existing cluster:

```
# Update an existing cluster to use Uptime SLA
az aks update --resource-group myResourceGroup --name myAKSCluster --uptime-sla
```

The following JSON snippet shows the paid tier for the SKU, indicating your cluster is enabled with Uptime SLA:

```
},
"sku": {
  "name": "Basic",
  "tier": "Paid"
},
```

Opt out of Uptime SLA

You can update your cluster to change to the free tier and opt out of Uptime SLA.

```
# Update an existing cluster to opt out of Uptime SLA
az aks update --resource-group myResourceGroup --name myAKSCluster --no-uptime-sla
```

Clean up

To avoid charges, clean up any resources you created. To delete the cluster, use the `az group delete` command to delete the AKS resource group:

```
az group delete --name myResourceGroup --yes --no-wait
```

Next steps

Use [Availability Zones](#) to increase high availability with your AKS cluster workloads.

Configure your cluster to [limit egress traffic](#).

Reduce latency with proximity placement groups

4/21/2021 • 4 minutes to read • [Edit Online](#)

NOTE

When using proximity placement groups on AKS, colocation only applies to the agent nodes. Node to node and the corresponding hosted pod to pod latency is improved. The colocation does not affect the placement of a cluster's control plane.

When deploying your application in Azure, spreading Virtual Machine (VM) instances across regions or availability zones creates network latency, which may impact the overall performance of your application. A proximity placement group is a logical grouping used to make sure Azure compute resources are physically located close to each other. Some applications like gaming, engineering simulations, and high-frequency trading (HFT) require low latency and tasks that complete quickly. For high-performance computing (HPC) scenarios such as these, consider using [proximity placement groups](#) (PPG) for your cluster's node pools.

Before you begin

This article requires that you are running the Azure CLI version 2.14 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Limitations

- A proximity placement group can map to at most one availability zone.
- A node pool must use Virtual Machine Scale Sets to associate a proximity placement group.
- A node pool can associate a proximity placement group at node pool create time only.

Node pools and proximity placement groups

The first resource you deploy with a proximity placement group attaches to a specific data center. Additional resources deployed with the same proximity placement group are colocated in the same data center. Once all resources using the proximity placement group have been stopped (deallocated) or deleted, it's no longer attached.

- Many node pools can be associated with a single proximity placement group.
- A node pool may only be associated with a single proximity placement group.

Configure proximity placement groups with availability zones

NOTE

While proximity placement groups require a node pool to use at most one availability zone, the [baseline Azure VM SLA of 99.9%](#) is still in effect for VMs in a single zone.

Proximity placement groups are a node pool concept and associated with each individual node pool. Using a PPG resource has no impact on AKS control plane availability. This can impact how a cluster should be designed with zones. To ensure a cluster is spread across multiple zones the following design is recommended.

- Provision a cluster with the first system pool using 3 zones and no proximity placement group associated. This ensures the system pods land in a dedicated node pool which will spread across multiple zones.
- Add additional user node pools with a unique zone and proximity placement group associated to each pool.

An example is nodepool1 in zone 1 and PPG1, nodepool2 in zone 2 and PPG2, nodepool3 in zone 3 with PPG3. This ensures at a cluster level, nodes are spread across multiple zones and each individual node pool is colocated in the designated zone with a dedicated PPG resource.

Create a new AKS cluster with a proximity placement group

The following example uses the `az group create` command to create a resource group named *myResourceGroup* in the *centralus* region. An AKS cluster named *myAKSCluster* is then created using the `az aks create` command.

Accelerated networking greatly improves networking performance of virtual machines. Ideally, use proximity placement groups in conjunction with accelerated networking. By default, AKS uses accelerated networking on [supported virtual machine instances](#), which include most Azure virtual machine with two or more vCPUs.

Create a new AKS cluster with a proximity placement group associated to the first system node pool:

```
# Create an Azure resource group
az group create --name myResourceGroup --location centralus
```

Run the following command, and store the ID that is returned:

```
# Create proximity placement group
az ppg create -n myPPG -g myResourceGroup -l centralus -t standard
```

The command produces output, which includes the *id* value you need for upcoming CLI commands:

```
{
  "availabilitySets": null,
  "colocationStatus": null,
  "id": "/subscriptions/yourSubscriptionID/resourceGroups/myResourceGroup/providers/Microsoft.Compute/proximityPlacementGroups/myPPG",
  "location": "centralus",
  "name": "myPPG",
  "proximityPlacementGroupType": "Standard",
  "resourceGroup": "myResourceGroup",
  "tags": {},
  "type": "Microsoft.Compute/proximityPlacementGroups",
  "virtualMachineScaleSets": null,
  "virtualMachines": null
}
```

Use the proximity placement group resource ID for the *myPPGResourceID* value in the below command:

```
# Create an AKS cluster that uses a proximity placement group for the initial system node pool only. The PPG has no effect on the cluster control plane.
az aks create \
  --resource-group myResourceGroup \
  --name myAKScluster \
  --ppg myPPGResourceID
```

Add a proximity placement group to an existing cluster

You can add a proximity placement group to an existing cluster by creating a new node pool. You can then optionally migrate existing workloads to the new node pool, and then delete the original node pool.

Use the same proximity placement group that you created earlier, and this will ensure agent nodes in both node pools in your AKS cluster are physically located in the same data center.

Use the resource ID from the proximity placement group you created earlier, and add a new node pool with the [az aks nodepool add](#) command:

```
# Add a new node pool that uses a proximity placement group, use a --node-count = 1 for testing
az aks nodepool add \
    --resource-group myResourceGroup \
    --cluster-name myAKSCluster \
    --name mynodepool \
    --node-count 1 \
    --ppg myPPGResourceID
```

Clean up

To delete the cluster, use the [az group delete](#) command to delete the AKS resource group:

```
az group delete --name myResourceGroup --yes --no-wait
```

Next steps

- Learn more about [proximity placement groups](#).

Azure Kubernetes Service (AKS) node image upgrade

4/21/2021 • 3 minutes to read • [Edit Online](#)

AKS supports upgrading the images on a node so you're up to date with the newest OS and runtime updates. AKS provides one new image per week with the latest updates, so it's beneficial to upgrade your node's images regularly for the latest features, including Linux or Windows patches. This article shows you how to upgrade AKS cluster node images and how to update node pool images without upgrading the version of Kubernetes.

For more information about the latest images provided by AKS, see the [AKS release notes](#).

For information on upgrading the Kubernetes version for your cluster, see [Upgrade an AKS cluster](#).

NOTE

The AKS cluster must use virtual machine scale sets for the nodes.

Check if your node pool is on the latest node image

You can see what is the latest node image version available for your node pool with the following command:

```
az aks nodepool get-upgrades \
--nodepool-name mynodepool \
--cluster-name myAKSCluster \
--resource-group myResourceGroup
```

In the output you can see the `latestNodeImageVersion` like on the example below:

```
{
  "id": "/subscriptions/XXXX-XXX-XXX-XXX-
XXXXXXXX/resourcegroups/myResourceGroup/providers/Microsoft.ContainerService/managedClusters/myAKSCluster/agent
Pools/nodepool1/upgradeProfiles/default",
  "kubernetesVersion": "1.17.11",
  "latestNodeImageVersion": "AKSUBuntu-1604-2020.10.28",
  "name": "default",
  "osType": "Linux",
  "resourceGroup": "myResourceGroup",
  "type": "Microsoft.ContainerService/managedClusters/agentPools/upgradeProfiles",
  "upgrades": null
}
```

So for `nodepool1` the latest node image available is `AKSUBuntu-1604-2020.10.28`. You can now compare it with the current node image version in use by your node pool by running:

```
az aks nodepool show \
--resource-group myResourceGroup \
--cluster-name myAKSCluster \
--name mynodepool \
--query nodeImageVersion
```

An example output would be:

```
"AKSUbuntu-1604-2020.10.08"
```

So in this example you could upgrade from the current `AKSUbuntu-1604-2020.10.08` image version to the latest version `AKSUbuntu-1604-2020.10.28`.

Upgrade all nodes in all node pools

Upgrading the node image is done with `az aks upgrade`. To upgrade the node image, use the following command:

```
az aks upgrade \
--resource-group myResourceGroup \
--name myAKSCluster \
--node-image-only
```

During the upgrade, check the status of the node images with the following `kubectl` command to get the labels and filter out the current node image information:

```
kubectl get nodes -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.metadata.labels.kubernetes\\.azure\\.com\\/node-image-version}{"\n"}{end}'
```

When the upgrade is complete, use `az aks show` to get the updated node pool details. The current node image is shown in the `nodeImageVersion` property.

```
az aks show \
--resource-group myResourceGroup \
--name myAKSCluster
```

Upgrade a specific node pool

Upgrading the image on a node pool is similar to upgrading the image on a cluster.

To update the OS image of the node pool without doing a Kubernetes cluster upgrade, use the `--node-image-only` option in the following example:

```
az aks nodepool upgrade \
--resource-group myResourceGroup \
--cluster-name myAKSCluster \
--name mynodepool \
--node-image-only
```

During the upgrade, check the status of the node images with the following `kubectl` command to get the labels and filter out the current node image information:

```
kubectl get nodes -o jsonpath='{range .items[*]}{.metadata.name} {"\t"}{.metadata.labels.kubernetes\\.azure\\.com\\/node-image-version}{"\n"}{end}'
```

When the upgrade is complete, use `az aks nodepool show` to get the updated node pool details. The current node image is shown in the `nodeImageVersion` property.

```
az aks nodepool show \
    --resource-group myResourceGroup \
    --cluster-name myAKSCluster \
    --name mynodepool
```

Upgrade node images with node surge

To speed up the node image upgrade process, you can upgrade your node images using a customizable node surge value. By default, AKS uses one additional node to configure upgrades.

If you'd like to increase the speed of upgrades, use the `--max-surge` value to configure the number of nodes to be used for upgrades so they complete faster. To learn more about the trade-offs of various `--max-surge` settings, see [Customize node surge upgrade](#).

The following command sets the max surge value for performing a node image upgrade:

```
az aks nodepool upgrade \
    --resource-group myResourceGroup \
    --cluster-name myAKSCluster \
    --name mynodepool \
    --max-surge 33% \
    --node-image-only \
    --no-wait
```

During the upgrade, check the status of the node images with the following `kubectl` command to get the labels and filter out the current node image information:

```
kubectl get nodes -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.metadata.labels.kubernetes\.azure\.com\/node-image-version}{"\n"}{end}'
```

Use `az aks nodepool show` to get the updated node pool details. The current node image is shown in the `nodeImageVersion` property.

```
az aks nodepool show \
    --resource-group myResourceGroup \
    --cluster-name myAKSCluster \
    --name mynodepool
```

Next steps

- See the [AKS release notes](#) for information about the latest node images.
- Learn how to upgrade the Kubernetes version with [Upgrade an AKS cluster](#).
- [Automatically apply cluster and node pool upgrades with GitHub Actions](#)
- Learn more about multiple node pools and how to upgrade node pools with [Create and manage multiple node pools](#).

Apply security updates to Azure Kubernetes Service (AKS) nodes automatically using GitHub Actions

3/5/2021 • 6 minutes to read • [Edit Online](#)

Security updates are a key part of maintaining your AKS cluster's security and compliance with the latest fixes for the underlying OS. These updates include OS security fixes or kernel updates. Some updates require a node reboot to complete the process.

Running `az aks upgrade` gives you a zero downtime way to apply updates. The command handles applying the latest updates to all your cluster's nodes, cordoning and draining traffic to the nodes, and restarting the nodes, then allowing traffic to the updated nodes. If you update your nodes using a different method, AKS will not automatically restart your nodes.

NOTE

The main difference between `az aks upgrade` when used with the `--node-image-only` flag is that, when it's used, only the node images will be upgraded. If omitted, both the node images and the Kubernetes control plane version will be upgraded. You can check [the docs for managed upgrades on nodes](#) and [the docs for cluster upgrades](#) for more in-depth information.

All Kubernetes' nodes run in a standard Azure virtual machine (VM). These VMs can be Windows or Linux-based. The Linux-based VMs use an Ubuntu image, with the OS configured to automatically check for updates every night.

When you use the `az aks upgrade` command, Azure CLI creates a surge of new nodes with the latest security and kernel updates, these nodes are initially cordoned to prevent any apps from being scheduled to them until the update is finished. After completion, Azure cordons (makes the node unavailable for scheduling of new workloads) and drains (moves the existent workloads to other node) the older nodes and uncordon the new ones, effectively transferring all the scheduled applications to the new nodes.

This process is better than updating Linux-based kernels manually because Linux requires a reboot when a new kernel update is installed. If you update the OS manually, you also need to reboot the VM, manually cordoning and draining all the apps.

This article shows you how you can automate the update process of AKS nodes. You'll use GitHub Actions and Azure CLI to create an update task based on `cron` that runs automatically.

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart [using the Azure CLI](#) or [using the Azure portal](#).

You also need the Azure CLI version 2.0.59 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

This article also assumes you have a [GitHub](#) account to create your actions in.

Create a timed GitHub Action

`cron` is a utility that allows you to run a set of commands, or job, on an automated schedule. To create job to

update your AKS nodes on an automated schedule, you'll need a repository to host your actions. Usually, GitHub actions are configured in the same repository as your application, but you can use any repository. For this article we'll be using your [profile repository](#). If you don't have one, create a new repository with the same name as your GitHub username.

1. Navigate to your repository on GitHub
2. Click on the **Actions** tab at the top of the page.
3. If you already set up a workflow in this repository, you'll be directed to the list of completed runs, in this case, click on the **New Workflow** button. If this is your first workflow in the repository, GitHub will present you with some project templates, click on the **Set up a workflow yourself** link below the description text.
4. Change the workflow `name` and `on` tags similar to the below. GitHub Actions use the same [POSIX cron syntax](#) as any Linux-based system. In this schedule, we're telling the workflow to run every 15 days at 3am.

```
name: Upgrade cluster node images
on:
  schedule:
    - cron: '0 3 */15 * *'
```

5. Create a new job using the below. This job is named `upgrade-node`, runs on an Ubuntu agent, and will connect to your Azure CLI account to execute the needed steps to upgrade the nodes.

```
name: Upgrade cluster node images
on:
  schedule:
    - cron: '0 3 */15 * *'
jobs:
  upgrade-node:
    runs-on: ubuntu-latest
```

Set up the Azure CLI in the workflow

In the `steps` key, you'll define all the work the workflow will execute to upgrade the nodes.

Download and sign in to the Azure CLI.

1. On the right-hand side of the GitHub Actions screen, find the *marketplace search bar* and type "Azure Login".
2. You'll get as a result, an Action called **Azure Login** published by **Azure**:

The screenshot shows the GitHub Marketplace search results for 'Azure Login'. At the top, there's a search bar with 'Azure Login' typed in. Below it, the text 'Marketplace / Search results' is displayed. The first result is 'Azure Login' by Azure, which has a blue checkmark indicating it is a verified action. The description says: 'Authenticate to Azure and run your Az CLI or Az PowerShell based Actions or scripts. gith...'. To the right of the action card, there's a star icon followed by the number '66', indicating the number of reviews. Below this, another action is listed: 'Azure Container Registry Login' by Azure, also with a blue checkmark, and a description: 'Log in to Azure Container Registry (ACR) or any private container registry'.

3. Click on **Azure Login**. On the next screen, click the **copy icon** in the top right of the code sample.

[Marketplace / Search results / Azure Login](#)

 **Azure Login**
By Azure  v1.1  43

Authenticate to Azure and run your Az CLI or Az PowerShell based Actions or scripts. [github.com/Azure/Actions](#)

[View full Marketplace listing](#)

Installation

Copy and paste the following snippet into your `.yml` file.

Version: v1.1 ▾ 

```
- name: Azure Login
  uses: Azure/login@v1.1
  with:
    # Paste output of `az ad sp create-for-rbac` as value of secret variable
    creds:
    # Set this value to true to enable Azure PowerShell Login in addition to
    enable-AzPSSession: # optional
    # Set this value to true to enable support for accessing tenants without
    allow-no-subscriptions: # optional
```

4. Paste the following under the `steps` key:

```
name: Upgrade cluster node images

on:
  schedule:
    - cron: '0 3 */15 * *'

jobs:
  upgrade-node:
    runs-on: ubuntu-latest

  steps:
    - name: Azure Login
      uses: Azure/login@v1.1
      with:
        creds: ${{ secrets.AZURE_CREDENTIALS }}
```

5. From the Azure CLI, run the following command to generate a new username and password.

```
az ad sp create-for-rbac -o json
```

The output should be similar to the following json:

```
{
  "appId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "displayName": "azure-cli-xxxx-xx-xx-xx-xx-xx",
  "name": "http://azure-cli-xxxx-xx-xx-xx-xx-xx",
  "password": "xXxXxXxXx",
  "tenant": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
}
```

6. In a new browser window navigate to your GitHub repository and open the **Settings** tab of the

repository. Click **Secrets** then, click on **New Repository Secret**.

7. For *Name*, use `AZURE_CREDENTIALS`.

8. For *Value*, add the entire contents from the output of the previous step where you created a new username and password.

Secrets / New secret

Name

`AZURE_CREDENTIALS`

Value

```
{  
  "appId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",  
  "displayName": "azure-cli-xxxx-xx-xx-xx-xx",  
  "name": "http://azure-cli-xxxx-xx-xx-xx-xx",  
  "password": "xXxXxXxX",  
  "tenant": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"  
}
```

Add secret

9. Click **Add Secret**.

The CLI used by your action will be logged to your Azure account and ready to run commands.

To create the steps to execute Azure CLI commands.

1. Navigate to the **search page** on *GitHub marketplace* on the right-hand side of the screen and search *Azure CLI Action*. Choose *Azure CLI Action by Azure*.

Azure CLI Action

[Marketplace / Search results](#)



Azure CLI Action

By Azure

☆ 24

Automate your GitHub workflows using Azure CLI scripts

2. Click the copy button on the *GitHub marketplace result* and paste the contents of the action in the main editor, below the *Azure Login* step, similar to the following:

```

name: Upgrade cluster node images

on:
  schedule:
    - cron: '0 3 */15 * *'

jobs:
  upgrade-node:
    runs-on: ubuntu-latest

    steps:
      - name: Azure Login
        uses: Azure/login@v1.1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}
      - name: Upgrade node images
        uses: Azure/cli@v1.0.0
        with:
          inlineScript: az aks upgrade -g {resourceGroupName} -n {aksClusterName} --node-image-only -
-yes

```

TIP

You can decouple the `-g` and `-n` parameters from the command by adding them to secrets similar to the previous steps. Replace the `{resourceGroupName}` and `{aksClusterName}` placeholders by their secret counterparts, for example `${{secrets.RESOURCE_GROUP_NAME}}` and `${{secrets.AKS_CLUSTER_NAME}}`

3. Rename the file to `upgrade-node-images`.

4. Click **Start Commit**, add a message title, and save the workflow.

Once you create the commit, the workflow will be saved and ready for execution.

NOTE

To upgrade a single node pool instead of all node pools on the cluster, add the `--name` parameter to the

`az aks nodepool upgrade` command to specify the node pool name. For example:

```
az aks nodepool upgrade -g {resourceGroupName} --cluster-name {aksClusterName} --name {{nodePoolName}} -
-node-image-only
```

Run the GitHub Action manually

You can run the workflow manually, in addition to the scheduled run, by adding a new `on` trigger called `workflow_dispatch`. The finished file should look like the YAML below:

```
name: Upgrade cluster node images

on:
  schedule:
    - cron: '0 3 */15 * *'
  workflow_dispatch:

jobs:
  upgrade-node:
    runs-on: ubuntu-latest

    steps:
      - name: Azure Login
        uses: Azure/login@v1.1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}

    # Code for upgrading one or more node pools
```

Next steps

- See the [AKS release notes](#) for information about the latest node images.
- Learn how to upgrade the Kubernetes version with [Upgrade an AKS cluster](#).
- Learn more about multiple node pools and how to upgrade node pools with [Create and manage multiple node pools](#).
- Learn more about [system node pools](#)
- To learn how to save costs using Spot instances, see [add a spot node pool to AKS](#)

Apply security and kernel updates to Linux nodes in Azure Kubernetes Service (AKS)

11/2/2020 • 4 minutes to read • [Edit Online](#)

To protect your clusters, security updates are automatically applied to Linux nodes in AKS. These updates include OS security fixes or kernel updates. Some of these updates require a node reboot to complete the process. AKS doesn't automatically reboot these Linux nodes to complete the update process.

The process to keep Windows Server nodes up to date is a little different. Windows Server nodes don't receive daily updates. Instead, you perform an AKS upgrade that deploys new nodes with the latest base Window Server image and patches. For AKS clusters that use Windows Server nodes, see [Upgrade a node pool in AKS](#).

This article shows you how to use the open-source [kured \(KUBernetes REboot Daemon\)](#) to watch for Linux nodes that require a reboot, then automatically handle the rescheduling of running pods and node reboot process.

NOTE

Kured is an open-source project by Weaveworks. Support for this project in AKS is provided on a best-effort basis. Additional support can be found in the #weave-community Slack channel.

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart using the [Azure CLI](#) or [using the Azure portal](#).

You also need the Azure CLI version 2.0.59 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Understand the AKS node update experience

In an AKS cluster, your Kubernetes nodes run as Azure virtual machines (VMs). These Linux-based VMs use an Ubuntu image, with the OS configured to automatically check for updates every night. If security or kernel updates are available, they are automatically downloaded and installed.



Some security updates, such as kernel updates, require a node reboot to finalize the process. A Linux node that requires a reboot creates a file named `/var/run/reboot-required`. This reboot process doesn't happen

automatically.

You can use your own workflows and processes to handle node reboots, or use `kured` to orchestrate the process. With `kured`, a [DaemonSet](#) is deployed that runs a pod on each Linux node in the cluster. These pods in the DaemonSet watch for existence of the `/var/run/reboot-required` file, and then initiate a process to reboot the nodes.

Node upgrades

There is an additional process in AKS that lets you *upgrade* a cluster. An upgrade is typically to move to a newer version of Kubernetes, not just apply node security updates. An AKS upgrade performs the following actions:

- A new node is deployed with the latest security updates and Kubernetes version applied.
- An old node is cordoned and drained.
- Pods are scheduled on the new node.
- The old node is deleted.

You can't remain on the same Kubernetes version during an upgrade event. You must specify a newer version of Kubernetes. To upgrade to the latest version of Kubernetes, you can [upgrade your AKS cluster](#).

Deploy kured in an AKS cluster

To deploy the `kured` DaemonSet, install the following official Kured Helm chart. This creates a role and cluster role, bindings, and a service account, then deploys the DaemonSet using `kured`.

```
# Add the Kured Helm repository
helm repo add kured https://weaveworks.github.io/kured

# Update your local Helm chart repository cache
helm repo update

# Create a dedicated namespace where you would like to deploy kured into
kubectl create namespace kured

# Install kured in that namespace with Helm 3 (only on Linux nodes, kured is not working on Windows nodes)
helm install kured kured/kured --namespace kured --set nodeSelector."beta\\.kubernetes\\.io/os"=linux
```

You can also configure additional parameters for `kured`, such as integration with Prometheus or Slack. For more information about additional configuration parameters, see the [kured Helm chart](#).

Update cluster nodes

By default, Linux nodes in AKS check for updates every evening. If you don't want to wait, you can manually perform an update to check that `kured` runs correctly. First, follow the steps to [SSH to one of your AKS nodes](#). Once you have an SSH connection to the Linux node, check for updates and apply them as follows:

```
sudo apt-get update && sudo apt-get upgrade -y
```

If updates were applied that require a node reboot, a file is written to `/var/run/reboot-required`. `kured` checks for nodes that require a reboot every 60 minutes by default.

Monitor and review reboot process

When one of the replicas in the DaemonSet has detected that a node reboot is required, a lock is placed on the node through the Kubernetes API. This lock prevents additional pods being scheduled on the node. The lock also indicates that only one node should be rebooted at a time. With the node cordoned off, running pods are

drained from the node, and the node is rebooted.

You can monitor the status of the nodes using the [kubectl get nodes](#) command. The following example output shows a node with a status of *SchedulingDisabled* as the node prepares for the reboot process:

NAME	STATUS	ROLES	AGE	VERSION
aks-nodepool1-28993262-0	Ready,SchedulingDisabled	agent	1h	v1.11.7

Once the update process is complete, you can view the status of the nodes using the [kubectl get nodes](#) command with the `--output wide` parameter. This additional output lets you see a difference in *KERNEL-VERSION* of the underlying nodes, as shown in the following example output. The *aks-nodepool1-28993262-0* was updated in a previous step and shows kernel version *4.15.0-1039-azure*. The node *aks-nodepool1-28993262-1* that hasn't been updated shows kernel version *4.15.0-1037-azure*.

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE
AKS-NODEPOOL1-28993262-0	Ready	agent	1h	v1.11.7	10.240.0.4	<none>	Ubuntu
16.04.6 LTS	4.15.0-1039-azure	docker://3.0.4					
AKS-NODEPOOL1-28993262-1	Ready	agent	1h	v1.11.7	10.240.0.5	<none>	Ubuntu
16.04.6 LTS	4.15.0-1037-azure	docker://3.0.4					

Next steps

This article detailed how to use [kured](#) to reboot Linux nodes automatically as part of the security update process. To upgrade to the latest version of Kubernetes, you can [upgrade your AKS cluster](#).

For AKS clusters that use Windows Server nodes, see [Upgrade a node pool in AKS](#).

Configure an AKS cluster

4/21/2021 • 11 minutes to read • [Edit Online](#)

As part of creating an AKS cluster, you may need to customize your cluster configuration to suit your needs. This article introduces a few options for customizing your AKS cluster.

OS configuration

AKS now supports Ubuntu 18.04 as the default node operating system (OS) in general availability (GA) for clusters in Kubernetes versions higher than 1.18. For versions below 1.18, AKS Ubuntu 16.04 is still the default base image. From Kubernetes v1.18 and higher, the default base is AKS Ubuntu 18.04.

IMPORTANT

Node pools created on Kubernetes v1.18 or greater default to `AKS Ubuntu 18.04` node image. Node pools on a supported Kubernetes version less than 1.18 receive `AKS Ubuntu 16.04` as the node image, but will be updated to `AKS Ubuntu 18.04` once the node pool Kubernetes version is updated to v1.18 or greater.

It is highly recommended to test your workloads on AKS Ubuntu 18.04 node pools prior to using clusters on 1.18 or greater.

Use AKS Ubuntu 18.04 (GA) on new clusters

Clusters created on Kubernetes v1.18 or greater default to `AKS Ubuntu 18.04` node image. Node pools on a supported Kubernetes version less than 1.18 will still receive `AKS Ubuntu 16.04` as the node image, but will be updated to `AKS Ubuntu 18.04` once the cluster or node pool Kubernetes version is updated to v1.18 or greater.

It is highly recommended to test your workloads on AKS Ubuntu 18.04 node pools prior to using clusters on 1.18 or greater.

To create a cluster using `AKS Ubuntu 18.04` node image, simply create a cluster running Kubernetes v1.18 or greater as shown below

```
az aks create --name myAKSCluster --resource-group myResourceGroup --kubernetes-version 1.18.14
```

Use AKS Ubuntu 18.04 (GA) on existing clusters

Clusters created on Kubernetes v1.18 or greater default to `AKS Ubuntu 18.04` node image. Node pools on a supported Kubernetes version less than 1.18 will still receive `AKS Ubuntu 16.04` as the node image, but will be updated to `AKS Ubuntu 18.04` once the cluster or node pool Kubernetes version is updated to v1.18 or greater.

It is highly recommended to test your workloads on AKS Ubuntu 18.04 node pools prior to using clusters on 1.18 or greater.

If your clusters or node pools are ready for `AKS Ubuntu 18.04` node image, you can simply upgrade them to a v1.18 or higher as below.

```
az aks upgrade --name myAKSCluster --resource-group myResourceGroup --kubernetes-version 1.18.14
```

If you just want to upgrade just one node pool:

```
az aks nodepool upgrade -name ubuntu1804 --cluster-name myAKScluster --resource-group myResourceGroup --kubernetes-version 1.18.14
```

Test AKS Ubuntu 18.04 (GA) on existing clusters

Node pools created on Kubernetes v1.18 or greater default to **AKS Ubuntu 18.04** node image. Node pools on a supported Kubernetes version less than 1.18 will still receive **AKS Ubuntu 16.04** as the node image, but will be updated to **AKS Ubuntu 18.04** once the node pool Kubernetes version is updated to v1.18 or greater.

It is highly recommended to test your workloads on AKS Ubuntu 18.04 node pools prior to upgrading your production node pools.

To create a node pool using **AKS Ubuntu 18.04** node image, simply create a node pool running kubernetes v1.18 or greater. Your cluster control plane needs to be at least on v1.18 or greater as well but your other node pools can remain on an older kubernetes version. Below we are first upgrading the control plane and then creating a new node pool with v1.18 that will receive the new node image OS version.

```
az aks upgrade --name myAKScluster --resource-group myResourceGroup --kubernetes-version 1.18.14 --control-plane-only
```

```
az aks nodepool add --name ubuntu1804 --cluster-name myAKScluster --resource-group myResourceGroup --kubernetes-version 1.18.14
```

Container runtime configuration

A container runtime is software that executes containers and manages container images on a node. The runtime helps abstract away sys-calls or operating system (OS) specific functionality to run containers on Linux or Windows. AKS clusters using Kubernetes version 1.19 node pools and greater use **containerd** as its container runtime. AKS clusters using Kubernetes prior to v1.19 for node pools use **Moby** (upstream docker) as its container runtime.



Containerd is an **OCI** (Open Container Initiative) compliant core container runtime that provides the minimum set of required functionality to execute containers and manage images on a node. It was **donated** to the Cloud Native Compute Foundation (CNCF) in March of 2017. The current Moby version that AKS uses already leverages and is built on top of **containerd**, as shown above.

With a **containerd**-based node and node pools, instead of talking to the **Dockershim**, the kubelet will talk directly to **containerd** via the CRI (container runtime interface) plugin, removing extra hops on the flow when compared to the Docker CRI implementation. As such, you'll see better pod startup latency and less resource (CPU and memory) usage.

By using **containerd** for AKS nodes, pod startup latency improves and node resource consumption by the container runtime decreases. These improvements are enabled by this new architecture where kubelet talks directly to **containerd** through the CRI plugin while in Moby/docker architecture kubelet would talk to the **Dockershim** and docker engine before reaching **containerd**, thus having extra hops on the flow.



`Containerd` works on every GA version of Kubernetes in AKS, and in every upstream kubernetes version above v1.19, and supports all Kubernetes and AKS features.

IMPORTANT

Clusters with node pools created on Kubernetes v1.19 or greater default to `containerd` for its container runtime. Clusters with node pools on a supported Kubernetes version less than 1.19 receive `Moby` for its container runtime, but will be updated to `ContainerD` once the node pool Kubernetes version is updated to v1.19 or greater. You can still use `Moby` node pools and clusters on older supported versions until those fall off support.

It is highly recommended to test your workloads on AKS node pools with `containerd` prior to using clusters on 1.19 or greater.

`Containerd` limitations/differences

- To use `containerd` as the container runtime you must use AKS Ubuntu 18.04 as your base OS image.
- While the docker toolset is still present on the nodes, Kubernetes uses `containerd` as the container runtime. Therefore, since Moby/Docker doesn't manage the Kubernetes-created containers on the nodes, you can't view or interact with your containers using Docker commands (like `docker ps`) or the Docker API.
- For `containerd`, we recommend using `crlctl` as a replacement CLI instead of the Docker CLI for troubleshooting pods, containers, and container images on Kubernetes nodes (for example, `crlctl ps`).
 - It doesn't provide the complete functionality of the docker CLI. It's intended for troubleshooting only.
 - `crlctl` offers a more kubernetes-friendly view of containers, with concepts like pods, etc. being present.
- `Containerd` sets up logging using the standardized `cri` logging format (which is different from what you currently get from docker's json driver). Your logging solution needs to support the `cri` logging format (like [Azure Monitor for Containers](#))
- You can no longer access the docker engine, `/var/run/docker.sock`, or use Docker-in-Docker (DinD).
 - If you currently extract application logs or monitoring data from Docker Engine, please use something like [Azure Monitor for Containers](#) instead. Additionally AKS doesn't support running any out of band commands on the agent nodes that could cause instability.
 - Even when using Moby/docker, building images and directly leveraging the docker engine via the methods above is strongly discouraged. Kubernetes isn't fully aware of those consumed resources, and those approaches present numerous issues detailed [here](#) and [here](#), for example.
- Building images - You can continue to use your current docker build workflow as normal, unless you are building images inside your AKS cluster. In this case, please consider switching to the recommended approach for building images using [ACR Tasks](#), or a more secure in-cluster option like `docker buildx`.

Generation 2 virtual machines

Azure supports [Generation 2 \(Gen2\) virtual machines \(VMs\)](#). Generation 2 VMs support key features that aren't supported in generation 1 VMs (Gen1). These features include increased memory, Intel Software Guard Extensions (Intel SGX), and virtualized persistent memory (vPMEM).

Generation 2 VMs use the new UEFI-based boot architecture rather than the BIOS-based architecture used by generation 1 VMs. Only specific SKUs and sizes support Gen2 VMs. Check the [list of supported sizes](#), to see if your SKU supports or requires Gen2.

Additionally not all VM images support Gen2, on AKS Gen2 VMs will use the new [AKS Ubuntu 18.04 image](#). This image supports all Gen2 SKUs and sizes.

Ephemeral OS

By default, Azure automatically replicates the operating system disk for a virtual machine to Azure storage to avoid data loss should the VM need to be relocated to another host. However, since containers aren't designed to have local state persisted, this behavior offers limited value while providing some drawbacks, including slower node provisioning and higher read/write latency.

By contrast, ephemeral OS disks are stored only on the host machine, just like a temporary disk. This provides lower read/write latency, along with faster node scaling and cluster upgrades.

Like the temporary disk, an ephemeral OS disk is included in the price of the virtual machine, so you incur no additional storage costs.

IMPORTANT

When a user does not explicitly request managed disks for the OS, AKS will default to ephemeral OS if possible for a given nodepool configuration.

When using ephemeral OS, the OS disk must fit in the VM cache. The sizes for VM cache are available in the [Azure documentation](#) in parentheses next to IO throughput ("cache size in GiB").

Using the AKS default VM size Standard_DS2_v2 with the default OS disk size of 100GB as an example, this VM size supports ephemeral OS but only has 86GB of cache size. This configuration would default to managed disks if the user does not specify explicitly. If a user explicitly requested ephemeral OS, they would receive a validation error.

If a user requests the same Standard_DS2_v2 with a 60GB OS disk, this configuration would default to ephemeral OS: the requested size of 60GB is smaller than the maximum cache size of 86GB.

Using Standard_D8s_v3 with 100GB OS disk, this VM size supports ephemeral OS and has 200GB of cache space. If a user does not specify the OS disk type, the nodepool would receive ephemeral OS by default.

Ephemeral OS requires at least version 2.15.0 of the Azure CLI.

Use Ephemeral OS on new clusters

Configure the cluster to use Ephemeral OS disks when the cluster is created. Use the `--node-osdisk-type` flag to set Ephemeral OS as the OS disk type for the new cluster.

```
az aks create --name myAKScluster --resource-group myResourceGroup -s Standard_DS3_v2 --node-osdisk-type Ephemeral
```

If you want to create a regular cluster using network-attached OS disks, you can do so by specifying `--node-osdisk-type=Managed`. You can also choose to add more ephemeral OS node pools as per below.

Use Ephemeral OS on existing clusters

Configure a new node pool to use Ephemeral OS disks. Use the `--node-osdisk-type` flag to set as the OS disk type as the OS disk type for that node pool.

```
az aks nodepool add --name ephemeral --cluster-name myAKSCluster --resource-group myResourceGroup -s Standard_DS3_v2 --node-osdisk-type Ephemeral
```

IMPORTANT

With ephemeral OS you can deploy VM and instance images up to the size of the VM cache. In the AKS case, the default node OS disk configuration uses 128GB, which means that you need a VM size that has a cache larger than 128GB. The default Standard_DS2_v2 has a cache size of 86GB, which is not large enough. The Standard_DS3_v2 has a cache size of 172GB, which is large enough. You can also reduce the default size of the OS disk by using `--node-osdisk-size`. The minimum size for AKS images is 30GB.

If you want to create node pools with network-attached OS disks, you can do so by specifying

```
--node-osdisk-type Managed
```

Custom resource group name

When you deploy an Azure Kubernetes Service cluster in Azure, a second resource group gets created for the worker nodes. By default, AKS will name the node resource group `MC_resourcegroupname_clusternamespace_location`, but you can also provide your own name.

To specify your own resource group name, install the `aks-preview` Azure CLI extension version 0.3.2 or later. Using the Azure CLI, use the `--node-resource-group` parameter of the `az aks create` command to specify a custom name for the resource group. If you use an Azure Resource Manager template to deploy an AKS cluster, you can define the resource group name by using the `nodeResourceGroup` property.

```
az aks create --name myAKSCluster --resource-group myResourceGroup --node-resource-group myNodeResourceGroup
```

The secondary resource group is automatically created by the Azure resource provider in your own subscription. You can only specify the custom resource group name when the cluster is created.

As you work with the node resource group, keep in mind that you can't:

- Specify an existing resource group for the node resource group.
- Specify a different subscription for the node resource group.
- Change the node resource group name after the cluster has been created.
- Specify names for the managed resources within the node resource group.
- Modify or delete Azure-created tags of managed resources within the node resource group.

Next steps

- Learn how [upgrade the node images](#) in your cluster.
- See [Upgrade an Azure Kubernetes Service \(AKS\) cluster](#) to learn how to upgrade your cluster to the latest version of Kubernetes.
- Read more about [containerd](#) and [Kubernetes](#)
- See the list of [Frequently asked questions about AKS](#) to find answers to some common AKS questions.
- Read more about [Ephemeral OS disks](#).

Customize node configuration for Azure Kubernetes Service (AKS) node pools (preview)

4/21/2021 • 9 minutes to read • [Edit Online](#)

Customizing your node configuration allows you to configure or tune your operating system (OS) settings or the kubelet parameters to match the needs of the workloads. When you create an AKS cluster or add a node pool to your cluster, you can customize a subset of commonly used OS and kubelet settings. To configure settings beyond this subset, [use a daemon set to customize your needed configurations without loosing AKS support for your nodes.](#)

Register the `CustomNodeConfigPreview` preview feature

To create an AKS cluster or node pool that can customize the kubelet parameters or OS settings, you must enable the `CustomNodeConfigPreview` feature flag on your subscription.

Register the `CustomNodeConfigPreview` feature flag by using the [az feature register](#) command, as shown in the following example:

```
az feature register --namespace "Microsoft.ContainerService" --name "CustomNodeConfigPreview"
```

It takes a few minutes for the status to show *Registered*. Verify the registration status by using the [az feature list](#) command:

```
az feature list -o table --query "[?contains(name, 'Microsoft.ContainerService/CustomNodeConfigPreview')].{Name:name, State:properties.state}"
```

When ready, refresh the registration of the *Microsoft.ContainerService* resource provider by using the [az provider register](#) command:

```
az provider register --namespace Microsoft.ContainerService
```

IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

Install aks-preview CLI extension

To create an AKS cluster or a node pool that can customize the kubelet parameters or OS settings, you need the latest *aks-preview* Azure CLI extension. Install the *aks-preview* Azure CLI extension by using the [az extension add](#) command. Or install any available updates by using the [az extension update](#) command.

```
# Install the aks-preview extension
az extension add --name aks-preview

# Update the extension to make sure you have the latest version installed
az extension update --name aks-preview
```

Use custom node configuration

Kubelet custom configuration

The supported Kubelet parameters and accepted values are listed below.

PARAMETER	ALLOWED VALUES/INTERVAL	DEFAULT	DESCRIPTION
<code>cpuManagerPolicy</code>	none, static	none	The static policy allows containers in Guaranteed pods with integer CPU requests access to exclusive CPUs on the node.
<code>cpuCfsQuota</code>	true, false	true	Enable/Disable CPU CFS quota enforcement for containers that specify CPU limits.
<code>cpuCfsQuotaPeriod</code>	Interval in milliseconds (ms)	<code>100ms</code>	Sets CPU CFS quota period value.
<code>imageGcHighThreshold</code>	0-100	85	The percent of disk usage after which image garbage collection is always run. Minimum disk usage that will trigger garbage collection. To disable image garbage collection, set to 100.
<code>imageGcLowThreshold</code>	0-100, no higher than <code>imageGcHighThreshold</code>	80	The percent of disk usage before which image garbage collection is never run. Minimum disk usage that can trigger garbage collection.
<code>topologyManagerPolicy</code>	none, best-effort, restricted, single-numa-node	none	Optimize NUMA node alignment, see more here . Only kubernetes v1.18+.
<code>allowedUnsafeSysctls</code>	<code>kernel.shm*</code> , <code>kernel.msg*</code> , <code>kernel.sem</code> , <code>fs.mqueue.*</code> , <code>net.*</code>	None	Allowed list of unsafe sysctls or unsafe sysctl patterns.

Linux OS custom configuration

The supported OS settings and accepted values are listed below.

File handle limits

When you're serving a lot of traffic, it's common that the traffic you're serving is coming from a large number of local files. You can tweak the below kernel settings and built-in limits to allow you to handle more, at the cost of some system memory.

SETTING	ALLOWED VALUES/INTERVAL	DEFAULT	DESCRIPTION
<code>fs.file-max</code>	8192 - 12000500	709620	Maximum number of file-handles that the Linux kernel will allocate, by increasing this value you can increase the maximum number of open files permitted.
<code>fs.inotify.max_user_watches</code>	781250 - 2097152	1048576	Maximum number of file watches allowed by the system. Each <i>watch</i> is roughly 90 bytes on a 32-bit kernel, and roughly 160 bytes on a 64-bit kernel.
<code>fs.aio-max-nr</code>	65536 - 6553500	65536	The aio-nr shows the current system-wide number of asynchronous io requests. aio-max-nr allows you to change the maximum value aio-nr can grow to.
<code>fs.nr_open</code>	8192 - 20000500	1048576	The maximum number of file-handles a process can allocate.

Socket and network tuning

For agent nodes, which are expected to handle very large numbers of concurrent sessions, you can use the subset of TCP and network options below that you can tweak per node pool.

SETTING	ALLOWED VALUES/INTERVAL	DEFAULT	DESCRIPTION
<code>net.core.somaxconn</code>	4096 - 3240000	16384	Maximum number of connection requests that can be queued for any given listening socket. An upper limit for the value of the backlog parameter passed to the listen(2) function. If the backlog argument is greater than the <code>somaxconn</code> , then it's silently truncated to this limit.
<code>net.core.netdev_max_backlog</code>	1000 - 3240000	1000	Maximum number of packets, queued on the INPUT side, when the interface receives packets faster than kernel can process them.

SETTING	ALLOWED VALUES/INTERVAL	DEFAULT	DESCRIPTION
<code>net.core.rmem_max</code>	212992 - 134217728	212992	The maximum receive socket buffer size in bytes.
<code>net.core.wmem_max</code>	212992 - 134217728	212992	The maximum send socket buffer size in bytes.
<code>net.core.optmem_max</code>	20480 - 4194304	20480	Maximum ancillary buffer size (option memory buffer) allowed per socket. Socket option memory is used in a few cases to store extra structures relating to usage of the socket.
<code>net.ipv4.tcp_max_syn_backlog</code>	128 - 3240000	16384	The maximum number of queued connection requests that have still not received an acknowledgment from the connecting client. If this number is exceeded, the kernel will begin dropping requests.
<code>net.ipv4.tcp_max_tw_buckets</code>	8000 - 1440000	32768	Maximal number of <code>timewait</code> sockets held by system simultaneously. If this number is exceeded, time-wait socket is immediately destroyed and warning is printed.
<code>net.ipv4.tcp_fin_timeout</code>	5 - 120	60	The length of time an orphaned (no longer referenced by any application) connection will remain in the FIN_WAIT_2 state before it's aborted at the local end.
<code>net.ipv4.tcp_keepalive_time</code>	30 - 432000	7200	How often TCP sends out <code>keepalive</code> messages when <code>keepalive</code> is enabled.
<code>net.ipv4.tcp_keepalive_probes</code>	1 - 15	9	How many <code>keepalive</code> probes TCP sends out, until it decides that the connection is broken.

SETTING	ALLOWED VALUES/INTERVAL	DEFAULT	DESCRIPTION
<code>net.ipv4.tcp_keepalive_intvl</code>	- 75	75	How frequently the probes are sent out. Multiplied by <code>tcp_keepalive_probes</code> it makes up the time to kill a connection that isn't responding, after probes started.
<code>net.ipv4.tcp_tw_reuse</code>	0 or 1	0	Allow to reuse <code>TIME-WAIT</code> sockets for new connections when it's safe from protocol viewpoint.
<code>net.ipv4.ip_local_port_range</code>	[First: 1024 - 60999 and Last: 32768 - 65000]	First: 32768 and Last: 60999	The local port range that is used by TCP and UDP traffic to choose the local port. Comprised of two numbers: The first number is the first local port allowed for TCP and UDP traffic on the agent node, the second is the last local port number.
<code>net.ipv4.neigh.default.gc_thresh1</code>	128000	4096	Minimum number of entries that may be in the ARP cache. Garbage collection won't be triggered if the number of entries is below this setting.
<code>net.ipv4.neigh.default.gc_thresh2</code>	180000	8192	Soft maximum number of entries that may be in the ARP cache. This setting is arguably the most important, as ARP garbage collection will be triggered about 5 seconds after reaching this soft maximum.
<code>net.ipv4.neigh.default.gc_thresh3</code>	100000	16384	Hard maximum number of entries in the ARP cache.
<code>net.netfilter.nf_conntrack_max</code>	131072 - 589824	131072	<code>nf_conntrack</code> is a module that tracks connection entries for NAT within Linux. The <code>nf_conntrack</code> module uses a hash table to record the <i>established connection</i> record of the TCP protocol. <code>nf_conntrack_max</code> is the maximum number of nodes in the hash table, that is, the maximum number of connections supported by the <code>nf_conntrack</code> module or the size of connection tracking table.

SETTING	ALLOWED VALUES/INTERVAL	DEFAULT	DESCRIPTION
<code>net.netfilter.nf_conntrack_buckets</code>	65536 - 147456	65536	<code>nf_conntrack</code> is a module that tracks connection entries for NAT within Linux. The <code>nf_conntrack</code> module uses a hash table to record the <i>established connection</i> record of the TCP protocol. <code>nf_conntrack_buckets</code> is the size of hash table.

Worker limits

Like file descriptor limits, the number of workers or threads that a process can create are limited by both a kernel setting and user limits. The user limit on AKS is unlimited.

SETTING	ALLOWED VALUES/INTERVAL	DEFAULT	DESCRIPTION
<code>kernel.threads-max</code>	20 - 513785	55601	Processes can spin up worker threads. The maximum number of all threads that can be created is set with the kernel setting <code>kernel.threads-max</code> .

Virtual memory

The settings below can be used to tune the operation of the virtual memory (VM) subsystem of the Linux kernel and the `writeout` of dirty data to disk.

SETTING	ALLOWED VALUES/INTERVAL	DEFAULT	DESCRIPTION
<code>vm.max_map_count</code>	65530 - 262144	65530	This file contains the maximum number of memory map areas a process may have. Memory map areas are used as a side-effect of calling <code>malloc</code> , directly by <code>mmap</code> , <code>mprotect</code> , and <code>madvise</code> , and also when loading shared libraries.
<code>vm.vfs_cache_pressure</code>	1 - 500	100	This percentage value controls the tendency of the kernel to reclaim the memory, which is used for caching of directory and inode objects.

SETTING	ALLOWED VALUES/INTERVAL	DEFAULT	DESCRIPTION
<code>vm.swappiness</code>	0 - 100	60	This control is used to define how aggressive the kernel will swap memory pages. Higher values will increase aggressiveness, lower values decrease the amount of swap. A value of 0 instructs the kernel not to initiate swap until the amount of free and file-backed pages is less than the high water mark in a zone.
<code>swapFileSizeMB</code>	1 MB - Size of the temporary disk (<code>/dev/sdb</code>)	None	<code>SwapFileSizeMB</code> specifies size in MB of a swap file will be created on the agent nodes from this node pool.
<code>transparentHugePageEnabled</code>	<code>always</code> , <code>madvise</code> , <code>never</code>	<code>always</code>	Transparent Hugepages is a Linux kernel feature intended to improve performance by making more efficient use of your processor's memory-mapping hardware. When enabled the kernel attempts to allocate <code>hugepages</code> whenever possible and any Linux process will receive 2-MB pages if the <code>mmap</code> region is 2 MB naturally aligned. In certain cases when <code>hugepages</code> are enabled system wide, applications may end up allocating more memory resources. An application may <code>mmap</code> a large region but only touch 1 byte of it, in that case a 2-MB page might be allocated instead of a 4k page for no good reason. This scenario is why it's possible to disable <code>hugepages</code> system-wide or to only have them inside <code>MADV_HUGEPAGE madvise</code> regions.
<code>transparentHugePageDefrag</code>	<code>always</code> , <code>defer</code> , <code>defer+madvise</code> , <code>madvise</code> , <code>never</code>	<code>madvise</code>	This value controls whether the kernel should make aggressive use of memory compaction to make more <code>hugepages</code> available.

IMPORTANT

For ease of search and readability the OS settings are displayed in this document by their name but should be added to the configuration json file or AKS API using [camelCase capitalization convention](#).

Create a `kubeletconfig.json` file with the following contents:

```
{  
  "cpuManagerPolicy": "static",  
  "cpuCfsQuota": true,  
  "cpuCfsQuotaPeriod": "200ms",  
  "imageGcHighThreshold": 90,  
  "imageGcLowThreshold": 70,  
  "topologyManagerPolicy": "best-effort",  
  "allowedUnsafeSysctls": [  
    "kernel.msg*",  
    "net.*"  
,  
  "failSwapOn": false  
}
```

Create a `linuxosconfig.json` file with the following contents:

```
{  
  "transparentHugePageEnabled": "madvise",  
  "transparentHugePageDefrag": "defer+madvise",  
  "swapFileSizeMB": 1500,  
  "sysctls": {  
    "netCoreSomaxconn": 163849,  
    "netIpv4TcpTwReuse": true,  
    "netIpv4IpLocalPortRange": "32000 60000"  
  }  
}
```

Create a new cluster specifying the kubelet and OS configurations using the JSON files created in the previous step.

NOTE

When you create a cluster, you can specify the kubelet configuration, OS configuration, or both. If you specify a configuration when creating a cluster, only the nodes in the initial node pool will have that configuration applied. Any settings not configured in the JSON file will retain the default value.

```
az aks create --name myAKScluster --resource-group myResourceGroup --kubelet-config ./kubeletconfig.json --  
linux-os-config ./linuxosconfig.json
```

Add a new node pool specifying the Kubelet parameters using the JSON file you created.

NOTE

When you add a node pool to an existing cluster, you can specify the kubelet configuration, OS configuration, or both. If you specify a configuration when adding a node pool, only the nodes in the new node pool will have that configuration applied. Any settings not configured in the JSON file will retain the default value.

```
az aks nodepool add --name mynodepool1 --cluster-name myAKSCluster --resource-group myResourceGroup --  
kubelet-config ./kubeletconfig.json
```

Next steps

- Learn [how to configure your AKS cluster](#).
- Learn how [upgrade the node images](#) in your cluster.
- See [Upgrade an Azure Kubernetes Service \(AKS\) cluster](#) to learn how to upgrade your cluster to the latest version of Kubernetes.
- See the list of [Frequently asked questions about AKS](#) to find answers to some common AKS questions.

Authenticate with Azure Container Registry from Azure Kubernetes Service

5/17/2021 • 3 minutes to read • [Edit Online](#)

When you're using Azure Container Registry (ACR) with Azure Kubernetes Service (AKS), an authentication mechanism needs to be established. This operation is implemented as part of the CLI and Portal experience by granting the required permissions to your ACR. This article provides examples for configuring authentication between these two Azure services.

You can set up the AKS to ACR integration in a few simple commands with the Azure CLI. This integration assigns the `AcrPull` role to the managed identity associated to the AKS Cluster.

NOTE

This article covers automatic authentication between AKS and ACR. If you need to pull an image from a private external registry, use an [image pull secret](#).

Before you begin

These examples require:

- **Owner or Azure account administrator** role on the **Azure subscription**
- Azure CLI version 2.7.0 or later

To avoid needing an **Owner or Azure account administrator** role, you can configure a managed identity manually or use an existing managed identity to authenticate ACR from AKS. For more information, see [Use an Azure managed identity to authenticate to an Azure container registry](#).

Create a new AKS cluster with ACR integration

You can set up AKS and ACR integration during the initial creation of your AKS cluster. To allow an AKS cluster to interact with ACR, an Azure Active Directory **managed identity** is used. The following CLI command allows you to authorize an existing ACR in your subscription and configures the appropriate **ACRPull** role for the managed identity. Supply valid values for your parameters below.

```
# set this to the name of your Azure Container Registry. It must be globally unique
MYACR=myContainerRegistry

# Run the following line to create an Azure Container Registry if you do not already have one
az acr create -n $MYACR -g myContainerRegistryResourceGroup --sku basic

# Create an AKS cluster with ACR integration
az aks create -n myAKSCluster -g myResourceGroup --generate-ssh-keys --attach-acr $MYACR
```

Alternatively, you can specify the ACR name using an ACR resource ID, which has the following format:

```
/subscriptions/\<subscription-id\>/resourceGroups/\<resource-group-name\>/providers/Microsoft.ContainerRegistry/registries/\<name\>
```

NOTE

If you are using an ACR that is located in a different subscription from your AKS cluster, use the ACR resource ID when attaching or detaching from an AKS cluster.

```
az aks create -n myAKScluster -g myResourceGroup --generate-ssh-keys --attach-acr  
/subscriptions/<subscription-id>/resourceGroups/myContainerRegistryResourceGroup/providers/Microsoft.ContainerRegistry/registries/myContainerRegistry
```

This step may take several minutes to complete.

Configure ACR integration for existing AKS clusters

Integrate an existing ACR with existing AKS clusters by supplying valid values for **acr-name** or **acr-resource-id** as below.

```
az aks update -n myAKScluster -g myResourceGroup --attach-acr <acr-name>
```

or,

```
az aks update -n myAKScluster -g myResourceGroup --attach-acr <acr-resource-id>
```

You can also remove the integration between an ACR and an AKS cluster with the following

```
az aks update -n myAKScluster -g myResourceGroup --detach-acr <acr-name>
```

or

```
az aks update -n myAKScluster -g myResourceGroup --detach-acr <acr-resource-id>
```

Working with ACR & AKS

Import an image into your ACR

Import an image from docker hub into your ACR by running the following:

```
az acr import -n <acr-name> --source docker.io/library/nginx:latest --image nginx:v1
```

Deploy the sample image from ACR to AKS

Ensure you have the proper AKS credentials

```
az aks get-credentials -g myResourceGroup -n myAKScluster
```

Create a file called **acr-nginx.yaml** that contains the following. Substitute the resource name of your registry for **acr-name**. Example: *myContainerRegistry*.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx0-deployment
  labels:
    app: nginx0-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx0
  template:
    metadata:
      labels:
        app: nginx0
    spec:
      containers:
        - name: nginx
          image: <acr-name>.azurecr.io/nginx:v1
          ports:
            - containerPort: 80
```

Next, run this deployment in your AKS cluster:

```
kubectl apply -f acr-nginx.yaml
```

You can monitor the deployment by running:

```
kubectl get pods
```

You should have two running pods.

NAME	READY	STATUS	RESTARTS	AGE
nginx0-deployment-669dfc4d4b-x74kr	1/1	Running	0	20s
nginx0-deployment-669dfc4d4b-xdpd6	1/1	Running	0	20s

Troubleshooting

- Run the [az aks check-acr](#) command to validate that the registry is accessible from the AKS cluster.
- Learn more about [ACR Monitoring](#)
- Learn more about [ACR Health](#)

Create and configure an Azure Kubernetes Services (AKS) cluster to use virtual nodes

3/5/2021 • 2 minutes to read • [Edit Online](#)

To rapidly scale application workloads in an AKS cluster, you can use virtual nodes. With virtual nodes, you have quick provisioning of pods, and only pay per second for their execution time. You don't need to wait for Kubernetes cluster autoscaler to deploy VM compute nodes to run the additional pods. Virtual nodes are only supported with Linux pods and nodes.

The virtual nodes add-on for AKS, is based on the open source project [Virtual Kubelet](#).

This article gives you an overview of the region availability and networking requirements for using virtual nodes, as well as the known limitations.

Regional availability

All regions, where ACI supports VNET SKUs, are supported for virtual nodes deployments.

For available CPU and Memory SKUs in each region, please check the [Azure Container Instances Resource availability for Azure Container Instances in Azure regions - Linux container groups](#)

Network requirements

Virtual nodes enable network communication between pods that run in Azure Container Instances (ACI) and the AKS cluster. To provide this communication, a virtual network subnet is created and delegated permissions are assigned. Virtual nodes only work with AKS clusters created using *advanced* networking (Azure CNI). By default, AKS clusters are created with *basic* networking (kubenet).

Pods running in Azure Container Instances (ACI) need access to the AKS API server endpoint, in order to configure networking.

Known limitations

Virtual Nodes functionality is heavily dependent on ACI's feature set. In addition to the [quotas and limits for Azure Container Instances](#), the following scenarios are not yet supported with Virtual nodes:

- Using service principal to pull ACR images. [Workaround](#) is to use [Kubernetes secrets](#)
- [Virtual Network Limitations](#) including VNet peering, Kubernetes network policies, and outbound traffic to the internet with network security groups.
- Init containers
- [Host aliases](#)
- [Arguments](#) for exec in ACI
- [DaemonSets](#) will not deploy pods to the virtual nodes
- Virtual nodes support scheduling Linux pods. You can manually install the open source [Virtual Kubelet ACI](#) provider to schedule Windows Server containers to ACI.
- Virtual nodes require AKS clusters with Azure CNI networking.
- Using api server authorized ip ranges for AKS.
- Volume mounting Azure Files share support [General-purpose V1](#). Follow the instructions for mounting [a volume with Azure Files share](#)

- Using IPv6 is not supported.

Next steps

Configure virtual nodes for your clusters:

- [Create virtual nodes using Azure CLI](#)
- [Create virtual nodes using the portal in Azure Kubernetes Services \(AKS\)](#)

Virtual nodes are often one component of a scaling solution in AKS. For more information on scaling solutions, see the following articles:

- [Use the Kubernetes horizontal pod autoscaler](#)
- [Use the Kubernetes cluster autoscaler](#)
- [Check out the Autoscale sample for Virtual Nodes](#)
- [Read more about the Virtual Kubelet open source library](#)

Create and configure an Azure Kubernetes Services (AKS) cluster to use virtual nodes using the Azure CLI

4/21/2021 • 8 minutes to read • [Edit Online](#)

This article shows you how to use the Azure CLI to create and configure the virtual network resources and AKS cluster, then enable virtual nodes.

Before you begin

Virtual nodes enable network communication between pods that run in Azure Container Instances (ACI) and the AKS cluster. To provide this communication, a virtual network subnet is created and delegated permissions are assigned. Virtual nodes only work with AKS clusters created using *advanced* networking (Azure CNI). By default, AKS clusters are created with *basic* networking (kubenet). This article shows you how to create a virtual network and subnets, then deploy an AKS cluster that uses advanced networking.

IMPORTANT

Before using virtual nodes with AKS, review both the [limitations of AKS virtual nodes](#) and the [virtual networking limitations of ACI](#). These limitations affect the location, networking configuration, and other configuration details of both your AKS cluster and the virtual nodes.

If you have not previously used ACI, register the service provider with your subscription. You can check the status of the ACI provider registration using the [az provider list](#) command, as shown in the following example:

```
az provider list --query "[?contains(namespace,'Microsoft.ContainerInstance')]" -o table
```

The *Microsoft.ContainerInstance* provider should report as *Registered*, as shown in the following example output:

Namespace	RegistrationState	RegistrationPolicy
Microsoft.ContainerInstance	Registered	RegistrationRequired

If the provider shows as *NotRegistered*, register the provider using the [az provider register](#) as shown in the following example:

```
az provider register --namespace Microsoft.ContainerInstance
```

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to <https://shell.azure.com/bash>. Select **Copy** to copy the blocks of code,

paste it into the Cloud Shell, and press enter to run it.

If you prefer to install and use the CLI locally, this article requires Azure CLI version 2.0.49 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Create a resource group

An Azure resource group is a logical group in which Azure resources are deployed and managed. Create a resource group with the `az group create` command. The following example creates a resource group named `myResourceGroup` in the `westus` location.

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

Create a virtual network

Create a virtual network using the `az network vnet create` command. The following example creates a virtual network name `myVnet` with an address prefix of `10.0.0.0/8`, and a subnet named `myAKSSubnet`. The address prefix of this subnet defaults to `10.240.0.0/16`:

```
az network vnet create \
--resource-group myResourceGroup \
--name myVnet \
--address-prefixes 10.0.0.0/8 \
--subnet-name myAKSSubnet \
--subnet-prefix 10.240.0.0/16
```

Now create an additional subnet for virtual nodes using the `az network vnet subnet create` command. The following example creates a subnet named `myVirtualNodeSubnet` with the address prefix of `10.241.0.0/16`.

```
az network vnet subnet create \
--resource-group myResourceGroup \
--vnet-name myVnet \
--name myVirtualNodeSubnet \
--address-prefixes 10.241.0.0/16
```

Create a service principal or use a managed identity

To allow an AKS cluster to interact with other Azure resources, a cluster identity is used. This cluster identity can be automatically created by the Azure CLI or portal, or you can pre-create one and assign additional permissions. By default, this cluster identity is a managed identity. For more information, see [Use managed identities](#). You can also use a service principal as your cluster identity. The following steps show you how to manually create and assign the service principal to your cluster.

Create a service principal using the `az ad sp create-for-rbac` command. The `--skip-assignment` parameter limits any additional permissions from being assigned.

```
az ad sp create-for-rbac --skip-assignment
```

The output is similar to the following example:

```
{  
  "appId": "bef76eb3-d743-4a97-9534-03e9388811fc",  
  "displayName": "azure-cli-2018-11-21-18-42-00",  
  "name": "http://azure-cli-2018-11-21-18-42-00",  
  "password": "1d257915-8714-4ce7-a7fb-0e5a5411df7f",  
  "tenant": "72f988bf-86f1-41af-91ab-2d7cd011db48"  
}
```

Make a note of the *appId* and *password*. These values are used in the following steps.

Assign permissions to the virtual network

To allow your cluster to use and manage the virtual network, you must grant the AKS service principal the correct rights to use the network resources.

First, get the virtual network resource ID using [az network vnet show](#):

```
az network vnet show --resource-group myResourceGroup --name myVnet --query id -o tsv
```

To grant the correct access for the AKS cluster to use the virtual network, create a role assignment using the [az role assignment create](#) command. Replace `<appId>` and `<vnetId>` with the values gathered in the previous two steps.

```
az role assignment create --assignee <appId> --scope <vnetId> --role Contributor
```

Create an AKS cluster

You deploy an AKS cluster into the AKS subnet created in a previous step. Get the ID of this subnet using [az network vnet subnet show](#):

```
az network vnet subnet show --resource-group myResourceGroup --vnet-name myVnet --name myAKSSubnet --query id -o tsv
```

Use the [az aks create](#) command to create an AKS cluster. The following example creates a cluster named *myAKSCluster* with one node. Replace `<subnetId>` with the ID obtained in the previous step, and then `<appId>` and `<password>` with the values gathered in the previous section.

```
az aks create \  
  --resource-group myResourceGroup \  
  --name myAKScluster \  
  --node-count 1 \  
  --network-plugin azure \  
  --service-cidr 10.0.0.0/16 \  
  --dns-service-ip 10.0.0.10 \  
  --docker-bridge-address 172.17.0.1/16 \  
  --vnet-subnet-id <subnetId> \  
  --service-principal <appId> \  
  --client-secret <password>
```

After several minutes, the command completes and returns JSON-formatted information about the cluster.

Enable virtual nodes addon

To enable virtual nodes, now use the [az aks enable-addons](#) command. The following example uses the subnet

named *myVirtualNodeSubnet* created in a previous step:

```
az aks enable-addons \
--resource-group myResourceGroup \
--name myAKSCluster \
--addons virtual-node \
--subnet-name myVirtualNodeSubnet
```

Connect to the cluster

To configure `kubectl` to connect to your Kubernetes cluster, use the [az aks get-credentials](#) command. This step downloads credentials and configures the Kubernetes CLI to use them.

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```

To verify the connection to your cluster, use the [kubectl get](#) command to return a list of the cluster nodes.

```
kubectl get nodes
```

The following example output shows the single VM node created and then the virtual node for Linux, *virtual-node-aci-linux*.

NAME	STATUS	ROLES	AGE	VERSION
virtual-node-aci-linux	Ready	agent	28m	v1.11.2
aks-agentpool-14693408-0	Ready	agent	32m	v1.11.2

Deploy a sample app

Create a file named `virtual-node.yaml` and copy in the following YAML. To schedule the container on the node, a `nodeSelector` and `toleration` are defined.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: aci-helloworld
spec:
  replicas: 1
  selector:
    matchLabels:
      app: aci-helloworld
  template:
    metadata:
      labels:
        app: aci-helloworld
    spec:
      containers:
        - name: aci-helloworld
          image: mcr.microsoft.com/azuredocs/aci-helloworld
          ports:
            - containerPort: 80
      nodeSelector:
        kubernetes.io/role: agent
        beta.kubernetes.io/os: linux
        type: virtual-kubelet
      tolerations:
        - key: virtual-kubelet.io/provider
          operator: Exists
        - key: azure.com/aci
          effect: NoSchedule
```

Run the application with the [kubectl apply](#) command.

```
kubectl apply -f virtual-node.yaml
```

Use the [kubectl get pods](#) command with the `-o wide` argument to output a list of pods and the scheduled node. Notice that the `aci-helloworld` pod has been scheduled on the `virtual-node-aci-linux` node.

```
kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
aci-helloworld-9b55975f-bnmfl	1/1	Running	0	4m	10.241.0.4	virtual-node-aci-linux

The pod is assigned an internal IP address from the Azure virtual network subnet delegated for use with virtual nodes.

NOTE

If you use images stored in Azure Container Registry, [configure and use a Kubernetes secret](#). A current limitation of virtual nodes is that you can't use integrated Azure AD service principal authentication. If you don't use a secret, pods scheduled on virtual nodes fail to start and report the error `HTTP response status code 400 error code "InaccessibleImage"`.

Test the virtual node pod

To test the pod running on the virtual node, browse to the demo application with a web client. As the pod is assigned an internal IP address, you can quickly test this connectivity from another pod on the AKS cluster. Create a test pod and attach a terminal session to it:

```
kubectl run -it --rm testvk --image=mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11
```

Install `curl` in the pod using `apt-get`:

```
apt-get update && apt-get install -y curl
```

Now access the address of your pod using `curl`, such as <http://10.241.0.4>. Provide your own internal IP address shown in the previous `kubectl get pods` command:

```
curl -L http://10.241.0.4
```

The demo application is displayed, as shown in the following condensed example output:

```
<html>
<head>
  <title>Welcome to Azure Container Instances!</title>
</head>
[...]
```

Close the terminal session to your test pod with `exit`. When your session is ended, the pod is deleted.

Remove virtual nodes

If you no longer wish to use virtual nodes, you can disable them using the `az aks disable-addons` command.

If necessary, go to <https://shell.azure.com> to open Azure Cloud Shell in your browser.

First, delete the `aci-helloworld` pod running on the virtual node:

```
kubectl delete -f virtual-node.yaml
```

The following example command disables the Linux virtual nodes:

```
az aks disable-addons --resource-group myResourceGroup --name myAKSCluster --addons virtual-node
```

Now, remove the virtual network resources and resource group:

```
# Change the name of your resource group, cluster and network resources as needed
RES_GROUP=myResourceGroup
AKS_CLUSTER=myAKScluster
AKS_VNET=myVnet
AKS_SUBNET=myVirtualNodeSubnet

# Get AKS node resource group
NODE_RES_GROUP=$(az aks show --resource-group $RES_GROUP --name $AKS_CLUSTER --query nodeResourceGroup --
output tsv)

# Get network profile ID
NETWORK_PROFILE_ID=$(az network profile list --resource-group $NODE_RES_GROUP --query [0].id --output tsv)

# Delete the network profile
az network profile delete --id $NETWORK_PROFILE_ID -y

# Delete the subnet delegation to Azure Container Instances
az network vnet subnet update --resource-group $RES_GROUP --vnet-name $AKS_VNET --name $AKS_SUBNET --remove
delegations 0
```

Next steps

In this article, a pod was scheduled on the virtual node and assigned a private, internal IP address. You could instead create a service deployment and route traffic to your pod through a load balancer or ingress controller. For more information, see [Create a basic ingress controller in AKS](#).

Virtual nodes are often one component of a scaling solution in AKS. For more information on scaling solutions, see the following articles:

- [Use the Kubernetes horizontal pod autoscaler](#)
- [Use the Kubernetes cluster autoscaler](#)
- [Check out the Autoscale sample for Virtual Nodes](#)
- [Read more about the Virtual Kubelet open source library](#)

Create and configure an Azure Kubernetes Services (AKS) cluster to use virtual nodes in the Azure portal

4/21/2021 • 5 minutes to read • [Edit Online](#)

This article shows you how to use the Azure portal to create and configure the virtual network resources and an AKS cluster with virtual nodes enabled.

NOTE

This article gives you an overview of the region availability and limitations using virtual nodes.

Before you begin

Virtual nodes enable network communication between pods that run in Azure Container Instances (ACI) and the AKS cluster. To provide this communication, a virtual network subnet is created and delegated permissions are assigned. Virtual nodes only work with AKS clusters created using *advanced* networking (Azure CNI). By default, AKS clusters are created with *basic* networking (kubenet). This article shows you how to create a virtual network and subnets, then deploy an AKS cluster that uses advanced networking.

If you have not previously used ACI, register the service provider with your subscription. You can check the status of the ACI provider registration using the [az provider list](#) command, as shown in the following example:

```
az provider list --query "[?contains(namespace, 'Microsoft.ContainerInstance')]" -o table
```

The *Microsoft.ContainerInstance* provider should report as *Registered*, as shown in the following example output:

Namespace	RegistrationState	RegistrationPolicy
Microsoft.ContainerInstance	Registered	RegistrationRequired

If the provider shows as *NotRegistered*, register the provider using the [az provider register](#) as shown in the following example:

```
az provider register --namespace Microsoft.ContainerInstance
```

Sign in to Azure

Sign in to the Azure portal at <https://portal.azure.com>.

Create an AKS cluster

In the top left-hand corner of the Azure portal, select **Create a resource > Kubernetes Service**.

On the **Basics** page, configure the following options:

- **PROJECT DETAILS:** Select an Azure subscription, then select or create an Azure resource group, such as `myResourceGroup`. Enter a **Kubernetes cluster name**, such as `myAKSCluster`.
- **CLUSTER DETAILS:** Select a region and Kubernetes version for the AKS cluster.
- **PRIMARY NODE POOL:** Select a VM size for the AKS nodes. The VM size **cannot** be changed once an AKS cluster has been deployed.
 - Select the number of nodes to deploy into the cluster. For this article, set **Node count** to **1**. Node count **can** be adjusted after the cluster has been deployed.

Click **Next: Node Pools**.

On the **Node Pools** page, select *Enable virtual nodes*.

[Home](#) > [Kubernetes services](#) >

Create Kubernetes cluster ... X

[Basics](#) [**Node pools**](#) [Authentication](#) [Networking](#) [Integrations](#) [Tags](#) [Review + create](#)

Node pools

In addition to the required primary node pool configured on the Basics tab, you can also add optional node pools to handle a variety of workloads [Learn more about multiple node pools](#)

[+ Add node pool](#) [Delete](#)

Name	Mode	OS type	Node count	Node size
<input type="checkbox"/> <code>agentpool</code>	System	Linux	1	Standard_DS2_v2

Enable virtual nodes

Virtual nodes allow burstable scaling backed by serverless Azure Container Instances. [Learn more about virtual nodes](#)

`Enable virtual nodes` (i) ✓

Enable virtual machine scale sets

Enabling virtual machine scale sets will create a cluster that uses virtual machine scale sets instead of individual virtual machines for the cluster nodes. Virtual machine scale sets are required for scenarios including autoscaling, multiple node pools, and Windows support. [Learn more about virtual machine scale sets in AKS](#)

`Enable virtual machine scale sets` (i) ✓
 (i) Virtual machine scale sets are required for availability zones

[Review + create](#)

[< Previous](#)

[Next : Authentication >](#)

By default, a cluster identity is created. This cluster identity is used for cluster communication and integration with other Azure services. By default, this cluster identity is a managed identity. For more information, see [Use managed identities](#). You can also use a service principal as your cluster identity.

The cluster is also configured for advanced networking. The virtual nodes are configured to use their own Azure virtual network subnet. This subnet has delegated permissions to connect Azure resources between the AKS cluster. If you don't already have delegated subnet, the Azure portal creates and configures the Azure virtual network and subnet for use with the virtual nodes.

Select **Review + create**. After the validation is complete, select **Create**.

It takes a few minutes to create the AKS cluster and to be ready for use.

Connect to the cluster

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account. To manage a Kubernetes cluster, use `kubectl`, the Kubernetes command-line client. The `kubectl` client is pre-installed in the Azure Cloud Shell.

To open the Cloud Shell, select Try it from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to <https://shell.azure.com/bash>. Select Copy to copy the blocks of code, paste it into the Cloud Shell, and press enter to run it.

Use the az aks get-credentials command to configure kubectl to connect to your Kubernetes cluster. The following example gets credentials for the cluster name *myAKSCluster* in the resource group named *myResourceGroup*.

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```

To verify the connection to your cluster, use the kubectl get command to return a list of the cluster nodes.

```
kubectl get nodes
```

The following example output shows the single VM node created and then the virtual node for Linux, *virtual-node-aci-linux*.

NAME	STATUS	ROLES	AGE	VERSION
virtual-node-aci-linux	Ready	agent	28m	v1.11.2
aks-agentpool-14693408-0	Ready	agent	32m	v1.11.2

Deploy a sample app

In the Azure Cloud Shell, create a file named `virtual-node.yaml` and copy in the following YAML. To schedule the container on the node, a `nodeSelector` and `toleration` are defined. These settings allow the pod to be scheduled on the virtual node and confirm that the feature is successfully enabled.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: aci-helloworld
spec:
  replicas: 1
  selector:
    matchLabels:
      app: aci-helloworld
  template:
    metadata:
      labels:
        app: aci-helloworld
    spec:
      containers:
        - name: aci-helloworld
          image: mcr.microsoft.com/azuredocs/aci-helloworld
          ports:
            - containerPort: 80
      nodeSelector:
        kubernetes.io/role: agent
        beta.kubernetes.io/os: linux
        type: virtual-kubelet
      tolerations:
        - key: virtual-kubelet.io/provider
          operator: Exists
```

Run the application with the kubectl apply command.

```
kubectl apply -f virtual-node.yaml
```

Use the `kubectl get pods` command with the `-o wide` argument to output a list of pods and the scheduled node.

Notice that the `virtual-node-helloworld` pod has been scheduled on the `virtual-node-linux` node.

```
kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
virtual-node-helloworld-9b55975f-bnmfl	1/1	Running	0	4m	10.241.0.4	virtual-node-aci-linux

The pod is assigned an internal IP address from the Azure virtual network subnet delegated for use with virtual nodes.

NOTE

If you use images stored in Azure Container Registry, [configure and use a Kubernetes secret](#). A current limitation of virtual nodes is that you can't use integrated Azure AD service principal authentication. If you don't use a secret, pods scheduled on virtual nodes fail to start and report the error `HTTP response status code 400 error code "InaccessibleImage"`.

Test the virtual node pod

To test the pod running on the virtual node, browse to the demo application with a web client. As the pod is assigned an internal IP address, you can quickly test this connectivity from another pod on the AKS cluster. Create a test pod and attach a terminal session to it:

```
kubectl run -it --rm virtual-node-test --image=mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11
```

Install `curl` in the pod using `apt-get`:

```
apt-get update && apt-get install -y curl
```

Now access the address of your pod using `curl`, such as `http://10.241.0.4`. Provide your own internal IP address shown in the previous `kubectl get pods` command:

```
curl -L http://10.241.0.4
```

The demo application is displayed, as shown in the following condensed example output:

```
<html>
<head>
  <title>Welcome to Azure Container Instances!</title>
</head>
[...]
```

Close the terminal session to your test pod with `exit`. When your session is ended, the pod is deleted.

Next steps

In this article, a pod was scheduled on the virtual node and assigned a private, internal IP address. You could instead create a service deployment and route traffic to your pod through a load balancer or ingress controller. For more information, see [Create a basic ingress controller in AKS](#).

Virtual nodes are one component of a scaling solution in AKS. For more information on scaling solutions, see the following articles:

- [Use the Kubernetes horizontal pod autoscaler](#)
- [Use the Kubernetes cluster autoscaler](#)
- [Check out the Autoscale sample for Virtual Nodes](#)
- [Read more about the Virtual Kubelet open source library](#)

Automatically scale a cluster to meet application demands on Azure Kubernetes Service (AKS)

4/21/2021 • 12 minutes to read • [Edit Online](#)

To keep up with application demands in Azure Kubernetes Service (AKS), you may need to adjust the number of nodes that run your workloads. The cluster autoscaler component can watch for pods in your cluster that can't be scheduled because of resource constraints. When issues are detected, the number of nodes in a node pool is increased to meet the application demand. Nodes are also regularly checked for a lack of running pods, with the number of nodes then decreased as needed. This ability to automatically scale up or down the number of nodes in your AKS cluster lets you run an efficient, cost-effective cluster.

This article shows you how to enable and manage the cluster autoscaler in an AKS cluster.

Before you begin

This article requires that you're running the Azure CLI version 2.0.76 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

About the cluster autoscaler

To adjust to changing application demands, such as between the workday and evening or on a weekend, clusters often need a way to automatically scale. AKS clusters can scale in one of two ways:

- The cluster autoscaler watches for pods that can't be scheduled on nodes because of resource constraints. The cluster then automatically increases the number of nodes.
- The horizontal pod autoscaler uses the Metrics Server in a Kubernetes cluster to monitor the resource demand of pods. If an application needs more resources, the number of pods is automatically increased to meet the demand.



Both the horizontal pod autoscaler and cluster autoscaler can also decrease the number of pods and nodes as needed. The cluster autoscaler decreases the number of nodes when there has been unused capacity for a period of time. Pods on a node to be removed by the cluster autoscaler are safely scheduled elsewhere in the cluster. The cluster autoscaler may be unable to scale down if pods can't move, such as in the following situations:

- A pod is directly created and isn't backed by a controller object, such as a deployment or replica set.
- A pod disruption budget (PDB) is too restrictive and doesn't allow the number of pods to be fall below a certain threshold.
- A pod uses node selectors or anti-affinity that can't be honored if scheduled on a different node.

For more information about how the cluster autoscaler may be unable to scale down, see [What types of pods can prevent the cluster autoscaler from removing a node?](#)

The cluster autoscaler uses startup parameters for things like time intervals between scale events and resource thresholds. For more information on what parameters the cluster autoscaler uses, see [Using the autoscaler profile](#).

The cluster and horizontal pod autoscalers can work together, and are often both deployed in a cluster. When combined, the horizontal pod autoscaler is focused on running the number of pods required to meet application demand. The cluster autoscaler is focused on running the number of nodes required to support the scheduled pods.

NOTE

Manual scaling is disabled when you use the cluster autoscaler. Let the cluster autoscaler determine the required number of nodes. If you want to manually scale your cluster, [disable the cluster autoscaler](#).

Create an AKS cluster and enable the cluster autoscaler

If you need to create an AKS cluster, use the `az aks create` command. To enable and configure the cluster autoscaler on the node pool for the cluster, use the `--enable-cluster-autoscaler` parameter, and specify a node `--min-count` and `--max-count`.

IMPORTANT

The cluster autoscaler is a Kubernetes component. Although the AKS cluster uses a virtual machine scale set for the nodes, don't manually enable or edit settings for scale set autoscale in the Azure portal or using the Azure CLI. Let the Kubernetes cluster autoscaler manage the required scale settings. For more information, see [Can I modify the AKS resources in the node resource group?](#)

The following example creates an AKS cluster with a single node pool backed by a virtual machine scale set. It also enables the cluster autoscaler on the node pool for the cluster and sets a minimum of 1 and maximum of 3 nodes:

```
# First create a resource group
az group create --name myResourceGroup --location eastus

# Now create the AKS cluster and enable the cluster autoscaler
az aks create \
  --resource-group myResourceGroup \
  --name myAKSCluster \
  --node-count 1 \
  --vm-set-type VirtualMachineScaleSets \
  --load-balancer-sku standard \
  --enable-cluster-autoscaler \
  --min-count 1 \
  --max-count 3
```

It takes a few minutes to create the cluster and configure the cluster autoscaler settings.

Update an existing AKS cluster to enable the cluster autoscaler

Use the `az aks update` command to enable and configure the cluster autoscaler on the node pool for the existing cluster. Use the `--enable-cluster-autoscaler` parameter, and specify a node `--min-count` and `--max-count`.

IMPORTANT

The cluster autoscaler is a Kubernetes component. Although the AKS cluster uses a virtual machine scale set for the nodes, don't manually enable or edit settings for scale set autoscale in the Azure portal or using the Azure CLI. Let the Kubernetes cluster autoscaler manage the required scale settings. For more information, see [Can I modify the AKS resources in the node resource group?](#)

The following example updates an existing AKS cluster to enable the cluster autoscaler on the node pool for the cluster and sets a minimum of 1 and maximum of 3 nodes:

```
az aks update \
--resource-group myResourceGroup \
--name myAKSCluster \
--enable-cluster-autoscaler \
--min-count 1 \
--max-count 3
```

It takes a few minutes to update the cluster and configure the cluster autoscaler settings.

Change the cluster autoscaler settings

IMPORTANT

If you have multiple node pools in your AKS cluster, skip to the [autoscale with multiple agent pools section](#). Clusters with multiple agent pools require use of the `az aks nodepool` command set to change node pool specific properties instead of `az aks`.

In the previous step to create an AKS cluster or update an existing node pool, the cluster autoscaler minimum node count was set to 1, and the maximum node count was set to 3. As your application demands change, you may need to adjust the cluster autoscaler node count.

To change the node count, use the `az aks update` command.

```
az aks update \
--resource-group myResourceGroup \
--name myAKSCluster \
--update-cluster-autoscaler \
--min-count 1 \
--max-count 5
```

The above example updates cluster autoscaler on the single node pool in `myAKSCluster` to a minimum of 1 and maximum of 5 nodes.

NOTE

The cluster autoscaler makes scaling decisions based on the minimum and maximum counts set on each node pool, but it does not enforce them after updating the min or max counts. For example, setting a min count of 5 when the current node count is 3 will not immediately scale the pool up to 5. If the minimum count on the node pool has a value higher than the current number of nodes, the new min or max settings will be respected when there are enough unschedulable pods present that would require 2 new additional nodes and trigger an autoscaler event. After the scale event, the new count limits are respected.

Monitor the performance of your applications and services, and adjust the cluster autoscaler node counts to match the required performance.

Using the autoscaler profile

You can also configure more granular details of the cluster autoscaler by changing the default values in the cluster-wide autoscaler profile. For example, a scale down event happens after nodes are under-utilized after 10 minutes. If you had workloads that ran every 15 minutes, you may want to change the autoscaler profile to scale down under utilized nodes after 15 or 20 minutes. When you enable the cluster autoscaler, a default profile is used unless you specify different settings. The cluster autoscaler profile has the following settings that you can update:

SETTING	DESCRIPTION	DEFAULT VALUE
scan-interval	How often cluster is reevaluated for scale up or down	10 seconds
scale-down-delay-after-add	How long after scale up that scale down evaluation resumes	10 minutes
scale-down-delay-after-delete	How long after node deletion that scale down evaluation resumes	scan-interval
scale-down-delay-after-failure	How long after scale down failure that scale down evaluation resumes	3 minutes
scale-down-unneeded-time	How long a node should be unneeded before it is eligible for scale down	10 minutes
scale-down-unready-time	How long an unready node should be unneeded before it is eligible for scale down	20 minutes
scale-down-utilization-threshold	Node utilization level, defined as sum of requested resources divided by capacity, below which a node can be considered for scale down	0.5
max-graceful-termination-sec	Maximum number of seconds the cluster autoscaler waits for pod termination when trying to scale down a node	600 seconds
balance-similar-node-groups	Detects similar node pools and balances the number of nodes between them	false
expander	Type of node pool expander to be used in scale up. Possible values: most-pods , random , least-waste , priority	random
skip-nodes-with-local-storage	If true cluster autoscaler will never delete nodes with pods with local storage, for example, EmptyDir or HostPath	true

SETTING	DESCRIPTION	DEFAULT VALUE
skip-nodes-with-system-pods	If true cluster autoscaler will never delete nodes with pods from kube-system (except for DaemonSet or mirror pods)	true
max-empty-bulk-delete	Maximum number of empty nodes that can be deleted at the same time	10 nodes
new-pod-scale-up-delay	For scenarios like burst/batch scale where you don't want CA to act before the kubernetes scheduler could schedule all the pods, you can tell CA to ignore unscheduled pods before they're a certain age.	0 seconds
max-total-unready-percentage	Maximum percentage of unready nodes in the cluster. After this percentage is exceeded, CA halts operations	45%
max-node-provision-time	Maximum time the autoscaler waits for a node to be provisioned	15 minutes
ok-total-unready-count	Number of allowed unready nodes, irrespective of max-total-unready-percentage	3 nodes

IMPORTANT

The cluster autoscaler profile affects all node pools that use the cluster autoscaler. You can't set an autoscaler profile per node pool.

The cluster autoscaler profile requires version 2.11.7 or greater of the Azure CLI. If you need to install or upgrade, see [Install Azure CLI](#).

Set the cluster autoscaler profile on an existing AKS cluster

Use the `az aks update` command with the `cluster-autoscaler-profile` parameter to set the cluster autoscaler profile on your cluster. The following example configures the scan interval setting as 30s in the profile.

```
az aks update \
--resource-group myResourceGroup \
--name myAKSCluster \
--cluster-autoscaler-profile scan-interval=30s
```

When you enable the cluster autoscaler on node pools in the cluster, those clusters will also use the cluster autoscaler profile. For example:

```
az aks nodepool update \
--resource-group myResourceGroup \
--cluster-name myAKSCluster \
--name mynodepool \
--enable-cluster-autoscaler \
--min-count 1 \
--max-count 3
```

IMPORTANT

When you set the cluster autoscaler profile, any existing node pools with the cluster autoscaler enabled will start using the profile immediately.

Set the cluster autoscaler profile when creating an AKS cluster

You can also use the `cluster-autoscaler-profile` parameter when you create your cluster. For example:

```
az aks create \
    --resource-group myResourceGroup \
    --name myAKSCluster \
    --node-count 1 \
    --enable-cluster-autoscaler \
    --min-count 1 \
    --max-count 3 \
    --cluster-autoscaler-profile scan-interval=30s
```

The above command creates an AKS cluster and defines the scan interval as 30 seconds for the cluster-wide autoscaler profile. The command also enables the cluster autoscaler on the initial node pool, sets the minimum node count to 1 and the maximum node count to 3.

Reset cluster autoscaler profile to default values

Use the `az aks update` command to reset the cluster autoscaler profile on your cluster.

```
az aks update \
    --resource-group myResourceGroup \
    --name myAKSCluster \
    --cluster-autoscaler-profile ""
```

Disable the cluster autoscaler

If you no longer wish to use the cluster autoscaler, you can disable it using the `az aks update` command, specifying the `--disable-cluster-autoscaler` parameter. Nodes aren't removed when the cluster autoscaler is disabled.

```
az aks update \
    --resource-group myResourceGroup \
    --name myAKSCluster \
    --disable-cluster-autoscaler
```

You can manually scale your cluster after disabling the cluster autoscaler by using the `az aks scale` command. If you use the horizontal pod autoscaler, that feature continues to run with the cluster autoscaler disabled, but pods may end up unable to be scheduled if all node resources are in use.

Re-enable a disabled cluster autoscaler

If you wish to re-enable the cluster autoscaler on an existing cluster, you can re-enable it using the `az aks update` command, specifying the `--enable-cluster-autoscaler`, `--min-count`, and `--max-count` parameters.

Retrieve cluster autoscaler logs and status

To diagnose and debug autoscaler events, logs and status can be retrieved from the autoscaler add-on.

AKS manages the cluster autoscaler on your behalf and runs it in the managed control plane. You can enable

control plane node to see the logs and operations from CA.

To configure logs to be pushed from the cluster autoscaler into Log Analytics, follow these steps.

1. Set up a rule for resource logs to push cluster-autoscaler logs to Log Analytics. **Instructions are detailed here**, ensure you check the box for `cluster-autoscaler` when selecting options for "Logs".
 2. Select the "Logs" section on your cluster via the **Azure portal**.
 3. Input the following example query into Log Analytics:

```
AzureDiagnostics  
| where Category == "cluster-autoscaler"
```

You should see logs similar to the following example as long as there are logs to retrieve.

The cluster autoscaler will also write out health status to a configmap named `cluster-autoscaler-status`. To retrieve these logs, execute the following `kubectl` command. A health status will be reported for each node pool configured with the cluster autoscaler.

```
kubectl get configmap -n kube-system cluster-autoscaler-status -o yaml
```

To learn more about what is logged from the autoscaler, read the FAQ on the [Kubernetes/autoscaler GitHub project](#).

Use the cluster autoscaler with multiple node pools enabled

The cluster autoscaler can be used together with multiple node pools enabled. Follow that document to learn how to enable multiple node pools and add additional node pools to an existing cluster. When using both features together, you enable the cluster autoscaler on each individual node pool in the cluster and can pass unique autoscaling rules to each.

The below command assumes you followed the initial instructions earlier in this document and you want to update an existing node pool's max-count from 3 to 5. Use the `az aks nodepool update` command to update an existing node pool's settings.

```
az aks nodepool update \
--resource-group myResourceGroup \
--cluster-name myAKSCluster \
--name nodepool1 \
--update-cluster-autoscaler \
--min-count 1 \
--max-count 5
```

The cluster autoscaler can be disabled with `az aks nodepool update` and passing the

`--disable-cluster-autoscaler` parameter.

```
az aks nodepool update \
--resource-group myResourceGroup \
--cluster-name myAKSCluster \
--name nodepool1 \
--disable-cluster-autoscaler
```

If you wish to re-enable the cluster autoscaler on an existing cluster, you can re-enable it using the `az aks nodepool update` command, specifying the

`--enable-cluster-autoscaler`, `--min-count`, and `--max-count` parameters.

NOTE

If you are planning on using the cluster autoscaler with nodepools that span multiple zones and leverage scheduling features related to zones such as volume topological scheduling, the recommendation is to have one nodepool per zone and enable the `--balance-similar-node-groups` through the autoscaler profile. This will ensure that the autoscaler will scale up successfully and try and keep the sizes of the nodepools balanced.

Next steps

This article showed you how to automatically scale the number of AKS nodes. You can also use the horizontal pod autoscaler to automatically adjust the number of pods that run your application. For steps on using the horizontal pod autoscaler, see [Scale applications in AKS](#).

Create an Azure Kubernetes Service (AKS) cluster that uses availability zones

5/18/2021 • 7 minutes to read • [Edit Online](#)

An Azure Kubernetes Service (AKS) cluster distributes resources such as nodes and storage across logical sections of underlying Azure infrastructure. This deployment model when using availability zones, ensures nodes in a given availability zone are physically separated from those defined in another availability zone. AKS clusters deployed with multiple availability zones configured across a cluster provide a higher level of availability to protect against a hardware failure or a planned maintenance event.

By defining node pools in a cluster to span multiple zones, nodes in a given node pool are able to continue operating even if a single zone has gone down. Your applications can continue to be available even if there is a physical failure in a single datacenter if orchestrated to tolerate failure of a subset of nodes.

This article shows you how to create an AKS cluster and distribute the node components across availability zones.

Before you begin

You need the Azure CLI version 2.0.76 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Limitations and region availability

AKS clusters can currently be created using availability zones in the following regions:

- Australia East
- Brazil South
- Canada Central
- Central US
- East US
- East US 2
- France Central
- Germany West Central
- Japan East
- North Europe
- Southeast Asia
- South Central US
- UK South
- US Gov Virginia
- West Europe
- West US 2

The following limitations apply when you create an AKS cluster using availability zones:

- You can only define availability zones when the cluster or node pool is created.
- Availability zone settings can't be updated after the cluster is created. You also can't update an existing, non-availability zone cluster to use availability zones.

- The chosen node size (VM SKU) selected must be available across all availability zones selected.
- Clusters with availability zones enabled require use of Azure Standard Load Balancers for distribution across zones. This load balancer type can only be defined at cluster create time. For more information and the limitations of the standard load balancer, see [Azure load balancer standard SKU limitations](#).

Azure disks limitations

Volumes that use Azure managed disks are currently not zone-redundant resources. Volumes cannot be attached across zones and must be co-located in the same zone as a given node hosting the target pod.

Kubernetes is aware of Azure availability zones since version 1.12. You can deploy a `PersistentVolumeClaim` object referencing an Azure Managed Disk in a multi-zone AKS cluster and [Kubernetes will take care of scheduling](#) any pod that claims this PVC in the correct availability zone.

Azure Resource Manager templates and availability zones

When *creating* an AKS cluster, if you explicitly define a `null value in a template` with syntax such as `"availabilityZones": null`, the Resource Manager template treats the property as if it doesn't exist, which means your cluster won't have availability zones enabled. Also, if you create a cluster with a Resource Manager template that omits the availability zones property, availability zones are disabled.

You can't update settings for availability zones on an existing cluster, so the behavior is different when updating an AKS cluster with Resource Manager templates. If you explicitly set a null value in your template for availability zones and *update* your cluster, there are no changes made to your cluster for availability zones. However, if you omit the availability zones property with syntax such as `"availabilityZones": []`, the deployment attempts to disable availability zones on your existing AKS cluster and **fails**.

Overview of availability zones for AKS clusters

Availability zones are a high-availability offering that protects your applications and data from datacenter failures. Zones are unique physical locations within an Azure region. Each zone is made up of one or more datacenters equipped with independent power, cooling, and networking. To ensure resiliency, there's always more than one zone in all zone enabled regions. The physical separation of availability zones within a region protects applications and data from datacenter failures.

For more information, see [What are availability zones in Azure?](#).

AKS clusters that are deployed using availability zones can distribute nodes across multiple zones within a single region. For example, a cluster in the *East US 2* region can create nodes in all three availability zones in *East US 2*. This distribution of AKS cluster resources improves cluster availability as they're resilient to failure of a specific zone.



If a single zone becomes unavailable, your applications continue to run if the cluster is spread across multiple zones.

Create an AKS cluster across availability zones

When you create a cluster using the `az aks create` command, the `--zones` parameter defines which zones agent nodes are deployed into. The control plane components such as etcd or the API are spread across the available zones in the region if you define the `--zones` parameter at cluster creation time. The specific zones which the control plane components are spread across are independent of what explicit zones are selected for the initial node pool.

If you don't define any zones for the default agent pool when you create an AKS cluster, control plane components are not guaranteed to spread across availability zones. You can add additional node pools using the `az aks nodepool add` command and specify `--zones` for new nodes, but it will not change how the control plane has been spread across zones. Availability zone settings can only be defined at cluster or node pool create-time.

The following example creates an AKS cluster named `myAKSCluster` in the resource group named `myResourceGroup`. A total of 3 nodes are created - one agent in zone 1, one in 2, and then one in 3.

```
az group create --name myResourceGroup --location eastus2

az aks create \
    --resource-group myResourceGroup \
    --name myAKSCluster \
    --generate-ssh-keys \
    --vm-set-type VirtualMachineScaleSets \
    --load-balancer-sku standard \
    --node-count 3 \
    --zones 1 2 3
```

It takes a few minutes to create the AKS cluster.

When deciding what zone a new node should belong to, a given AKS node pool will use a [best effort zone balancing offered by underlying Azure Virtual Machine Scale Sets](#). A given AKS node pool is considered "balanced" if each zone has the same number of VMs or +- 1 VM in all other zones for the scale set.

Verify node distribution across zones

When the cluster is ready, list the agent nodes in the scale set to see what availability zone they're deployed in.

First, get the AKS cluster credentials using the `az aks get-credentials` command:

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```

Next, use the `kubectl describe` command to list the nodes in the cluster and filter on the `failure-domain.beta.kubernetes.io/zone` value. The following example is for a Bash shell.

```
kubectl describe nodes | grep -e "Name:" -e "failure-domain.beta.kubernetes.io/zone"
```

The following example output shows the three nodes distributed across the specified region and availability zones, such as `eastus2-1` for the first availability zone and `eastus2-2` for the second availability zone:

```
Name:      aks-nodepool1-28993262-vmss000000
           failure-domain.beta.kubernetes.io/zone=eastus2-1
Name:      aks-nodepool1-28993262-vmss000001
           failure-domain.beta.kubernetes.io/zone=eastus2-2
Name:      aks-nodepool1-28993262-vmss000002
           failure-domain.beta.kubernetes.io/zone=eastus2-3
```

As you add additional nodes to an agent pool, the Azure platform automatically distributes the underlying VMs across the specified availability zones.

Note that in newer Kubernetes versions (1.17.0 and later), AKS is using the newer label `topology.kubernetes.io/zone` in addition to the deprecated `failure-domain.beta.kubernetes.io/zone`. You can get the same result as above with by running the following script:

```
kubectl get nodes -o custom-
columns=NAME:'{.metadata.name}',REGION:'{.metadata.labels.topology\\.kubernetes\\.io/region}',ZONE:'{metadata.labels.topology\\.kubernetes\\.io/zone}'
```

Which will give you a more succinct output:

NAME	REGION	ZONE
aks-nodepool1-34917322-vmss000000	eastus	eastus-1
aks-nodepool1-34917322-vmss000001	eastus	eastus-2
aks-nodepool1-34917322-vmss000002	eastus	eastus-3

Verify pod distribution across zones

As documented in [Well-Known Labels, Annotations and Taints](#), Kubernetes uses the `failure-domain.beta.kubernetes.io/zone` label to automatically distribute pods in a replication controller or service across the different zones available. In order to test this, you can scale up your cluster from 3 to 5 nodes, to verify correct pod spreading:

```
az aks scale \
--resource-group myResourceGroup \
--name myAKSCluster \
--node-count 5
```

When the scale operation completes after a few minutes, the command

```
kubectl describe nodes | grep -e "Name:" -e "failure-domain.beta.kubernetes.io/zone"
```

 in a Bash shell should give an output similar to this sample:

Name:	aks-nodepool1-28993262-vmss000000
	failure-domain.beta.kubernetes.io/zone=eastus2-1
Name:	aks-nodepool1-28993262-vmss000001
	failure-domain.beta.kubernetes.io/zone=eastus2-2
Name:	aks-nodepool1-28993262-vmss000002
	failure-domain.beta.kubernetes.io/zone=eastus2-3
Name:	aks-nodepool1-28993262-vmss000003
	failure-domain.beta.kubernetes.io/zone=eastus2-1
Name:	aks-nodepool1-28993262-vmss000004
	failure-domain.beta.kubernetes.io/zone=eastus2-2

We now have two additional nodes in zones 1 and 2. You can deploy an application consisting of three replicas. We will use NGINX as an example:

```
kubectl create deployment nginx --image=mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
kubectl scale deployment nginx --replicas=3
```

By viewing nodes where your pods are running, you see pods are running on the nodes corresponding to three different availability zones. For example, with the command

```
kubectl describe pod | grep -e "Name:" -e "Node:"
```

 in a Bash shell you would get an output similar to this:

```
Name:      nginx-6db489d4b7-ktdwg
Node:      aks-nodepool1-28993262-vmss00000/10.240.0.4
Name:      nginx-6db489d4b7-v7zvj
Node:      aks-nodepool1-28993262-vmss00002/10.240.0.6
Name:      nginx-6db489d4b7-xz6wj
Node:      aks-nodepool1-28993262-vmss00004/10.240.0.8
```

As you can see from the previous output, the first pod is running on node 0, which is located in the availability zone `eastus2-1`. The second pod is running on node 2, which corresponds to `eastus2-3`, and the third one in node 4, in `eastus2-2`. Without any additional configuration, Kubernetes is spreading the pods correctly across all three availability zones.

Next steps

This article detailed how to create an AKS cluster that uses availability zones. For more considerations on highly available clusters, see [Best practices for business continuity and disaster recovery in AKS](#).

Create and manage multiple node pools for a cluster in Azure Kubernetes Service (AKS)

5/17/2021 • 26 minutes to read • [Edit Online](#)

In Azure Kubernetes Service (AKS), nodes of the same configuration are grouped together into *node pools*. These node pools contain the underlying VMs that run your applications. The initial number of nodes and their size (SKU) is defined when you create an AKS cluster, which creates a [system node pool](#). To support applications that have different compute or storage demands, you can create additional [user node pools](#). System node pools serve the primary purpose of hosting critical system pods such as CoreDNS and tunnelfront. User node pools serve the primary purpose of hosting your application pods. However, application pods can be scheduled on system node pools if you wish to only have one pool in your AKS cluster. User node pools are where you place your application-specific pods. For example, use these additional user node pools to provide GPUs for compute-intensive applications, or access to high-performance SSD storage.

NOTE

This feature enables higher control over how to create and manage multiple node pools. As a result, separate commands are required for create/update/delete. Previously cluster operations through `az aks create` or `az aks update` used the managedCluster API and were the only option to change your control plane and a single node pool. This feature exposes a separate operation set for agent pools through the agentPool API and require use of the `az aks nodepool` command set to execute operations on an individual node pool.

This article shows you how to create and manage multiple node pools in an AKS cluster.

Before you begin

You need the Azure CLI version 2.2.0 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Limitations

The following limitations apply when you create and manage AKS clusters that support multiple node pools:

- See [Quotas, virtual machine size restrictions, and region availability in Azure Kubernetes Service \(AKS\)](#).
- You can delete system node pools, provided you have another system node pool to take its place in the AKS cluster.
- System pools must contain at least one node, and user node pools may contain zero or more nodes.
- The AKS cluster must use the Standard SKU load balancer to use multiple node pools, the feature is not supported with Basic SKU load balancers.
- The AKS cluster must use virtual machine scale sets for the nodes.
- The name of a node pool may only contain lowercase alphanumeric characters and must begin with a lowercase letter. For Linux node pools the length must be between 1 and 12 characters, for Windows node pools the length must be between 1 and 6 characters.
- All node pools must reside in the same virtual network.
- When creating multiple node pools at cluster create time, all Kubernetes versions used by node pools must match the version set for the control plane. This can be updated after the cluster has been provisioned by using per node pool operations.

Create an AKS cluster

IMPORTANT

If you run a single system node pool for your AKS cluster in a production environment, we recommend you use at least three nodes for the node pool.

To get started, create an AKS cluster with a single node pool. The following example uses the [az group create](#) command to create a resource group named *myResourceGroup* in the *eastus* region. An AKS cluster named *myAKSCluster* is then created using the [az aks create](#) command.

NOTE

The *Basic* load balancer SKU is **not supported** when using multiple node pools. By default, AKS clusters are created with the *Standard* load balancer SKU from the Azure CLI and Azure portal.

```
# Create a resource group in East US
az group create --name myResourceGroup --location eastus

# Create a basic single-node AKS cluster
az aks create \
    --resource-group myResourceGroup \
    --name myAKSCluster \
    --vm-set-type VirtualMachineScaleSets \
    --node-count 2 \
    --generate-ssh-keys \
    --load-balancer-sku standard
```

It takes a few minutes to create the cluster.

NOTE

To ensure your cluster operates reliably, you should run at least 2 (two) nodes in the default node pool, as essential system services are running across this node pool.

When the cluster is ready, use the [az aks get-credentials](#) command to get the cluster credentials for use with `kubectl`:

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```

Add a node pool

The cluster created in the previous step has a single node pool. Let's add a second node pool using the [az aks nodepool add](#) command. The following example creates a node pool named *mynodepool* that runs 3 nodes:

```
az aks nodepool add \
    --resource-group myResourceGroup \
    --cluster-name myAKSCluster \
    --name mynodepool \
    --node-count 3
```

NOTE

The name of a node pool must start with a lowercase letter and can only contain alphanumeric characters. For Linux node pools the length must be between 1 and 12 characters, for Windows node pools the length must be between 1 and 6 characters.

To see the status of your node pools, use the [az aks node pool list](#) command and specify your resource group and cluster name:

```
az aks nodepool list --resource-group myResourceGroup --cluster-name myAKSCluster
```

The following example output shows that *mynodepool* has been successfully created with three nodes in the node pool. When the AKS cluster was created in the previous step, a default *nodepool1* was created with a node count of 2.

```
[  
 {  
 ...  
 "count": 3,  
 ...  
 "name": "mynodepool",  
 "orchestratorVersion": "1.15.7",  
 ...  
 "vmSize": "Standard_DS2_v2",  
 ...  
 },  
 {  
 ...  
 "count": 2,  
 ...  
 "name": "nodepool1",  
 "orchestratorVersion": "1.15.7",  
 ...  
 "vmSize": "Standard_DS2_v2",  
 ...  
 }  
 ]
```

TIP

If no *VmSize* is specified when you add a node pool, the default size is *Standard_D2s_v3* for Windows node pools and *Standard_DS2_v2* for Linux node pools. If no *OrchestratorVersion* is specified, it defaults to the same version as the control plane.

Add a node pool with a unique subnet (preview)

A workload may require splitting a cluster's nodes into separate pools for logical isolation. This isolation can be supported with separate subnets dedicated to each node pool in the cluster. This can address requirements such as having non-contiguous virtual network address space to split across node pools.

Limitations

- All subnets assigned to nodepools must belong to the same virtual network.
- System pods must have access to all nodes/pods in the cluster to provide critical functionality such as DNS resolution and tunneling kubectl logs/exec/port-forward proxy.
- If you expand your VNET after creating the cluster you must update your cluster (perform any managed cluster operation but node pool operations don't count) before adding a subnet outside the original cidr. AKS

will error out on the agent pool add now though we originally allowed it. If you don't know how to reconcile your cluster file a support ticket.

- Calico Network Policy is not supported.
- Azure Network Policy is not supported.
- Kube-proxy expects a single contiguous cidr and uses it this for three optimizations. See this [K.E.P](#). and --cluster-cidr [here](#) for details. In azure cnf your first node pool's subnet will be given to kube-proxy.

To create a node pool with a dedicated subnet, pass the subnet resource ID as an additional parameter when creating a node pool.

```
az aks nodepool add \
--resource-group myResourceGroup \
--cluster-name myAKSCluster \
--name mynodepool \
--node-count 3 \
--vnet-subnet-id <YOUR_SUBNET_RESOURCE_ID>
```

Upgrade a node pool

NOTE

Upgrade and scale operations on a cluster or node pool cannot occur simultaneously, if attempted an error is returned. Instead, each operation type must complete on the target resource prior to the next request on that same resource. Read more about this on our [troubleshooting guide](#).

The commands in this section explain how to upgrade a single specific node pool. The relationship between upgrading the Kubernetes version of the control plane and the node pool are explained in the [section below](#).

NOTE

The node pool OS image version is tied to the Kubernetes version of the cluster. You will only get OS image upgrades, following a cluster upgrade.

Since there are two node pools in this example, we must use [az aks nodepool upgrade](#) to upgrade a node pool. To see the available upgrades use [az aks get-upgrades](#)

```
az aks get-upgrades --resource-group myResourceGroup --name myAKSCluster
```

Let's upgrade the *mynodepool*. Use the [az aks nodepool upgrade](#) command to upgrade the node pool, as shown in the following example:

```
az aks nodepool upgrade \
--resource-group myResourceGroup \
--cluster-name myAKSCluster \
--name mynodepool \
--kubernetes-version KUBERNETES_VERSION \
--no-wait
```

List the status of your node pools again using the [az aks node pool list](#) command. The following example shows that *mynodepool* is in the *Upgrading* state to *KUBERNETES_VERSION*:

```
az aks nodepool list -g myResourceGroup --cluster-name myAKSCluster
```

```
[  
 {  
 ...  
 "count": 3,  
 ...  
 "name": "mynodepool",  
 "orchestratorVersion": "KUBERNETES_VERSION",  
 ...  
 "provisioningState": "Upgrading",  
 ...  
 "vmSize": "Standard_DS2_v2",  
 ...  
 },  
 {  
 ...  
 "count": 2,  
 ...  
 "name": "nodepool1",  
 "orchestratorVersion": "1.15.7",  
 ...  
 "provisioningState": "Succeeded",  
 ...  
 "vmSize": "Standard_DS2_v2",  
 ...  
 }  
]
```

It takes a few minutes to upgrade the nodes to the specified version.

As a best practice, you should upgrade all node pools in an AKS cluster to the same Kubernetes version. The default behavior of `az aks upgrade` is to upgrade all node pools together with the control plane to achieve this alignment. The ability to upgrade individual node pools lets you perform a rolling upgrade and schedule pods between node pools to maintain application uptime within the above constraints mentioned.

Upgrade a cluster control plane with multiple node pools

NOTE

Kubernetes uses the standard [Semantic Versioning](#) versioning scheme. The version number is expressed as $x.y.z$ where x is the major version, y is the minor version, and z is the patch version. For example, in version `1.12.6`, 1 is the major version, 12 is the minor version, and 6 is the patch version. The Kubernetes version of the control plane and the initial node pool are set during cluster creation. All additional node pools have their Kubernetes version set when they are added to the cluster. The Kubernetes versions may differ between node pools as well as between a node pool and the control plane.

An AKS cluster has two cluster resource objects with Kubernetes versions associated.

1. A cluster control plane Kubernetes version.
2. A node pool with a Kubernetes version.

A control plane maps to one or many node pools. The behavior of an upgrade operation depends on which Azure CLI command is used.

Upgrading an AKS control plane requires using `az aks upgrade`. This command upgrades the control plane version and all node pools in the cluster.

Issuing the `az aks upgrade` command with the `--control-plane-only` flag upgrades only the cluster control

plane. None of the associated node pools in the cluster are changed.

Upgrading individual node pools requires using `az aks nodepool upgrade`. This command upgrades only the target node pool with the specified Kubernetes version

Validation rules for upgrades

The valid Kubernetes upgrades for a cluster's control plane and node pools are validated by the following sets of rules.

- Rules for valid versions to upgrade node pools:
 - The node pool version must have the same *major* version as the control plane.
 - The node pool *minor* version must be within two *minor* versions of the control plane version.
 - The node pool version cannot be greater than the control `major.minor.patch` version.
- Rules for submitting an upgrade operation:
 - You cannot downgrade the control plane or a node pool Kubernetes version.
 - If a node pool Kubernetes version is not specified, behavior depends on the client being used.
Declaration in Resource Manager templates falls back to the existing version defined for the node pool if used, if none is set the control plane version is used to fall back on.
 - You can either upgrade or scale a control plane or a node pool at a given time, you cannot submit multiple operations on a single control plane or node pool resource simultaneously.

Scale a node pool manually

As your application workload demands change, you may need to scale the number of nodes in a node pool. The number of nodes can be scaled up or down.

To scale the number of nodes in a node pool, use the `az aks node pool scale` command. The following example scales the number of nodes in *mynodepool* to 5:

```
az aks nodepool scale \
    --resource-group myResourceGroup \
    --cluster-name myAKSCluster \
    --name mynodepool \
    --node-count 5 \
    --no-wait
```

List the status of your node pools again using the `az aks node pool list` command. The following example shows that *mynodepool* is in the *Scaling* state with a new count of 5 nodes:

```
az aks nodepool list -g myResourceGroup --cluster-name myAKSCluster
```

```
[  
  {  
    ...  
    "count": 5,  
    ...  
    "name": "mynodepool",  
    "orchestratorVersion": "1.15.7",  
    ...  
    "provisioningState": "Scaling",  
    ...  
    "vmSize": "Standard_DS2_v2",  
    ...  
  },  
  {  
    ...  
    "count": 2,  
    ...  
    "name": "nodepool1",  
    "orchestratorVersion": "1.15.7",  
    ...  
    "provisioningState": "Succeeded",  
    ...  
    "vmSize": "Standard_DS2_v2",  
    ...  
  }  
]
```

It takes a few minutes for the scale operation to complete.

Scale a specific node pool automatically by enabling the cluster autoscaler

AKS offers a separate feature to automatically scale node pools with a feature called the [cluster autoscaler](#). This feature can be enabled per node pool with unique minimum and maximum scale counts per node pool. Learn how to [use the cluster autoscaler per node pool](#).

Delete a node pool

If you no longer need a pool, you can delete it and remove the underlying VM nodes. To delete a node pool, use the [az aks node pool delete](#) command and specify the node pool name. The following example deletes the *mynodepool* created in the previous steps:

Caution

There are no recovery options for data loss that may occur when you delete a node pool. If pods can't be scheduled on other node pools, those applications are unavailable. Make sure you don't delete a node pool when in-use applications don't have data backups or the ability to run on other node pools in your cluster.

```
az aks nodepool delete -g myResourceGroup --cluster-name myAKSCluster --name mynodepool --no-wait
```

The following example output from the [az aks node pool list](#) command shows that *mynodepool* is in the *Deleting* state:

```
az aks nodepool list -g myResourceGroup --cluster-name myAKSCluster
```

```
[  
 {  
 ...  
 "count": 5,  
 ...  
 "name": "mynodepool",  
 "orchestratorVersion": "1.15.7",  
 ...  
 "provisioningState": "Deleting",  
 ...  
 "vmSize": "Standard_DS2_v2",  
 ...  
 },  
 {  
 ...  
 "count": 2,  
 ...  
 "name": "nodepool1",  
 "orchestratorVersion": "1.15.7",  
 ...  
 "provisioningState": "Succeeded",  
 ...  
 "vmSize": "Standard_DS2_v2",  
 ...  
 }  
]
```

It takes a few minutes to delete the nodes and the node pool.

Specify a VM size for a node pool

In the previous examples to create a node pool, a default VM size was used for the nodes created in the cluster. A more common scenario is for you to create node pools with different VM sizes and capabilities. For example, you may create a node pool that contains nodes with large amounts of CPU or memory, or a node pool that provides GPU support. In the next step, you [use taints and tolerations](#) to tell the Kubernetes scheduler how to limit access to pods that can run on these nodes.

In the following example, create a GPU-based node pool that uses the *Standard_NC6* VM size. These VMs are powered by the NVIDIA Tesla K80 card. For information on available VM sizes, see [Sizes for Linux virtual machines in Azure](#).

Create a node pool using the `az aks node pool add` command again. This time, specify the name *gpunodepool*, and use the `--node-vm-size` parameter to specify the *Standard_NC6* size:

```
az aks nodepool add \  
 --resource-group myResourceGroup \  
 --cluster-name myAKSCluster \  
 --name gpunodepool \  
 --node-count 1 \  
 --node-vm-size Standard_NC6 \  
 --no-wait
```

The following example output from the `az aks node pool list` command shows that *gpunodepool* is *Creating* nodes with the specified *VmSize*:

```
az aks nodepool list -g myResourceGroup --cluster-name myAKSCluster
```

```
[  
  {  
    ...  
    "count": 1,  
    ...  
    "name": "gpunodepool",  
    "orchestratorVersion": "1.15.7",  
    ...  
    "provisioningState": "Creating",  
    ...  
    "vmSize": "Standard_NC6",  
    ...  
  },  
  {  
    ...  
    "count": 2,  
    ...  
    "name": "nodepool1",  
    "orchestratorVersion": "1.15.7",  
    ...  
    "provisioningState": "Succeeded",  
    ...  
    "vmSize": "Standard_DS2_v2",  
    ...  
  }  
]
```

It takes a few minutes for the *gpunodepool* to be successfully created.

Specify a taint, label, or tag for a node pool

When creating a node pool, you can add taints, labels, or tags to that node pool. When you add a taint, label, or tag, all nodes within that node pool also get that taint, label, or tag.

IMPORTANT

Adding taints, labels, or tags to nodes should be done for the entire node pool using `az aks nodepool`. Applying taints, labels, or tags to individual nodes in a node pool using `kubectl` is not recommended.

Setting nodepool taints

To create a node pool with a taint, use `az aks nodepool add`. Specify the name *taintnp* and use the `--node-taints` parameter to specify `sku=gpu:NoSchedule` for the taint.

```
az aks nodepool add \  
  --resource-group myResourceGroup \  
  --cluster-name myAKSCluster \  
  --name taintnp \  
  --node-count 1 \  
  --node-taints sku=gpu:NoSchedule \  
  --no-wait
```

NOTE

A taint can only be set for node pools during node pool creation.

The following example output from the `az aks nodepool list` command shows that *taintnp* is *Creating* nodes with the specified *nodeTaints*:

```
$ az aks nodepool list -g myResourceGroup --cluster-name myAKSCluster

[
  {
    ...
    "count": 1,
    ...
    "name": "taintnp",
    "orchestratorVersion": "1.15.7",
    ...
    "provisioningState": "Creating",
    ...
    "nodeTaints": [
      "sku=gpu:NoSchedule"
    ],
    ...
  },
  ...
]
```

The taint information is visible in Kubernetes for handling scheduling rules for nodes. The Kubernetes scheduler can use taints and tolerations to restrict what workloads can run on nodes.

- A **taint** is applied to a node that indicates only specific pods can be scheduled on them.
- A **toleration** is then applied to a pod that allows them to *tolerate* a node's taint.

For more information on how to use advanced Kubernetes scheduled features, see [Best practices for advanced scheduler features in AKS](#)

In the previous step, you applied the *sku=gpu:NoSchedule* taint when you created your node pool. The following basic example YAML manifest uses a toleration to allow the Kubernetes scheduler to run an NGINX pod on a node in that node pool.

Create a file named `nginx-toleration.yaml` and copy in the following example YAML:

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - image: mcr.microsoft.com/oss/nginx/nginx:1.15.9-alpine
    name: mypod
    resources:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 1
        memory: 2G
  tolerations:
  - key: "sku"
    operator: "Equal"
    value: "gpu"
    effect: "NoSchedule"
```

Schedule the pod using the `kubectl apply -f nginx-toleration.yaml` command:

```
kubectl apply -f nginx-toleration.yaml
```

It takes a few seconds to schedule the pod and pull the NGINX image. Use the `kubectl describe pod` command to

view the pod status. The following condensed example output shows the `sku=gpu:NoSchedule` toleration is applied. In the events section, the scheduler has assigned the pod to the `aks-taintnp-28993262-vmss000000` node:

```
kubectl describe pod mypod
```

```
[...]
Tolerations:    node.kubernetes.io/not-ready:NoExecute for 300s
                  node.kubernetes.io/unreachable:NoExecute for 300s
                  sku=gpu:NoSchedule
Events:
  Type      Reason     Age   From           Message
  ----      ----     --   --   -----
  Normal    Scheduled  4m48s  default-scheduler  Successfully assigned default/mypod to aks-taintnp-28993262-
  vmss000000
  Normal    Pulling    4m47s  kubelet         pulling image "mcr.microsoft.com/oss/nginx/nginx:1.15.9-
  alpine"
  Normal    Pulled    4m43s  kubelet         Successfully pulled image
  "mcr.microsoft.com/oss/nginx/nginx:1.15.9-alpine"
  Normal    Created    4m40s  kubelet         Created container
  Normal    Started    4m40s  kubelet         Started container
```

Only pods that have this toleration applied can be scheduled on nodes in `taintnp`. Any other pod would be scheduled in the `nodepool1` node pool. If you create additional node pools, you can use additional taints and tolerations to limit what pods can be scheduled on those node resources.

Setting nodepool labels

You can also add labels to a node pool during node pool creation. Labels set at the node pool are added to each node in the node pool. These [labels are visible in Kubernetes](#) for handling scheduling rules for nodes.

To create a node pool with a label, use `az aks nodepool add`. Specify the name `labelnp` and use the `--labels` parameter to specify `dept=IT` and `costcenter=9999` for labels.

```
az aks nodepool add \
  --resource-group myResourceGroup \
  --cluster-name myAKSCluster \
  --name labelnp \
  --node-count 1 \
  --labels dept=IT costcenter=9999 \
  --no-wait
```

NOTE

Label can only be set for node pools during node pool creation. Labels must also be a key/value pair and have a [valid syntax](#).

The following example output from the `az aks nodepool list` command shows that `labelnp` is *Creating* nodes with the specified `nodeLabels`:

```
$ az aks nodepool list -g myResourceGroup --cluster-name myAKScluster

[
  {
    ...
    "count": 1,
    ...
    "name": "labelnp",
    "orchestratorVersion": "1.15.7",
    ...
    "provisioningState": "Creating",
    ...
    "nodeLabels": {
      "dept": "IT",
      "costcenter": "9999"
    },
    ...
  },
  ...
]
```

Setting nodepool Azure tags

You can apply an Azure tag to node pools in your AKS cluster. Tags applied to a node pool are applied to each node within the node pool and are persisted through upgrades. Tags are also applied to new nodes added to a node pool during scale-out operations. Adding a tag can help with tasks such as policy tracking or cost estimation.

Azure tags have keys which are case-insensitive for operations, such as when retrieving a tag by searching the key. In this case a tag with the given key will be updated or retrieved regardless of casing. Tag values are case-sensitive.

In AKS, if multiple tags are set with identical keys but different casing, the tag used is the first in alphabetical order. For example, `{"Key1": "val1", "kEy1": "val2", "key1": "val3"}` results in `Key1` and `val1` being set.

Create a node pool using the [az aks nodepool add](#). Specify the name `tagnodepool` and use the `--tag` parameter to specify `dept=IT` and `costcenter=9999` for tags.

```
az aks nodepool add \
  --resource-group myResourceGroup \
  --cluster-name myAKScluster \
  --name tagnodepool \
  --node-count 1 \
  --tags dept=IT costcenter=9999 \
  --no-wait
```

NOTE

You can also use the `--tags` parameter when using [az aks nodepool update](#) command as well as during cluster creation. During cluster creation, the `--tags` parameter applies the tag to the initial node pool created with the cluster. All tag names must adhere to the limitations in [Use tags to organize your Azure resources](#). Updating a node pool with the `--tags` parameter updates any existing tag values and appends any new tags. For example, if your node pool had `dept=IT` and `costcenter=9999` for tags and you updated it with `team=dev` and `costcenter=111` for tags, your nodepool would have `dept=IT`, `costcenter=111`, and `team=dev` for tags.

The following example output from the [az aks nodepool list](#) command shows that `tagnodepool` is *Creating* nodes with the specified `tag`.

```
az aks nodepool list -g myResourceGroup --cluster-name myAKSCluster
```

```
[  
 {  
 ...  
 "count": 1,  
 ...  
 "name": "tagnodepool",  
 "orchestratorVersion": "1.15.7",  
 ...  
 "provisioningState": "Creating",  
 ...  
 "tags": {  
 "dept": "IT",  
 "costcenter": "9999"  
 },  
 ...  
 },  
 ...  
 ]
```

Add a FIPS-enabled node pool (preview)

The Federal Information Processing Standard (FIPS) 140-2 is a US government standard that defines minimum security requirements for cryptographic modules in information technology products and systems. AKS allows you to create Linux-based node pools with FIPS 140-2 enabled. Deployments running on FIPS-enabled node pools can use those cryptographic modules to provide increased security and help meet security controls as part of FedRAMP compliance. For more details on FIPS 140-2, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

FIPS-enabled node pools are currently in preview.

IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

You will need the `aks-preview` Azure CLI extension version *0.5.11* or later. Install the `aks-preview` Azure CLI extension by using the `az extension add` command. Or install any available updates by using the `az extension update` command.

```
# Install the aks-preview extension  
az extension add --name aks-preview  
  
# Update the extension to make sure you have the latest version installed  
az extension update --name aks-preview
```

To use the feature, you must also enable the `FIPSPreview` feature flag on your subscription.

Register the `FIPSPreview` feature flag by using the `az feature register` command, as shown in the following example:

```
az feature register --namespace "Microsoft.ContainerService" --name "FIPSPreview"
```

It takes a few minutes for the status to show *Registered*. Verify the registration status by using the [az feature list](#) command:

```
az feature list -o table --query "[?contains(name, 'Microsoft.ContainerService/FIPSPreview')].{Name:name,State:properties.state}"
```

When ready, refresh the registration of the *Microsoft.ContainerService* resource provider by using the [az provider register](#) command:

```
az provider register --namespace Microsoft.ContainerService
```

FIPS-enabled node pools have the following limitations:

- Currently, you can only have FIPS-enabled Linux-based node pools running on Ubuntu 18.04.
- FIPS-enabled node pools require Kubernetes version 1.19 and greater.
- To update the underlying packages or modules used for FIPS, you must use [Node Image Upgrade](#).

IMPORTANT

The FIPS-enabled Linux image is a different image than the default Linux image used for Linux-based node pools. To enable FIPS on a node pool, you must create a new Linux-based node pool. You can't enable FIPS on existing node pools.

FIPS-enabled node images may have different version numbers, such as kernel version, than images that are not FIPS-enabled. Also, the update cycle for FIPS-enabled node pools and node images may differ from node pools and images that are not FIPS-enabled.

To create a FIPS-enabled node pool, use [az aks nodepool add](#) with the *--enable-fips-image* parameter when creating a node pool.

```
az aks nodepool add \
    --resource-group myResourceGroup \
    --cluster-name myAKSCluster \
    --name fipsnp \
    --enable-fips-image
```

NOTE

You can also use the *--enable-fips-image* parameter with [az aks create](#) when creating a cluster to enable FIPS on the default node pool. When adding node pools to a cluster created in this way, you still must use the *--enable-fips-image* parameter when adding node pools to create a FIPS-enabled node pool.

To verify your node pool is FIPS-enabled, use [az aks show](#) to check the *enableFIPS* value in *agentPoolProfiles*.

```
az aks show --resource-group myResourceGroup --cluster-name myAKSCluster --query="agentPoolProfiles[].{Name:name enableFips:enableFips}" -o table
```

The following example output shows the *fipsnp* node pool is FIPS-enabled and *nodepool1* is not.

Name	enableFips
<hr/>	
fipsnp	True
nodepool1	False

You can also verify deployments have access to the FIPS cryptographic libraries using `kubectl debug` on a node in the FIPS-enabled node pool. Use `kubectl get nodes` to list the nodes:

```
$ kubectl get nodes
NAME                      STATUS   ROLES      AGE     VERSION
aks-fipsnp-12345678-vmss000000  Ready    agent      6m4s    v1.19.9
aks-fipsnp-12345678-vmss000001  Ready    agent      5m21s   v1.19.9
aks-fipsnp-12345678-vmss000002  Ready    agent      6m8s    v1.19.9
aks-nodepool1-12345678-vmss000000  Ready    agent      34m    v1.19.9
```

In the above example, the nodes starting with `aks-fipsnp` are part of the FIPS-enabled node pool. Use `kubectl debug` to run a deployment with an interactive session on one of those nodes in the FIPS-enabled node pool.

```
kubectl debug node/aks-fipsnp-12345678-vmss000000 -it --image=mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11
```

From the interactive session, you can verify the FIPS cryptographic libraries are enabled:

```
root@aks-fipsnp-12345678-vmss000000:/# cat /proc/sys/crypto/fips_enabled
1
```

FIPS-enabled node pools also have a `kubernetes.azure.com/fips_enabled=true` label, which can be used by deployments to target those node pools.

Manage node pools using a Resource Manager template

When you use an Azure Resource Manager template to create and managed resources, you can typically update the settings in your template and redeploy to update the resource. With node pools in AKS, the initial node pool profile can't be updated once the AKS cluster has been created. This behavior means that you can't update an existing Resource Manager template, make a change to the node pools, and redeploy. Instead, you must create a separate Resource Manager template that updates only the node pools for an existing AKS cluster.

Create a template such as `aks-agentpools.json` and paste the following example manifest. This example template configures the following settings:

- Updates the `Linux` node pool named `myagentpool` to run three nodes.
- Sets the nodes in the node pool to run Kubernetes version `1.15.7`.
- Defines the node size as `Standard_DS2_v2`.

Edit these values as need to update, add, or delete node pools as needed:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "clusterName": {
            "type": "string",
            "metadata": {
                "description": "The name of your existing AKS cluster."
            }
        },
        "location": {
            "type": "string",
            "metadata": {
                "description": "The location of your existing AKS cluster."
            }
        },
        "agentPoolName": {
            "type": "string",
            "defaultValue": "myagentpool",
            "metadata": {
                "description": "The name of the agent pool to create or update."
            }
        },
        "vnetSubnetId": {
            "type": "string",
            "defaultValue": "",
            "metadata": {
                "description": "The Vnet subnet resource ID for your existing AKS cluster."
            }
        }
    },
    "variables": {
        "apiVersion": {
            "aks": "2020-01-01"
        },
        "agentPoolProfiles": {
            "maxPods": 30,
            "osDiskSizeGB": 0,
            "agentCount": 3,
            "agentVmSize": "Standard_DS2_v2",
            "osType": "Linux",
            "vnetSubnetId": "[parameters('vnetSubnetId')]"
        }
    },
    "resources": [
        {
            "apiVersion": "2020-01-01",
            "type": "Microsoft.ContainerService/managedClusters/agentPools",
            "name": "[concat(parameters('clusterName'), '/', parameters('agentPoolName'))]",
            "location": "[parameters('location')]",
            "properties": {
                "maxPods": "[variables('agentPoolProfiles').maxPods]",
                "osDiskSizeGB": "[variables('agentPoolProfiles').osDiskSizeGB]",
                "count": "[variables('agentPoolProfiles').agentCount]",
                "vmSize": "[variables('agentPoolProfiles').agentVmSize]",
                "osType": "[variables('agentPoolProfiles').osType]",
                "storageProfile": "ManagedDisks",
                "type": "VirtualMachineScaleSets",
                "vnetSubnetID": "[variables('agentPoolProfiles').vnetSubnetId]",
                "orchestratorVersion": "1.15.7"
            }
        }
    ]
}
```

Deploy this template using the [az deployment group create](#) command, as shown in the following example. You

are prompted for the existing AKS cluster name and location:

```
az deployment group create \
--resource-group myResourceGroup \
--template-file aks-agentpools.json
```

TIP

You can add a tag to your node pool by adding the *tag* property in the template, as shown in the following example.

```
...
"resources": [
{
  ...
  "properties": {
    ...
    "tags": {
      "name1": "val1"
    },
    ...
  }
}
...
}
```

It may take a few minutes to update your AKS cluster depending on the node pool settings and operations you define in your Resource Manager template.

Assign a public IP per node for your node pools

AKS nodes do not require their own public IP addresses for communication. However, scenarios may require nodes in a node pool to receive their own dedicated public IP addresses. A common scenario is for gaming workloads, where a console needs to make a direct connection to a cloud virtual machine to minimize hops. This scenario can be achieved on AKS by using Node Public IP.

First, create a new resource group.

```
az group create --name myResourceGroup2 --location eastus
```

Create a new AKS cluster and attach a public IP for your nodes. Each of the nodes in the node pool receives a unique public IP. You can verify this by looking at the Virtual Machine Scale Set instances.

```
az aks create -g MyResourceGroup2 -n MyManagedCluster -l eastus --enable-node-public-ip
```

For existing AKS clusters, you can also add a new node pool, and attach a public IP for your nodes.

```
az aks nodepool add -g MyResourceGroup2 --cluster-name MyManagedCluster -n nodepool2 --enable-node-public-ip
```

Use a public IP prefix

There are a number of [benefits to using a public IP prefix](#). AKS supports using addresses from an existing public IP prefix for your nodes by passing the resource ID with the flag `node-public-ip-prefix` when creating a new cluster or adding a node pool.

First, create a public IP prefix using [az network public-ip prefix create](#):

```
az network public-ip prefix create --length 28 --location eastus --name MyPublicIPPrefix --resource-group MyResourceGroup3
```

View the output, and take note of the `id` for the prefix:

```
{  
  ...  
  "id": "/subscriptions/<subscription-id>/resourceGroups/myResourceGroup3/providers/Microsoft.Network/publicIPPrefixes/MyPublicIPPrefix",  
  ...  
}
```

Finally, when creating a new cluster or adding a new node pool, use the flag `--node-public-ip-prefix` and pass in the prefix's resource ID:

```
az aks create -g MyResourceGroup3 -n MyManagedCluster -l eastus --enable-node-public-ip --node-public-ip-prefix /subscriptions/<subscription-id>/resourcegroups/MyResourceGroup3/providers/Microsoft.Network/publicIPPrefixes/MyPublicIPPrefix
```

Locate public IPs for nodes

You can locate the public IPs for your nodes in various ways:

- Use the Azure CLI command [az vmss list-instance-public-ips](#).
- Use [PowerShell or Bash commands](#).
- You can also view the public IPs in the Azure portal by viewing the instances in the Virtual Machine Scale Set.

IMPORTANT

The [node resource group](#) contains the nodes and their public IPs. Use the node resource group when executing commands to find the public IPs for your nodes.

```
az vmss list-instance-public-ips -g MC_MyResourceGroup2_MyManagedCluster_eastus -n YourVirtualMachineScaleSetName
```

Clean up resources

In this article, you created an AKS cluster that includes GPU-based nodes. To reduce unnecessary cost, you may want to delete the `gpunodepool`, or the whole AKS cluster.

To delete the GPU-based node pool, use the [az aks nodepool delete](#) command as shown in following example:

```
az aks nodepool delete -g myResourceGroup --cluster-name myAKScluster --name gpunodepool
```

To delete the cluster itself, use the [az group delete](#) command to delete the AKS resource group:

```
az group delete --name myResourceGroup --yes --no-wait
```

You can also delete the additional cluster you created for the public IP for node pools scenario.

```
az group delete --name myResourceGroup2 --yes --no-wait
```

Next steps

Learn more about [system node pools](#).

In this article, you learned how to create and manage multiple node pools in an AKS cluster. For more information about how to control pods across node pools, see [Best practices for advanced scheduler features in AKS](#).

To create and use Windows Server container node pools, see [Create a Windows Server container in AKS](#).

Use [proximity placement groups](#) to reduce latency for your AKS applications.

Add a spot node pool to an Azure Kubernetes Service (AKS) cluster

4/21/2021 • 6 minutes to read • [Edit Online](#)

A spot node pool is a node pool backed by a [spot virtual machine scale set](#). Using spot VMs for nodes with your AKS cluster allows you to take advantage of unutilized capacity in Azure at a significant cost savings. The amount of available unutilized capacity will vary based on many factors, including node size, region, and time of day.

When deploying a spot node pool, Azure will allocate the spot nodes if there's capacity available. But there's no SLA for the spot nodes. A spot scale set that backs the spot node pool is deployed in a single fault domain and offers no high availability guarantees. At any time when Azure needs the capacity back, the Azure infrastructure will evict spot nodes.

Spot nodes are great for workloads that can handle interruptions, early terminations, or evictions. For example, workloads such as batch processing jobs, development and testing environments, and large compute workloads may be good candidates to be scheduled on a spot node pool.

In this article, you add a secondary spot node pool to an existing Azure Kubernetes Service (AKS) cluster.

This article assumes a basic understanding of Kubernetes and Azure Load Balancer concepts. For more information, see [Kubernetes core concepts for Azure Kubernetes Service \(AKS\)](#).

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

Before you begin

When you create a cluster to use a spot node pool, that cluster must also use Virtual Machine Scale Sets for node pools and the *Standard* SKU load balancer. You must also add an additional node pool after you create your cluster to use a spot node pool. Adding an additional node pool is covered in a later step.

This article requires that you are running the Azure CLI version 2.14 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Limitations

The following limitations apply when you create and manage AKS clusters with a spot node pool:

- A spot node pool can't be the cluster's default node pool. A spot node pool can only be used for a secondary pool.
- You can't upgrade a spot node pool since spot node pools can't guarantee cordon and drain. You must replace your existing spot node pool with a new one to do operations such as upgrading the Kubernetes version. To replace a spot node pool, create a new spot node pool with a different version of Kubernetes, wait until its status is *Ready*, then remove the old node pool.
- The control plane and node pools cannot be upgraded at the same time. You must upgrade them separately or remove the spot node pool to upgrade the control plane and remaining node pools at the same time.
- A spot node pool must use Virtual Machine Scale Sets.
- You cannot change `ScaleSetPriority` or `SpotMaxPrice` after creation.
- When setting `SpotMaxPrice`, the value must be -1 or a positive value with up to five decimal places.
- A spot node pool will have the label `kubernetes.azure.com/scalesetpriority:spot`, the taint `kubernetes.azure.com/scalesetpriority=spot:NoSchedule`, and system pods will have anti-affinity.

- You must add a [corresponding toleration](#) to schedule workloads on a spot node pool.

Add a spot node pool to an AKS cluster

You must add a spot node pool to an existing cluster that has multiple node pools enabled. More details on creating an AKS cluster with multiple node pools are available [here](#).

Create a node pool using the [az aks nodepool add](#).

```
az aks nodepool add \
    --resource-group myResourceGroup \
    --cluster-name myAKSCluster \
    --name spotnodepool \
    --priority Spot \
    --eviction-policy Delete \
    --spot-max-price -1 \
    --enable-cluster-autoscaler \
    --min-count 1 \
    --max-count 3 \
    --no-wait
```

By default, you create a node pool with a *priority* of *Regular* in your AKS cluster when you create a cluster with multiple node pools. The above command adds an auxiliary node pool to an existing AKS cluster with a *priority* of *Spot*. The *priority* of *Spot* makes the node pool a spot node pool. The *eviction-policy* parameter is set to *Delete* in the above example, which is the default value. When you set the [eviction policy](#) to *Delete*, nodes in the underlying scale set of the node pool are deleted when they're evicted. You can also set the eviction policy to *Deallocate*. When you set the eviction policy to *Deallocate*, nodes in the underlying scale set are set to the stopped-deallocated state upon eviction. Nodes in the stopped-deallocated state count against your compute quota and can cause issues with cluster scaling or upgrading. The *priority* and *eviction-policy* values can only be set during node pool creation. Those values can't be updated later.

The command also enables the [cluster autoscaler](#), which is recommended to use with spot node pools. Based on the workloads running in your cluster, the cluster autoscaler scales up and scales down the number of nodes in the node pool. For spot node pools, the cluster autoscaler will scale up the number of nodes after an eviction if additional nodes are still needed. If you change the maximum number of nodes a node pool can have, you also need to adjust the `maxCount` value associated with the cluster autoscaler. If you do not use a cluster autoscaler, upon eviction, the spot pool will eventually decrease to zero and require a manual operation to receive any additional spot nodes.

IMPORTANT

Only schedule workloads on spot node pools that can handle interruptions, such as batch processing jobs and testing environments. It is recommended that you set up [taints and tolerations](#) on your spot node pool to ensure that only workloads that can handle node evictions are scheduled on a spot node pool. For example, the above command by default adds a taint of `kubernetes.azure.com/scalesetpriority=spot:NoSchedule` so only pods with a corresponding toleration are scheduled on this node.

Verify the spot node pool

To verify your node pool has been added as a spot node pool:

```
az aks nodepool show --resource-group myResourceGroup --cluster-name myAKSCluster --name spotnodepool
```

Confirm `scaleSetPriority` is *Spot*.

To schedule a pod to run on a spot node, add a toleration that corresponds to the taint applied to your spot node. The following example shows a portion of a yaml file that defines a toleration that corresponds to a `kubernetes.azure.com/scalesetpriority=spot:NoSchedule` taint used in the previous step.

```
spec:  
  containers:  
    - name: spot-example  
  tolerations:  
    - key: "kubernetes.azure.com/scalesetpriority"  
      operator: "Equal"  
      value: "spot"  
      effect: "NoSchedule"  
    ...
```

When a pod with this toleration is deployed, Kubernetes can successfully schedule the pod on the nodes with the taint applied.

Max price for a spot pool

[Pricing for spot instances is variable](#), based on region and SKU. For more information, see pricing for [Linux](#) and [Windows](#).

With variable pricing, you have option to set a max price, in US dollars (USD), using up to 5 decimal places. For example, the value `0.98765` would be a max price of \$0.98765 USD per hour. If you set the max price to `-1`, the instance won't be evicted based on price. The price for the instance will be the current price for Spot or the price for a standard instance, whichever is less, as long as there is capacity and quota available.

Next steps

In this article, you learned how to add a spot node pool to an AKS cluster. For more information about how to control pods across node pools, see [Best practices for advanced scheduler features in AKS](#).

Manage system node pools in Azure Kubernetes Service (AKS)

4/21/2021 • 7 minutes to read • [Edit Online](#)

In Azure Kubernetes Service (AKS), nodes of the same configuration are grouped together into *node pools*. Node pools contain the underlying VMs that run your applications. System node pools and user node pools are two different node pool modes for your AKS clusters. System node pools serve the primary purpose of hosting critical system pods such as `CoreDNS` and `metrics-server`. User node pools serve the primary purpose of hosting your application pods. However, application pods can be scheduled on system node pools if you wish to only have one pool in your AKS cluster. Every AKS cluster must contain at least one system node pool with at least one node.

IMPORTANT

If you run a single system node pool for your AKS cluster in a production environment, we recommend you use at least three nodes for the node pool.

Before you begin

- You need the Azure CLI version 2.3.1 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Limitations

The following limitations apply when you create and manage AKS clusters that support system node pools.

- See [Quotas, virtual machine size restrictions, and region availability in Azure Kubernetes Service \(AKS\)](#).
- The AKS cluster must be built with virtual machine scale sets as the VM type and the *Standard* SKU load balancer.
- The name of a node pool may only contain lowercase alphanumeric characters and must begin with a lowercase letter. For Linux node pools, the length must be between 1 and 12 characters. For Windows node pools, the length must be between 1 and 6 characters.
- An API version of 2020-03-01 or greater must be used to set a node pool mode. Clusters created on API versions older than 2020-03-01 contain only user node pools, but can be migrated to contain system node pools by following [update pool mode steps](#).
- The mode of a node pool is a required property and must be explicitly set when using ARM templates or direct API calls.

System and user node pools

For a system node pool, AKS automatically assigns the label `kubernetes.azure.com/mode: system` to its nodes. This causes AKS to prefer scheduling system pods on node pools that contain this label. This label does not prevent you from scheduling application pods on system node pools. However, we recommend you isolate critical system pods from your application pods to prevent misconfigured or rogue application pods from accidentally killing system pods. You can enforce this behavior by creating a dedicated system node pool. Use the `CriticalAddonsOnly=true:NoSchedule` taint to prevent application pods from being scheduled on system node pools.

System node pools have the following restrictions:

- System pools osType must be Linux.
- User node pools osType may be Linux or Windows.
- System pools must contain at least one node, and user node pools may contain zero or more nodes.
- System node pools require a VM SKU of at least 2 vCPUs and 4GB memory. But burstable-VM(B series) is not recommended.
- A minimum of two nodes 4 vCPUs is recommended(e.g. Standard_DS4_v2), especially for large clusters (Multiple CoreDNS Pod replicas, 3-4+ add-ons, etc.).
- System node pools must support at least 30 pods as described by the [minimum and maximum value formula for pods](#).
- Spot node pools require user node pools.
- Adding an additional system node pool or changing which node pool is a system node pool will *NOT* automatically move system pods. System pods can continue to run on the same node pool even if you change it to a user node pool. If you delete or scale down a node pool running system pods that was previously a system node pool, those system pods are redeployed with preferred scheduling to the new system node pool.

You can do the following operations with node pools:

- Create a dedicated system node pool (prefer scheduling of system pods to node pools of `mode:system`)
- Change a system node pool to be a user node pool, provided you have another system node pool to take its place in the AKS cluster.
- Change a user node pool to be a system node pool.
- Delete user node pools.
- You can delete system node pools, provided you have another system node pool to take its place in the AKS cluster.
- An AKS cluster may have multiple system node pools and requires at least one system node pool.
- If you want to change various immutable settings on existing node pools, you can create new node pools to replace them. One example is to add a new node pool with a new maxPods setting and delete the old node pool.

Create a new AKS cluster with a system node pool

When you create a new AKS cluster, you automatically create a system node pool with a single node. The initial node pool defaults to a mode of type system. When you create new node pools with `az aks nodepool add`, those node pools are user node pools unless you explicitly specify the mode parameter.

The following example creates a resource group named *myResourceGroup* in the *eastus* region.

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

Use the `az aks create` command to create an AKS cluster. The following example creates a cluster named *myAKSCluster* with one dedicated system pool containing one node. For your production workloads, ensure you are using system node pools with at least three nodes. This operation may take several minutes to complete.

```
# Create a new AKS cluster with a single system pool
az aks create -g myResourceGroup --name myAKSCluster --node-count 1 --generate-ssh-keys
```

Add a dedicated system node pool to an existing AKS cluster

IMPORTANT

You can't change node taints through the CLI after the node pool is created.

You can add one or more system node pools to existing AKS clusters. It's recommended to schedule your application pods on user node pools, and dedicate system node pools to only critical system pods. This prevents rogue application pods from accidentally killing system pods. Enforce this behavior with the `CriticalAddonsOnly=true:NoSchedule` taint for your system node pools.

The following command adds a dedicated node pool of mode type system with a default count of three nodes.

```
az aks nodepool add \
    --resource-group myResourceGroup \
    --cluster-name myAKSCluster \
    --name systempool \
    --node-count 3 \
    --node-taints CriticalAddonsOnly=true:NoSchedule \
    --mode System
```

Show details for your node pool

You can check the details of your node pool with the following command.

```
az aks nodepool show -g myResourceGroup --cluster-name myAKSCluster -n systempool
```

A mode of type **System** is defined for system node pools, and a mode of type **User** is defined for user node pools. For a system pool, verify the taint is set to `CriticalAddonsOnly=true:NoSchedule`, which will prevent application pods from being scheduled on this node pool.

```
{  
  "agentPoolType": "VirtualMachineScaleSets",  
  "availabilityZones": null,  
  "count": 1,  
  "enableAutoScaling": null,  
  "enableNodePublicIp": false,  
  "id":  
    "/subscriptions/yourSubscriptionId/resourcegroups/myResourceGroup/providers/Microsoft.ContainerService/managedClusters/myAKSCluster/agentPools/systempool",  
  "maxCount": null,  
  "maxPods": 110,  
  "minCount": null,  
  "mode": "System",  
  "name": "systempool",  
  "nodeImageVersion": "AKSUbuntu-1604-2020.06.30",  
  "nodeLabels": {},  
  "nodeTaints": [  
    "CriticalAddonsOnly=true:NoSchedule"  
,  
  ],  
  "orchestratorVersion": "1.16.10",  
  "osDiskSizeGb": 128,  
  "osType": "Linux",  
  "provisioningState": "Failed",  
  "proximityPlacementGroupId": null,  
  "resourceGroup": "myResourceGroup",  
  "scaleSetEvictionPolicy": null,  
  "scaleSetPriority": null,  
  "spotMaxPrice": null,  
  "tags": null,  
  "type": "Microsoft.ContainerService/managedClusters/agentPools",  
  "upgradeSettings": {  
    "maxSurge": null  
,  
  },  
  "vmSize": "Standard_DS2_v2",  
  "vnetSubnetId": null  
}  
}
```

Update existing cluster system and user node pools

NOTE

An API version of 2020-03-01 or greater must be used to set a system node pool mode. Clusters created on API versions older than 2020-03-01 contain only user node pools as a result. To receive system node pool functionality and benefits on older clusters, update the mode of existing node pools with the following commands on the latest Azure CLI version.

You can change modes for both system and user node pools. You can change a system node pool to a user pool only if another system node pool already exists on the AKS cluster.

This command changes a system node pool to a user node pool.

```
az aks nodepool update -g myResourceGroup --cluster-name myAKSCluster -n mynodepool --mode user
```

This command changes a user node pool to a system node pool.

```
az aks nodepool update -g myResourceGroup --cluster-name myAKSCluster -n mynodepool --mode system
```

Delete a system node pool

NOTE

To use system node pools on AKS clusters before API version 2020-03-02, add a new system node pool, then delete the original default node pool.

You must have at least two system node pools on your AKS cluster before you can delete one of them.

```
az aks nodepool delete -g myResourceGroup --cluster-name myAKSCluster -n mynodepool
```

Clean up resources

To delete the cluster, use the [az group delete](#) command to delete the AKS resource group:

```
az group delete --name myResourceGroup --yes --no-wait
```

Next steps

In this article, you learned how to create and manage system node pools in an AKS cluster. For more information about how to use multiple node pools, see [use multiple node pools](#).

Access Kubernetes resources from the Azure portal

3/5/2021 • 4 minutes to read • [Edit Online](#)

The Azure portal includes a Kubernetes resource view for easy access to the Kubernetes resources in your Azure Kubernetes Service (AKS) cluster. Viewing Kubernetes resources from the Azure portal reduces context switching between the Azure portal and the `kubectl` command-line tool, streamlining the experience for viewing and editing your Kubernetes resources. The resource viewer currently includes multiple resource types, such as deployments, pods, and replica sets.

The Kubernetes resource view from the Azure portal replaces the [AKS dashboard add-on](#), which is deprecated.

Prerequisites

To view Kubernetes resources in the Azure portal, you need an AKS cluster. Any cluster is supported, but if using Azure Active Directory (Azure AD) integration, your cluster must use [AKS-managed Azure AD integration](#). If your cluster uses legacy Azure AD, you can upgrade your cluster in the portal or with the [Azure CLI](#). You can also [use the Azure portal](#) to create a new AKS cluster.

View Kubernetes resources

To see the Kubernetes resources, navigate to your AKS cluster in the Azure portal. The navigation pane on the left is used to access your resources. The resources include:

- **Namespaces** displays the namespaces of your cluster. The filter at the top of the namespace list provides a quick way to filter and display your namespace resources.
- **Workloads** shows information about deployments, pods, replica sets, stateful sets, daemon sets, jobs, and cron jobs deployed to your cluster. The screenshot below shows the default system pods in an example AKS cluster.
- **Services and ingresses** shows all of your cluster's service and ingress resources.
- **Storage** shows your Azure storage classes and persistent volume information.
- **Configuration** shows your cluster's config maps and secrets.

Name	Namespace	Ready	Up-to-date	Available	Age
coredns-autoscaler	kube-system	✓ 1/1	1	1	49 minutes
coredns	kube-system	✓ 2/2	2	2	49 minutes
metrics-server	kube-system	✓ 1/1	1	1	49 minutes
omsagent-ss	kube-system	✓ 1/1	1	1	49 minutes
tunnelfront	kube-system	✓ 1/1	1	1	49 minutes

Deploy an application

In this example, we'll use our sample AKS cluster to deploy the Azure Vote application from the [AKS quickstart](#).

1. Select **Add** from any of the resource views (Namespace, Workloads, Services and ingresses, Storage, or Configuration).
2. Paste the YAML for the Azure Vote application from the [AKS quickstart](#).
3. Select **Add** at the bottom of the YAML editor to deploy the application.

Once the YAML file is added, the resource viewer shows both Kubernetes services that were created: the internal service (azure-vote-back), and the external service (azure-vote-front) to access the Azure Vote application. The external service includes a linked external IP address so you can easily view the application in your browser.

Name	Namespace	Status	Type	Cluster IP	External IP	Ports	Age
kubernetes	default	Ok	ClusterIP	10.0.0.1		443/TCP	55 minutes
healthmodel-replicaset-service	kube-system	Ok	ClusterIP	10.0.226.252		25227/TCP	55 minutes
kube-dns	kube-system	Ok	ClusterIP	10.0.0.10		53/UDP,53/TCP	55 minutes
metrics-server	kube-system	Ok	ClusterIP	10.0.75.143		443/TCP	55 minutes
azure-vote-back	default	Ok	ClusterIP	10.0.223.33		6379/TCP	8 seconds
azure-vote-front	default	Ok	LoadBalancer	10.0.57.52	52.154.169.60	80:30864/TCP	7 seconds

Monitor deployment insights

AKS clusters with [Azure Monitor for containers](#) enabled can quickly view deployment and other insights. From the Kubernetes resources view, users can see the live status of individual deployments, including CPU and memory usage, as well as transition to Azure monitor for more in-depth information about specific nodes and containers. Here's an example of deployment insights from a sample AKS cluster:

Name	Namespace	Ready	Up-To-Date	Available	Age
azure-vote-back	default	1/1	1	1	12 minutes
azure-vote-front	default	1/1	1	1	12 minutes
coredns	kube-system	2/2	2	2	an hour
coredns-autoscaler	kube-system	1/1	1	1	an hour
metrics-server	kube-system	1/1	1	1	an hour
omsagent-rs	kube-system	1/1	1	1	an hour
tunnelfront	kube-system	1/1	1	1	an hour

Edit YAML

The Kubernetes resource view also includes a YAML editor. A built-in YAML editor means you can update or create services and deployments from within the portal and apply changes immediately.

azure-vote-front | YAML

Service

The screenshot shows the Azure Portal interface for managing a Kubernetes service named 'azure-vote-front'. The 'YAML' tab is selected, displaying the following YAML code:

```
1 kind: Service
2 apiVersion: v1
3 metadata:
4   name: azure-vote-front
5   namespace: default
6   selfLink: /api/v1/namespaces/default/services/azure-vote-front
7   uid:
8   resourceVersion: '857494'
9   creationTimestamp: '2020-08-04T21:27:12Z'
10  finalizers:
11    - service.kubernetes.io/load-balancer-cleanup
12  managedFields:
13    - manager: Mozilla
14      operation: Update
15      apiVersion: v1
16      time: '2020-08-04T21:27:12Z'
17      fieldsType: FieldsV1
18      fieldsV1:
19        'f:spec':
20          'f:externalTrafficPolicy': {}
21        'f:ports':
22          .: {}
23          'k:{"port":80,"protocol":"TCP"}':
24            .: {}
25            'f:port': {}
26            'f:protocol': {}
```

Below the code editor are two buttons: 'Review + save' and 'Discard'.

After editing the YAML, changes are applied by selecting **Review + save**, confirming the changes, and then saving again.

WARNING

Performing direct production changes via UI or CLI is not recommended, you should leverage [continuous integration \(CI\) and continuous deployment \(CD\) best practices](#). The Azure Portal Kubernetes management capabilities and the YAML editor are built for learning and flighting new deployments in a development and testing setting.

Troubleshooting

This section addresses common problems and troubleshooting steps.

Unauthorized access

To access the Kubernetes resources, you must have access to the AKS cluster, the Kubernetes API, and the Kubernetes objects. Ensure that you're either a cluster administrator or a user with the appropriate permissions to access the AKS cluster. For more information on cluster security, see [Access and identity options for AKS](#).

NOTE

The kubernetes resource view in the Azure Portal is only supported by [managed-AAD enabled clusters](#) or non-AAD enabled clusters. If you are using a managed-AAD enabled cluster, your AAD user or identity needs to have the respective roles/role bindings to access the kubernetes API, in addition to the permission to pull the user `kubeconfig`.

Enable resource view

For existing clusters, you may need to enable the Kubernetes resource view. To enable the resource view, follow the prompts in the portal for your cluster.

TIP

The AKS feature for [API server authorized IP ranges](#) can be added to limit API server access to only the firewall's public endpoint. Another option for such clusters is updating `--api-server-authorized-ip-ranges` to include access for a local client computer or IP address range (from which portal is being browsed). To allow this access, you need the computer's public IPv4 address. You can find this address with below command or by searching "what is my IP address" in an internet browser.

```
# Retrieve your IP address
CURRENT_IP=$(dig @resolver1.opendns.com ANY myip.opendns.com +short)

# Add to AKS approved list
az aks update -g $RG -n $AKSNAME --api-server-authorized-ip-ranges $CURRENT_IP/32
```

Next steps

This article showed you how to access Kubernetes resources for your AKS cluster. See [Deployments and YAML manifests](#) for a deeper understanding of cluster resources and the YAML files that are accessed with the Kubernetes resource viewer.

Access the Kubernetes web dashboard in Azure Kubernetes Service (AKS)

4/21/2021 • 5 minutes to read • [Edit Online](#)

Kubernetes includes a web dashboard that can be used for basic management operations. This dashboard lets you view basic health status and metrics for your applications, create and deploy services, and edit existing applications. This article shows you how to access the Kubernetes dashboard using the Azure CLI, then guides you through some basic dashboard operations.

For more information on the Kubernetes dashboard, see [Kubernetes Web UI Dashboard](#). AKS uses version 2.0 and greater of the open-source dashboard.

WARNING

The AKS dashboard add-on is set for deprecation. Use the [Kubernetes resource view in the Azure portal \(preview\)](#) instead.

- The Kubernetes dashboard is enabled by default for clusters running a Kubernetes version less than 1.18.
- The dashboard add-on will be disabled by default for all new clusters created on Kubernetes 1.18 or greater.
- Starting with Kubernetes 1.19 in preview, AKS will no longer support installation of the managed kube-dashboard addon.
- Existing clusters with the add-on enabled will not be impacted. Users will continue to be able to manually install the open-source dashboard as user-installed software.

Before you begin

The steps detailed in this document assume that you've created an AKS cluster and have established a `kubectl` connection with the cluster. If you need to create an AKS cluster, see [Quickstart: Deploy an Azure Kubernetes Service cluster using the Azure CLI](#).

You also need the Azure CLI version 2.6.0 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Disable the Kubernetes dashboard

The kube-dashboard addon is **enabled by default on clusters older than K8s 1.18**. The addon can be disabled by running the following command.

```
az aks disable-addons -g myRG -n myAKScluster -a kube-dashboard
```

Start the Kubernetes dashboard

WARNING

The AKS dashboard add-on is deprecated for versions 1.19+. Please use the [Kubernetes resource view in the Azure portal \(preview\)](#) instead.

- The following command will now open the Azure Portal resource view instead of the kubernetes dashboard for versions 1.19 and above.

To start the Kubernetes dashboard on a cluster, use the `az aks browse` command. This command requires the installation of the kube-dashboard addon on the cluster, which is included by default on clusters running any version older than Kubernetes 1.18.

The following example opens the dashboard for the cluster named *myAKSCluster* in the resource group named *myResourceGroup*.

```
az aks browse --resource-group myResourceGroup --name myAKSCluster
```

This command creates a proxy between your development system and the Kubernetes API, and opens a web browser to the Kubernetes dashboard. If a web browser doesn't open to the Kubernetes dashboard, copy and paste the URL address noted in the Azure CLI, typically `http://127.0.0.1:8001`.

NOTE

If you do not see the dashboard at `http://127.0.0.1:8001` you can manually route to the following addresses. Clusters on 1.16 or greater use https and require a separate endpoint.

- K8s 1.16 or greater:

```
http://127.0.0.1:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy
```

- K8s 1.15 and below:

```
http://127.0.0.1:8001/api/v1/namespaces/kube-system/services/kubernetes-dashboard:/proxy
```

Sign in to the dashboard (kubernetes 1.16+)

IMPORTANT

As of [v1.10.1 of the Kubernetes dashboard](#) or kubernetes v1.16+ the service account "kubernetes-dashboard" can no longer be used to retrieve resources due to a [security fix in that release](#). As a result, requests without auth info return a [401 unauthorized error](#). A bearer token retrieved from a service account can still be used as in this [Kubernetes Dashboard example](#), but this impacts the login flow of the dashboard add-on compared to older versions.

If you still run a version prior to 1.16 you can still give permissions to the "kubernetes-dashboard" service account, but this is **not recommended**:

```
kubectl create clusterrolebinding kubernetes-dashboard --clusterrole=cluster-admin --serviceaccount=kube-system:kubernetes-dashboard
```

The initial screen presented requires a kubeconfig or token. Both options require resource permissions to display those resources in the dashboard.

Kubernetes Dashboard

Kubeconfig

Please select the kubeconfig file that you have created to configure access to the cluster. To find out more about how to configure and use kubeconfig file, please refer to the [Configure Access to Multiple Clusters](#) section.

Token

Every Service Account has a Secret with valid Bearer Token that can be used to log in to Dashboard. To find out more about how to configure and use Bearer Tokens, please refer to the [Authentication](#) section.

Choose kubeconfig file

...

[Sign in](#)

Use a kubeconfig

For both Azure AD enabled and non-Azure AD enabled clusters, a kubeconfig can be passed in. Ensure access tokens are valid, if your tokens are expired you can refresh tokens via kubectl.

1. Set the admin kubeconfig with `az aks get-credentials -a --resource-group <RG_NAME> --name <CLUSTER_NAME>`
2. Select `Kubeconfig` and click `Choose kubeconfig file` to open file selector
3. Select your kubeconfig file (defaults to \$HOME/.kube/config)
4. Click `Sign In`

Use a token

1. For **non-Azure AD enabled cluster**, run `kubectl config view` and copy the token associated with the user account of your cluster.
2. Paste into the token option at sign in.
3. Click `Sign In`

For Azure AD enabled clusters, retrieve your AAD token with the following command. Validate you've replaced the resource group and cluster name in the command.

```
## Update <RESOURCE_GROUP> and <AKS_NAME> with your input.  
  
kubectl config view -o jsonpath='{.users[?(@.name == "clusterUser_<RESOURCE_GROUP>_<AKS_NAME>")].user.auth-provider.config.access-token}'
```

Once successful, a page similar to the below will be displayed.



Create an application

The following steps require the user to have permissions to the respective resources.

To see how the Kubernetes dashboard can reduce the complexity of management tasks, let's create an application. You can create an application from the Kubernetes dashboard by providing text input, a YAML file, or through a graphical wizard.

To create an application, complete the following steps:

1. Select the **Create** button in the upper right window.
2. To use the graphical wizard, choose to **Create an app**.
3. Provide a name for the deployment, such as *nginx*
4. Enter the name for the container image to use, such as *nginx:1.15.5*
5. To expose port 80 for web traffic, you create a Kubernetes service. Under **Service**, select **External**, then enter **80** for both the port and target port.
6. When ready, select **Deploy** to create the app.

Cluster

- Namespaces
- Nodes
- Persistent Volumes
- Roles
- Storage Classes

Namespace

- default

Overview

Workloads

- Cron Jobs
- Daemon Sets
- Deployments
- Jobs
- Pods
- Replica Sets
- Replication Controllers
- Stateful Sets

Discovery and Load Balancing

CREATE FROM TEXT INPUT **CREATE FROM FILE** **CREATE AN APP**

App name * **nginx** 5 / 24

Container image * **nginx:1.15.5**

Number of pods * **1**

Service * **External**

Port *	Target port *	Protocol *
80	80	TCP

Port Target port Protocol *

SHOW ADVANCED OPTIONS

DEPLOY **CANCEL**

It takes a minute or two for a public external IP address to be assigned to the Kubernetes service. On the left-hand side, under **Discovery and Load Balancing** select **Services**. Your application's service is listed, including the *External endpoints*, as shown in the following example:

Discovery and load balancing > Services

Overview

Workloads

- Cron Jobs
- Daemon Sets
- Deployments
- Jobs
- Pods
- Replica Sets
- Replication Controllers
- Stateful Sets

Discovery and Load Balancing

Ingresses

Services

Services

Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
nginx	k8s-app: nginx	10.0.180.34	nginx:80 TCP nginx:30344 TCP	23.96.99.120:80	41 minutes
kubernetes	component: ap... provider: kuber...	10.0.0.1	kubernetes:443 T kubernetes:0 TCP	-	7 days

Select the endpoint address to open a web browser window to the default NGINX page:



View pod information

The Kubernetes dashboard can provide basic monitoring metrics and troubleshooting information such as logs.

To see more information about your application pods, select **Pods** in the left-hand menu. The list of available pods is shown. Choose your *nginx* pod to view information, such as resource consumption:



Edit the application

In addition to creating and viewing applications, the Kubernetes dashboard can be used to edit and update application deployments. To provide additional redundancy for the application, let's increase the number of NGINX replicas.

To edit a deployment:

1. Select **Deployments** in the left-hand menu, and then choose your *nginx* deployment.
2. Select **Edit** in the upper right-hand navigation bar.
3. Locate the `spec.replicas` value, at around line 20. To increase the number of replicas for the application, change this value from `1` to `3`.
4. Select **Update** when ready.

```

15    "annotations": {
16      "deployment.kubernetes.io/revision": "1"
17    }
18  },
19  "spec": {
20    "replicas": 3,
21    "selector": {
22      "matchLabels": {
23        "k8s-app": "nginx"
24      }
25    },
26    "template": {
27      "metadata": {
28        "name": "nginx",
29        "creationTimestamp": null,
30        "labels": {

```

CANCEL COPY UPDATE

It takes a few moments for the new pods to be created inside a replica set. On the left-hand menu, choose **Replica Sets**, and then choose your *nginx* replica set. The list of pods now reflects the updated replica count, as shown in the following example output:

Name	Node	Status	Restarts	Age	CPU (cores)	Memory (bytes)
nginx-f597dd9c-aks-nodepool1-79590246-0		Running	0	56 minutes	0	2.098 Mi
nginx-f597dd9c-aks-nodepool1-79590246-0		Running	0	10 seconds	-	2.059 Mi
nginx-f597dd9c-aks-nodepool1-79590246-0		Running	0	10 seconds	-	1.980 Mi

Next steps

For more information about the Kubernetes dashboard, see the [Kubernetes Web UI Dashboard](#).

Dynamically create and use a persistent volume with Azure disks in Azure Kubernetes Service (AKS)

4/21/2021 • 7 minutes to read • [Edit Online](#)

A persistent volume represents a piece of storage that has been provisioned for use with Kubernetes pods. A persistent volume can be used by one or many pods, and can be dynamically or statically provisioned. This article shows you how to dynamically create persistent volumes with Azure disks for use by a single pod in an Azure Kubernetes Service (AKS) cluster.

NOTE

An Azure disk can only be mounted with *Access mode* type *ReadWriteOnce*, which makes it available to one node in AKS. If you need to share a persistent volume across multiple nodes, use [Azure Files](#).

For more information on Kubernetes volumes, see [Storage options for applications in AKS](#).

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart using the [Azure CLI](#) or [using the Azure portal](#).

You also need the Azure CLI version 2.0.59 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Built-in storage classes

A storage class is used to define how a unit of storage is dynamically created with a persistent volume. For more information on Kubernetes storage classes, see [Kubernetes Storage Classes](#).

Each AKS cluster includes four pre-created storage classes, two of them configured to work with Azure disks:

- The *default* storage class provisions a standard SSD Azure disk.
 - Standard storage is backed by Standard SSDs and delivers cost-effective storage while still delivering reliable performance.
- The *managed-premium* storage class provisions a premium Azure disk.
 - Premium disks are backed by SSD-based high-performance, low-latency disk. Perfect for VMs running production workload. If the AKS nodes in your cluster use premium storage, select the *managed-premium* class.

If you use one of the default storage classes, you can't update the volume size after the storage class is created. To be able to update the volume size after a storage class is created, add the line `allowVolumeExpansion: true` to one of the default storage classes, or you can create your own custom storage class. Note that it's not supported to reduce the size of a PVC (to prevent data loss). You can edit an existing storage class by using the `kubectl edit sc` command.

For example, if you want to use a disk of size 4 TiB, you must create a storage class that defines `cachingmode: None` because [disk caching isn't supported for disks 4 TiB and larger](#).

For more information about storage classes and creating your own storage class, see [Storage options for applications in AKS](#).

Use the `kubectl get sc` command to see the pre-created storage classes. The following example shows the pre-create storage classes available within an AKS cluster:

```
$ kubectl get sc

NAME          PROVISIONER           AGE
default (default)  kubernetes.io/azure-disk  1h
managed-premium   kubernetes.io/azure-disk  1h
```

NOTE

Persistent volume claims are specified in GiB but Azure managed disks are billed by SKU for a specific size. These SKUs range from 32GiB for S4 or P4 disks to 32TiB for S80 or P80 disks (in preview). The throughput and IOPS performance of a Premium managed disk depends on the both the SKU and the instance size of the nodes in the AKS cluster. For more information, see [Pricing and Performance of Managed Disks](#).

Create a persistent volume claim

A persistent volume claim (PVC) is used to automatically provision storage based on a storage class. In this case, a PVC can use one of the pre-created storage classes to create a standard or premium Azure managed disk.

Create a file named `azure-premium.yaml`, and copy in the following manifest. The claim requests a disk named `azure-managed-disk` that is *5GB* in size with *ReadWriteOnce* access. The *managed-premium* storage class is specified as the storage class.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: azure-managed-disk
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: managed-premium
  resources:
    requests:
      storage: 5Gi
```

TIP

To create a disk that uses standard storage, use `storageClassName: default` rather than *managed-premium*.

Create the persistent volume claim with the `kubectl apply` command and specify your `azure-premium.yaml` file:

```
$ kubectl apply -f azure-premium.yaml

persistentvolumeclaim/azure-managed-disk created
```

Use the persistent volume

Once the persistent volume claim has been created and the disk successfully provisioned, a pod can be created with access to the disk. The following manifest creates a basic NGINX pod that uses the persistent volume claim named *azure-managed-disk* to mount the Azure disk at the path `/mnt/azure`. For Windows Server containers, specify a *mountPath* using the Windows path convention, such as '*D:*'.

Create a file named `azure-pvc-disk.yaml`, and copy in the following manifest.

```
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
    resources:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 250m
        memory: 256Mi
    volumeMounts:
    - mountPath: "/mnt/azure"
      name: volume
  volumes:
  - name: volume
    persistentVolumeClaim:
      claimName: azure-managed-disk
```

Create the pod with the `kubectl apply` command, as shown in the following example:

```
$ kubectl apply -f azure-pvc-disk.yaml
pod/mypod created
```

You now have a running pod with your Azure disk mounted in the `/mnt/azure` directory. This configuration can be seen when inspecting your pod via `kubectl describe pod mypod`, as shown in the following condensed example:

```
$ kubectl describe pod mypod
[...]
Volumes:
  volume:
    Type:     PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  azure-managed-disk
    ReadOnly:   false
  default-token-smm2n:
    Type:     Secret (a volume populated by a Secret)
    SecretName: default-token-smm2n
    Optional:   false
[...]
Events:
  Type  Reason          Age   From           Message
  ----  -----          ----  --  -----
  Normal Scheduled      2m    default-scheduler  Successfully assigned mypod to
  aks-nodepool1-79590246-0
  Normal SuccessfulMountVolume 2m    kubelet, aks-nodepool1-79590246-0  MountVolume.SetUp succeeded for
  volume "default-token-smm2n"
  Normal SuccessfulMountVolume 1m    kubelet, aks-nodepool1-79590246-0  MountVolume.SetUp succeeded for
  volume "pvc-faf0f176-8b8d-11e8-923b-deb28c58d242"
[...]
```

Use Ultra Disks

To leverage ultra disk see [Use Ultra Disks on Azure Kubernetes Service \(AKS\)](#).

Back up a persistent volume

To back up the data in your persistent volume, take a snapshot of the managed disk for the volume. You can then use this snapshot to create a restored disk and attach to pods as a means of restoring the data.

First, get the volume name with the `kubectl get pvc` command, such as for the PVC named *azure-managed-disk*.

```
$ kubectl get pvc azure-managed-disk

NAME                  STATUS    VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS          AGE
azure-managed-disk   Bound    pvc-faf0f176-8b8d-11e8-923b-deb28c58d242   5Gi        RWO           managed-
premium              3m
```

This volume name forms the underlying Azure disk name. Query for the disk ID with [az disk list](#) and provide your PVC volume name, as shown in the following example:

```
$ az disk list --query '[].id | [?contains(@,`pvc-faf0f176-8b8d-11e8-923b-deb28c58d242`)]' -o tsv

/subscriptions/<guid>/resourceGroups/MC_MYRESOURCEGROUP_MYAKSCLUSTER_EASTUS/providers/MicrosoftCompute/disks
/kubernetes-dynamic-pvc-faf0f176-8b8d-11e8-923b-deb28c58d242
```

Use the disk ID to create a snapshot disk with [az snapshot create](#). The following example creates a snapshot named *pvcSnapshot* in the same resource group as the AKS cluster (*MC_myResourceGroup_myAKSCluster_eastus*). You may encounter permission issues if you create snapshots and restore disks in resource groups that the AKS cluster does not have access to.

```
$ az snapshot create \
--resource-group MC_myResourceGroup_myAKSCluster_eastus \
--name pvcSnapshot \
--source
/subscriptions/<guid>/resourceGroups/MC_myResourceGroup_myAKSCluster_eastus/providers/MicrosoftCompute/disks
/kubernetes-dynamic-pvc-faf0f176-8b8d-11e8-923b-deb28c58d242
```

Depending on the amount of data on your disk, it may take a few minutes to create the snapshot.

Restore and use a snapshot

To restore the disk and use it with a Kubernetes pod, use the snapshot as a source when you create a disk with [az disk create](#). This operation preserves the original resource if you then need to access the original data snapshot. The following example creates a disk named *pvcRestored* from the snapshot named *pvcSnapshot*.

```
az disk create --resource-group MC_myResourceGroup_myAKSCluster_eastus --name pvcRestored --source
pvcSnapshot
```

To use the restored disk with a pod, specify the ID of the disk in the manifest. Get the disk ID with the [az disk show](#) command. The following example gets the disk ID for *pvcRestored* created in the previous step:

```
az disk show --resource-group MC_myResourceGroup_myAKSCluster_eastus --name pvcRestored --query id -o tsv
```

Create a pod manifest named `azure-restored.yaml` and specify the disk URI obtained in the previous step. The

following example creates a basic NGINX web server, with the restored disk mounted as a volume at `/mnt/azure`.

```
kind: Pod
apiVersion: v1
metadata:
  name: mypodrestored
spec:
  containers:
    - name: mypodrestored
      image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
      resources:
        requests:
          cpu: 100m
          memory: 128Mi
        limits:
          cpu: 250m
          memory: 256Mi
      volumeMounts:
        - mountPath: "/mnt/azure"
          name: volume
    volumes:
      - name: volume
        azureDisk:
          kind: Managed
          diskName: pvcRestored
          diskURI:
            /subscriptions/<guid>/resourceGroups/MC_myResourceGroupAKS_myAKSCluster_eastus/providers/Microsoft.Compute/disks/pvcRestored
```

Create the pod with the [kubectl apply](#) command, as shown in the following example:

```
$ kubectl apply -f azure-restored.yaml
pod/mypodrestored created
```

You can use [kubectl describe pod mypodrestored](#) to view details of the pod, such as the following condensed example that shows the volume information:

```
$ kubectl describe pod mypodrestored
[...]
Volumes:
  volume:
    Type:      AzureDisk (an Azure Data Disk mount on the host and bind mount to the pod)
    DiskName:  pvcRestored
    DiskURI:   /subscriptions/19da35d3-9a1a-4f3b-9b9c-
3c56ef409565/resourceGroups/MC_myResourceGroupAKS_myAKSCluster_eastus/providers/Microsoft.Compute/disks/pvcR
estored
    Kind:      Managed
    FSType:    ext4
    CachingMode:  ReadWrite
    ReadOnly:  false
[...]
```

Next steps

For associated best practices, see [Best practices for storage and backups in AKS](#).

Learn more about Kubernetes persistent volumes using Azure disks.

[Kubernetes plugin for Azure disks](#)

Manually create and use a volume with Azure disks in Azure Kubernetes Service (AKS)

4/21/2021 • 3 minutes to read • [Edit Online](#)

Container-based applications often need to access and persist data in an external data volume. If a single pod needs access to storage, you can use Azure disks to present a native volume for application use. This article shows you how to manually create an Azure disk and attach it to a pod in AKS.

NOTE

An Azure disk can only be mounted to a single pod at a time. If you need to share a persistent volume across multiple pods, use [Azure Files](#).

For more information on Kubernetes volumes, see [Storage options for applications in AKS](#).

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart using the [Azure CLI](#) or [using the Azure portal](#).

You also need the Azure CLI version 2.0.59 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Create an Azure disk

When you create an Azure disk for use with AKS, you can create the disk resource in the **node** resource group. This approach allows the AKS cluster to access and manage the disk resource. If you instead create the disk in a separate resource group, you must grant the Azure Kubernetes Service (AKS) managed identity for your cluster the `Contributor` role to the disk's resource group.

For this article, create the disk in the node resource group. First, get the resource group name with the `az aks show` command and add the `--query nodeResourceGroup` query parameter. The following example gets the node resource group for the AKS cluster name *myAKSCluster* in the resource group name *myResourceGroup*.

```
$ az aks show --resource-group myResourceGroup --name myAKSCluster --query nodeResourceGroup -o tsv  
MC_myResourceGroup_myAKSCluster_eastus
```

Now create a disk using the `az disk create` command. Specify the node resource group name obtained in the previous command, and then a name for the disk resource, such as *myAKSDisk*. The following example creates a 20GiB disk, and outputs the ID of the disk once created. If you need to create a disk for use with Windows Server containers, add the `--os-type windows` parameter to correctly format the disk.

```
az disk create \  
--resource-group MC_myResourceGroup_myAKSCluster_eastus \  
--name myAKSDisk \  
--size-gb 20 \  
--query id --output tsv
```

NOTE

Azure disks are billed by SKU for a specific size. These SKUs range from 32GiB for S4 or P4 disks to 32TiB for S80 or P80 disks (in preview). The throughput and IOPS performance of a Premium managed disk depends on both the SKU and the instance size of the nodes in the AKS cluster. See [Pricing and Performance of Managed Disks](#).

The disk resource ID is displayed once the command has successfully completed, as shown in the following example output. This disk ID is used to mount the disk in the next step.

```
/subscriptions/<subscriptionID>/resourceGroups/MC_myAKSCluster_myAKSCluster_eastus/providers/Microsoft.Compute/disks/myAKSDisk
```

Mount disk as volume

To mount the Azure disk into your pod, configure the volume in the container spec. Create a new file named `azure-disk-pod.yaml` with the following contents. Update `diskName` with the name of the disk created in the previous step, and `diskURI` with the disk ID shown in output of the disk create command. If desired, update the `mountPath`, which is the path where the Azure disk is mounted in the pod. For Windows Server containers, specify a *mountPath* using the Windows path convention, such as '*D:*'.

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
      name: mypod
      resources:
        requests:
          cpu: 100m
          memory: 128Mi
        limits:
          cpu: 250m
          memory: 256Mi
      volumeMounts:
        - name: azure
          mountPath: /mnt/azure
  volumes:
    - name: azure
      azureDisk:
        kind: Managed
        diskName: myAKSDisk
        diskURI:
/subscriptions/<subscriptionID>/resourceGroups/MC_myAKSCluster_myAKSCluster_eastus/providers/Microsoft.Compute/disks/myAKSDisk
```

Use the `kubectl` command to create the pod.

```
kubectl apply -f azure-disk-pod.yaml
```

You now have a running pod with an Azure disk mounted at `/mnt/azure`. You can use `kubectl describe pod mypod` to verify the disk is mounted successfully. The following condensed example output shows the volume mounted in the container:

```
[...]
Volumes:
  azure:
    Type:          AzureDisk (an Azure Data Disk mount on the host and bind mount to the pod)
    DiskName:      myAKSDisk
    DiskURI:
/subscriptions/<subscriptionID/resourceGroups/MC_myResourceGroupAKS_myAKSCluster_eastus/providers/Microsoft.Compute/disks/myAKSDisk
    Kind:          Managed
    FSType:        ext4
    CachingMode:   ReadWrite
    ReadOnly:      false
  default-token-z5sd7:
    Type:          Secret (a volume populated by a Secret)
    SecretName:   default-token-z5sd7
    Optional:     false
[...]
Events:
  Type  Reason           Age   From            Message
  ----  -----          ----  --              -----
  Normal Scheduled       1m    default-scheduler  Successfully assigned mypod to
aks-nodepool1-79590246-0
  Normal SuccessfulMountVolume 1m    kubelet, aks-nodepool1-79590246-0  MountVolume.SetUp succeeded for
volume "default-token-z5sd7"
  Normal SuccessfulMountVolume 41s   kubelet, aks-nodepool1-79590246-0  MountVolume.SetUp succeeded for
volume "azure"
[...]
```

Next steps

For associated best practices, see [Best practices for storage and backups in AKS](#).

For more information about AKS clusters interact with Azure disks, see the [Kubernetes plugin for Azure Disks](#).

Dynamically create and use a persistent volume with Azure Files in Azure Kubernetes Service (AKS)

4/21/2021 • 4 minutes to read • [Edit Online](#)

A persistent volume represents a piece of storage that has been provisioned for use with Kubernetes pods. A persistent volume can be used by one or many pods, and can be dynamically or statically provisioned. If multiple pods need concurrent access to the same storage volume, you can use Azure Files to connect using the [Server Message Block \(SMB\) protocol](#). This article shows you how to dynamically create an Azure Files share for use by multiple pods in an Azure Kubernetes Service (AKS) cluster.

For more information on Kubernetes volumes, see [Storage options for applications in AKS](#).

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart using the [Azure CLI](#) or [using the Azure portal](#).

You also need the Azure CLI version 2.0.59 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Create a storage class

A storage class is used to define how an Azure file share is created. A storage account is automatically created in the [node resource group](#) for use with the storage class to hold the Azure file shares. Choose of the following [Azure storage redundancy](#) for *skuName*:

- *Standard_LRS* - standard locally redundant storage (LRS)
- *Standard_GRS* - standard geo-redundant storage (GRS)
- *Standard_ZRS* - standard zone redundant storage (ZRS)
- *Standard_RAGRS* - standard read-access geo-redundant storage (RA-GRS)
- *Premium_LRS* - premium locally redundant storage (LRS)
- *Premium_ZRS* - premium zone redundant storage (ZRS)

NOTE

Azure Files support premium storage in AKS clusters that run Kubernetes 1.13 or higher, minimum premium file share is 100GB

For more information on Kubernetes storage classes for Azure Files, see [Kubernetes Storage Classes](#).

Create a file named `azure-file-sc.yaml` and copy in the following example manifest. For more information on *mountOptions*, see the [Mount options](#) section.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: my-azurefile
provisioner: kubernetes.io/azure-file
mountOptions:
  - dir_mode=0777
  - file_mode=0777
  - uid=0
  - gid=0
  - mfsymlinks
  - cache=strict
  - actimeo=30
parameters:
  skuName: Standard_LRS
```

Create the storage class with the [kubectl apply](#) command:

```
kubectl apply -f azure-file-sc.yaml
```

Create a persistent volume claim

A persistent volume claim (PVC) uses the storage class object to dynamically provision an Azure file share. The following YAML can be used to create a persistent volume claim *5 GB* in size with *ReadWriteMany* access. For more information on access modes, see the [Kubernetes persistent volume](#) documentation.

Now create a file named `azure-file-pvc.yaml` and copy in the following YAML. Make sure that the *storageClassName* matches the storage class created in the last step:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-azurefile
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: my-azurefile
  resources:
    requests:
      storage: 5Gi
```

NOTE

If using the *Premium_LRS* sku for your storage class, the minimum value for *storage* must be *100Gi*.

Create the persistent volume claim with the [kubectl apply](#) command:

```
kubectl apply -f azure-file-pvc.yaml
```

Once completed, the file share will be created. A Kubernetes secret is also created that includes connection information and credentials. You can use the [kubectl get](#) command to view the status of the PVC:

```
$ kubectl get pvc my-azurefile

NAME          STATUS    VOLUME                                     CAPACITY   ACCESS MODES  STORAGECLASS
AGE
my-azurefile  Bound     pvc-8436e62e-a0d9-11e5-8521-5a8664dc0477  5Gi        RWX           my-azurefile
5m
```

Use the persistent volume

The following YAML creates a pod that uses the persistent volume claim *my-azurefile* to mount the Azure file share at the */mnt/azure* path. For Windows Server containers, specify a *mountPath* using the Windows path convention, such as '*D:*'.

Create a file named `azure-pvc-files.yaml`, and copy in the following YAML. Make sure that the *claimName* matches the PVC created in the last step.

```
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
    resources:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 250m
        memory: 256Mi
    volumeMounts:
    - mountPath: "/mnt/azure"
      name: volume
  volumes:
  - name: volume
    persistentVolumeClaim:
      claimName: my-azurefile
```

Create the pod with the [kubectl apply](#) command.

```
kubectl apply -f azure-pvc-files.yaml
```

You now have a running pod with your Azure Files share mounted in the */mnt/azure* directory. This configuration can be seen when inspecting your pod via `kubectl describe pod mypod`. The following condensed example output shows the volume mounted in the container:

```
Containers:
  mypod:
    Container ID: docker://053bc9c0df72232d755aa040bfba8b533fa696b123876108dec400e364d2523e
    Image:         mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
    Image ID:      docker-
    pullable://nginx@sha256:d85914d547a6c92faa39ce7058bd7529baacab7e0cd4255442b04577c4d1f424
    State:        Running
    Started:     Fri, 01 Mar 2019 23:56:16 +0000
    Ready:       True
    Mounts:
      /mnt/azure from volume (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-8rv4z (ro)
  [...]
Volumes:
  volume:
    Type:      PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName: my-azurefile
    ReadOnly:   false
  [...]
```

Mount options

The default value for *fileMode* and *dirMode* is *0777* for Kubernetes version 1.13.0 and above. If dynamically creating the persistent volume with a storage class, mount options can be specified on the storage class object. The following example sets *0777*:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: my-azurefile
provisioner: kubernetes.io/azure-file
mountOptions:
  - dir_mode=0777
  - file_mode=0777
  - uid=0
  - gid=0
  - mfsymlinks
  - cache=strict
  - actimeo=30
parameters:
  skuName: Standard_LRS
```

Next steps

For associated best practices, see [Best practices for storage and backups in AKS](#).

For storage class parameters, see [Dynamic Provision](#).

Learn more about Kubernetes persistent volumes using Azure Files.

[Kubernetes plugin for Azure Files](#)

Manually create and use a volume with Azure Files share in Azure Kubernetes Service (AKS)

4/21/2021 • 4 minutes to read • [Edit Online](#)

Container-based applications often need to access and persist data in an external data volume. If multiple pods need concurrent access to the same storage volume, you can use Azure Files to connect using the [Server Message Block \(SMB\) protocol](#). This article shows you how to manually create an Azure Files share and attach it to a pod in AKS.

For more information on Kubernetes volumes, see [Storage options for applications in AKS](#).

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart [using the Azure CLI](#) or [using the Azure portal](#).

You also need the Azure CLI version 2.0.59 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Create an Azure file share

Before you can use Azure Files as a Kubernetes volume, you must create an Azure Storage account and the file share. The following commands create a resource group named *myAKSShare*, a storage account, and a Files share named *aksshare*:

```
# Change these four parameters as needed for your own environment
$AKS_PERS_STORAGE_ACCOUNT_NAME=mystorageaccount$RANDOM
$AKS_PERS_RESOURCE_GROUP=myAKSShare
$AKS_PERS_LOCATION=eastus
$AKS_PERS_SHARE_NAME=aksshare

# Create a resource group
az group create --name $AKS_PERS_RESOURCE_GROUP --location $AKS_PERS_LOCATION

# Create a storage account
az storage account create -n $AKS_PERS_STORAGE_ACCOUNT_NAME -g $AKS_PERS_RESOURCE_GROUP -l
$AKS_PERS_LOCATION --sku Standard_LRS

# Export the connection string as an environment variable, this is used when creating the Azure file share
export AZURE_STORAGE_CONNECTION_STRING=$(az storage account show-connection-string -n
$AKS_PERS_STORAGE_ACCOUNT_NAME -g $AKS_PERS_RESOURCE_GROUP -o tsv)

# Create the file share
az storage share create -n $AKS_PERS_SHARE_NAME --connection-string $AZURE_STORAGE_CONNECTION_STRING

# Get storage account key
$STORAGE_KEY=$(az storage account keys list --resource-group $AKS_PERS_RESOURCE_GROUP --account-name
$AKS_PERS_STORAGE_ACCOUNT_NAME --query "[0].value" -o tsv)

# Echo storage account name and key
echo Storage account name: $AKS_PERS_STORAGE_ACCOUNT_NAME
echo Storage account key: $STORAGE_KEY
```

Make a note of the storage account name and key shown at the end of the script output. These values are needed when you create the Kubernetes volume in one of the following steps.

Create a Kubernetes secret

Kubernetes needs credentials to access the file share created in the previous step. These credentials are stored in a [Kubernetes secret](#), which is referenced when you create a Kubernetes pod.

Use the `kubectl create secret` command to create the secret. The following example creates a shared named `azure-secret` and populates the `azurerestorageaccountname` and `azurerestorageaccountkey` from the previous step. To use an existing Azure storage account, provide the account name and key.

```
kubectl create secret generic azure-secret --from-literal=azurerestorageaccountname=$AKS_PERS_STORAGE_ACCOUNT_NAME --from-literal=azurerestorageaccountkey=$STORAGE_KEY
```

Mount file share as an inline volume

Note: starting from 1.18.15, 1.19.7, 1.20.2, 1.21.0, secret namespace in inline `azureFile` volume can only be set as `default` namespace, to specify a different secret namespace, please use below persistent volume example instead.

To mount the Azure Files share into your pod, configure the volume in the container spec. Create a new file named `azure-files-pod.yaml` with the following contents. If you changed the name of the Files share or secret name, update the `shareName` and `secretName`. If desired, update the `mountPath`, which is the path where the Files share is mounted in the pod. For Windows Server containers, specify a `mountPath` using the Windows path convention, such as '`D:`'.

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
    name: mypod
    resources:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 250m
        memory: 256Mi
    volumeMounts:
    - name: azure
      mountPath: /mnt/azure
  volumes:
  - name: azure
    azureFile:
      secretName: azure-secret
      shareName: aksshare
      readOnly: false
```

Use the `kubectl` command to create the pod.

```
kubectl apply -f azure-files-pod.yaml
```

You now have a running pod with an Azure Files share mounted at `/mnt/azure`. You can use `kubectl describe pod mypod` to verify the share is mounted successfully. The following condensed example

output shows the volume mounted in the container:

```
Containers:
  mypod:
    Container ID: docker://86d244cf7c4822401e88f55fd75217d213aa9c3c6a3df169e76e8e25ed28166
    Image:          mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
    Image ID:       docker-
    pullable://nginx@sha256:9ad0746d8f2ea6df3a17ba89eca40b48c47066dfab55a75e08e2b70fc80d929e
    State:          Running
    Started:        Sat, 02 Mar 2019 00:05:47 +0000
    Ready:          True
    Mounts:
      /mnt/azure from azure (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-z5sd7 (ro)
    [...]
Volumes:
  azure:
    Type:          AzureFile (an Azure File Service mount on the host and bind mount to the pod)
    SecretName:    azure-secret
    ShareName:     aksshare
    ReadOnly:      false
  default-token-z5sd7:
    Type:          Secret (a volume populated by a Secret)
    SecretName:    default-token-z5sd7
  [...]
```

Mount file share as an persistent volume

- Mount options

The default value for *fileMode* and *dirMode* is *0777* for Kubernetes version 1.15 and above. The following example sets *0755* on the *PersistentVolume* object:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: azurefile
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  azureFile:
    secretName: azure-secret
    secretNamespace: default
    shareName: aksshare
    readOnly: false
  mountOptions:
    - dir_mode=0755
    - file_mode=0755
    - uid=1000
    - gid=1000
    - mfsymlinks
    - nobrl
```

To update your mount options, create a *azurefile-mount-options-pv.yaml* file with a *PersistentVolume*. For example:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: azurefile
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  azureFile:
    secretName: azure-secret
    shareName: aksshare
    readOnly: false
  mountOptions:
    - dir_mode=0777
    - file_mode=0777
    - uid=1000
    - gid=1000
    - mfsymlinks
    - nobrl
```

Create a `azurefile-mount-options-pvc.yaml` file with a `PersistentVolumeClaim` that uses the `PersistentVolume`. For example:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: azurefile
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 5Gi
```

Use the `kubectl` commands to create the `PersistentVolume` and `PersistentVolumeClaim`.

```
kubectl apply -f azurefile-mount-options-pv.yaml
kubectl apply -f azurefile-mount-options-pvc.yaml
```

Verify your `PersistentVolumeClaim` is created and bound to the `PersistentVolume`.

```
$ kubectl get pvc azurefile
NAME      STATUS  VOLUME      CAPACITY  ACCESS MODES  STORAGECLASS  AGE
azurefile  Bound   azurefile   5Gi       RWX        azurefile    5s
```

Update your container spec to reference your `PersistentVolumeClaim` and update your pod. For example:

```
...
volumes:
- name: azure
  persistentVolumeClaim:
    claimName: azurefile
```

Next steps

For associated best practices, see [Best practices for storage and backups in AKS](#).

For more information about AKS clusters interact with Azure Files, see the [Kubernetes plugin for Azure Files](#).

For storage class parameters, see [Static Provision\(bring your own file share\)](#).

Manually create and use an NFS (Network File System) Linux Server volume with Azure Kubernetes Service (AKS)

11/2/2020 • 4 minutes to read • [Edit Online](#)

Sharing data between containers is often a necessary component of container-based services and applications. You usually have various pods that need access to the same information on an external persistent volume. While Azure files are an option, creating an NFS Server on an Azure VM is another form of persistent shared storage.

This article will show you how to create an NFS Server on an Ubuntu virtual machine. And also give your AKS containers access to this shared file system.

Before you begin

This article assumes that you have an existing AKS Cluster. If you need an AKS Cluster, see the AKS quickstart [using the Azure CLI](#) or [using the Azure portal](#).

Your AKS Cluster will need to live in the same or peered virtual networks as the NFS Server. The cluster must be created in an existing VNET, which can be the same VNET as your VM.

The steps for configuring with an existing VNET are described in the documentation: [creating AKS Cluster in existing VNET](#) and [connecting virtual networks with VNET peering](#)

It also assumes you've created an Ubuntu Linux Virtual Machine (for example, 18.04 LTS). Settings and size can be to your liking and can be deployed through Azure. For Linux quickstart, see [Linux VM management](#).

If you deploy your AKS Cluster first, Azure will automatically populate the virtual network field when deploying your Ubuntu machine, making them live within the same VNET. But if you want to work with peered networks instead, consult the documentation above.

Deploying the NFS Server onto a Virtual Machine

Here is the script to set up an NFS Server within your Ubuntu virtual machine:

```

#!/bin/bash

# This script should be executed on Linux Ubuntu Virtual Machine

EXPORT_DIRECTORY=${1:-/export/data}
DATA_DIRECTORY=${2:-/data}
AKS_SUBNET=${3:-*}

echo "Updating packages"
apt-get -y update

echo "Installing NFS kernel server"

apt-get -y install nfs-kernel-server

echo "Making data directory ${DATA_DIRECTORY}"
mkdir -p ${DATA_DIRECTORY}

echo "Making new directory to be exported and linked to data directory: ${EXPORT_DIRECTORY}"
mkdir -p ${EXPORT_DIRECTORY}

echo "Mount binding ${DATA_DIRECTORY} to ${EXPORT_DIRECTORY}"
mount --bind ${DATA_DIRECTORY} ${EXPORT_DIRECTORY}

echo "Giving 777 permissions to ${EXPORT_DIRECTORY} directory"
chmod 777 ${EXPORT_DIRECTORY}

parentdir="$(dirname ${EXPORT_DIRECTORY})"
echo "Giving 777 permissions to parent: ${parentdir} directory"
chmod 777 $parentdir

echo "Appending bound directories into fstab"
echo "${DATA_DIRECTORY}    ${EXPORT_DIRECTORY}    none    bind    0    0" >> /etc/fstab

echo "Appending localhost and Kubernetes subnet address ${AKS_SUBNET} to exports configuration file"
echo "/export    ${AKS_SUBNET}(rw,async,insecure,fsid=0,crossmnt,no_subtree_check)" >> /etc/exports
echo "/export    localhost(rw,async,insecure,fsid=0,crossmnt,no_subtree_check)" >> /etc/exports

nohup service nfs-kernel-server restart

```

The server will restart (because of the script) and you can mount the NFS Server to AKS.

IMPORTANT

Make sure to replace the **AKS_SUBNET** with the correct one from your cluster or else "*" will open your NFS Server to all ports and connections.

After you've created your VM, copy the script above into a file. Then, you can move it from your local machine, or wherever the script is, into the VM using:

```
scp /path/to/script_file username@vm-ip-address:/home/{username}
```

Once your script is in your VM, you can ssh into the VM and execute it via the command:

```
sudo ./nfs-server-setup.sh
```

If its execution fails because of a permission denied error, set execution permission via the command:

```
chmod +x ~/nfs-server-setup.sh
```

Connecting AKS Cluster to NFS Server

We can connect the NFS Server to our cluster by provisioning a persistent volume and persistent volume claim that specifies how to access the volume.

Connecting the two services in the same or peered virtual networks is necessary. Instructions for setting up the cluster in the same VNET are here: [Creating AKS Cluster in existing VNET](#)

Once they are in the same virtual network (or peered), you need to provision a persistent volume and a persistent volume claim in your AKS Cluster. The containers can then mount the NFS drive to their local directory.

Here is an example Kubernetes definition for the persistent volume (This definition assumes your cluster and VM are in the same VNET):

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: <NFS_NAME>
  labels:
    type: nfs
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  nfs:
    server: <NFS_INTERNAL_IP>
    path: <NFS_EXPORT_FILE_PATH>
```

Replace **NFS_INTERNAL_IP**, **NFS_NAME** and **NFS_EXPORT_FILE_PATH** with NFS Server information.

You'll also need a persistent volume claim file. Here is an example of what to include:

IMPORTANT

"**storageClassName**" needs to remain an empty string or the claim won't work.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: <NFS_NAME>
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 1Gi
  selector:
    matchLabels:
      type: nfs
```

Troubleshooting

If you can't connect to the server from a cluster, an issue might be the exported directory, or its parent, doesn't have sufficient permissions to access the server.

Check that both your export directory and its parent directory have 777 permissions.

You can check permissions by running the command below and the directories should have '*drwxrwxrwx*' permissions:

```
ls -l
```

More information

To get a full walkthrough or to help you debug your NFS Server setup, here is an in-depth tutorial:

- [NFS Tutorial](#)

Next steps

For associated best practices, see [Best practices for storage and backups in AKS](#).

Integrate Azure NetApp Files with Azure Kubernetes Service

4/21/2021 • 5 minutes to read • [Edit Online](#)

Azure NetApp Files is an enterprise-class, high-performance, metered file storage service running on Azure. This article shows you how to integrate Azure NetApp Files with Azure Kubernetes Service (AKS).

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart using the [Azure CLI](#) or [using the Azure portal](#).

IMPORTANT

Your AKS cluster must also be [in a region that supports Azure NetApp Files](#).

You also need the Azure CLI version 2.0.59 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Limitations

The following limitations apply when you use Azure NetApp Files:

- Azure NetApp Files is only available [in selected Azure regions](#).
- Before you can use Azure NetApp Files, you must be granted access to the Azure NetApp Files service. To apply for access, you can use the [Azure NetApp Files waitlist submission form](#) or go to <https://azure.microsoft.com/services/netapp/#getting-started>. You can't access the Azure NetApp Files service until you receive the official confirmation email from the Azure NetApp Files team.
- After the initial deployment of an AKS cluster, only static provisioning for Azure NetApp Files is supported.
- To use dynamic provisioning with Azure NetApp Files, install and configure [NetApp Trident](#) version 19.07 or later.

Configure Azure NetApp Files

IMPORTANT

Before you can register the *Microsoft.NetApp* resource provider, you must complete the [Azure NetApp Files waitlist submission form](#) or go to <https://azure.microsoft.com/services/netapp/#getting-started> for your subscription. You can't register the resource provider until you receive the official confirmation email from the Azure NetApp Files team.

Register the *Microsoft.NetApp* resource provider:

```
az provider register --namespace Microsoft.NetApp --wait
```

NOTE

This can take some time to complete.

When you create an Azure NetApp account for use with AKS, you need to create the account in the **node** resource group. First, get the resource group name with the [az aks show](#) command and add the `--query nodeResourceGroup` query parameter. The following example gets the node resource group for the AKS cluster named *myAKSCluster* in the resource group name *myResourceGroup*:

```
az aks show --resource-group myResourceGroup --name myAKSCluster --query nodeResourceGroup -o tsv
```

```
MC_myResourceGroup_myAKSCluster_eastus
```

Create an Azure NetApp Files account in the **node** resource group and same region as your AKS cluster using [az netappfiles account create](#). The following example creates an account named *myaccount1* in the *MC_myResourceGroup_myAKSCluster_eastus* resource group and *eastus* region:

```
az netappfiles account create \
--resource-group MC_myResourceGroup_myAKSCluster_eastus \
--location eastus \
--account-name myaccount1
```

Create a new capacity pool by using [az netappfiles pool create](#). The following example creates a new capacity pool named *mypool1* with 4 TB in size and *Premium* service level:

```
az netappfiles pool create \
--resource-group MC_myResourceGroup_myAKSCluster_eastus \
--location eastus \
--account-name myaccount1 \
--pool-name mypool1 \
--size 4 \
--service-level Premium
```

Create a subnet to [delegate to Azure NetApp Files](#) using [az network vnet subnet create](#). *This subnet must be in the same virtual network as your AKS cluster.*

```
RESOURCE_GROUP=MC_myResourceGroup_myAKSCluster_eastus
VNET_NAME=$(az network vnet list --resource-group $RESOURCE_GROUP --query [].name -o tsv)
VNET_ID=$(az network vnet show --resource-group $RESOURCE_GROUP --name $VNET_NAME --query "id" -o tsv)
SUBNET_NAME=MyNetAppSubnet
az network vnet subnet create \
--resource-group $RESOURCE_GROUP \
--vnet-name $VNET_NAME \
--name $SUBNET_NAME \
--delegations "Microsoft.NetApp/volumes" \
--address-prefixes 10.0.0.0/28
```

Create a volume by using [az netappfiles volume create](#).

```

RESOURCE_GROUP=MC_myResourceGroup_myAKSCluster_eastus
LOCATION=eastus
ANF_ACCOUNT_NAME=myaccount1
POOL_NAME=mypool1
SERVICE_LEVEL=Premium
VNET_NAME=$(az network vnet list --resource-group $RESOURCE_GROUP --query [].name -o tsv)
VNET_ID=$(az network vnet show --resource-group $RESOURCE_GROUP --name $VNET_NAME --query "id" -o tsv)
SUBNET_NAME=MyNetAppSubnet
SUBNET_ID=$(az network vnet subnet show --resource-group $RESOURCE_GROUP --vnet-name $VNET_NAME --name $SUBNET_NAME --query "id" -o tsv)
VOLUME_SIZE_GiB=100 # 100 GiB
UNIQUE_FILE_PATH="myfilepath2" # Please note that file path needs to be unique within all ANF Accounts

az netappfiles volume create \
    --resource-group $RESOURCE_GROUP \
    --location $LOCATION \
    --account-name $ANF_ACCOUNT_NAME \
    --pool-name $POOL_NAME \
    --name "myvol1" \
    --service-level $SERVICE_LEVEL \
    --vnet $VNET_ID \
    --subnet $SUBNET_ID \
    --usage-threshold $VOLUME_SIZE_GiB \
    --file-path $UNIQUE_FILE_PATH \
    --protocol-types "NFSv3"

```

Create the PersistentVolume

List the details of your volume using `az netappfiles volume show`

```

az netappfiles volume show --resource-group $RESOURCE_GROUP --account-name $ANF_ACCOUNT_NAME --pool-name
$POOL_NAME --volume-name "myvol1"

```

```
{
  ...
  "creationToken": "myfilepath2",
  ...
  "mountTargets": [
    {
      ...
      "ipAddress": "10.0.0.4",
      ...
    }
  ],
  ...
}
```

Create a `pv-nfs.yaml` defining a PersistentVolume. Replace `path` with the `creationToken` and `server` with `ipAddress` from the previous command. For example:

```
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-nfs
spec:
  capacity:
    storage: 100Gi
  accessModes:
    - ReadWriteMany
  mountOptions:
    - vers=3
  nfs:
    server: 10.0.0.4
    path: /myfilepath2
```

Update the *server* and *path* to the values of your NFS (Network File System) volume you created in the previous step. Create the PersistentVolume with the [kubectl apply](#) command:

```
kubectl apply -f pv-nfs.yaml
```

Verify the *Status* of the PersistentVolume is *Available* using the [kubectl describe](#) command:

```
kubectl describe pv pv-nfs
```

Create the PersistentVolumeClaim

Create a `pvc-nfs.yaml` defining a PersistentVolume. For example:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-nfs
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 1Gi
```

Create the PersistentVolumeClaim with the [kubectl apply](#) command:

```
kubectl apply -f pvc-nfs.yaml
```

Verify the *Status* of the PersistentVolumeClaim is *Bound* using the [kubectl describe](#) command:

```
kubectl describe pvc pvc-nfs
```

Mount with a pod

Create a `nginx-nfs.yaml` defining a pod that uses the PersistentVolumeClaim. For example:

```
kind: Pod
apiVersion: v1
metadata:
  name: nginx-nfs
spec:
  containers:
    - image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
      name: nginx-nfs
      command:
        - "/bin/sh"
        - "-c"
        - "while true; do echo $(date) >> /mnt/azure/outfile; sleep 1; done"
      volumeMounts:
        - name: disk01
          mountPath: /mnt/azure
  volumes:
    - name: disk01
      persistentVolumeClaim:
        claimName: pvc-nfs
```

Create the pod with the [kubectl apply](#) command:

```
kubectl apply -f nginx-nfs.yaml
```

Verify the pod is *Running* using the [kubectl describe](#) command:

```
kubectl describe pod nginx-nfs
```

Verify your volume has been mounted in the pod by using [kubectl exec](#) to connect to the pod then `df -h` to check if the volume is mounted.

```
$ kubectl exec -it nginx-nfs -- sh
```

```
/ # df -h
Filesystem           Size  Used Avail Use% Mounted on
...
10.0.0.4:/myfilepath2  100T  384K  100T   1% /mnt/azure
...
```

Next steps

For more information on Azure NetApp Files, see [What is Azure NetApp Files](#). For more information on using NFS with AKS, see [Manually create and use an NFS \(Network File System\) Linux Server volume with Azure Kubernetes Service \(AKS\)](#).

Use Azure ultra disks on Azure Kubernetes Service (preview)

4/21/2021 • 5 minutes to read • [Edit Online](#)

Azure ultra disks offer high throughput, high IOPS, and consistent low latency disk storage for your stateful applications. One major benefit of ultra disks is the ability to dynamically change the performance of the SSD along with your workloads without the need to restart your agent nodes. Ultra disks are suited for data-intensive workloads.

Before you begin

This feature can only be set at cluster creation or node pool creation time.

IMPORTANT

Azure ultra disks require nodepools deployed in availability zones and regions that support these disks as well as only specific VM series. See the [Ultra disks GA scope and limitations](#).

Register the `EnableUltraSSD` preview feature

To create an AKS cluster or a node pool that can leverage Ultra disks, you must enable the `EnableUltraSSD` feature flag on your subscription.

Register the `EnableUltraSSD` feature flag using the `az feature register` command as shown in the following example:

```
az feature register --namespace "Microsoft.ContainerService" --name "EnableUltraSSD"
```

It takes a few minutes for the status to show *Registered*. You can check on the registration status using the `az feature list` command:

```
az feature list -o table --query "[?contains(name, 'Microsoft.ContainerService/EnableUltraSSD')].{Name:name,State:properties.state}"
```

When ready, refresh the registration of the `Microsoft.ContainerService` resource provider using the `az provider register` command:

```
az provider register --namespace Microsoft.ContainerService
```

IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

Install aks-preview CLI extension

To create an AKS cluster or a node pool that can use Ultra Disks, you need the latest *aks-preview* CLI extension.

Install the *aks-preview* Azure CLI extension using the [az extension add](#) command, or install any available updates using the [az extension update](#) command:

```
# Install the aks-preview extension
az extension add --name aks-preview

# Update the extension to make sure you have the latest version installed
az extension update --name aks-preview
```

Limitations

- See the [Ultra disks GA scope and limitations](#)
- The supported size range for a Ultra disks is between 100 and 1500

Create a new cluster that can use Ultra disks

Create an AKS cluster that is able to leverage Ultra Disks by using the following CLI commands. Use the

```
--aks-custom-headers
```

```
EnableUltraSSD
```

Create an Azure resource group:

```
# Create an Azure resource group
az group create --name myResourceGroup --location westus2
```

Create the AKS cluster with support for Ultra Disks.

```
# Create an AKS-managed Azure AD cluster
az aks create -g MyResourceGroup -n MyManagedCluster -l westus2 --node-vm-size Standard_L8s_v2 --zones 1 2 --node-count 2 --aks-custom-headers EnableUltraSSD=true
```

If you want to create clusters without ultra disk support, you can do so by omitting the custom

```
--aks-custom-headers
```

```
parameter.
```

Enable Ultra disks on an existing cluster

You can enable ultra disks on existing clusters by adding a new node pool to your cluster that support ultra disks. Configure a new node pool to use ultra disks by using the [--aks-custom-headers](#) flag.

```
az aks nodepool add --name ultradisk --cluster-name myAKSCluster --resource-group myResourceGroup --node-vm-size Standard_L8s_v2 --zones 1 2 --node-count 2 --aks-custom-headers EnableUltraSSD=true
```

If you want to create new node pools without support for ultra disks, you can do so by omitting the custom

```
--aks-custom-headers
```

```
parameter.
```

Use ultra disks dynamically with a storage class

To use ultra disks in our deployments or stateful sets you can use a [storage class for dynamic provisioning](#).

Create the storage class

A storage class is used to define how a unit of storage is dynamically created with a persistent volume. For more information on Kubernetes storage classes, see [Kubernetes Storage Classes](#).

In this case, we'll create a storage class that references ultra disks. Create a file named `azure-ultra-disk-sc.yaml`, and copy in the following manifest.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: ultra-disk-sc
provisioner: kubernetes.io/azure-disk
volumeBindingMode: WaitForFirstConsumer # optional, but recommended if you want to wait until the pod that
will use this disk is created
parameters:
  skuname: UltraSSD_LRS
  kind: managed
  cachingmode: None
  diskIopsReadWrite: "2000" # minimum value: 2 IOPS/GiB
  diskMbpsReadWrite: "320" # minimum value: 0.032/GiB
```

Create the storage class with the `kubectl apply` command and specify your `azure-ultra-disk-sc.yaml` file:

```
$ kubectl apply -f azure-ultra-disk-sc.yaml

storageclass.storage.k8s.io/ultra-disk-sc created
```

Create a persistent volume claim

A persistent volume claim (PVC) is used to automatically provision storage based on a storage class. In this case, a PVC can use the previously created storage class to create an ultra disk.

Create a file named `azure-ultra-disk-pvc.yaml`, and copy in the following manifest. The claim requests a disk named `ultra-disk` that is *1000 GB* in size with *ReadWriteOnce* access. The `ultra-disk-sc` storage class is specified as the storage class.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ultra-disk
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: ultra-disk-sc
  resources:
    requests:
      storage: 1000Gi
```

Create the persistent volume claim with the `kubectl apply` command and specify your `azure-ultra-disk-pvc.yaml` file:

```
$ kubectl apply -f azure-ultra-disk-pvc.yaml

persistentvolumeclaim/ultra-disk created
```

Use the persistent volume

Once the persistent volume claim has been created and the disk successfully provisioned, a pod can be created with access to the disk. The following manifest creates a basic NGINX pod that uses the persistent volume claim

named *ultra-disk* to mount the Azure disk at the path `/mnt/azure`.

Create a file named `nginx-ultra.yaml`, and copy in the following manifest.

```
kind: Pod
apiVersion: v1
metadata:
  name: nginx-ultra
spec:
  containers:
    - name: nginx-ultra
      image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
      resources:
        requests:
          cpu: 100m
          memory: 128Mi
        limits:
          cpu: 250m
          memory: 256Mi
      volumeMounts:
        - mountPath: "/mnt/azure"
          name: volume
      volumes:
        - name: volume
          persistentVolumeClaim:
            claimName: ultra-disk
```

Create the pod with the `kubectl apply` command, as shown in the following example:

```
$ kubectl apply -f nginx-ultra.yaml
pod/nginx-ultra created
```

You now have a running pod with your Azure disk mounted in the `/mnt/azure` directory. This configuration can be seen when inspecting your pod via `kubectl describe pod nginx-ultra`, as shown in the following condensed example:

```
$ kubectl describe pod nginx-ultra
[...]
Volumes:
volume:
  Type:     PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
  ClaimName:  azure-managed-disk
  ReadOnly:   false
default-token-smm2n:
  Type:     Secret (a volume populated by a Secret)
  SecretName: default-token-smm2n
  Optional:  false
[...]
Events:
  Type  Reason           Age    From                  Message
  ----  -----          ----   ----
  Normal Scheduled       2m    default-scheduler   Successfully assigned mypod to
  aks-nodepool1-79590246-0
  Normal SuccessfulMountVolume 2m    kubelet, aks-nodepool1-79590246-0  MountVolume.SetUp succeeded for
  volume "default-token-smm2n"
  Normal SuccessfulMountVolume 1m    kubelet, aks-nodepool1-79590246-0  MountVolume.SetUp succeeded for
  volume "pvc-faf0f176-8b8d-11e8-923b-deb28c58d24"
[...]
```

Next steps

- For more about ultra disks, see [Using Azure ultra disks](#).
- For more about storage best practices, see [Best practices for storage and backups in Azure Kubernetes Service \(AKS\)](#)

Enable Container Storage Interface (CSI) drivers for Azure disks and Azure Files on Azure Kubernetes Service (AKS) (preview)

4/21/2021 • 4 minutes to read • [Edit Online](#)

The Container Storage Interface (CSI) is a standard for exposing arbitrary block and file storage systems to containerized workloads on Kubernetes. By adopting and using CSI, Azure Kubernetes Service (AKS) can write, deploy, and iterate plug-ins to expose new or improve existing storage systems in Kubernetes without having to touch the core Kubernetes code and wait for its release cycles.

The CSI storage driver support on AKS allows you to natively use:

- [Azure disks](#), which can be used to create a Kubernetes *DataDisk* resource. Disks can use Azure Premium Storage, backed by high-performance SSDs, or Azure Standard Storage, backed by regular HDDs or Standard SSDs. For most production and development workloads, use Premium Storage. Azure disks are mounted as *ReadWriteOnce*, so are only available to a single pod. For storage volumes that can be accessed by multiple pods simultaneously, use Azure Files.
- [Azure Files](#), which can be used to mount an SMB 3.0 share backed by an Azure Storage account to pods. With Azure Files, you can share data across multiple nodes and pods. Azure Files can use Azure Standard Storage backed by regular HDDs or Azure Premium Storage backed by high-performance SSDs.

IMPORTANT

Starting in Kubernetes version 1.21, Kubernetes will use CSI drivers only and by default. These drivers are the future of storage support in Kubernetes.

In-tree drivers refers to the current storage drivers that are part of the core Kubernetes code versus the new CSI drivers, which are plug-ins.

Limitations

- This feature can only be set at cluster creation time.
- The minimum Kubernetes minor version that supports CSI drivers is v1.17.
- During the preview, the default storage class will still be the [same in-tree storage class](#). After this feature is generally available, the default storage class will be the `managed-csi` storage class and in-tree storage classes will be removed.
- During the first preview phase, only Azure CLI is supported.

IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

Register the `EnableAzureDiskFileCSI` preview feature

To create an AKS cluster that can use CSI drivers for Azure disks and Azure Files, you must enable the `EnableAzureDiskFileCSI` feature flag on your subscription.

Register the `EnableAzureDiskFileCSI` feature flag by using the [az feature register](#) command, as shown in the following example:

```
az feature register --namespace "Microsoft.ContainerService" --name "EnableAzureDiskFileCSI"
```

It takes a few minutes for the status to show *Registered*. Verify the registration status by using the [az feature list](#) command:

```
az feature list -o table --query "[?contains(name, 'Microsoft.ContainerService/EnableAzureDiskFileCSI')].{Name:name,State:properties.state}"
```

When ready, refresh the registration of the *Microsoft.ContainerService* resource provider by using the [az provider register](#) command:

```
az provider register --namespace Microsoft.ContainerService
```

IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

Install `aks-preview` CLI extension

To create an AKS cluster or a node pool that can use the CSI storage drivers, you need the latest *aks-preview* Azure CLI extension. Install the *aks-preview* Azure CLI extension by using the [az extension add](#) command. Or install any available updates by using the [az extension update](#) command.

```
# Install the aks-preview extension
az extension add --name aks-preview

# Update the extension to make sure you have the latest version installed
az extension update --name aks-preview
```

Create a new cluster that can use CSI storage drivers

Create a new cluster that can use CSI storage drivers for Azure disks and Azure Files by using the following CLI commands. Use the `--aks-custom-headers` flag to set the `EnableAzureDiskFileCSI` feature.

Create an Azure resource group:

```
# Create an Azure resource group
az group create --name myResourceGroup --location canadacentral
```

Create the AKS cluster with support for CSI storage drivers:

```
# Create an AKS-managed Azure AD cluster
az aks create -g MyResourceGroup -n MyManagedCluster --network-plugin azure --aks-custom-headers
EnableAzureDiskFileCSI=true
```

If you want to create clusters in tree storage drivers instead of CSI storage drivers, you can do so by omitting the custom `--aks-custom-headers` parameter.

Check how many Azure disk-based volumes you can attach to this node by running:

```
$ kubectl get nodes
aks-nodepool1-25371499-vmss000000
aks-nodepool1-25371499-vmss000001
aks-nodepool1-25371499-vmss000002

$ echo $(kubectl get CSINode <NODE NAME> -o jsonpath=".spec.drivers[1].allocatable.count")
8
```

Next steps

- To use the CSI drive for Azure disks, see [Use Azure disks with CSI drivers](#).
- To use the CSI drive for Azure Files, see [Use Azure Files with CSI drivers](#).
- For more about storage best practices, see [Best practices for storage and backups in Azure Kubernetes Service](#).

Use the Azure disk Container Storage Interface (CSI) drivers in Azure Kubernetes Service (AKS) (preview)

4/28/2021 • 9 minutes to read • [Edit Online](#)

The Azure disk Container Storage Interface (CSI) driver is a [CSI specification](#)-compliant driver used by Azure Kubernetes Service (AKS) to manage the lifecycle of Azure disks.

The CSI is a standard for exposing arbitrary block and file storage systems to containerized workloads on Kubernetes. By adopting and using CSI, AKS can write, deploy, and iterate plug-ins to expose new or improve existing storage systems in Kubernetes without having to touch the core Kubernetes code and wait for its release cycles.

To create an AKS cluster with CSI driver support, see [Enable CSI drivers for Azure disks and Azure Files on AKS](#).

NOTE

In-tree drivers refers to the current storage drivers that are part of the core Kubernetes code versus the new CSI drivers, which are plug-ins.

Use CSI persistent volumes with Azure disks

A [persistent volume](#) (PV) represents a piece of storage that's provisioned for use with Kubernetes pods. A PV can be used by one or many pods and can be dynamically or statically provisioned. This article shows you how to dynamically create PVs with Azure disks for use by a single pod in an AKS cluster. For static provisioning, see [Manually create and use a volume with Azure disks](#).

IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

For more information on Kubernetes volumes, see [Storage options for applications in AKS](#).

Dynamically create Azure disk PVs by using the built-in storage classes

A storage class is used to define how a unit of storage is dynamically created with a persistent volume. For more information on Kubernetes storage classes, see [Kubernetes storage classes](#). When you use storage CSI drivers on AKS, there are two additional built-in `StorageClasses` that use the Azure disk CSI storage drivers. The additional CSI storage classes are created with the cluster alongside the in-tree default storage classes.

- `managed-csi` : Uses Azure Standard SSD locally redundant storage (LRS) to create a managed disk.

- `managed-csi-premium` : Uses Azure Premium LRS to create a managed disk.

The reclaim policy in both storage classes ensures that the underlying Azure disk is deleted when the respective PV is deleted. The storage classes also configure the PVs to be expandable. You just need to edit the persistent volume claim (PVC) with the new size.

To leverage these storage classes, create a [PVC](#) and respective pod that references and uses them. A PVC is used to automatically provision storage based on a storage class. A PVC can use one of the pre-created storage classes or a user-defined storage class to create an Azure-managed disk for the desired SKU and size. When you create a pod definition, the PVC is specified to request the desired storage.

Create an example pod and respective PVC with the [kubectl apply](#) command:

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/azuredisk-csi-driver/master/deploy/example/pvc-azuredisk-csi.yaml
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/azuredisk-csi-driver/master/deploy/example/nginx-pod-azuredisk.yaml

persistentvolumeclaim/pvc-azuredisk created
pod/nginx-azuredisk created
```

After the pod is in the running state, create a new file called `test.txt`.

```
$ kubectl exec nginx-azuredisk -- touch /mnt/azuredisk/test.txt
```

You can now validate that the disk is correctly mounted by running the following command and verifying you see the `test.txt` file in the output:

```
$ kubectl exec nginx-azuredisk -- ls /mnt/azuredisk

lost+found
outfile
test.txt
```

Create a custom storage class

The default storage classes suit the most common scenarios, but not all. For some cases, you might want to have your own storage class customized with your own parameters. For example, we have a scenario where you might want to change the `volumeBindingMode` class.

You can use a `volumeBindingMode: Immediate` class that guarantees that occurs immediately once the PVC is created. In cases where your node pools are topology constrained, for example, using availability zones, PVs would be bound or provisioned without knowledge of the pod's scheduling requirements (in this case to be in a specific zone).

To address this scenario, you can use `volumeBindingMode: WaitForFirstConsumer`, which delays the binding and provisioning of a PV until a pod that uses the PVC is created. In this way, the PV will conform and be provisioned in the availability zone (or other topology) that's specified by the pod's scheduling constraints. The default storage classes use `volumeBindingMode: WaitForFirstConsumer` class.

Create a file named `sc-azuredisk-csi-waitforfirstconsumer.yaml`, and paste the following manifest. The storage class is the same as our `managed-csi` storage class but with a different `volumeBindingMode` class.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azuredisk-csi-waitforfirstconsumer
provisioner: disk.csi.azure.com
parameters:
  skuname: StandardSSD_LRS
allowVolumeExpansion: true
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
```

Create the storage class with the [kubectl apply](#) command, and specify your

`sc-azuredisk-csi-waitforfirstconsumer.yaml` file:

```
$ kubectl apply -f sc-azuredisk-csi-waitforfirstconsumer.yaml
storageclass.storage.k8s.io/azuredisk-csi-waitforfirstconsumer created
```

Volume snapshots

The Azure disk CSI driver supports creating [snapshots of persistent volumes](#). As part of this capability, the driver can perform either *full* or *incremental* snapshots depending on the value set in the `incremental` parameter (by default, it's true).

For details on all the parameters, see [volume snapshot class parameters](#).

Create a volume snapshot

For an example of this capability, create a [volume snapshot class](#) with the [kubectl apply](#) command:

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/azuredisk-csi-
driver/master/deploy/example/snapshot/storageclass-azuredisk-snapshot.yaml
volumesnapshotclass.snapshot.storage.k8s.io/csi-azuredisk-vsc created
```

Now let's create a [volume snapshot](#) from the PVC that we dynamically created at the beginning of this tutorial, `pvc-azuredisk`.

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/azuredisk-csi-
driver/master/deploy/example/snapshot/azuredisk-volume-snapshot.yaml
volumesnapshot.snapshot.storage.k8s.io/azuredisk-volume-snapshot created
```

Check that the snapshot was created correctly:

```
$ kubectl describe volumesnapshot azuredisk-volume-snapshot

Name:          azuredisk-volume-snapshot
Namespace:     default
Labels:        <none>
Annotations:   API Version:  snapshot.storage.k8s.io/v1beta1
Kind:          VolumeSnapshot
Metadata:
  Creation Timestamp:  2020-08-27T05:27:58Z
  Finalizers:
    snapshot.storage.kubernetes.io/volumesnapshot-as-source-protection
    snapshot.storage.kubernetes.io/volumesnapshot-bound-protection
  Generation:      1
  Resource Version: 714582
  Self Link:       /apis/snapshot.storage.k8s.io/v1beta1/namespaces/default/volumesnapshots/azuredisk-
volume-snapshot
  UID:            dd953ab5-6c24-42d4-ad4a-f33180e0ef87
Spec:
  Source:
    Persistent Volume Claim Name:  pvc-azuredisk
    Volume Snapshot Class Name:     csi-azuredisk-vsc
Status:
  Bound Volume Snapshot Content Name: snapcontent-dd953ab5-6c24-42d4-ad4a-f33180e0ef87
  Creation Time:                  2020-08-31T05:27:59Z
  Ready To Use:                  true
  Restore Size:                  10Gi
  Events:                       <none>
```

Create a new PVC based on a volume snapshot

You can create a new PVC based on a volume snapshot. Use the snapshot created in the previous step, and create a [new PVC](#) and a [new pod](#) to consume it.

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/azuredisk-csi-
driver/master/deploy/example/snapshot/pvc-azuredisk-snapshot-restored.yaml

$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/azuredisk-csi-
driver/master/deploy/example/snapshot/nginx-pod-restored-snapshot.yaml

persistentvolumeclaim/pvc-azuredisk-snapshot-restored created
pod/nginx-restored created
```

Finally, let's make sure it's the same PVC created before by checking the contents.

```
$ kubectl exec nginx-restored -- ls /mnt/azuredisk

lost+found
outfile
test.txt
```

As expected, we can still see our previously created `test.txt` file.

Clone volumes

A cloned volume is defined as a duplicate of an existing Kubernetes volume. For more information on cloning volumes in Kubernetes, see the conceptual documentation for [volume cloning](#).

The CSI driver for Azure disks supports volume cloning. To demonstrate, create a [cloned volume](#) of the [previously created](#) `azuredisk-pvc` and [a new pod to consume it](#).

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/azuredisk-csi-driver/master/deploy/example/cloning/pvc-azuredisk-cloning.yaml

$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/azuredisk-csi-driver/master/deploy/example/cloning/nginx-pod-restored-cloning.yaml

persistentvolumeclaim/pvc-azuredisk-cloning created
pod/nginx-restored-cloning created
```

We can now check the content of the cloned volume by running the following command and confirming we still see our `test.txt` created file.

```
$ kubectl exec nginx-restored-cloning -- ls /mnt/azuredisk

lost+found
outfile
test.txt
```

Resize a persistent volume

You can instead request a larger volume for a PVC. Edit the PVC object, and specify a larger size. This change triggers the expansion of the underlying volume that backs the PV.

NOTE

A new PV is never created to satisfy the claim. Instead, an existing volume is resized.

In AKS, the built-in `managed-csi` storage class already allows for expansion, so use the [PVC created earlier with this storage class](#). The PVC requested a 10-Gi persistent volume. We can confirm that by running:

```
$ kubectl exec -it nginx-azuredisk -- df -h /mnt/azuredisk

Filesystem      Size  Used Avail Use% Mounted on
/dev/sdc        9.8G  42M  9.8G   1% /mnt/azuredisk
```

IMPORTANT

Currently, the Azure disk CSI driver only supports resizing PVCs with no pods associated (and the volume not mounted to a specific node).

As such, let's delete the pod we created earlier:

```
$ kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/azuredisk-csi-driver/master/deploy/example/nginx-pod-azuredisk.yaml

pod "nginx-azuredisk" deleted
```

Let's expand the PVC by increasing the `spec.resources.requests.storage` field:

```
$ kubectl patch pvc pvc-azuredisk --type merge --patch '{"spec": {"resources": {"requests": {"storage": "15Gi"}}}}'

persistentvolumeclaim/pvc-azuredisk patched
```

Let's confirm the volume is now larger:

```
$ kubectl get pv
```

NAME	STORAGECLASS	REASON	AGE	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM
pvc-391ea1a6-0191-4022-b915-c8dc4216174a	azuredisk	managed-csi	2d2h	15Gi	RWO	Delete	Bound	default/pvc- (...)

NOTE

The PVC won't reflect the new size until it has a pod associated to it again.

Let's create a new pod:

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/azuredisk-csi-  
driver/master/deploy/example/nginx-pod-azuredisk.yaml
```

pod/nginx-azuredisk created

And, finally, confirm the size of the PVC and inside the pod:

```
$ kubectl get pvc pvc-azuredisk
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS
pvc-azuredisk	Bound	pvc-391ea1a6-0191-4022-b915-c8dc4216174a	15Gi	RWO	managed-csi


```
$ kubectl exec -it nginx-azuredisk -- df -h /mnt/azuredisk
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sdc	15G	46M	15G	1%	/mnt/azuredisk

Shared disk

[Azure shared disks](#) is an Azure managed disks feature that enables attaching an Azure disk to agent nodes simultaneously. Attaching a managed disk to multiple agent nodes allows you, for example, to deploy new or migrate existing clustered applications to Azure.

IMPORTANT

Currently, only raw block device (`volumeMode: Block`) is supported by the Azure disk CSI driver. Applications should manage the coordination and control of writes, reads, locks, caches, mounts, and fencing on the shared disk, which is exposed as a raw block device.

Let's create a file called `shared-disk.yaml` by copying the following command that contains the shared disk storage class and PVC:

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: managed-csi-shared
provisioner: disk.csi.azure.com
parameters:
  skuname: Premium_LRS # Currently shared disk is only available with premium SSD
  maxShares: "2"
  cachingMode: None # ReadOnly cache is not available for premium SSD with maxShares>1
reclaimPolicy: Delete
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-azuredisk-shared
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 256Gi # minimum size of shared disk is 256GB (P15)
  volumeMode: Block
  storageClassName: managed-csi-shared

```

Create the storage class with the [kubectl apply](#) command, and specify your `shared-disk.yaml` file:

```

$ kubectl apply -f shared-disk.yaml

storageclass.storage.k8s.io/managed-csi-shared created
persistentvolumeclaim/pvc-azuredisk-shared created

```

Now let's create a file called `deployment-shared.yml` by copying the following command:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx
  name: deployment-azuredisk
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      name: deployment-azuredisk
    spec:
      containers:
        - name: deployment-azuredisk
          image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
          volumeDevices:
            - name: azuredisk
              devicePath: /dev/sdx
      volumes:
        - name: azuredisk
      persistentVolumeClaim:
        claimName: pvc-azuredisk-shared

```

Create the deployment with the [kubectl apply](#) command, and specify your `deployment-shared.yml` file:

```
$ kubectl apply -f deployment-shared.yml  
  
deployment/deployment-azuredisk created
```

Finally, let's check the block device inside the pod:

```
# kubectl exec -it deployment-sharedisk-7454978bc6-xh7jp sh  
/ # dd if=/dev/zero of=/dev/sdx bs=1024k count=100  
100+0 records in  
100+0 records out/s
```

Windows containers

The Azure disk CSI driver also supports Windows nodes and containers. If you want to use Windows containers, follow the [Windows containers tutorial](#) to add a Windows node pool.

After you have a Windows node pool, you can now use the built-in storage classes like `managed-csi`. You can deploy an example [Windows-based stateful set](#) that saves timestamps into the file `data.txt` by deploying the following command with the `kubectl apply` command:

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/azuredisk-csi-  
driver/master/deploy/example/windows/statefulset.yaml  
  
statefulset.apps/busybox-azuredisk created
```

You can now validate the contents of the volume by running:

```
$ kubectl exec -it busybox-azuredisk-0 -- cat c:\\mnt\\azuredisk\\data.txt # on Linux/MacOS Bash  
$ kubectl exec -it busybox-azuredisk-0 -- cat c:\\mnt\\azuredisk\\data.txt # on Windows Powershell/CMD  
  
2020-08-27 08:13:41Z  
2020-08-27 08:13:42Z  
2020-08-27 08:13:44Z  
(...)
```

Next steps

- To learn how to use CSI drivers for Azure Files, see [Use Azure Files with CSI drivers](#).
- For more information about storage best practices, see [Best practices for storage and backups in Azure Kubernetes Service](#).

Use Azure Files Container Storage Interface (CSI) drivers in Azure Kubernetes Service (AKS) (preview)

4/28/2021 • 8 minutes to read • [Edit Online](#)

The Azure Files Container Storage Interface (CSI) driver is a [CSI specification](#)-compliant driver used by Azure Kubernetes Service (AKS) to manage the lifecycle of Azure Files shares.

The CSI is a standard for exposing arbitrary block and file storage systems to containerized workloads on Kubernetes. By adopting and using CSI, AKS now can write, deploy, and iterate plug-ins to expose new or improve existing storage systems in Kubernetes without having to touch the core Kubernetes code and wait for its release cycles.

To create an AKS cluster with CSI driver support, see [Enable CSI drivers for Azure disks and Azure Files on AKS](#).

NOTE

In-tree drivers refers to the current storage drivers that are part of the core Kubernetes code versus the new CSI drivers, which are plug-ins.

Use a persistent volume with Azure Files

A [persistent volume \(PV\)](#) represents a piece of storage that's provisioned for use with Kubernetes pods. A PV can be used by one or many pods and can be dynamically or statically provisioned. If multiple pods need concurrent access to the same storage volume, you can use Azure Files to connect by using the [Server Message Block \(SMB\) protocol](#). This article shows you how to dynamically create an Azure Files share for use by multiple pods in an AKS cluster. For static provisioning, see [Manually create and use a volume with an Azure Files share](#).

For more information on Kubernetes volumes, see [Storage options for applications in AKS](#).

IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

Dynamically create Azure Files PVs by using the built-in storage classes

A storage class is used to define how an Azure Files share is created. A storage account is automatically created in the [node resource group](#) for use with the storage class to hold the Azure Files shares. Choose one of the following [Azure storage redundancy SKUs](#) for *skuName*:

- **Standard_LRS**: Standard locally redundant storage
- **Standard_GRS**: Standard geo-redundant storage
- **Standard_ZRS**: Standard zone-redundant storage

- **Standard_RAGRS**: Standard read-access geo-redundant storage
- **Premium_LRS**: Premium locally redundant storage

NOTE

Azure Files supports Azure Premium Storage. The minimum premium file share is 100 GB.

When you use storage CSI drivers on AKS, there are two additional built-in `storageClasses` that use the Azure Files CSI storage drivers. The additional CSI storage classes are created with the cluster alongside the in-tree default storage classes.

- `azurefile-csi` : Uses Azure Standard Storage to create an Azure Files share.
- `azurefile-csi-premium` : Uses Azure Premium Storage to create an Azure Files share.

The reclaim policy on both storage classes ensures that the underlying Azure Files share is deleted when the respective PV is deleted. The storage classes also configure the file shares to be expandable, you just need to edit the persistent volume claim (PVC) with the new size.

To use these storage classes, create a [PVC](#) and respective pod that references and uses them. A PVC is used to automatically provision storage based on a storage class. A PVC can use one of the pre-created storage classes or a user-defined storage class to create an Azure Files share for the desired SKU and size. When you create a pod definition, the PVC is specified to request the desired storage.

Create an [example PVC and pod that prints the current date into an `outfile`](#) with the `kubectl apply` command:

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/azurefile-csi-driver/master/deploy/example/pvc-azurefile-csi.yaml
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/azurefile-csi-driver/master/deploy/example/nginx-pod-azurefile.yaml

persistentvolumeclaim/pvc-azurefile created
pod/nginx-azurefile created
```

After the pod is in the running state, you can validate that the file share is correctly mounted by running the following command and verifying the output contains the `outfile`:

```
$ kubectl exec nginx-azurefile -- ls -l /mnt/azurefile
total 29
-rwxrwxrwx 1 root root 29348 Aug 31 21:59 outfile
```

Create a custom storage class

The default storage classes suit the most common scenarios, but not all. For some cases, you might want to have your own storage class customized with your own parameters. For example, use the following manifest to configure the `mountOptions` of the file share.

The default value for `fileMode` and `dirMode` is `0777` for Kubernetes mounted file shares. You can specify the different mount options on the storage class object.

Create a file named `azure-file-sc.yaml`, and paste the following example manifest:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: my-azurefile
provisioner: file.csi.azure.com
reclaimPolicy: Delete
volumeBindingMode: Immediate
allowVolumeExpansion: true
mountOptions:
  - dir_mode=0640
  - file_mode=0640
  - uid=0
  - gid=0
  - mfsymlinks
  - cache=strict # https://linux.die.net/man/8/mount.cifs
  - nosharesock
parameters:
  skuName: Standard_LRS
```

Create the storage class with the [kubectl apply](#) command:

```
kubectl apply -f azure-file-sc.yaml

storageclass.storage.k8s.io/my-azurefile created
```

The Azure Files CSI driver supports creating [snapshots of persistent volumes](#) and the underlying file shares.

Create a [volume snapshot class](#) with the [kubectl apply](#) command:

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/azurefile-csi-
driver/master/deploy/example/snapshot/volumesnapshotclass-azurefile.yaml

volumesnapshotclass.snapshot.storage.k8s.io/csi-azurefile-vsc created
```

Create a [volume snapshot](#) from the PVC we dynamically created at the beginning of this tutorial, `pvc-azurefile`.

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/azurefile-csi-
driver/master/deploy/example/snapshot/volumesnapshot-azurefile.yaml

volumesnapshot.snapshot.storage.k8s.io/azurefile-volume-snapshot created
```

Verify the snapshot was created correctly:

```
$ kubectl describe volumesnapshot azurefile-volume-snapshot

Name:          azurefile-volume-snapshot
Namespace:     default
Labels:        <none>
Annotations:   API Version:  snapshot.storage.k8s.io/v1beta1
Kind:          VolumeSnapshot
Metadata:
  Creation Timestamp:  2020-08-27T22:37:41Z
  Finalizers:
    snapshot.storage.kubernetes.io/volumesnapshot-as-source-protection
    snapshot.storage.kubernetes.io/volumesnapshot-bound-protection
  Generation:      1
  Resource Version: 955091
  Self Link:       /apis/snapshot.storage.k8s.io/v1beta1/namespaces/default/volumesnapshots/azurefile-
volume-snapshot
  UID:            c359a38f-35c1-4fb1-9da9-2c06d35ca0f4
Spec:
  Source:
    Persistent Volume Claim Name: pvc-azurefile
    Volume Snapshot Class Name:    csi-azurefile-vsc
Status:
  Bound Volume Snapshot Content Name: snapcontent-c359a38f-35c1-4fb1-9da9-2c06d35ca0f4
  Ready To Use:                 false
  Events:                      <none>
```

Resize a persistent volume

You can request a larger volume for a PVC. Edit the PVC object, and specify a larger size. This change triggers the expansion of the underlying volume that backs the PV.

NOTE

A new PV is never created to satisfy the claim. Instead, an existing volume is resized.

In AKS, the built-in `azurefile-csi` storage class already supports expansion, so use the [PVC created earlier with this storage class](#). The PVC requested a 100Gi file share. We can confirm that by running:

```
$ kubectl exec -it nginx-azurefile -- df -h /mnt/azurefile
Filesystem                                Size  Used  Avail
Use% Mounted on
//f149b5a219bd34caeb07de9.file.core.windows.net/pvc-5e5d9980-da38-492b-8581-17e3cad01770  100G  128K  100G
1% /mnt/azurefile
```

Expand the PVC by increasing the `spec.resources.requests.storage` field:

```
$ kubectl patch pvc pvc-azurefile --type merge --patch '{"spec": {"resources": {"requests": {"storage": "200Gi"}}}}'
persistentvolumeclaim/pvc-azurefile patched
```

Verify that both the PVC and the file system inside the pod show the new size:

```
$ kubectl get pvc pvc-azurefile
NAME           STATUS    VOLUME
AGE
pvc-azurefile  Bound     pvc-5e5d9980-da38-492b-8581-17e3cad01770
64m

$ kubectl exec -it nginx-azurefile -- df -h /mnt/azurefile
Filesystem      Size  Used Avail
Use% Mounted on
//f149b5a219bd34caeb07de9.file.core.windows.net/pvc-5e5d9980-da38-492b-8581-17e3cad01770  200G  128K  200G
1% /mnt/azurefile
```

NFS file shares

Azure Files now has support for NFS v4.1 protocol. NFS 4.1 support for Azure Files provides you with a fully managed NFS file system as a service built on a highly available and highly durable distributed resilient storage platform.

This option is optimized for random access workloads with in-place data updates and provides full POSIX file system support. This section shows you how to use NFS shares with the Azure File CSI driver on an AKS cluster.

Make sure to check the [limitations](#) and [region availability](#) during the preview phase.

Register the `AllowNfsFileShares` preview feature

To create a file share that leverages NFS 4.1, you must enable the `AllowNfsFileShares` feature flag on your subscription.

Register the `AllowNfsFileShares` feature flag by using the [az feature register](#) command, as shown in the following example:

```
az feature register --namespace "Microsoft.Storage" --name "AllowNfsFileShares"
```

It takes a few minutes for the status to show *Registered*. Verify the registration status by using the [az feature list](#) command:

```
az feature list -o table --query "[?contains(name, 'Microsoft.Storage/AllowNfsFileShares')].
{Name:name,State:properties.state}"
```

When ready, refresh the registration of the *Microsoft.Storage* resource provider by using the [az provider register](#) command:

```
az provider register --namespace Microsoft.Storage
```

Create a storage account for the NFS file share

Create a [Premium_LRS](#) Azure storage account with following configurations to support NFS shares:

- account kind: FileStorage
- secure transfer required(enable HTTPS traffic only): false
- select the virtual network of your agent nodes in Firewalls and virtual networks - so you might prefer to create the Storage Account in the MC_ resource group.

Create NFS file share storage class

Save a `nfs-sc.yaml` file with the manifest below editing the respective placeholders.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: azurefile-csi-nfs
provisioner: file.csi.azure.com
parameters:
  resourceGroup: EXISTING_RESOURCE_GROUP_NAME # optional, required only when storage account is not in the same resource group as your agent nodes
  storageAccount: EXISTING_STORAGE_ACCOUNT_NAME
  protocol: nfs
```

After editing and saving the file, create the storage class with the [kubectl apply](#) command:

```
$ kubectl apply -f nfs-sc.yaml

storageclass.storage.k8s.io/azurefile-csi created
```

Create a deployment with an NFS-backed file share

You can deploy an example [stateful set](#) that saves timestamps into a file `data.txt` by deploying the following command with the [kubectl apply](#) command:

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/azurefile-csi-driver/master/deploy/example/statefulset.yaml

statefulset.apps/statefulset-azurefile created
```

Validate the contents of the volume by running:

```
$ kubectl exec -it statefulset-azurefile-0 -- df -h

Filesystem      Size  Used Avail Use% Mounted on
...
/dev/sda1              29G   11G   19G
37% /etc/hosts
accountname.file.core.windows.net:/accountname/pvc-fa72ec43-ae64-42e4-a8a2-556606f5da38  100G     0  100G
0% /mnt/azurefile
...
...
```

NOTE

Note that since NFS file share is in Premium account, the minimum file share size is 100GB. If you create a PVC with a small storage size, you might encounter an error "failed to create file share ... size (5)...".

Windows containers

The Azure Files CSI driver also supports Windows nodes and containers. If you want to use Windows containers, follow the [Windows containers tutorial](#) to add a Windows node pool.

After you have a Windows node pool, use the built-in storage classes like `azurefile-csi` or create custom ones. You can deploy an example [Windows-based stateful set](#) that saves timestamps into a file `data.txt` by deploying the following command with the [kubectl apply](#) command:

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/azurefile-csi-driver/master/deploy/example/windows/statefulset.yaml

statefulset.apps/busybox-azurefile created
```

Validate the contents of the volume by running:

```
$ kubectl exec -it busybox-azurefile-0 -- cat c:\\mnt\\azurefile\\data.txt # on Linux/MacOS Bash
$ kubectl exec -it busybox-azurefile-0 -- cat c:\\mnt\\azurefile\\data.txt # on Windows Powershell/CMD

2020-08-27 22:11:01Z
2020-08-27 22:11:02Z
2020-08-27 22:11:04Z
(...)
```

Next steps

- To learn how to use CSI drivers for Azure disks, see [Use Azure disks with CSI drivers](#).
- For more about storage best practices, see [Best practices for storage and backups in Azure Kubernetes Service](#).

Use kubenet networking with your own IP address ranges in Azure Kubernetes Service (AKS)

4/21/2021 • 13 minutes to read • [Edit Online](#)

By default, AKS clusters use [kubenet](#), and an Azure virtual network and subnet are created for you. With *kubenet*, nodes get an IP address from the Azure virtual network subnet. Pods receive an IP address from a logically different address space to the Azure virtual network subnet of the nodes. Network address translation (NAT) is then configured so that the pods can reach resources on the Azure virtual network. The source IP address of the traffic is NAT'd to the node's primary IP address. This approach greatly reduces the number of IP addresses that you need to reserve in your network space for pods to use.

With [Azure Container Networking Interface \(CNI\)](#), every pod gets an IP address from the subnet and can be accessed directly. These IP addresses must be unique across your network space, and must be planned in advance. Each node has a configuration parameter for the maximum number of pods that it supports. The equivalent number of IP addresses per node are then reserved up front for that node. This approach requires more planning, and often leads to IP address exhaustion or the need to rebuild clusters in a larger subnet as your application demands grow. You can configure the maximum pods deployable to a node at cluster create time or when creating new node pools. If you don't specify maxPods when creating new node pools, you receive a default value of 110 for kubenet.

This article shows you how to use *kubenet* networking to create and use a virtual network subnet for an AKS cluster. For more information on network options and considerations, see [Network concepts for Kubernetes and AKS](#).

Prerequisites

- The virtual network for the AKS cluster must allow outbound internet connectivity.
- Don't create more than one AKS cluster in the same subnet.
- AKS clusters may not use `169.254.0.0/16`, `172.30.0.0/16`, `172.31.0.0/16`, or `192.0.2.0/24` for the Kubernetes service address range, pod address range or cluster virtual network address range.
- The cluster identity used by the AKS cluster must have at least [Network Contributor](#) role on the subnet within your virtual network. You must also have the appropriate permissions, such as the subscription owner, to create a cluster identity and assign it permissions. If you wish to define a [custom role](#) instead of using the built-in Network Contributor role, the following permissions are required:
 - `Microsoft.Network/virtualNetworks/subnets/join/action`
 - `Microsoft.Network/virtualNetworks/subnets/read`

WARNING

To use Windows Server node pools, you must use Azure CNI. The use of kubenet as the network model is not available for Windows Server containers.

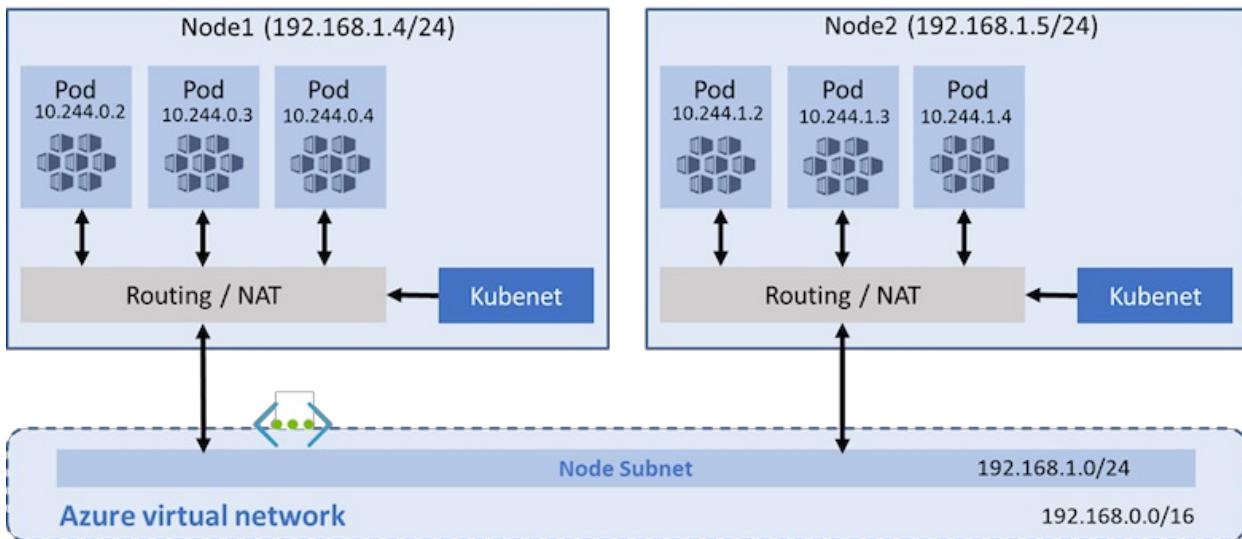
Before you begin

You need the Azure CLI version 2.0.65 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Overview of kubenet networking with your own subnet

In many environments, you have defined virtual networks and subnets with allocated IP address ranges. These virtual network resources are used to support multiple services and applications. To provide network connectivity, AKS clusters can use *kubenet* (basic networking) or Azure CNI (advanced networking).

With *kubenet*, only the nodes receive an IP address in the virtual network subnet. Pods can't communicate directly with each other. Instead, User Defined Routing (UDR) and IP forwarding is used for connectivity between pods across nodes. By default, UDRs and IP forwarding configuration is created and maintained by the AKS service, but you have the option to [bring your own route table for custom route management](#). You could also deploy pods behind a service that receives an assigned IP address and load balances traffic for the application. The following diagram shows how the AKS nodes receive an IP address in the virtual network subnet, but not the pods:



Azure supports a maximum of 400 routes in a UDR, so you can't have an AKS cluster larger than 400 nodes. AKS [Virtual Nodes](#) and Azure Network Policies aren't supported with *kubenet*. You can use [Calico Network Policies](#), as they are supported with *kubenet*.

With *Azure CNI*, each pod receives an IP address in the IP subnet, and can directly communicate with other pods and services. Your clusters can be as large as the IP address range you specify. However, the IP address range must be planned in advance, and all of the IP addresses are consumed by the AKS nodes based on the maximum number of pods that they can support. Advanced network features and scenarios such as [Virtual Nodes](#) or Network Policies (either Azure or Calico) are supported with *Azure CNI*.

Limitations & considerations for kubenet

- An additional hop is required in the design of kubenet, which adds minor latency to pod communication.
- Route tables and user-defined routes are required for using kubenet, which adds complexity to operations.
- Direct pod addressing isn't supported for kubenet due to kubenet design.
- Unlike Azure CNI clusters, multiple kubenet clusters can't share a subnet.
- Features **not supported on kubenet** include:
 - [Azure network policies](#), but Calico network policies are supported on kubenet
 - [Windows node pools](#)
 - [Virtual nodes add-on](#)

IP address availability and exhaustion

With *Azure CNI*, a common issue is the assigned IP address range is too small to then add additional nodes when you scale or upgrade a cluster. The network team may also not be able to issue a large enough IP address range to support your expected application demands.

As a compromise, you can create an AKS cluster that uses *kubenet* and connect to an existing virtual network subnet. This approach lets the nodes receive defined IP addresses, without the need to reserve a large number of IP addresses up front for all of the potential pods that could run in the cluster.

With *kubenet*, you can use a much smaller IP address range and be able to support large clusters and application demands. For example, even with a /27 IP address range on your subnet, you could run a 20-25 node cluster with enough room to scale or upgrade. This cluster size would support up to 2,200-2,750 pods (with a default maximum of 110 pods per node). The maximum number of pods per node that you can configure with *kubenet* in AKS is 110.

The following basic calculations compare the difference in network models:

- **kubenet** - a simple /24 IP address range can support up to 251 nodes in the cluster (each Azure virtual network subnet reserves the first three IP addresses for management operations)
 - This node count could support up to 27,610 pods (with a default maximum of 110 pods per node with *kubenet*)
- **Azure CNI** - that same basic /24 subnet range could only support a maximum of 8 nodes in the cluster
 - This node count could only support up to 240 pods (with a default maximum of 30 pods per node with *Azure CNI*)

NOTE

These maximums don't take into account upgrade or scale operations. In practice, you can't run the maximum number of nodes that the subnet IP address range supports. You must leave some IP addresses available for use during scale or upgrade operations.

Virtual network peering and ExpressRoute connections

To provide on-premises connectivity, both *kubenet* and *Azure-CNI* network approaches can use [Azure virtual network peering](#) or [ExpressRoute connections](#). Plan your IP address ranges carefully to prevent overlap and incorrect traffic routing. For example, many on-premises networks use a 10.0.0.0/8 address range that is advertised over the ExpressRoute connection. It's recommended to create your AKS clusters into Azure virtual network subnets outside of this address range, such as 172.16.0.0/16.

Choose a network model to use

The choice of which network plugin to use for your AKS cluster is usually a balance between flexibility and advanced configuration needs. The following considerations help outline when each network model may be the most appropriate.

Use *kubenet* when:

- You have limited IP address space.
- Most of the pod communication is within the cluster.
- You don't need advanced AKS features such as virtual nodes or Azure Network Policy. Use [Calico network policies](#).

Use *Azure CNI* when:

- You have available IP address space.
- Most of the pod communication is to resources outside of the cluster.
- You don't want to manage user defined routes for pod connectivity.
- You need AKS advanced features such as virtual nodes or Azure Network Policy. Use [Calico network policies](#).

For more information to help you decide which network model to use, see [Compare network models and their support scope](#).

Create a virtual network and subnet

To get started with using *kubenet* and your own virtual network subnet, first create a resource group using the [az group create](#) command. The following example creates a resource group named *myResourceGroup* in the *eastus* location:

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

If you don't have an existing virtual network and subnet to use, create these network resources using the [az network vnet create](#) command. In the following example, the virtual network is named *myVnet* with the address prefix of *192.168.0.0/16*. A subnet is created named *myAKSSubnet* with the address prefix *192.168.1.0/24*.

```
az network vnet create \
--resource-group myResourceGroup \
--name myAKSVnet \
--address-prefixes 192.168.0.0/16 \
--subnet-name myAKSSubnet \
--subnet-prefix 192.168.1.0/24
```

Create a service principal and assign permissions

To allow an AKS cluster to interact with other Azure resources, an Azure Active Directory service principal is used. The service principal needs to have permissions to manage the virtual network and subnet that the AKS nodes use. To create a service principal, use the [az ad sp create-for-rbac](#) command:

```
az ad sp create-for-rbac --skip-assignment
```

The following example output shows the application ID and password for your service principal. These values are used in additional steps to assign a role to the service principal and then create the AKS cluster:

```
{
  "appId": "476b3636-5eda-4c0e-9751-849e70b5cfad",
  "displayName": "azure-cli-2019-01-09-22-29-24",
  "name": "http://azure-cli-2019-01-09-22-29-24",
  "password": "a1024cd7-af7b-469f-8fd7-b293ecbb174e",
  "tenant": "72f998bf-85f1-41cf-92ab-2e7cd014db46"
}
```

To assign the correct delegations in the remaining steps, use the [az network vnet show](#) and [az network vnet subnet show](#) commands to get the required resource IDs. These resource IDs are stored as variables and referenced in the remaining steps:

```
VNET_ID=$(az network vnet show --resource-group myResourceGroup --name myAKSVnet --query id -o tsv)
SUBNET_ID=$(az network vnet subnet show --resource-group myResourceGroup --vnet-name myAKSVnet --name myAKSSubnet --query id -o tsv)
```

Now assign the service principal for your AKS cluster *Network Contributor* permissions on the virtual network using the [az role assignment create](#) command. Provide your own <appId> as shown in the output from the previous command to create the service principal:

```
az role assignment create --assignee <appId> --scope $VNET_ID --role "Network Contributor"
```

Create an AKS cluster in the virtual network

You've now created a virtual network and subnet, and created and assigned permissions for a service principal to use those network resources. Now create an AKS cluster in your virtual network and subnet using the `az aks create` command. Define your own service principal `<appId>` and `<password>`, as shown in the output from the previous command to create the service principal.

The following IP address ranges are also defined as part of the cluster create process:

- The `--service-cidr` is used to assign internal services in the AKS cluster an IP address. This IP address range should be an address space that isn't in use elsewhere in your network environment, including any on-premises network ranges if you connect, or plan to connect, your Azure virtual networks using Express Route or a Site-to-Site VPN connection.
- The `--dns-service-ip` address should be the `.10` address of your service IP address range.
- The `--pod-cidr` should be a large address space that isn't in use elsewhere in your network environment. This range includes any on-premises network ranges if you connect, or plan to connect, your Azure virtual networks using Express Route or a Site-to-Site VPN connection.
 - This address range must be large enough to accommodate the number of nodes that you expect to scale up to. You can't change this address range once the cluster is deployed if you need more addresses for additional nodes.
 - The pod IP address range is used to assign a `/24` address space to each node in the cluster. In the following example, the `--pod-cidr` of `10.244.0.0/16` assigns the first node `10.244.0.0/24`, the second node `10.244.1.0/24`, and the third node `10.244.2.0/24`.
 - As the cluster scales or upgrades, the Azure platform continues to assign a pod IP address range to each new node.
- The `--docker-bridge-address` lets the AKS nodes communicate with the underlying management platform. This IP address must not be within the virtual network IP address range of your cluster, and shouldn't overlap with other address ranges in use on your network.

```
az aks create \
    --resource-group myResourceGroup \
    --name myAKScluster \
    --node-count 3 \
    --network-plugin kubenet \
    --service-cidr 10.0.0.0/16 \
    --dns-service-ip 10.0.0.10 \
    --pod-cidr 10.244.0.0/16 \
    --docker-bridge-address 172.17.0.1/16 \
    --vnet-subnet-id $SUBNET_ID \
    --service-principal <appId> \
    --client-secret <password>
```

NOTE

If you wish to enable an AKS cluster to include a [Calico network policy](#) you can use the following command.

```
az aks create \
    --resource-group myResourceGroup \
    --name myAKScluster \
    --node-count 3 \
    --network-plugin kubenet --network-policy calico \
    --service-cidr 10.0.0.0/16 \
    --dns-service-ip 10.0.0.10 \
    --pod-cidr 10.244.0.0/16 \
    --docker-bridge-address 172.17.0.1/16 \
    --vnet-subnet-id $SUBNET_ID \
    --service-principal <appId> \
    --client-secret <password>
```

When you create an AKS cluster, a network security group and route table are automatically created. These network resources are managed by the AKS control plane. The network security group is automatically associated with the virtual NICs on your nodes. The route table is automatically associated with the virtual network subnet. Network security group rules and route tables are automatically updated as you create and expose services.

Bring your own subnet and route table with kubenet

With kubenet, a route table must exist on your cluster subnet(s). AKS supports bringing your own existing subnet and route table.

If your custom subnet does not contain a route table, AKS creates one for you and adds rules to it throughout the cluster lifecycle. If your custom subnet contains a route table when you create your cluster, AKS acknowledges the existing route table during cluster operations and adds/updates rules accordingly for cloud provider operations.

WARNING

Custom rules can be added to the custom route table and updated. However, rules are added by the Kubernetes cloud provider which must not be updated or removed. Rules such as 0.0.0.0/0 must always exist on a given route table and map to the target of your internet gateway, such as an NVA or other egress gateway. Take caution when updating rules that only your custom rules are being modified.

Learn more about setting up a [custom route table](#).

Kubenet networking requires organized route table rules to successfully route requests. Due to this design, route tables must be carefully maintained for each cluster which relies on it. Multiple clusters cannot share a route table because pod CIDRs from different clusters may overlap which causes unexpected and broken routing. When configuring multiple clusters on the same virtual network or dedicating a virtual network to each cluster, ensure the following limitations are considered.

Limitations:

- Permissions must be assigned before cluster creation, ensure you are using a service principal with write permissions to your custom subnet and custom route table.
- A custom route table must be associated to the subnet before you create the AKS cluster.
- The associated route table resource cannot be updated after cluster creation. While the route table resource cannot be updated, custom rules can be modified on the route table.
- Each AKS cluster must use a single, unique route table for all subnets associated with the cluster. You cannot reuse a route table with multiple clusters due to the potential for overlapping pod CIDRs and conflicting routing rules.

After you create a custom route table and associate it to your subnet in your virtual network, you can create a

new AKS cluster that uses your route table. You need to use the subnet ID for where you plan to deploy your AKS cluster. This subnet also must be associated with your custom route table.

```
# Find your subnet ID
az network vnet subnet list --resource-group
    --vnet-name
    [--subscription]
```

```
# Create a kubernetes cluster with with a custom subnet preconfigured with a route table
az aks create -g MyResourceGroup -n MyManagedCluster --vnet-subnet-id MySubnetID
```

Next steps

With an AKS cluster deployed into your existing virtual network subnet, you can now use the cluster as normal. Get started with [creating new apps using Helm](#) or [deploy existing apps using Helm](#).

Configure Azure CNI networking in Azure Kubernetes Service (AKS)

4/21/2021 • 19 minutes to read • [Edit Online](#)

By default, AKS clusters use [kubenet](#), and a virtual network and subnet are created for you. With *kubenet*, nodes get an IP address from a virtual network subnet. Network address translation (NAT) is then configured on the nodes, and pods receive an IP address "hidden" behind the node IP. This approach reduces the number of IP addresses that you need to reserve in your network space for pods to use.

With [Azure Container Networking Interface \(CNI\)](#), every pod gets an IP address from the subnet and can be accessed directly. These IP addresses must be unique across your network space, and must be planned in advance. Each node has a configuration parameter for the maximum number of pods that it supports. The equivalent number of IP addresses per node are then reserved up front for that node. This approach requires more planning, and often leads to IP address exhaustion or the need to rebuild clusters in a larger subnet as your application demands grow.

This article shows you how to use *Azure CNI* networking to create and use a virtual network subnet for an AKS cluster. For more information on network options and considerations, see [Network concepts for Kubernetes and AKS](#).

Prerequisites

- The virtual network for the AKS cluster must allow outbound internet connectivity.
- AKS clusters may not use `169.254.0.0/16`, `172.30.0.0/16`, `172.31.0.0/16`, or `192.0.2.0/24` for the Kubernetes service address range, pod address range, or cluster virtual network address range.
- The cluster identity used by the AKS cluster must have at least [Network Contributor](#) permissions on the subnet within your virtual network. If you wish to define a [custom role](#) instead of using the built-in Network Contributor role, the following permissions are required:
 - `Microsoft.Network/virtualNetworks/subnets/join/action`
 - `Microsoft.Network/virtualNetworks/subnets/read`
- The subnet assigned to the AKS node pool cannot be a [delegated subnet](#).

Plan IP addressing for your cluster

Clusters configured with Azure CNI networking require additional planning. The size of your virtual network and its subnet must accommodate the number of pods you plan to run and the number of nodes for the cluster.

IP addresses for the pods and the cluster's nodes are assigned from the specified subnet within the virtual network. Each node is configured with a primary IP address. By default, 30 additional IP addresses are pre-configured by Azure CNI that are assigned to pods scheduled on the node. When you scale out your cluster, each node is similarly configured with IP addresses from the subnet. You can also view the [maximum pods per node](#).

IMPORTANT

The number of IP addresses required should include considerations for upgrade and scaling operations. If you set the IP address range to only support a fixed number of nodes, you cannot upgrade or scale your cluster.

- When you **upgrade** your AKS cluster, a new node is deployed into the cluster. Services and workloads begin to run on the new node, and an older node is removed from the cluster. This rolling upgrade process requires a minimum of one additional block of IP addresses to be available. Your node count is then $n + 1$.
 - This consideration is particularly important when you use Windows Server node pools. Windows Server nodes in AKS do not automatically apply Windows Updates, instead you perform an upgrade on the node pool. This upgrade deploys new nodes with the latest Window Server 2019 base node image and security patches. For more information on upgrading a Windows Server node pool, see [Upgrade a node pool in AKS](#).
- When you **scale** an AKS cluster, a new node is deployed into the cluster. Services and workloads begin to run on the new node. Your IP address range needs to take into considerations how you may want to scale up the number of nodes and pods your cluster can support. One additional node for upgrade operations should also be included. Your node count is then $n + \text{number-of-additional-scaled-nodes-you-anticipate} + 1$.

If you expect your nodes to run the maximum number of pods, and regularly destroy and deploy pods, you should also factor in some additional IP addresses per node. These additional IP addresses take into consideration it may take a few seconds for a service to be deleted and the IP address released for a new service to be deployed and acquire the address.

The IP address plan for an AKS cluster consists of a virtual network, at least one subnet for nodes and pods, and a Kubernetes service address range.

ADDRESS RANGE / AZURE RESOURCE	LIMITS AND SIZING
Virtual network	The Azure virtual network can be as large as /8, but is limited to 65,536 configured IP addresses. Consider all your networking needs, including communicating with services in other virtual networks, before configuring your address space. For example, if you configure too large of an address space, you may run into issues with overlapping other address spaces within your network.

ADDRESS RANGE / AZURE RESOURCE	LIMITS AND SIZING
Subnet	<p>Must be large enough to accommodate the nodes, pods, and all Kubernetes and Azure resources that might be provisioned in your cluster. For example, if you deploy an internal Azure Load Balancer, its front-end IPs are allocated from the cluster subnet, not public IPs. The subnet size should also take into account upgrade operations or future scaling needs.</p> <p>To calculate the <i>minimum</i> subnet size including an additional node for upgrade operations:</p> <div style="border: 1px solid #ccc; padding: 5px; margin-left: 20px;">$(\text{number of nodes} + 1) + ((\text{number of nodes} + 1) * \text{maximum pods per node that you configure})$</div> <p>Example for a 50 node cluster:</p> <div style="border: 1px solid #ccc; padding: 5px; margin-left: 20px;">$(51) + (51 * 30 \text{ (default)}) = 1,581$ (/21 or larger)</div> <p>Example for a 50 node cluster that also includes provision to scale up an additional 10 nodes:</p> <div style="border: 1px solid #ccc; padding: 5px; margin-left: 20px;">$(61) + (61 * 30 \text{ (default)}) = 1,891$ (/21 or larger)</div> <p>If you don't specify a maximum number of pods per node when you create your cluster, the maximum number of pods per node is set to <i>30</i>. The minimum number of IP addresses required is based on that value. If you calculate your minimum IP address requirements on a different maximum value, see how to configure the maximum number of pods per node to set this value when you deploy your cluster.</p>
Kubernetes service address range	This range should not be used by any network element on or connected to this virtual network. Service address CIDR must be smaller than /12. You can reuse this range across different AKS clusters.
Kubernetes DNS service IP address	IP address within the Kubernetes service address range that will be used by cluster service discovery. Don't use the first IP address in your address range, such as .1. The first address in your subnet range is used for the <i>kubernetes.default.svc.cluster.local</i> /address.
Docker bridge address	The Docker bridge network address represents the default <i>docker0</i> bridge network address present in all Docker installations. While <i>docker0</i> bridge is not used by AKS clusters or the pods themselves, you must set this address to continue to support scenarios such as <i>docker build</i> within the AKS cluster. It is required to select a CIDR for the Docker bridge network address because otherwise Docker will pick a subnet automatically, which could conflict with other CIDRs. You must pick an address space that does not collide with the rest of the CIDRs on your networks, including the cluster's service CIDR and pod CIDR. Default of 172.17.0.1/16. You can reuse this range across different AKS clusters.

Maximum pods per node

The maximum number of pods per node in an AKS cluster is 250. The *default* maximum number of pods per node varies between *kubenet* and *Azure CNI* networking, and the method of cluster deployment.

DEPLOYMENT METHOD	KUBENET DEFAULT	AZURE CNI DEFAULT	CONFIGURABLE AT DEPLOYMENT
Azure CLI	110	30	Yes (up to 250)
Resource Manager template	110	30	Yes (up to 250)
Portal	110	110 (configured in the Node Pools tab)	No

Configure maximum - new clusters

You're able to configure the maximum number of pods per node at cluster deployment time or as you add new node pools. If you deploy with the Azure CLI or with a Resource Manager template, you can set the maximum pods per node value as high as 250.

If you don't specify maxPods when creating new node pools, you receive a default value of 30 for Azure CNI.

A minimum value for maximum pods per node is enforced to guarantee space for system pods critical to cluster health. The minimum value that can be set for maximum pods per node is 10 if and only if the configuration of each node pool has space for a minimum of 30 pods. For example, setting the maximum pods per node to the minimum of 10 requires each individual node pool to have a minimum of 3 nodes. This requirement applies for each new node pool created as well, so if 10 is defined as maximum pods per node each subsequent node pool added must have at least 3 nodes.

NETWORKING	MINIMUM	MAXIMUM
Azure CNI	10	250
Kubenet	10	110

NOTE

The minimum value in the table above is strictly enforced by the AKS service. You can not set a maxPods value lower than the minimum shown as doing so can prevent the cluster from starting.

- **Azure CLI:** Specify the `--max-pods` argument when you deploy a cluster with the [az aks create](#) command. The maximum value is 250.
- **Resource Manager template:** Specify the `maxPods` property in the [ManagedClusterAgentPoolProfile](#) object when you deploy a cluster with a Resource Manager template. The maximum value is 250.
- **Azure portal:** You can't change the maximum number of pods per node when you deploy a cluster with the Azure portal. Azure CNI networking clusters are limited to 30 pods per node when you deploy using the Azure portal.

Configure maximum - existing clusters

The maxPod per node setting can be defined when you create a new node pool. If you need to increase the maxPod per node setting on an existing cluster, add a new node pool with the new desired maxPod count. After migrating your pods to the new pool, delete the older pool. To delete any older pool in a cluster, ensure you are setting node pool modes as defined in the [system node pools document](#).

Deployment parameters

When you create an AKS cluster, the following parameters are configurable for Azure CNI networking:

Virtual network: The virtual network into which you want to deploy the Kubernetes cluster. If you want to create a new virtual network for your cluster, select *Create new* and follow the steps in the *Create virtual network* section. For information about the limits and quotas for an Azure virtual network, see [Azure subscription and service limits, quotas, and constraints](#).

Subnet: The subnet within the virtual network where you want to deploy the cluster. If you want to create a new subnet in the virtual network for your cluster, select *Create new* and follow the steps in the *Create subnet* section. For hybrid connectivity, the address range shouldn't overlap with any other virtual networks in your environment.

Azure Network Plugin: When Azure network plugin is used, the internal LoadBalancer service with "externalTrafficPolicy=Local" can't be accessed from VMs with an IP in clusterCIDR that does not belong to AKS cluster.

Kubernetes service address range: This parameter is the set of virtual IPs that Kubernetes assigns to internal [services](#) in your cluster. You can use any private address range that satisfies the following requirements:

- Must not be within the virtual network IP address range of your cluster
- Must not overlap with any other virtual networks with which the cluster virtual network peers
- Must not overlap with any on-premises IPs
- Must not be within the ranges `169.254.0.0/16`, `172.30.0.0/16`, `172.31.0.0/16`, or `192.0.2.0/24`

Although it's technically possible to specify a service address range within the same virtual network as your cluster, doing so is not recommended. Unpredictable behavior can result if overlapping IP ranges are used. For more information, see the [FAQ](#) section of this article. For more information on Kubernetes services, see [Services](#) in the Kubernetes documentation.

Kubernetes DNS service IP address: The IP address for the cluster's DNS service. This address must be within the *Kubernetes service address range*. Don't use the first IP address in your address range, such as .1. The first address in your subnet range is used for the `kubernetes.default.svc.cluster.local` address.

Docker Bridge address: The Docker bridge network address represents the default `docker0` bridge network address present in all Docker installations. While `docker0` bridge is not used by AKS clusters or the pods themselves, you must set this address to continue to support scenarios such as `docker build` within the AKS cluster. It is required to select a CIDR for the Docker bridge network address because otherwise Docker will pick a subnet automatically which could conflict with other CIDRs. You must pick an address space that does not collide with the rest of the CIDRs on your networks, including the cluster's service CIDR and pod CIDR.

Configure networking - CLI

When you create an AKS cluster with the Azure CLI, you can also configure Azure CNI networking. Use the following commands to create a new AKS cluster with Azure CNI networking enabled.

First, get the subnet resource ID for the existing subnet into which the AKS cluster will be joined:

```
$ az network vnet subnet list \
  --resource-group myVnet \
  --vnet-name myVnet \
  --query "[0].id" --output tsv

/subscriptions/<guid>/resourceGroups/myVnet/providers/Microsoft.Network/virtualNetworks/myVnet/subnets/defau
lt
```

Use the `az aks create` command with the `--network-plugin azure` argument to create a cluster with advanced networking. Update the `--vnet-subnet-id` value with the subnet ID collected in the previous step:

```
az aks create \
--resource-group myResourceGroup \
--name myAKSCluster \
--network-plugin azure \
--vnet-subnet-id <subnet-id> \
--docker-bridge-address 172.17.0.1/16 \
--dns-service-ip 10.2.0.10 \
--service-cidr 10.2.0.0/24 \
--generate-ssh-keys
```

Configure networking - portal

The following screenshot from the Azure portal shows an example of configuring these settings during AKS cluster creation:

The screenshot shows the 'Create Kubernetes cluster' page in the Azure portal. The 'Networking' tab is selected. It provides information about networking options: "Basic" sets up a simple default config with a VNet and internal IP addresses, while "Advanced" allows configuring your own VNet for automatic connectivity. The "Advanced" option is selected and highlighted with a red box. The form includes fields for Virtual network (set to 'aks-vnet-16033614'), Subnet ('aks-subnet'), Kubernetes service address range ('10.0.0.0/16'), Kubernetes DNS service IP address ('10.0.0.10'), and Docker Bridge address ('172.17.0.1/16'). All fields have green checkmarks indicating they are valid.

Dynamic allocation of IPs and enhanced subnet support (preview)

IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

NOTE

This preview feature is currently available in the following regions:

- West Central US

A drawback with the traditional CNI is the exhaustion of pod IP addresses as the AKS cluster grows, resulting in the need to rebuild the entire cluster in a bigger subnet. The new dynamic IP allocation capability in Azure CNI solves this problem by allotting pod IPs from a subnet separate from the subnet hosting the AKS cluster. It offers the following benefits:

- **Better IP utilization:** IPs are dynamically allocated to cluster Pods from the Pod subnet. This leads to better utilization of IPs in the cluster compared to the traditional CNI solution, which does static allocation of IPs for every node.
- **Scalable and flexible:** Node and pod subnets can be scaled independently. A single pod subnet can be shared across multiple node pools of a cluster or across multiple AKS clusters deployed in the same VNet. You can also configure a separate pod subnet for a node pool.
- **High performance:** Since pods have direct connectivity to other cluster pods and resources in the VNet. The solution supports very large clusters without any degradation in performance.
- **Separate VNet policies for pods:** Since pods have a separate subnet, you can configure separate VNet policies for them that are different from node policies. This enables many useful scenarios such as allowing internet connectivity only for pods and not for nodes, fixing the source IP for pod in a node pool using a VNet Network NAT, and using NSGs to filter traffic between node pools.
- **Kubernetes network policies:** Both the Azure Network Policies and Calico work with this new solution.

Install the `aks-preview` Azure CLI

You will need the `aks-preview` Azure CLI extension. Install the `aks-preview` Azure CLI extension by using the [az extension add](#) command. Or install any available updates by using the [az extension update](#) command.

```
# Install the aks-preview extension
az extension add --name aks-preview

# Update the extension to make sure you have the latest version installed
az extension update --name aks-preview
```

Register the `PodSubnetPreview` preview feature

To use the feature, you must also enable the `PodSubnetPreview` feature flag on your subscription.

Register the `PodSubnetPreview` feature flag by using the [az feature register](#) command, as shown in the following example:

```
az feature register --namespace "Microsoft.ContainerService" --name "PodSubnetPreview"
```

It takes a few minutes for the status to show *Registered*. Verify the registration status by using the [az feature list](#) command:

```
az feature list -o table --query "[?contains(name, 'Microsoft.ContainerService/PodSubnetPreview')].{Name:name,State:properties.state}"
```

When ready, refresh the registration of the *Microsoft.ContainerService* resource provider by using the [az provider register](#) command:

```
az provider register --namespace Microsoft.ContainerService
```

Additional prerequisites

The prerequisites already listed for Azure CNI still apply, but there are a few additional limitations:

- Only linux node clusters and node pools are supported.
- AKS Engine and DIY clusters are not supported.

Planning IP addressing

When using this feature, planning is much simpler. Since the nodes and pods scale independently, their address spaces can also be planned separately. Since pod subnets can be configured to the granularity of a node pool, customers can always add a new subnet when they add a node pool. The system pods in a cluster/node pool also receive IPs from the pod subnet, so this behavior needs to be accounted for.

The planning of IPs for K8S services and Docker bridge remain unchanged.

Maximum pods per node in a cluster with dynamic allocation of IPs and enhanced subnet support

The pods per node values when using Azure CNI with dynamic allocation of IPs have changed slightly from the traditional CNI behavior:

CNI	DEPLOYMENT METHOD	DEFAULT	CONFIGURABLE AT DEPLOYMENT
Traditional Azure CNI	Azure CLI	30	Yes (up to 250)
Azure CNI with dynamic allocation of IPs	Azure CLI	250	Yes (up to 250)

All other guidance related to configuring the maximum nodes per pod remains the same.

Additional deployment parameters

The deployment parameters described above are all still valid, with one exception:

- The **subnet** parameter now refers to the subnet related to the cluster's nodes.
- An additional parameter **pod subnet** is used to specify the subnet whose IP addresses will be dynamically allocated to pods.

Configure networking - CLI with dynamic allocation of IPs and enhanced subnet support

Using dynamic allocation of IPs and enhanced subnet support in your cluster is similar to the default method for configuring a cluster Azure CNI. The following example walks through creating a new virtual network with a subnet for nodes and a subnet for pods, and creating a cluster that uses Azure CNI with dynamic allocation of IPs and enhanced subnet support. Be sure to replace variables such as `$subscription` with your own values:

First, create the virtual network with two subnets:

```

$resourceGroup="myResourceGroup"
$vnet="myVirtualNetwork"

# Create our two subnet network
az network vnet create -g $rg --name $vnet --address-prefixes 10.0.0.0/8 -o none
az network vnet subnet create -g $rg --vnet-name $vnet --name nodesubnet --address-prefixes 10.240.0.0/16 -o none
az network vnet subnet create -g $rg --vnet-name $vnet --name podsubnet --address-prefixes 10.241.0.0/16 -o none

```

Then, create the cluster, referencing the node subnet using `--vnet-subnet-id` and the pod subnet using `--pod-subnet-id`:

```

$clusterName="myAKSCluster"
$location="eastus"
$subscription="aaaaaaaa-aaaaa-aaaaaa-aaaa"

az aks create -n $clusterName -g $resourceGroup -l $location --max-pods 250 --node-count 2 --network-plugin
azure --vnet-subnet-id
/subscriptions/$subscription/resourceGroups/$resourceGroup/providers/Microsoft.Network/virtualNetworks/$vnet
/subnets/nodesubnet --pod-subnet-id
/subscriptions/$subscription/resourceGroups/$resourceGroup/providers/Microsoft.Network/virtualNetworks/$vnet
/subnets/podsubnet

```

Adding node pool

When adding node pool, reference the node subnet using `--vnet-subnet-id` and the pod subnet using `--pod-subnet-id`. The following example creates two new subnets that are then referenced in the creation of a new node pool:

```

az network vnet subnet create -g $resourceGroup --vnet-name $vnet --name node2subnet --address-prefixes
10.242.0.0/16 -o none
az network vnet subnet create -g $resourceGroup --vnet-name $vnet --name pod2subnet --address-prefixes
10.243.0.0/16 -o none

az aks nodepool add --cluster-name $clusterName -g $resourceGroup -n newNodepool --max-pods 250 --node-
count 2 --vnet-subnet-id
/subscriptions/$subscription/resourceGroups/$resourceGroup/providers/Microsoft.Network/virtualNetworks/$vnet
/subnets/node2subnet --pod-subnet-id
/subscriptions/$subscription/resourceGroups/$resourceGroup/providers/Microsoft.Network/virtualNetworks/$vnet
/subnets/pod2subnet --no-wait

```

Frequently asked questions

The following questions and answers apply to the **Azure CNI** networking configuration.

- *Can I deploy VMs in my cluster subnet?*

Yes.

- *What source IP do external systems see for traffic that originates in an Azure CNI-enabled pod?*

Systems in the same virtual network as the AKS cluster see the pod IP as the source address for any traffic from the pod. Systems outside the AKS cluster virtual network see the node IP as the source address for any traffic from the pod.

- *Can I configure per-pod network policies?*

Yes, Kubernetes network policy is available in AKS. To get started, see [Secure traffic between pods by using network policies in AKS](#).

- *Is the maximum number of pods deployable to a node configurable?*

Yes, when you deploy a cluster with the Azure CLI or a Resource Manager template. See [Maximum pods per node](#).

You can't change the maximum number of pods per node on an existing cluster.

- *How do I configure additional properties for the subnet that I created during AKS cluster creation? For example, service endpoints.*

The complete list of properties for the virtual network and subnets that you create during AKS cluster creation can be configured in the standard virtual network configuration page in the Azure portal.

- *Can I use a different subnet within my cluster virtual network for the Kubernetes service address range?*

It's not recommended, but this configuration is possible. The service address range is a set of virtual IPs (VIPs) that Kubernetes assigns to internal services in your cluster. Azure Networking has no visibility into the service IP range of the Kubernetes cluster. Because of the lack of visibility into the cluster's service address range, it's possible to later create a new subnet in the cluster virtual network that overlaps with the service address range. If such an overlap occurs, Kubernetes could assign a service an IP that's already in use by another resource in the subnet, causing unpredictable behavior or failures. By ensuring you use an address range outside the cluster's virtual network, you can avoid this overlap risk.

Dynamic allocation of IP addresses and enhanced subnet support FAQs

The following questions and answers apply to the [Azure CNI network configuration when using Dynamic allocation of IP addresses and enhanced subnet support](#).

- *Can I assign multiple pod subnets to a cluster/node pool?*

Only one subnet can be assigned to a cluster or node pool. However, multiple clusters or node pools can share a single subnet.

- *Can I assign Pod subnets from a different VNet altogether?*

The pod subnet should be from the same VNet as the cluster.

- *Can some node pools in a cluster use the traditional CNI while others use the new CNI?*

The entire cluster should use only one type of CNI.

AKS Engine

[Azure Kubernetes Service Engine \(AKS Engine\)](#) is an open-source project that generates Azure Resource Manager templates you can use for deploying Kubernetes clusters on Azure.

Kubernetes clusters created with AKS Engine support both the [kubenet](#) and [Azure CNI](#) plugins. As such, both networking scenarios are supported by AKS Engine.

Next steps

Learn more about networking in AKS in the following articles:

- [Use a static IP address with the Azure Kubernetes Service \(AKS\) load balancer](#)
- [Use an internal load balancer with Azure Container Service \(AKS\)](#)
- [Create a basic ingress controller with external network connectivity](#)
- [Enable the HTTP application routing add-on](#)

- Create an ingress controller that uses an internal, private network and IP address
- Create an ingress controller with a dynamic public IP and configure Let's Encrypt to automatically generate TLS certificates
- Create an ingress controller with a static public IP and configure Let's Encrypt to automatically generate TLS certificates

Use an internal load balancer with Azure Kubernetes Service (AKS)

4/21/2021 • 4 minutes to read • [Edit Online](#)

To restrict access to your applications in Azure Kubernetes Service (AKS), you can create and use an internal load balancer. An internal load balancer makes a Kubernetes service accessible only to applications running in the same virtual network as the Kubernetes cluster. This article shows you how to create and use an internal load balancer with Azure Kubernetes Service (AKS).

NOTE

Azure Load Balancer is available in two SKUs - *Basic* and *Standard*. By default, the Standard SKU is used when you create an AKS cluster. When creating a Service with type as LoadBalancer, you will get the same LB type as when you provision the cluster. For more information, see [Azure load balancer SKU comparison](#).

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart using the [Azure CLI](#) or [using the Azure portal](#).

You also need the Azure CLI version 2.0.59 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

The AKS cluster identity needs permission to manage network resources if you use an existing subnet or resource group. For information see [Use kubenet networking with your own IP address ranges in Azure Kubernetes Service \(AKS\)](#) or [Configure Azure CNI networking in Azure Kubernetes Service \(AKS\)](#). If you are configuring your load balancer to use an [IP address in a different subnet](#), ensure the the AKS cluster identity also has read access to that subnet.

For more information on permissions, see [Delegate AKS access to other Azure resources](#).

Create an internal load balancer

To create an internal load balancer, create a service manifest named `internal-lb.yaml` with the service type *LoadBalancer* and the `azure-load-balancer-internal` annotation as shown in the following example:

```
apiVersion: v1
kind: Service
metadata:
  name: internal-app
  annotations:
    service.beta.kubernetes.io/azure-load-balancer-internal: "true"
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: internal-app
```

Deploy the internal load balancer using the `kubectl apply` and specify the name of your YAML manifest:

```
kubectl apply -f internal-lb.yaml
```

An Azure load balancer is created in the node resource group and connected to the same virtual network as the AKS cluster.

When you view the service details, the IP address of the internal load balancer is shown in the *EXTERNAL-IP* column. In this context, *External*/is in relation to the external interface of the load balancer, not that it receives a public, external IP address. It may take a minute or two for the IP address to change from *<pending>* to an actual internal IP address, as shown in the following example:

```
$ kubectl get service internal-app
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
internal-app	LoadBalancer	10.0.248.59	10.240.0.7	80:30555/TCP	2m

Specify an IP address

If you would like to use a specific IP address with the internal load balancer, add the *loadBalancerIP* property to the load balancer YAML manifest. In this scenario, the specified IP address must reside in the same subnet as the AKS cluster and must not already be assigned to a resource. For example, you shouldn't use an IP address in the range designated for the Kubernetes subnet.

```
apiVersion: v1
kind: Service
metadata:
  name: internal-app
  annotations:
    service.beta.kubernetes.io/azure-load-balancer-internal: "true"
spec:
  type: LoadBalancer
  loadBalancerIP: 10.240.0.25
  ports:
  - port: 80
  selector:
    app: internal-app
```

When deployed and you view the service details, the IP address in the *EXTERNAL-IP* column reflects your specified IP address:

```
$ kubectl get service internal-app
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
internal-app	LoadBalancer	10.0.184.168	10.240.0.25	80:30225/TCP	4m

For more information on configuring your load balancer in a different subnet, see [Specify a different subnet](#)

Use private networks

When you create your AKS cluster, you can specify advanced networking settings. This approach lets you deploy the cluster into an existing Azure virtual network and subnets. One scenario is to deploy your AKS cluster into a private network connected to your on-premises environment and run services only accessible internally. For more information, see [configure your own virtual network subnets with Kubenet or Azure CNI](#).

No changes to the previous steps are needed to deploy an internal load balancer in an AKS cluster that uses a private network. The load balancer is created in the same resource group as your AKS cluster but connected to

your private virtual network and subnet, as shown in the following example:

```
$ kubectl get service internal-app
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
internal-app	LoadBalancer	10.1.15.188	10.0.0.35	80:31669/TCP	1m

NOTE

You may need to grant the cluster identity for your AKS cluster the *Network Contributor* role to the resource group where your Azure virtual network resources are deployed. View the cluster identity with `az aks show`, such as

```
az aks show --resource-group myResourceGroup --name myAKSCluster --query "identity". To create a role assignment, use the az role assignment create command.
```

Specify a different subnet

To specify a subnet for your load balancer, add the `azure-load-balancer-internal-subnet` annotation to your service. The subnet specified must be in the same virtual network as your AKS cluster. When deployed, the load balancer *EXTERNAL-IP* address is part of the specified subnet.

```
apiVersion: v1
kind: Service
metadata:
  name: internal-app
  annotations:
    service.beta.kubernetes.io/azure-load-balancer-internal: "true"
    service.beta.kubernetes.io/azure-load-balancer-internal-subnet: "apps-subnet"
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: internal-app
```

Delete the load balancer

When all services that use the internal load balancer are deleted, the load balancer itself is also deleted.

You can also directly delete a service as with any Kubernetes resource, such as

```
kubectl delete service internal-app, which also then deletes the underlying Azure load balancer.
```

Next steps

Learn more about Kubernetes services at the [Kubernetes services documentation](#).

Use a public Standard Load Balancer in Azure Kubernetes Service (AKS)

4/21/2021 • 19 minutes to read • [Edit Online](#)

The Azure Load Balancer is on L4 of the Open Systems Interconnection (OSI) model that supports both inbound and outbound scenarios. It distributes inbound flows that arrive at the load balancer's front end to the backend pool instances.

A **public** Load Balancer when integrated with AKS serves two purposes:

1. To provide outbound connections to the cluster nodes inside the AKS virtual network. It achieves this objective by translating the nodes private IP address to a public IP address that is part of its *Outbound Pool*.
2. To provide access to applications via Kubernetes services of type `LoadBalancer`. With it, you can easily scale your applications and create highly available services.

An **internal (or private)** load balancer is used where only private IPs are allowed as frontend. Internal load balancers are used to load balance traffic inside a virtual network. A load balancer frontend can also be accessed from an on-premises network in a hybrid scenario.

This document covers the integration with Public Load balancer. For internal Load Balancer integration, see the [AKS Internal Load balancer documentation](#).

Before you begin

Azure Load Balancer is available in two SKUs - *Basic* and *Standard*. By default, *Standard* SKU is used when you create an AKS cluster. Use the *Standard* SKU to have access to added functionality, such as a larger backend pool, [multiple node pools](#), and [Availability Zones](#). It's the recommended Load Balancer SKU for AKS.

For more information on the *Basic* and *Standard* SKUs, see [Azure load balancer SKU comparison](#).

This article assumes you have an AKS cluster with the *Standard* SKU Azure Load Balancer and walks through how to use and configure some of the capabilities and features of the load balancer. If you need an AKS cluster, see the AKS quickstart [using the Azure CLI](#) or [using the Azure portal](#).

IMPORTANT

If you prefer not to leverage the Azure Load Balancer to provide outbound connection and instead have your own gateway, firewall or proxy for that purpose you can skip the creation of the load balancer outbound pool and respective frontend IP by using [Outbound type as UserDefinedRouting \(UDR\)](#). The Outbound type defines the egress method for a cluster and it defaults to type: load balancer.

Use the public standard load balancer

After creating an AKS cluster with Outbound Type: Load Balancer (default), the cluster is ready to use the load balancer to expose services as well.

For that you can create a public Service of type `LoadBalancer` as shown in the following example. Start by creating a service manifest named `public-svc.yaml`:

```
apiVersion: v1
kind: Service
metadata:
  name: public-svc
spec:
  type: LoadBalancer
  ports:
  - port: 80
    selector:
      app: public-app
```

Deploy the public service manifest by using [kubectl apply](#) and specify the name of your YAML manifest:

```
kubectl apply -f public-svc.yaml
```

The Azure Load Balancer will be configured with a new public IP that will front this new service. Since the Azure Load Balancer can have multiple Frontend IPs, each new service deployed will get a new dedicated frontend IP to be uniquely accessed.

You can confirm your service is created and the load balancer is configured by running for example:

```
kubectl get service public-svc
```

NAMESPACE	NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
default	public-svc	LoadBalancer	10.0.39.110	52.156.88.187	80:32068/TCP	52s

When you view the service details, the public IP address created for this service on the load balancer is shown in the *EXTERNAL-IP* column. It may take a minute or two for the IP address to change from *<pending>* to an actual public IP address, as shown in the above example.

Configure the public standard load balancer

When using the Standard SKU public load balancer, there's a set of options that can be customized at creation time or by updating the cluster. These options allow you to customize the Load Balancer to meet your workloads needs and should be reviewed accordingly. With the Standard load balancer you can:

- Set or scale the number of Managed Outbound IPs
- Bring your own custom [Outbound IPs or Outbound IP Prefix](#)
- Customize the number of allocated outbound ports to each node of the cluster
- Configure the timeout setting for idle connections

IMPORTANT

Only one outbound IP option (managed IPs, bring your own IP, or IP Prefix) can be used at a given time.

Scale the number of managed outbound public IPs

Azure Load Balancer provides outbound connectivity from a virtual network in addition to inbound. Outbound rules make it simple to configure public Standard Load Balancer's outbound network address translation.

Like all Load Balancer rules, outbound rules follow the same familiar syntax as load balancing and inbound NAT rules:

frontend IPs + parameters + backend pool

An outbound rule configures outbound NAT for all virtual machines identified by the backend pool to be translated to the frontend. And parameters provide additional fine grained control over the outbound NAT algorithm.

While an outbound rule can be used with just a single public IP address, outbound rules ease the configuration burden for scaling outbound NAT. You can use multiple IP addresses to plan for large-scale scenarios and you can use outbound rules to mitigate SNAT exhaustion prone patterns. Each additional IP address provided by a frontend provides 64k ephemeral ports for Load Balancer to use as SNAT ports.

When using a *Standard* SKU load balancer with managed outbound public IPs, which are created by default, you can scale the number of managed outbound public IPs using the `load-balancer-managed-ip-count` parameter.

To update an existing cluster, run the following command. This parameter can also be set at cluster create-time to have multiple managed outbound public IPs.

```
az aks update \
--resource-group myResourceGroup \
--name myAKSCluster \
--load-balancer-managed-outbound-ip-count 2
```

The above example sets the number of managed outbound public IPs to 2 for the *myAKSCluster* cluster in *myResourceGroup*.

You can also use the `load-balancer-managed-ip-count` parameter to set the initial number of managed outbound public IPs when creating your cluster by appending the `--load-balancer-managed-outbound-ip-count` parameter and setting it to your desired value. The default number of managed outbound public IPs is 1.

Provide your own outbound public IPs or prefixes

When you use a *Standard* SKU load balancer, by default the AKS cluster automatically creates a public IP in the AKS-managed infrastructure resource group and assigns it to the load balancer outbound pool.

A public IP created by AKS is considered an AKS managed resource. This means the lifecycle of that public IP is intended to be managed by AKS and requires no user action directly on the public IP resource. Alternatively, you can assign your own custom public IP or public IP prefix at cluster creation time. Your custom IPs can also be updated on an existing cluster's load balancer properties.

Requirements for using your own public IP or prefix:

- Custom public IP addresses must be created and owned by the user. Managed public IP addresses created by AKS cannot be reused as bringing your own custom IP as it can cause management conflicts.
- You must ensure the AKS cluster identity (Service Principal or Managed Identity) has permissions to access the outbound IP. As per the [required public IP permissions list](#).
- Make sure you meet the [pre-requisites and constraints](#) necessary to configure Outbound IPs or Outbound IP prefixes.

Update the cluster with your own outbound public IP

Use the `az network public-ip show` command to list the IDs of your public IPs.

```
az network public-ip show --resource-group myResourceGroup --name myPublicIP --query id -o tsv
```

The above command shows the ID for the *myPublicIP* public IP in the *myResourceGroup* resource group.

Use the `az aks update` command with the `load-balancer-outbound-ips` parameter to update your cluster with your public IPs.

The following example uses the `load-balancer-outbound-ips` parameter with the IDs from the previous

command.

```
az aks update \
--resource-group myResourceGroup \
--name myAKSCluster \
--load-balancer-outbound-ips <publicIpId1>,<publicIpId2>
```

Update the cluster with your own outbound public IP prefix

You can also use public IP prefixes for egress with your *Standard* SKU load balancer. The following example uses the [az network public-ip prefix show](#) command to list the IDs of your public IP prefixes:

```
az network public-ip prefix show --resource-group myResourceGroup --name myPublicIPPrefix --query id -o tsv
```

The above command shows the ID for the *myPublicIPPrefix* public IP prefix in the *myResourceGroup* resource group.

The following example uses the *load-balancer-outbound-ip-prefixes* parameter with the IDs from the previous command.

```
az aks update \
--resource-group myResourceGroup \
--name myAKSCluster \
--load-balancer-outbound-ip-prefixes <publicIpPrefixId1>,<publicIpPrefixId2>
```

Create the cluster with your own public IP or prefixes

You may wish to bring your own IP addresses or IP prefixes for egress at cluster creation time to support scenarios like adding egress endpoints to an allow list. Append the same parameters shown above to your cluster creation step to define your own public IPs and IP prefixes at the start of a cluster's lifecycle.

Use the [az aks create](#) command with the *load-balancer-outbound-ips* parameter to create a new cluster with your public IPs at the start.

```
az aks create \
--resource-group myResourceGroup \
--name myAKSCluster \
--load-balancer-outbound-ips <publicIpId1>,<publicIpId2>
```

Use the [az aks create](#) command with the *load-balancer-outbound-ip-prefixes* parameter to create a new cluster with your public IP prefixes at the start.

```
az aks create \
--resource-group myResourceGroup \
--load-balancer-outbound-ip-prefixes <publicIpPrefixId1>,<publicIpPrefixId2>
```

Configure the allocated outbound ports

IMPORTANT

If you have applications on your cluster which are expected to establish a large number of connection to small set of destinations, eg. many frontend instances connecting to an SQL DB, you have a scenario very susceptible to encounter SNAT Port exhaustion (run out of ports to connect from). For these scenarios it's highly recommended to increase the allocated outbound ports and outbound frontend IPs on the load balancer. The increase should consider that one (1) additional IP address adds 64k additional ports to distribute across all cluster nodes.

Unless otherwise specified, AKS will use the default value of Allocated Outbound Ports that Standard Load Balancer defines when configuring it. This value is **null** on the AKS API or **0** on the SLB API as shown by the below command:

```
NODE_RG=$(az aks show --resource-group myResourceGroup --name myAKSCluster --query nodeResourceGroup -o tsv)
az network lb outbound-rule list --resource-group $NODE_RG --lb-name kubernetes -o table
```

The previous commands will list the outbound rule for your load balancer, for example:

AllocatedOutboundPorts	EnableTcpReset	IdleTimeoutInMinutes	Name	Protocol
ProvisioningState	ResourceGroup			
0	True	30	aksOutboundRule	All
MC_myResourceGroup_myAKSCluster_eastus				Succeeded

This output does not mean that you have 0 ports but instead that you are leveraging the [automatic outbound port assignment based on backend pool size](#), so for example if a cluster has 50 or less nodes, 1024 ports for each node are allocated, as you increase the number of nodes from there you'll gradually get fewer ports per node.

To define or increase the number of Allocated Outbound ports, you can follow the below example:

```
az aks update \
--resource-group myResourceGroup \
--name myAKSCluster \
--load-balancer-managed-outbound-ip-count 7 \
--load-balancer-outbound-ports 4000
```

This example would give you 4000 Allocated Outbound Ports for each node in my cluster, and with 7 IPs you would have $4000 \text{ ports per node} * 100 \text{ nodes} = 400k \text{ total ports} \leq 448k \text{ total ports} = 7 \text{ IPs} * 64k \text{ ports per IP}$. This would allow you to safely scale to 100 nodes and have a default upgrade operation. It is critical to allocate sufficient ports for additional nodes needed for upgrade and other operations. AKS defaults to one buffer node for upgrade, in this example this requires 4000 free ports at any given point in time. If using [maxSurge values](#), multiply the outbound ports per node by your maxSurge value.

To safely go above 100 nodes, you'd have to add more IPs.

IMPORTANT

You must [calculate your required quota and check the requirements](#) before customizing `allocatedOutboundPorts` to avoid connectivity or scaling issues.

You can also use the `load-balancer-outbound-ports` parameters when creating a cluster, but you must also specify either `load-balancer-managed-outbound-ip-count`, `load-balancer-outbound-ips`, or `load-balancer-outbound-ip-prefixes` as well. For example:

```
az aks create \
--resource-group myResourceGroup \
--name myAKSCluster \
--load-balancer-sku standard \
--load-balancer-managed-outbound-ip-count 2 \
--load-balancer-outbound-ports 1024
```

Configure the load balancer idle timeout

When SNAT port resources are exhausted, outbound flows fail until existing flows release SNAT ports. Load Balancer reclaims SNAT ports when the flow closes and the AKS-configured load balancer uses a 30-minute idle timeout for reclaiming SNAT ports from idle flows. You can also use transport (for example, `TCP keepalives`) or `application-layer keepalives` to refresh an idle flow and reset this idle timeout if necessary. You can configure this timeout following the below example:

```
az aks update \
--resource-group myResourceGroup \
--name myAKSCluster \
--load-balancer-idle-timeout 4
```

If you expect to have numerous short lived connections, and no connections that are long lived and might have long times of idle, like leveraging `kubectl proxy` or `kubectl port-forward` consider using a low timeout value such as 4 minutes. Also, when using TCP keepalives, it's sufficient to enable them on one side of the connection. For example, it's sufficient to enable them on the server side only to reset the idle timer of the flow and it's not necessary for both sides to start TCP keepalives. Similar concepts exist for application layer, including database client-server configurations. Check the server side for what options exist for application-specific keepalives.

IMPORTANT

AKS enables TCP Reset on idle by default and recommends you keep this configuration on and leverage it for more predictable application behavior on your scenarios. TCP RST is only sent during TCP connection in ESTABLISHED state. Read more about it [here](#).

Requirements for customizing allocated outbound ports and idle timeout

- The value you specify for `allocatedOutboundPorts` must also be a multiple of 8.
- You must have enough outbound IP capacity based on the number of your node VMs and required allocated outbound ports. To validate you have enough outbound IP capacity, use the following formula:

$outboundIPs * 64,000 > nodeVMs * desiredAllocatedOutboundPorts$.

For example, if you have 3 `nodeVMs`, and 50,000 `desiredAllocatedOutboundPorts`, you need to have at least 3 `outboundIPs`. It is recommended that you incorporate additional outbound IP capacity beyond what you need. Additionally, you must account for the cluster autoscaler and the possibility of node pool upgrades when calculating outbound IP capacity. For the cluster autoscaler, review the current node count and the maximum node count and use the higher value. For upgrading, account for an additional node VM for every node pool that allows upgrading.

- When setting `IdleTimeoutInMinutes` to a different value than the default of 30 minutes, consider how long your workloads will need an outbound connection. Also consider the default timeout value for a *Standard* SKU load balancer used outside of AKS is 4 minutes. An `IdleTimeoutInMinutes` value that more accurately reflects your specific AKS workload can help decrease SNAT exhaustion caused by tying up connections no longer being used.

WARNING

Altering the values for `AllocatedOutboundPorts` and `IdleTimeoutInMinutes` may significantly change the behavior of the outbound rule for your load balancer and should not be done lightly, without understanding the tradeoffs and your application's connection patterns, check the [SNAT Troubleshooting section below](#) and review the [Load Balancer outbound rules](#) and [outbound connections in Azure](#) before updating these values to fully understand the impact of your changes.

Restrict inbound traffic to specific IP ranges

The following manifest uses `loadBalancerSourceRanges` to specify a new IP range for inbound external traffic:

```
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-front
spec:
  type: LoadBalancer
  ports:
  - port: 80
    selector:
      app: azure-vote-front
  loadBalancerSourceRanges:
  - MY_EXTERNAL_IP_RANGE
```

NOTE

Inbound, external traffic flows from the load balancer to the virtual network for your AKS cluster. The virtual network has a Network Security Group (NSG) which allows all inbound traffic from the load balancer. This NSG uses a [service tag](#) of type `LoadBalancer` to allow traffic from the load balancer.

Maintain the client's IP on inbound connections

By default, a service of type `LoadBalancer` in Kubernetes and in AKS won't persist the client's IP address on the connection to the pod. The source IP on the packet that's delivered to the pod will be the private IP of the node. To maintain the client's IP address, you must set `service.spec.externalTrafficPolicy` to `local` in the service definition. The following manifest shows an example:

```
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-front
spec:
  type: LoadBalancer
  externalTrafficPolicy: Local
  ports:
  - port: 80
    selector:
      app: azure-vote-front
```

Additional customizations via Kubernetes Annotations

Below is a list of annotations supported for Kubernetes services with type `LoadBalancer`, these annotations only apply to INBOUND flows:

ANNOTATION	VALUE	DESCRIPTION
<code>service.beta.kubernetes.io/azure-load-balancer-internal</code>	<code>true</code> or <code>false</code>	Specify whether the load balancer should be internal. It's defaulting to public if not set.
<code>service.beta.kubernetes.io/azure-load-balancer-internal-subnet</code>	Name of the subnet	Specify which subnet the internal load balancer should be bound to. It's defaulting to the subnet configured in cloud config file if not set.

ANNOTATION	VALUE	DESCRIPTION
<code>service.beta.kubernetes.io/azure-dns-label-name</code>	Name of the DNS label on Public IPs	Specify the DNS label name for the public service. If it is set to empty string, the DNS entry in the Public IP will not be used.
<code>service.beta.kubernetes.io/azure-shared-securityrule</code>	<code>true</code> or <code>false</code>	Specify that the service should be exposed using an Azure security rule that may be shared with another service, trading specificity of rules for an increase in the number of services that can be exposed. This annotation relies on the Azure Augmented Security Rules feature of Network Security groups.
<code>service.beta.kubernetes.io/azure-load-balancer-resource-group</code>	Name of the resource group	Specify the resource group of load balancer public IPs that aren't in the same resource group as the cluster infrastructure (node resource group).
<code>service.beta.kubernetes.io/azure-allowed-service-tags</code>	List of allowed service tags	Specify a list of allowed service tags separated by comma.
<code>service.beta.kubernetes.io/azure-load-balancer-tcp-idle-timeout</code>	TCP idle timeouts in minutes	Specify the time, in minutes, for TCP connection idle timeouts to occur on the load balancer. Default and minimum value is 4. Maximum value is 30. Must be an integer.
<code>service.beta.kubernetes.io/azure-load-balancer-disable-tcp-reset</code>	<code>true</code>	Disable <code>enableTcpReset</code> for SLB

Troubleshooting SNAT

If you know that you're starting many outbound TCP or UDP connections to the same destination IP address and port, and you observe failing outbound connections or are advised by support that you're exhausting SNAT ports (preallocated ephemeral ports used by PAT), you have several general mitigation options. Review these options and decide what is available and best for your scenario. It's possible that one or more can help manage this scenario. For detailed information, review the [Outbound Connections Troubleshooting Guide](#).

Frequently the root cause of SNAT exhaustion is an anti-pattern for how outbound connectivity is established, managed, or configurable timers changed from their default values. Review this section carefully.

Steps

1. Check if your connections remain idle for a long time and rely on the default idle timeout for releasing that port. If so the default timeout of 30 min might need to be reduced for your scenario.
2. Investigate how your application is creating outbound connectivity (for example, code review or packet capture).
3. Determine if this activity is expected behavior or whether the application is misbehaving. Use [metrics](#) and [logs](#) in Azure Monitor to substantiate your findings. Use "Failed" category for SNAT Connections metric for example.
4. Evaluate if appropriate [patterns](#) are followed.
5. Evaluate if SNAT port exhaustion should be mitigated with [additional Outbound IP addresses + additional Allocated Outbound Ports](#) .

Design patterns

Always take advantage of connection reuse and connection pooling whenever possible. These patterns will avoid resource exhaustion problems and result in predictable behavior. Primitives for these patterns can be found in many development libraries and frameworks.

- Atomic requests (one request per connection) are generally not a good design choice. Such anti-pattern limits scale, reduces performance, and decreases reliability. Instead, reuse HTTP/S connections to reduce the numbers of connections and associated SNAT ports. The application scale will increase and performance improve because of reduced handshakes, overhead, and cryptographic operation cost when using TLS.
- If you're using out of cluster/custom DNS, or custom upstream servers on coreDNS have in mind that DNS can introduce many individual flows at volume when the client isn't caching the DNS resolvers result. Make sure to customize coreDNS first instead of using custom DNS servers, and define a good caching value.
- UDP flows (for example DNS lookups) allocate SNAT ports for the duration of the idle timeout. The longer the idle timeout, the higher the pressure on SNAT ports. Use short idle timeout (for example 4 minutes). Use connection pools to shape your connection volume.
- Never silently abandon a TCP flow and rely on TCP timers to clean up flow. If you don't let TCP explicitly close the connection, state remains allocated at intermediate systems and endpoints and makes SNAT ports unavailable for other connections. This pattern can trigger application failures and SNAT exhaustion.
- Don't change OS-level TCP close related timer values without expert knowledge of impact. While the TCP stack will recover, your application performance can be negatively affected when the endpoints of a connection have mismatched expectations. Wishing to change timers is usually a sign of an underlying design problem. Review following recommendations.

The above example updates the rule to only allow inbound external traffic from the *MY_EXTERNAL_IP_RANGE* range. If you replace *MY_EXTERNAL_IP_RANGE* with the internal subnet IP address, traffic is restricted to cluster internal IPs only. This will not allow clients from outside of your Kubernetes cluster to access the load balancer.

Moving from a basic SKU load balancer to standard SKU

If you have an existing cluster with the Basic SKU Load Balancer, there are important behavioral differences to note when migrating to use a cluster with the Standard SKU Load Balancer.

For example, making blue/green deployments to migrate clusters is a common practice given the `load-balancer-sku` type of a cluster can only be defined at cluster create time. However, *Basic SKU* Load Balancers use *Basic SKU* IP Addresses, which aren't compatible with *Standard SKU* Load Balancers as they require *Standard SKU* IP Addresses. When migrating clusters to upgrade Load Balancer SKUs, a new IP address with a compatible IP Address SKU will be required.

For more considerations on how to migrate clusters, visit [our documentation on migration considerations](#) to view a list of important topics to consider when migrating. The below limitations are also important behavioral differences to note when using Standard SKU Load Balancers in AKS.

Limitations

The following limitations apply when you create and manage AKS clusters that support a load balancer with the *Standard SKU*:

- At least one public IP or IP prefix is required for allowing egress traffic from the AKS cluster. The public IP or IP prefix is also required to maintain connectivity between the control plane and agent nodes and to maintain compatibility with previous versions of AKS. You have the following options for specifying public IPs or IP prefixes with a *Standard SKU* load balancer:
 - Provide your own public IPs.
 - Provide your own public IP prefixes.

- Specify a number up to 100 to allow the AKS cluster to create that many *Standard* SKU public IPs in the same resource group created as the AKS cluster, which is usually named with *MC_* at the beginning. AKS assigns the public IP to the *Standard* SKU load balancer. By default, one public IP will automatically be created in the same resource group as the AKS cluster, if no public IP, public IP prefix, or number of IPs is specified. You also must allow public addresses and avoid creating any Azure Policy that bans IP creation.
- A public IP created by AKS cannot be reused as a custom bring your own public IP address. All custom IP addresses must be created and managed by the user.
- Defining the load balancer SKU can only be done when you create an AKS cluster. You can't change the load balancer SKU after an AKS cluster has been created.
- You can only use one type of load balancer SKU (Basic or Standard) in a single cluster.
- *Standard* SKU Load Balancers only support *Standard* SKU IP Addresses.

Next steps

Learn more about Kubernetes services at the [Kubernetes services documentation](#).

Learn more about using Internal Load Balancer for Inbound traffic at the [AKS Internal Load Balancer documentation](#).

Use a static public IP address and DNS label with the Azure Kubernetes Service (AKS) load balancer

4/21/2021 • 4 minutes to read • [Edit Online](#)

By default, the public IP address assigned to a load balancer resource created by an AKS cluster is only valid for the lifespan of that resource. If you delete the Kubernetes service, the associated load balancer and IP address are also deleted. If you want to assign a specific IP address or retain an IP address for redeployed Kubernetes services, you can create and use a static public IP address.

This article shows you how to create a static public IP address and assign it to your Kubernetes service.

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart using the [Azure CLI](#) or [using the Azure portal](#).

You also need the Azure CLI version 2.0.59 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

This article covers using a *Standard* SKU IP with a *Standard* SKU load balancer. For more information, see [IP address types and allocation methods in Azure](#).

Create a static IP address

Create a static public IP address with the `az network public-ip create` command. The following creates a static IP resource named *myAKSPublicIP* in the *myResourceGroup* resource group:

```
az network public-ip create \
    --resource-group myResourceGroup \
    --name myAKSPublicIP \
    --sku Standard \
    --allocation-method static
```

NOTE

If you are using a *Basic* SKU load balancer in your AKS cluster, use *Basic* for the *sku* parameter when defining a public IP. Only *Basic* SKU IPs work with the *Basic* SKU load balancer and only *Standard* SKU IPs work with *Standard* SKU load balancers.

The IP address is displayed, as shown in the following condensed example output:

```
{
  "publicIp": {
    ...
    "ipAddress": "40.121.183.52",
    ...
  }
}
```

You can later get the public IP address using the `az network public-ip list` command. Specify the name of the

node resource group and public IP address you created, and query for the *ipAddress* as shown in the following example:

```
$ az network public-ip show --resource-group myResourceGroup --name myAKSPublicIP --query ipAddress --output tsv  
40.121.183.52
```

Create a service using the static IP address

Before creating a service, ensure the cluster identity used by the AKS cluster has delegated permissions to the other resource group. For example:

```
az role assignment create \  
  --assignee <Client ID> \  
  --role "Network Contributor" \  
  --scope /subscriptions/<subscription id>/resourceGroups/<resource group name>
```

IMPORTANT

If you customized your outbound IP make sure your cluster identity has permissions to both the outbound public IP and this inbound public IP.

To create a *LoadBalancer* service with the static public IP address, add the `loadBalancerIP` property and the value of the static public IP address to the YAML manifest. Create a file named `load-balancer-service.yaml` and copy in the following YAML. Provide your own public IP address created in the previous step. The following example also sets the annotation to the resource group named *myResourceGroup*. Provide your own resource group name.

```
apiVersion: v1  
kind: Service  
metadata:  
  annotations:  
    service.beta.kubernetes.io/azure-load-balancer-resource-group: myResourceGroup  
  name: azure-load-balancer  
spec:  
  loadBalancerIP: 40.121.183.52  
  type: LoadBalancer  
  ports:  
  - port: 80  
  selector:  
    app: azure-load-balancer
```

Create the service and deployment with the `kubectl apply` command.

```
kubectl apply -f load-balancer-service.yaml
```

Apply a DNS label to the service

If your service is using a dynamic or static public IP address, you can use the service annotation `service.beta.kubernetes.io/azure-dns-label-name` to set a public-facing DNS label. This publishes a fully qualified domain name for your service using Azure's public DNS servers and top-level domain. The annotation value must be unique within the Azure location, so it's recommended to use a sufficiently qualified label.

Azure will then automatically append a default subnet, such as <location>.cloudapp.azure.com (where location is the region you selected), to the name you provide, to create the fully qualified DNS name. For example:

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/azure-dns-label-name: myserviceuniquelabel
  name: azure-load-balancer
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: azure-load-balancer
```

NOTE

To publish the service on your own domain, see [Azure DNS](#) and the [external-dns](#) project.

Troubleshoot

If the static IP address defined in the *loadBalancerIP* property of the Kubernetes service manifest does not exist, or has not been created in the node resource group and no additional delegations configured, the load balancer service creation fails. To troubleshoot, review the service creation events with the `kubectl describe` command. Provide the name of the service as specified in the YAML manifest, as shown in the following example:

```
kubectl describe service azure-load-balancer
```

Information about the Kubernetes service resource is displayed. The *Events* at the end of the following example output indicate that the *user supplied IP Address was not found*. In these scenarios, verify that you have created the static public IP address in the node resource group and that the IP address specified in the Kubernetes service manifest is correct.

```
Name:                  azure-load-balancer
Namespace:             default
Labels:                <none>
Annotations:           <none>
Selector:              app=azure-load-balancer
Type:                 LoadBalancer
IP:                   10.0.18.125
IP:                   40.121.183.52
Port:                 <unset>  80/TCP
TargetPort:            80/TCP
NodePort:              <unset>  32582/TCP
Endpoints:             <none>
Session Affinity:     None
External Traffic Policy: Cluster
Events:
  Type  Reason          Age           From            Message
  ----  ----          ----          ----          -----
  Normal CreatingLoadBalancer  7s (x2 over 22s)  service-controller  Creating load balancer
  Warning CreatingLoadBalancerFailed 6s (x2 over 12s)  service-controller  Error creating load balancer
(will retry): Failed to create load balancer for service default/azure-load-balancer: user supplied IP
Address 40.121.183.52 was not found
```

Next steps

For additional control over the network traffic to your applications, you may want to instead [create an ingress controller](#). You can also [create an ingress controller with a static public IP address](#).

Create an ingress controller in Azure Kubernetes Service (AKS)

4/26/2021 • 6 minutes to read • [Edit Online](#)

An ingress controller is a piece of software that provides reverse proxy, configurable traffic routing, and TLS termination for Kubernetes services. Kubernetes ingress resources are used to configure the ingress rules and routes for individual Kubernetes services. Using an ingress controller and ingress rules, a single IP address can be used to route traffic to multiple services in a Kubernetes cluster.

This article shows you how to deploy the NGINX ingress controller in an Azure Kubernetes Service (AKS) cluster. Two applications are then run in the AKS cluster, each of which is accessible over the single IP address.

You can also:

- [Enable the HTTP application routing add-on](#)
- [Create an ingress controller that uses an internal, private network and IP address](#)
- [Create an ingress controller that uses your own TLS certificates](#)
- [Create an ingress controller that uses Let's Encrypt to automatically generate TLS certificates with a dynamic public IP address or with a static public IP address](#)

Before you begin

This article uses Helm 3 to install the NGINX ingress controller. Make sure that you are using the latest release of Helm and have access to the *ingress-nginx* Helm repository.

This article also requires that you are running the Azure CLI version 2.0.64 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Create an ingress controller

To create the ingress controller, use Helm to install *nginx-ingress*. For added redundancy, two replicas of the NGINX ingress controllers are deployed with the `--set controller.replicaCount` parameter. To fully benefit from running replicas of the ingress controller, make sure there's more than one node in your AKS cluster.

The ingress controller also needs to be scheduled on a Linux node. Windows Server nodes shouldn't run the ingress controller. A node selector is specified using the `--set nodeSelector` parameter to tell the Kubernetes scheduler to run the NGINX ingress controller on a Linux-based node.

TIP

The following example creates a Kubernetes namespace for the ingress resources named *ingress-basic*. Specify a namespace for your own environment as needed.

TIP

If you would like to enable client source IP preservation for requests to containers in your cluster, add `--set controller.service.externalTrafficPolicy=Local` to the Helm install command. The client source IP is stored in the request header under *X-Forwarded-For*. When using an ingress controller with client source IP preservation enabled, SSL pass-through will not work.

```
# Create a namespace for your ingress resources
kubectl create namespace ingress-basic

# Add the ingress-nginx repository
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

# Use Helm to deploy an NGINX ingress controller
helm install nginx-ingress ingress-nginx/nginx-inginx \
--namespace ingress-basic \
--set controller.replicaCount=2 \
--set controller.nodeSelector."beta\.kubernetes\.io/os"=linux \
--set defaultBackend.nodeSelector."beta\.kubernetes\.io/os"=linux \
--set controller.admissionWebhooks.patch.nodeSelector."beta\.kubernetes\.io/os"=linux
```

When the Kubernetes load balancer service is created for the NGINX ingress controller, a dynamic public IP address is assigned, as shown in the following example output:

```
$ kubectl --namespace ingress-basic get services -o wide -w nginx-ingress-ingress-nginx-controller

NAME                                TYPE      CLUSTER-IP   EXTERNAL-IP      PORT(S)
AGE      SELECTOR
nginx-ingress-ingress-nginx-controller   LoadBalancer   10.0.74.133   EXTERNAL_IP
80:32486/TCP,443:30953/TCP   44s    app.kubernetes.io/component=controller,app.kubernetes.io/instance=nginx-
ingress,app.kubernetes.io/name=ingress-nginx
```

No ingress rules have been created yet, so the NGINX ingress controller's default 404 page is displayed if you browse to the internal IP address. Ingress rules are configured in the following steps.

Run demo applications

To see the ingress controller in action, run two demo applications in your AKS cluster. In this example, you use `kubectl apply` to deploy two instances of a simple *Hello world* application.

Create a `aks-helloworld-one.yaml` file and copy in the following example YAML:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: aks-helloworld-one
spec:
  replicas: 1
  selector:
    matchLabels:
      app: aks-helloworld-one
  template:
    metadata:
      labels:
        app: aks-helloworld-one
    spec:
      containers:
        - name: aks-helloworld-one
          image: mcr.microsoft.com/azuredocs/aks-helloworld:v1
          ports:
            - containerPort: 80
          env:
            - name: TITLE
              value: "Welcome to Azure Kubernetes Service (AKS)"
---
apiVersion: v1
kind: Service
metadata:
  name: aks-helloworld-one
spec:
  type: ClusterIP
  ports:
    - port: 80
  selector:
    app: aks-helloworld-one
```

Create a `aks-helloworld-two.yaml` file and copy in the following example YAML:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: aks-helloworld-two
spec:
  replicas: 1
  selector:
    matchLabels:
      app: aks-helloworld-two
  template:
    metadata:
      labels:
        app: aks-helloworld-two
    spec:
      containers:
        - name: aks-helloworld-two
          image: mcr.microsoft.com/azuredocs/aks-helloworld:v1
          ports:
            - containerPort: 80
          env:
            - name: TITLE
              value: "AKS Ingress Demo"
---
apiVersion: v1
kind: Service
metadata:
  name: aks-helloworld-two
spec:
  type: ClusterIP
  ports:
    - port: 80
  selector:
    app: aks-helloworld-two

```

Run the two demo applications using `kubectl apply`:

```

kubectl apply -f aks-helloworld-one.yaml --namespace ingress-basic
kubectl apply -f aks-helloworld-two.yaml --namespace ingress-basic

```

Create an ingress route

Both applications are now running on your Kubernetes cluster. To route traffic to each application, create a Kubernetes ingress resource. The ingress resource configures the rules that route traffic to one of the two applications.

In the following example, traffic to `EXTERNAL_IP` is routed to the service named `aks-helloworld-one`. Traffic to `EXTERNAL_IP/hello-world-two` is routed to the `aks-helloworld-two` service. Traffic to `EXTERNAL_IP/static` is routed to the service named `aks-helloworld-one` for static assets.

Create a file named `hello-world-ingress.yaml` and copy in the following example YAML.

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hello-world-ingress
  namespace: ingress-basic
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
  - http:
      paths:
      - path: /hello-world-one(/|$(.))
        pathType: Prefix
        backend:
          service:
            name: aks-helloworld-one
            port:
              number: 80
      - path: /hello-world-two(/|$(.))
        pathType: Prefix
        backend:
          service:
            name: aks-helloworld-two
            port:
              number: 80
      - path: /(.*)
        pathType: Prefix
        backend:
          service:
            name: aks-helloworld-one
            port:
              number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hello-world-ingress-static
  namespace: ingress-basic
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    nginx.ingress.kubernetes.io/rewrite-target: /static/$2
spec:
  rules:
  - http:
      paths:
      - path:
          pathType: Prefix
          backend:
            service:
              name: aks-helloworld-one
              port:
                number: 80
          path: /static(/|$(.))

```

Create the **ingress** resource using the `kubectl apply -f hello-world-ingress.yaml` command.

```

$ kubectl apply -f hello-world-ingress.yaml

ingress.extensions/hello-world-ingress created
ingress.extensions/hello-world-ingress-static created

```

Test the ingress controller

To test the routes for the ingress controller, browse to the two applications. Open a web browser to the IP address of your NGINX ingress controller, such as `EXTERNAL_IP`. The first demo application is displayed in the web browser, as shown in the follow example:



Now add the `/hello-world-two` path to the IP address, such as `EXTERNAL_IP/hello-world-two`. The second demo application with the custom title is displayed:



Clean up resources

This article used Helm to install the ingress components and sample apps. When you deploy a Helm chart, a number of Kubernetes resources are created. These resources includes pods, deployments, and services. To clean up these resources, you can either delete the entire sample namespace, or the individual resources.

Delete the sample namespace and all resources

To delete the entire sample namespace, use the `kubectl delete` command and specify your namespace name. All the resources in the namespace are deleted.

```
kubectl delete namespace ingress-basic
```

Delete resources individually

Alternatively, a more granular approach is to delete the individual resources created. List the Helm releases with the `helm list` command. Look for charts named `nginx-ingress` and `aks-helloworld`, as shown in the following example output:

```
$ helm list --namespace ingress-basic
```

NAME	NAMESPACE	REVISION	UPDATED	STATUS
CHART	APP VERSION			
nginx-ingress	ingress-basic	1	2020-01-06 19:55:46.358275 -0600 CST	deployed
nginx-ingress-1.27.1	0.26.1			

Uninstall the releases with the `helm uninstall` command. The following example uninstalls the NGINX ingress deployment.

```
$ helm uninstall nginx-ingress --namespace ingress-basic  
release "nginx-ingress" uninstalled
```

Next, remove the two sample applications:

```
kubectl delete -f aks-helloworld-one.yaml --namespace ingress-basic  
kubectl delete -f aks-helloworld-two.yaml --namespace ingress-basic
```

Remove the ingress route that directed traffic to the sample apps:

```
kubectl delete -f hello-world-ingress.yaml
```

Finally, you can delete the itself namespace. Use the `kubectl delete` command and specify your namespace name:

```
kubectl delete namespace ingress-basic
```

Next steps

This article included some external components to AKS. To learn more about these components, see the following project pages:

- [Helm CLI](#)
- [NGINX ingress controller](#)

You can also:

- [Enable the HTTP application routing add-on](#)
- [Create an ingress controller that uses an internal, private network and IP address](#)
- [Create an ingress controller that uses your own TLS certificates](#)
- [Create an ingress controller that uses Let's Encrypt to automatically generate TLS certificates with a dynamic public IP address or with a static public IP address](#)

HTTP application routing

4/26/2021 • 6 minutes to read • [Edit Online](#)

The HTTP application routing solution makes it easy to access applications that are deployed to your Azure Kubernetes Service (AKS) cluster. When the solution's enabled, it configures an [Ingress controller](#) in your AKS cluster. As applications are deployed, the solution also creates publicly accessible DNS names for application endpoints.

When the add-on is enabled, it creates a DNS Zone in your subscription. For more information about DNS cost, see [DNS pricing](#).

Caution

The HTTP application routing add-on is designed to let you quickly create an ingress controller and access your applications. This add-on is not currently designed for use in a production environment and is not recommended for production use. For production-ready ingress deployments that include multiple replicas and TLS support, see [Create an HTTPS ingress controller](#).

HTTP routing solution overview

The add-on deploys two components: a [Kubernetes Ingress controller](#) and an [External-DNS](#) controller.

- **Ingress controller:** The Ingress controller is exposed to the internet by using a Kubernetes service of type LoadBalancer. The Ingress controller watches and implements [Kubernetes Ingress resources](#), which creates routes to application endpoints.
- **External-DNS controller:** Watches for Kubernetes Ingress resources and creates DNS A records in the cluster-specific DNS zone.

Deploy HTTP routing: CLI

The HTTP application routing add-on can be enabled with the Azure CLI when deploying an AKS cluster. To do so, use the `az aks create` command with the `--enable-addons` argument.

```
az aks create --resource-group myResourceGroup --name myAKSCluster --enable-addons http_application_routing
```

TIP

If you want to enable multiple add-ons, provide them as a comma-separated list. For example, to enable HTTP application routing and monitoring, use the format `--enable-addons http_application_routing,monitoring`.

You can also enable HTTP routing on an existing AKS cluster using the `az aks enable-addons` command. To enable HTTP routing on an existing cluster, add the `--addons` parameter and specify `http_application_routing` as shown in the following example:

```
az aks enable-addons --resource-group myResourceGroup --name myAKSCluster --addons http_application_routing
```

After the cluster is deployed or updated, use the `az aks show` command to retrieve the DNS zone name.

```
az aks show --resource-group myResourceGroup --name myAKSCluster --query
addonProfiles.httpApplicationRouting.config.HTTPApplicationRoutingZoneName -o table
```

This name is needed to deploy applications to the AKS cluster and is shown in the following example output:

```
9f9c1fe7-21a1-416d-99cd-3543bb92e4c3.eastus.aksapp.io
```

Deploy HTTP routing: Portal

The HTTP application routing add-on can be enabled through the Azure portal when deploying an AKS cluster.

Home > Kubernetes services > Create Kubernetes cluster

Create Kubernetes cluster

Basics Scale Authentication Networking Monitoring Tags Review + create

You can change networking settings for your cluster, including enabling HTTP application routing and configuring your network using either the 'Basic' or 'Advanced' options:

- 'Basic' networking creates a new VNet for your cluster using default values.
- 'Advanced' networking allows clusters to use a new or existing VNet with customizable addresses. Application pods are connected directly to the VNet, which allows for native integration with VNet features.

Learn more about networking in Azure Kubernetes Service

HTTP application routing ⓘ Yes No

Load balancer ⓘ Standard

Network configuration ⓘ Basic Advanced

Review + create < Previous Next : Monitoring >

After the cluster is deployed, browse to the auto-created AKS resource group and select the DNS zone. Take note of the DNS zone name. This name is needed to deploy applications to the AKS cluster.

Home > Resource groups > MC_myAKSCluster_myAKSCluster_westeurope > 8d61f730-e2a7-4e57-93ab-48a331a3ee54.westeurope.aksapp.io

DNS zone

8d61f730-e2a7-4e57-93ab-48a331a3ee54.westeurope.aksapp.io

Record set Move Delete zone Refresh

Resource group (change)
mc_myakscluster_myakscluster_westeurope

Subscription (change)
Microsoft Internal - Billable

Subscription ID

Name server 1
ns1-01.azure-dns.com.
Name server 2
ns2-01.azure-dns.net.
Name server 3
ns3-01.azure-dns.org.
Name server 4
ns4-01.azure-dns.info.

Tags (change)
Click here to add tags

NAME	TYPE	TTL	VALUE
@	NS	172800	ns1-01.azure-dns.com. ns2-01.azure-dns.net. ns3-01.azure-dns.org. ns4-01.azure-dns.info.
@	SOA	3600	Email: azuredns-hostmaster.microsoft.com Host: ns1-01.azure-dns.com. Refresh: 3600 Retry: 300 Expire: 2419200 Minimum TTL: 300 Serial number: 1

Connect to your AKS cluster

To connect to the Kubernetes cluster from your local computer, you use `kubectl`, the Kubernetes command-line client.

If you use the Azure Cloud Shell, `kubectl` is already installed. You can also install it locally using the `az aks install-cli` command:

```
az aks install-cli
```

To configure `kubectl` to connect to your Kubernetes cluster, use the `az aks get-credentials` command. The following example gets credentials for the AKS cluster named *MyAKSCluster* in the *MyResourceGroup*.

```
az aks get-credentials --resource-group MyResourceGroup --name MyAKSCluster
```

Use HTTP routing

The HTTP application routing solution may only be triggered on Ingress resources that are annotated as follows:

```
annotations:  
  kubernetes.io/ingress.class: addon-http-application-routing
```

Create a file named **samples-http-application-routing.yaml** and copy in the following YAML. On line 43, update `<CLUSTER_SPECIFIC_DNS_ZONE>` with the DNS zone name collected in the previous step of this article.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: aks-helloworld
spec:
  replicas: 1
  selector:
    matchLabels:
      app: aks-helloworld
  template:
    metadata:
      labels:
        app: aks-helloworld
    spec:
      containers:
        - name: aks-helloworld
          image: mcr.microsoft.com/azuredocs/aks-helloworld:v1
          ports:
            - containerPort: 80
          env:
            - name: TITLE
              value: "Welcome to Azure Kubernetes Service (AKS)"
---
apiVersion: v1
kind: Service
metadata:
  name: aks-helloworld
spec:
  type: ClusterIP
  ports:
    - port: 80
  selector:
    app: aks-helloworld
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: aks-helloworld
  annotations:
    kubernetes.io/ingress.class: addon-http-application-routing
spec:
  rules:
    - host: aks-helloworld.<CLUSTER_SPECIFIC_DNS_ZONE>
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: aks-helloworld
                port:
                  number: 80

```

Use the [kubectl apply](#) command to create the resources.

```
kubectl apply -f samples-http-application-routing.yaml
```

The following example shows the created resources:

```
$ kubectl apply -f samples-http-application-routing.yaml

deployment.apps/aks-helloworld created
service/aks-helloworld created
ingress.networking.k8s.io/aks-helloworld created
```

Open a web browser to *aks-helloworld.<CLUSTER_SPECIFIC_DNS_ZONE>*, for example *aks-helloworld.9f9c1fe7-21a1-416d-99cd-3543bb92e4c3.eastus.aksapp.io* and verify you see the demo application. The application may take a few minutes to appear.

Remove HTTP routing

The HTTP routing solution can be removed using the Azure CLI. To do so run the following command, substituting your AKS cluster and resource group name.

```
az aks disable-addons --addons http_application_routing --name myAKSCluster --resource-group myResourceGroup
--no-wait
```

When the HTTP application routing add-on is disabled, some Kubernetes resources may remain in the cluster. These resources include *configMaps* and *secrets*, and are created in the *kube-system* namespace. To maintain a clean cluster, you may want to remove these resources.

Look for *addon-http-application-routing* resources using the following [kubectl get](#) commands:

```
kubectl get deployments --namespace kube-system
kubectl get services --namespace kube-system
kubectl get configmaps --namespace kube-system
kubectl get secrets --namespace kube-system
```

The following example output shows configMaps that should be deleted:

```
$ kubectl get configmaps --namespace kube-system

NAMESPACE      NAME                                     DATA   AGE
kube-system    addon-http-application-routing-nginx-configuration   0      9m7s
kube-system    addon-http-application-routing-tcp-services        0      9m7s
kube-system    addon-http-application-routing-udp-services        0      9m7s
```

To delete resources, use the [kubectl delete](#) command. Specify the resource type, resource name, and namespace. The following example deletes one of the previous configmaps:

```
kubectl delete configmaps addon-http-application-routing-nginx-configuration --namespace kube-system
```

Repeat the previous [kubectl delete](#) step for all *addon-http-application-routing* resources that remained in your cluster.

Troubleshoot

Use the [kubectl logs](#) command to view the application logs for the External-DNS application. The logs should confirm that an A and TXT DNS record were created successfully.

```
$ kubectl logs -f deploy/addon-http-application-routing-external-dns -n kube-system

time="2018-04-26T20:36:19Z" level=info msg="Updating A record named 'aks-helloworld' to '52.242.28.189' for
Azure DNS zone '471756a6-e744-4aa0-aa01-89c4d162a7a7.canadaeast.aksapp.io'."
time="2018-04-26T20:36:21Z" level=info msg="Updating TXT record named 'aks-helloworld' to
'"heritage=external-dns,external-dns/owner=default"' for Azure DNS zone '471756a6-e744-4aa0-aa01-
89c4d162a7a7.canadaeast.aksapp.io'."
```

These records can also be seen on the DNS zone resource in the Azure portal.

NAME	TYPE	TTL	VALUE
@	NS	172800	ns1-01.azure-dns.com. ns2-01.azure-dns.net. ns3-01.azure-dns.org. ns4-01.azure-dns.info.
@	SOA	3600	Email: azuredns-hostmaster.microsoft.com Host: ns1-01.azure-dns.com. Refresh: 3600 Retry: 300 Expire: 2419200 Minimum TTL: 300 Serial number: 1
party-clippy	A	300	40.91.216.190
party-clippy	TXT	300	"heritage=external-dns,external-dns/owner=default"

Use the `kubectl logs` command to view the application logs for the Nginx Ingress controller. The logs should confirm the `CREATE` of an Ingress resource and the reload of the controller. All HTTP activity is logged.

```
$ kubectl logs -f deploy/addon-http-application-routing-nginx-ingress-controller -n kube-system

-----
NGINX Ingress controller
  Release:    0.13.0
  Build:      git-4bc943a
  Repository: https://github.com/kubernetes/ingress-nginx
-----

I0426 20:30:12.212936      9 flags.go:162] Watching for ingress class: addon-http-application-routing
W0426 20:30:12.213041      9 flags.go:165] only Ingress with class "addon-http-application-routing" will be
processed by this ingress controller
W0426 20:30:12.213505      9 client_config.go:533] Neither --kubeconfig nor --master was specified. Using
the inClusterConfig. This might not work.
I0426 20:30:12.213752      9 main.go:181] Creating API client for https://10.0.0.1:443
I0426 20:30:12.287928      9 main.go:225] Running in Kubernetes Cluster version v1.8 (v1.8.11) - git
(clean) commit 1df6a8381669a6c753f79cb31ca2e3d57ee7c8a3 - platform linux/amd64
I0426 20:30:12.290988      9 main.go:84] validated kube-system/addon-http-application-routing-default-http-
backend as the default backend
I0426 20:30:12.294314      9 main.go:105] service kube-system/addon-http-application-routing-nginx-ingress
validated as source of Ingress status
I0426 20:30:12.426443      9 stat_collector.go:77] starting new nginx stats collector for Ingress
controller running in namespace (class addon-http-application-routing)
I0426 20:30:12.426509      9 stat_collector.go:78] collector extracting information from port 18080
I0426 20:30:12.448779      9 nginx.go:281] starting Ingress controller
I0426 20:30:12.463585      9 event.go:218] Event(v1.ObjectReference{Kind:"ConfigMap", Namespace:"kube-
system", Name:"addon-http-application-routing-nginx-configuration", UID:"2588536c-4990-11e8-a5e1-
0a58ac1f0ef2", APIVersion:"v1", ResourceVersion:"559", FieldPath:""}): type: 'Normal' reason: 'CREATE'
ConfigMap kube-system/addon-http-application-routing-nginx-configuration
I0426 20:30:12.466945      9 event.go:218] Event(v1.ObjectReference{Kind:"ConfigMap", Namespace:"kube-
system", Name:"addon-http-application-routing-tcp-services", UID:"258ca065-4990-11e8-a5e1-0a58ac1f0ef2",
APIVersion:"v1", ResourceVersion:"561", FieldPath:""}): type: 'Normal' reason: 'CREATE' ConfigMap kube-
system/addon-http-application-routing-tcp-services
I0426 20:30:12.467053      9 event.go:218] Event(v1.ObjectReference{Kind:"ConfigMap", Namespace:"kube-
system", Name:"addon-http-application-routing-udp-services", UID:"259023bc-4990-11e8-a5e1-0a58ac1f0ef2",
APIVersion:"v1", ResourceVersion:"562", FieldPath:""}): type: 'Normal' reason: 'CREATE' ConfigMap kube-
system/addon-http-application-routing-udp-services
I0426 20:30:13.649195      9 nginx.go:302] starting NGINX process...
I0426 20:30:13.649347      9 leaderelection.go:175] attempting to acquire leader lease  kube-
system/ingress-controller-leader-addon-http-application-routing...
I0426 20:30:13.649776      9 controller.go:170] backend reload required
I0426 20:30:13.649800      9 stat_collector.go:34] changing prometheus collector from  to default
I0426 20:30:13.662191      9 leaderelection.go:184] successfully acquired lease kube-system/ingress-
controller-leader-addon-http-application-routing
I0426 20:30:13.662292      9 status.go:196] new leader elected: addon-http-application-routing-nginx-
ingress-controller-5cxntd6
I0426 20:30:13.763362      9 controller.go:179] ingress backend successfully reloaded...
I0426 21:51:55.249327      9 event.go:218] Event(v1.ObjectReference{Kind:"Ingress", Namespace:"default",
Name:"aks-helloworld", UID:"092c9599-499c-11e8-a5e1-0a58ac1f0ef2", APIVersion:"extensions",
ResourceVersion:"7346", FieldPath:""}): type: 'Normal' reason: 'CREATE' Ingress default/aks-helloworld
W0426 21:51:57.908771      9 controller.go:775] service default/aks-helloworld does not have any active
endpoints
I0426 21:51:57.908951      9 controller.go:170] backend reload required
I0426 21:51:58.042932      9 controller.go:179] ingress backend successfully reloaded...
167.220.24.46 - [167.220.24.46] - - [26/Apr/2018:21:53:20 +0000] "GET / HTTP/1.1" 200 234 "" "Mozilla/5.0
(compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0)" 197 0.001 [default-aks-helloworld-80] 10.244.0.13:8080
234 0.004 200
```

Clean up

Remove the associated Kubernetes objects created in this article using `kubectl delete`.

```
kubectl delete -f samples-http-application-routing.yaml
```

The example output shows Kubernetes objects have been removed.

```
$ kubectl delete -f samples-http-application-routing.yaml

deployment "aks-helloworld" deleted
service "aks-helloworld" deleted
ingress "aks-helloworld" deleted
```

Next steps

For information on how to install an HTTPS-secured Ingress controller in AKS, see [HTTPS Ingress on Azure Kubernetes Service \(AKS\)](#).

Create an ingress controller to an internal virtual network in Azure Kubernetes Service (AKS)

4/26/2021 • 8 minutes to read • [Edit Online](#)

An ingress controller is a piece of software that provides reverse proxy, configurable traffic routing, and TLS termination for Kubernetes services. Kubernetes ingress resources are used to configure the ingress rules and routes for individual Kubernetes services. Using an ingress controller and ingress rules, a single IP address can be used to route traffic to multiple services in a Kubernetes cluster.

This article shows you how to deploy the [NGINX ingress controller](#) in an Azure Kubernetes Service (AKS) cluster. The ingress controller is configured on an internal, private virtual network and IP address. No external access is allowed. Two applications are then run in the AKS cluster, each of which is accessible over the single IP address.

You can also:

- [Create a basic ingress controller with external network connectivity](#)
- [Enable the HTTP application routing add-on](#)
- [Create an ingress controller that uses your own TLS certificates](#)
- Create an ingress controller that uses Let's Encrypt to automatically generate TLS certificates [with a dynamic public IP address](#) or [with a static public IP address](#)

Before you begin

This article uses [Helm 3](#) to install the NGINX ingress controller. Make sure that you are using the latest release of Helm and have access to the *ingress-nginx* Helm repository. For more information on configuring and using Helm, see [Install applications with Helm in Azure Kubernetes Service \(AKS\)](#).

This article also requires that you are running the Azure CLI version 2.0.64 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Create an ingress controller

By default, an NGINX ingress controller is created with a dynamic public IP address assignment. A common configuration requirement is to use an internal, private network and IP address. This approach allows you to restrict access to your services to internal users, with no external access.

Create a file named *internal-ingress.yaml* using the following example manifest file. This example assigns **10.240.0.42** to the *loadBalancerIP* resource. Provide your own internal IP address for use with the ingress controller. Make sure that this IP address is not already in use within your virtual network. Also, if you are using an existing virtual network and subnet, you must configure your AKS cluster with the correct permissions to manage the virtual network and subnet. See [Use kubenet networking with your own IP address ranges in Azure Kubernetes Service \(AKS\)](#) or [Configure Azure CNI networking in Azure Kubernetes Service \(AKS\)](#) for more information.

```
controller:  
  service:  
    loadBalancerIP: 10.240.0.42  
    annotations:  
      service.beta.kubernetes.io/azure-load-balancer-internal: "true"
```

Now deploy the *nginx-ingress* chart with Helm. To use the manifest file created in the previous step, add the `-f internal-ingress.yaml` parameter. For added redundancy, two replicas of the NGINX ingress controllers are deployed with the `--set controller.replicaCount` parameter. To fully benefit from running replicas of the ingress controller, make sure there's more than one node in your AKS cluster.

The ingress controller also needs to be scheduled on a Linux node. Windows Server nodes shouldn't run the ingress controller. A node selector is specified using the `--set controller.nodeSelector` parameter to tell the Kubernetes scheduler to run the NGINX ingress controller on a Linux-based node.

TIP

The following example creates a Kubernetes namespace for the ingress resources named *ingress-basic*. Specify a namespace for your own environment as needed. If your AKS cluster is not Kubernetes RBAC enabled, add `--set rbac.create=false` to the Helm commands.

TIP

If you would like to enable [client source IP preservation](#) for requests to containers in your cluster, add `--set controller.service.externalTrafficPolicy=Local` to the Helm install command. The client source IP is stored in the request header under *X-Forwarded-For*. When using an ingress controller with client source IP preservation enabled, TLS pass-through will not work.

```
# Create a namespace for your ingress resources
kubectl create namespace ingress-basic

# Add the ingress-nginx repository
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

# Use Helm to deploy an NGINX ingress controller
helm install nginx-ingress ingress-nginx/ingress-nginx \
--namespace ingress-basic \
-f internal-ingress.yaml \
--set controller.replicaCount=2 \
--set controller.nodeSelector."beta\\.kubernetes\\.io/os"=linux \
--set defaultBackend.nodeSelector."beta\\.kubernetes\\.io/os"=linux \
--set controller.admissionWebhooks.patch.nodeSelector."beta\\.kubernetes\\.io/os"=linux
```

When the Kubernetes load balancer service is created for the NGINX ingress controller, your internal IP address is assigned. To get the public IP address, use the `kubectl get service` command.

```
kubectl --namespace ingress-basic get services -o wide -w nginx-ingress-ingress-nginx-controller
```

It takes a few minutes for the IP address to be assigned to the service, as shown in the following example output:

```
$ kubectl --namespace ingress-basic get services -o wide -w nginx-ingress-ingress-nginx-controller
NAME                                     TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE   SELECTOR
nginx-ingress-ingress-nginx-controller   LoadBalancer  10.0.74.133    EXTERNAL_IP
80:32486/TCP,443:30953/TCP   44s   app.kubernetes.io/component=controller,app.kubernetes.io/instance=nginx-
ingress,app.kubernetes.io/name=ingress-nginx
```

No ingress rules have been created yet, so the NGINX ingress controller's default 404 page is displayed if you browse to the internal IP address. Ingress rules are configured in the following steps.

Run demo applications

To see the ingress controller in action, run two demo applications in your AKS cluster. In this example, you use `kubectl apply` to deploy two instances of a simple *Hello world* application.

Create a *aks-helloworld.yaml* file and copy in the following example YAML:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: aks-helloworld
spec:
  replicas: 1
  selector:
    matchLabels:
      app: aks-helloworld
  template:
    metadata:
      labels:
        app: aks-helloworld
    spec:
      containers:
        - name: aks-helloworld
          image: mcr.microsoft.com/azuredocs/aks-helloworld:v1
          ports:
            - containerPort: 80
          env:
            - name: TITLE
              value: "Welcome to Azure Kubernetes Service (AKS)"
---
apiVersion: v1
kind: Service
metadata:
  name: aks-helloworld
spec:
  type: ClusterIP
  ports:
    - port: 80
  selector:
    app: aks-helloworld
```

Create a *ingress-demo.yaml* file and copy in the following example YAML:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: ingress-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ingress-demo
  template:
    metadata:
      labels:
        app: ingress-demo
    spec:
      containers:
        - name: ingress-demo
          image: mcr.microsoft.com/azuredocs/aks-helloworld:v1
          ports:
            - containerPort: 80
          env:
            - name: TITLE
              value: "AKS Ingress Demo"
---
apiVersion: v1
kind: Service
metadata:
  name: ingress-demo
spec:
  type: ClusterIP
  ports:
    - port: 80
  selector:
    app: ingress-demo

```

Run the two demo applications using `kubectl apply`:

```

kubectl apply -f aks-helloworld.yaml --namespace ingress-basic
kubectl apply -f ingress-demo.yaml --namespace ingress-basic

```

Create an ingress route

Both applications are now running on your Kubernetes cluster. To route traffic to each application, create a Kubernetes ingress resource. The ingress resource configures the rules that route traffic to one of the two applications.

In the following example, traffic to the address `http://10.240.0.42/` is routed to the service named `aks-helloworld`. Traffic to the address `http://10.240.0.42/hello-world-two` is routed to the `ingress-demo` service.

Create a file named `hello-world-ingress.yaml` and copy in the following example YAML.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hello-world-ingress
  namespace: ingress-basic
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
  - http:
      paths:
      - path: /hello-world-one(/|$(.))
        pathType: Prefix
        backend:
          service:
            name: aks-helloworld
            port:
              number: 80
      - path: /hello-world-two(/|$(.))
        pathType: Prefix
        backend:
          service:
            name: ingress-demo
            port:
              number: 80
      - path: /(.*)
        pathType: Prefix
        backend:
          service:
            name: aks-helloworld
            port:
              number: 80
```

Create the ingress resource using the `kubectl apply -f hello-world-ingress.yaml` command.

```
kubectl apply -f hello-world-ingress.yaml
```

The following example output shows the ingress resource is created.

```
$ kubectl apply -f hello-world-ingress.yaml
ingress.extensions/hello-world-ingress created
```

Test the ingress controller

To test the routes for the ingress controller, browse to the two applications with a web client. If needed, you can quickly test this internal-only functionality from a pod on the AKS cluster. Create a test pod and attach a terminal session to it:

```
kubectl run -it --rm aks-ingress-test --image=mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11 --
namespace ingress-basic
```

Install `curl` in the pod using `apt-get`:

```
apt-get update && apt-get install -y curl
```

Now access the address of your Kubernetes ingress controller using `curl`, such as <http://10.240.0.42>. Provide your own internal IP address specified when you deployed the ingress controller in the first step of this article.

```
curl -L http://10.240.0.42
```

No additional path was provided with the address, so the ingress controller defaults to the `/route`. The first demo application is returned, as shown in the following condensed example output:

```
$ curl -L http://10.240.0.42

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <link rel="stylesheet" type="text/css" href="/static/default.css">
    <title>Welcome to Azure Kubernetes Service (AKS)</title>
[...]
```

Now add `/hello-world-two` path to the address, such as <http://10.240.0.42/hello-world-two>. The second demo application with the custom title is returned, as shown in the following condensed example output:

```
$ curl -L -k http://10.240.0.42/hello-world-two

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <link rel="stylesheet" type="text/css" href="/static/default.css">
    <title>AKS Ingress Demo</title>
[...]
```

Clean up resources

This article used Helm to install the ingress components. When you deploy a Helm chart, a number of Kubernetes resources are created. These resources includes pods, deployments, and services. To clean up these resources, you can either delete the entire sample namespace, or the individual resources.

Delete the sample namespace and all resources

To delete the entire sample namespace, use the `kubectl delete` command and specify your namespace name. All the resources in the namespace are deleted.

```
kubectl delete namespace ingress-basic
```

Delete resources individually

Alternatively, a more granular approach is to delete the individual resources created. List the Helm releases with the `helm list` command.

```
helm list --namespace ingress-basic
```

Look for charts named `nginx-ingress` and `aks-helloworld`, as shown in the following example output:

\$ helm list --namespace ingress-basic					
NAME	NAMESPACE	REVISION	UPDATED	STATUS	
CHART	APP VERSION				
nginx-ingress	ingress-basic	1	2020-01-06 19:55:46.358275 -0600 CST		deployed
nginx-ingress-1.27.1	0.26.1				

Uninstall the releases with the `helm uninstall` command.

```
helm uninstall nginx-ingress --namespace ingress-basic
```

The following example uninstalls the NGINX ingress deployment.

```
$ helm uninstall nginx-ingress --namespace ingress-basic  
release "nginx-ingress" uninstalled
```

Next, remove the two sample applications:

```
kubectl delete -f aks-helloworld.yaml --namespace ingress-basic  
kubectl delete -f ingress-demo.yaml --namespace ingress-basic
```

Remove the ingress route that directed traffic to the sample apps:

```
kubectl delete -f hello-world-ingress.yaml
```

Finally, you can delete the itself namespace. Use the `kubectl delete` command and specify your namespace name:

```
kubectl delete namespace ingress-basic
```

Next steps

This article included some external components to AKS. To learn more about these components, see the following project pages:

- [Helm CLI](#)
- [NGINX ingress controller](#)

You can also:

- [Create a basic ingress controller with external network connectivity](#)
- [Enable the HTTP application routing add-on](#)
- [Create an ingress controller with a dynamic public IP and configure Let's Encrypt to automatically generate TLS certificates](#)
- [Create an ingress controller with a static public IP address and configure Let's Encrypt to automatically generate TLS certificates](#)

Tutorial: Enable Application Gateway Ingress Controller add-on for an existing AKS cluster with an existing Application Gateway

4/21/2021 • 6 minutes to read • [Edit Online](#)

You can use Azure CLI or Portal to enable the [Application Gateway Ingress Controller \(AGIC\)](#) add-on for an existing [Azure Kubernetes Services \(AKS\)](#) cluster. In this tutorial, you'll learn how to use AGIC add-on to expose your Kubernetes application in an existing AKS cluster through an existing Application Gateway deployed in separate virtual networks. You'll start by creating an AKS cluster in one virtual network and an Application Gateway in a separate virtual network to simulate existing resources. You'll then enable the AGIC add-on, peer the two virtual networks together, and deploy a sample application that will be exposed through the Application Gateway using the AGIC add-on. If you're enabling the AGIC add-on for an existing Application Gateway and existing AKS cluster in the same virtual network, then you can skip the peering step below. The add-on provides a much faster way of deploying AGIC for your AKS cluster than [previously through Helm](#) and also offers a fully managed experience.

In this tutorial, you learn how to:

- Create a resource group
- Create a new AKS cluster
- Create a new Application Gateway
- Enable the AGIC add-on in the existing AKS cluster through Azure CLI
- Enable the AGIC add-on in the existing AKS cluster through Portal
- Peer the Application Gateway virtual network with the AKS cluster virtual network
- Deploy a sample application using AGIC for Ingress on the AKS cluster
- Check that the application is reachable through Application Gateway

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

Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#).
A blue rectangular button with a white 'A' icon and the text "Launch Cloud Shell".
- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
 - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
 - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
 - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).

Create a resource group

In Azure, you allocate related resources to a resource group. Create a resource group by using [az group create](#).

The following example creates a resource group named *myResourceGroup* in the *canadacentral*/location (region).

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

Deploy a new AKS cluster

You'll now deploy a new AKS cluster, to simulate having an existing AKS cluster that you want to enable the AGIC add-on for.

In the following example, you'll be deploying a new AKS cluster named *myCluster* using [Azure CNI](#) and [Managed Identities](#) in the resource group you created, *myResourceGroup*.

```
az aks create -n myCluster -g myResourceGroup --network-plugin azure --enable-managed-identity
```

To configure additional parameters for the `az aks create` command, visit references [here](#).

Deploy a new Application Gateway

You'll now deploy a new Application Gateway, to simulate having an existing Application Gateway that you want to use to load balance traffic to your AKS cluster, *myCluster*. The name of the Application Gateway will be *myApplicationGateway*, but you will need to first create a public IP resource, named *myPublicIp*, and a new virtual network called *myVnet* with address space 11.0.0.0/8, and a subnet with address space 11.1.0.0/16 called *mySubnet*, and deploy your Application Gateway in *mySubnet* using *myPublicIp*.

When using an AKS cluster and Application Gateway in separate virtual networks, the address spaces of the two virtual networks must not overlap. The default address space that an AKS cluster deploys in is 10.0.0.0/8, so we set the Application Gateway virtual network address prefix to 11.0.0.0/8.

```
az network public-ip create -n myPublicIp -g MyResourceGroup --allocation-method Static --sku Standard
az network vnet create -n myVnet -g myResourceGroup --address-prefix 11.0.0.0/8 --subnet-name mySubnet --
subnet-prefix 11.1.0.0/16
az network application-gateway create -n myApplicationGateway -l canadacentral -g myResourceGroup --sku
Standard_v2 --public-ip-address myPublicIp --vnet-name myVnet --subnet mySubnet
```

NOTE

Application Gateway Ingress Controller (AGIC) add-on **only** supports Application Gateway v2 SKUs (Standard and WAF), and **not** the Application Gateway v1 SKUs.

Enable the AGIC add-on in existing AKS cluster through Azure CLI

If you'd like to continue using Azure CLI, you can continue to enable the AGIC add-on in the AKS cluster you created, *myCluster*, and specify the AGIC add-on to use the existing Application Gateway you created, *myApplicationGateway*.

```
appgwId=$(az network application-gateway show -n myApplicationGateway -g myResourceGroup -o tsv --query
"id")
az aks enable-addons -n myCluster -g myResourceGroup -a ingress-appgw --appgw-id $appgwId
```

Enable the AGIC add-on in existing AKS cluster through Portal

If you'd like to use Azure portal to enable AGIC add-on, go to (<https://aka.ms/azure/portal/aks/agic>) and navigate to your AKS cluster through the Portal link. From there, go to the Networking tab within your AKS cluster. You'll see an Application Gateway ingress controller section, which allows you to enable/disable the ingress controller add-on using the Portal UI. Check the box next to "Enable ingress controller", and select the Application Gateway you created, *myApplicationGateway* from the dropdown menu.



Application Gateway ingress controller

Enable ingress controller

Application gateway (New) ingress-appgateway

Create new

Peer the two virtual networks together

Since we deployed the AKS cluster in its own virtual network and the Application Gateway in another virtual network, you'll need to peer the two virtual networks together in order for traffic to flow from the Application Gateway to the pods in the cluster. Peering the two virtual networks requires running the Azure CLI command two separate times, to ensure that the connection is bi-directional. The first command will create a peering connection from the Application Gateway virtual network to the AKS virtual network; the second command will create a peering connection in the other direction.

```
nodeResourceGroup=$(az aks show -n myCluster -g myResourceGroup -o tsv --query "nodeResourceGroup")
aksVnetName=$(az network vnet list -g $nodeResourceGroup -o tsv --query "[0].name")

aksVnetId=$(az network vnet show -n $aksVnetName -g $nodeResourceGroup -o tsv --query "id")
az network vnet peering create -n AppGwtoAKSVnetPeering -g myResourceGroup --vnet-name myVnet --remote-vnet
$aksVnetId --allow-vnet-access

appGVnetId=$(az network vnet show -n myVnet -g myResourceGroup -o tsv --query "id")
az network vnet peering create -n AKStoAppGVnetPeering -g $nodeResourceGroup --vnet-name $aksVnetName --
remote-vnet $appGVnetId --allow-vnet-access
```

Deploy a sample application using AGIC

You'll now deploy a sample application to the AKS cluster you created that will use the AGIC add-on for Ingress and connect the Application Gateway to the AKS cluster. First, you'll get credentials to the AKS cluster you deployed by running the `az aks get-credentials` command.

```
az aks get-credentials -n myCluster -g myResourceGroup
```

Once you have the credentials to the cluster you created, run the following command to set up a sample application that uses AGIC for Ingress to the cluster. AGIC will update the Application Gateway you set up earlier with corresponding routing rules to the new sample application you deployed.

```
kubectl apply -f https://raw.githubusercontent.com/Azure/application-gateway-kubernetes-ingress/master/docs/examples/aspnetapp.yaml
```

Check that the application is reachable

Now that the Application Gateway is set up to serve traffic to the AKS cluster, let's verify that your application is

reachable. You'll first get the IP address of the Ingress.

```
kubectl get ingress
```

Check that the sample application you created is up and running by either visiting the IP address of the Application Gateway that you got from running the above command or check with `curl`. It may take Application Gateway a minute to get the update, so if the Application Gateway is still in an "Updating" state on Portal, then let it finish before trying to reach the IP address.

Clean up resources

When no longer needed, remove the resource group, application gateway, and all related resources.

```
az group delete --name myResourceGroup
```

Next steps

[Learn more about disabling the AGIC add-on](#)

Create an HTTPS ingress controller and use your own TLS certificates on Azure Kubernetes Service (AKS)

4/26/2021 • 9 minutes to read • [Edit Online](#)

An ingress controller is a piece of software that provides reverse proxy, configurable traffic routing, and TLS termination for Kubernetes services. Kubernetes ingress resources are used to configure the ingress rules and routes for individual Kubernetes services. Using an ingress controller and ingress rules, a single IP address can be used to route traffic to multiple services in a Kubernetes cluster.

This article shows you how to deploy the [NGINX ingress controller](#) in an Azure Kubernetes Service (AKS) cluster. You generate your own certificates, and create a Kubernetes secret for use with the ingress route. Finally, two applications are run in the AKS cluster, each of which is accessible over a single IP address.

You can also:

- [Create a basic ingress controller with external network connectivity](#)
- [Enable the HTTP application routing add-on](#)
- [Create an ingress controller that uses an internal, private network and IP address](#)
- Create an ingress controller that uses Let's Encrypt to automatically generate TLS certificates [with a dynamic public IP address](#) or [with a static public IP address](#)

Before you begin

This article uses [Helm 3](#) to install the NGINX ingress controller. Make sure that you are using the latest release of Helm and have access to the *ingress-nginx* Helm repository. For upgrade instructions, see the [Helm install docs](#). For more information on configuring and using Helm, see [Install applications with Helm in Azure Kubernetes Service \(AKS\)](#).

This article also requires that you are running the Azure CLI version 2.0.64 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Create an ingress controller

To create the ingress controller, use `Helm` to install *nginx-ingress*. For added redundancy, two replicas of the NGINX ingress controllers are deployed with the `--set controller.replicaCount` parameter. To fully benefit from running replicas of the ingress controller, make sure there's more than one node in your AKS cluster.

The ingress controller also needs to be scheduled on a Linux node. Windows Server nodes shouldn't run the ingress controller. A node selector is specified using the `--set nodeSelector` parameter to tell the Kubernetes scheduler to run the NGINX ingress controller on a Linux-based node.

TIP

The following example creates a Kubernetes namespace for the ingress resources named *ingress-basic*. Specify a namespace for your own environment as needed. If your AKS cluster is not Kubernetes RBAC enabled, add `--set rbac.create=false` to the Helm commands.

TIP

If you would like to enable [client source IP preservation](#) for requests to containers in your cluster, add

```
--set controller.service.externalTrafficPolicy=Local
```

 to the Helm install command. The client source IP is stored in the request header under *X-Forwarded-For*. When using an ingress controller with client source IP preservation enabled, TLS pass-through will not work.

```
# Create a namespace for your ingress resources
kubectl create namespace ingress-basic

# Add the ingress-nginx repository
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

# Use Helm to deploy an NGINX ingress controller
helm install nginx-ingress ingress-nginx/ingress-nginx \
--namespace ingress-basic \
--set controller.replicaCount=2 \
--set controller.nodeSelector."beta\.kubernetes\.io/os"=linux \
--set defaultBackend.nodeSelector."beta\.kubernetes\.io/os"=linux \
--set controller.admissionWebhooks.patch.nodeSelector."beta\.kubernetes\.io/os"=linux
```

During the installation, an Azure public IP address is created for the ingress controller. This public IP address is static for the life-span of the ingress controller. If you delete the ingress controller, the public IP address assignment is lost. If you then create an additional ingress controller, a new public IP address is assigned. If you wish to retain the use of the public IP address, you can instead [create an ingress controller with a static public IP address](#).

To get the public IP address, use the `kubectl get service` command.

```
kubectl --namespace ingress-basic get services -o wide -w nginx-ingress-ingress-nginx-controller
```

It takes a few minutes for the IP address to be assigned to the service.

```
$ kubectl --namespace ingress-basic get services -o wide -w nginx-ingress-ingress-nginx-controller
NAME                  TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE      SELECTOR
nginx-ingress-ingress-nginx-controller   LoadBalancer   10.0.74.133    EXTERNAL_IP
80:32486/TCP,443:30953/TCP   44s    app.kubernetes.io/component=controller,app.kubernetes.io/instance=nginx-
ingress,app.kubernetes.io/name=ingress-nginx
```

Make a note of this public IP address, as it's used in the last step to test the deployment.

No ingress rules have been created yet. If you browse to the public IP address, the NGINX ingress controller's default 404 page is displayed.

Generate TLS certificates

For this article, let's generate a self-signed certificate with `openssl`. For production use, you should request a trusted, signed certificate through a provider or your own certificate authority (CA). In the next step, you generate a Kubernetes *Secret* using the TLS certificate and private key generated by OpenSSL.

The following example generates a 2048-bit RSA X509 certificate valid for 365 days named *aks-ingress-tls.crt*. The private key file is named *aks-ingress-tls.key*. A Kubernetes TLS secret requires both of these files.

This article works with the *demo.azure.com* subject common name and doesn't need to be changed. For

production use, specify your own organizational values for the `-subj` parameter:

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
-out aks-ingress-tls.crt \
-keyout aks-ingress-tls.key \
-subj "/CN=demo.azure.com/O=aks-ingress-tls"
```

Create Kubernetes secret for the TLS certificate

To allow Kubernetes to use the TLS certificate and private key for the ingress controller, you create and use a Secret. The secret is defined once, and uses the certificate and key file created in the previous step. You then reference this secret when you define ingress routes.

The following example creates a Secret name *aks-ingress-tls*:

```
kubectl create secret tls aks-ingress-tls \
--namespace ingress-basic \
--key aks-ingress-tls.key \
--cert aks-ingress-tls.crt
```

Run demo applications

An ingress controller and a Secret with your certificate have been configured. Now let's run two demo applications in your AKS cluster. In this example, Helm is used to deploy two instances of a simple 'Hello world' application.

To see the ingress controller in action, run two demo applications in your AKS cluster. In this example, you use `kubectl apply` to deploy two instances of a simple *Hello world* application.

Create a *aks-helloworld.yaml* file and copy in the following example YAML:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: aks-helloworld
spec:
  replicas: 1
  selector:
    matchLabels:
      app: aks-helloworld
  template:
    metadata:
      labels:
        app: aks-helloworld
    spec:
      containers:
        - name: aks-helloworld
          image: mcr.microsoft.com/azuredocs/aks-helloworld:v1
          ports:
            - containerPort: 80
          env:
            - name: TITLE
              value: "Welcome to Azure Kubernetes Service (AKS)"
---
apiVersion: v1
kind: Service
metadata:
  name: aks-helloworld
spec:
  type: ClusterIP
  ports:
    - port: 80
  selector:
    app: aks-helloworld
```

Create a *ingress-demo.yaml*/file and copy in the following example YAML:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: ingress-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ingress-demo
  template:
    metadata:
      labels:
        app: ingress-demo
    spec:
      containers:
        - name: ingress-demo
          image: mcr.microsoft.com/azuredocs/aks-helloworld:v1
          ports:
            - containerPort: 80
          env:
            - name: TITLE
              value: "AKS Ingress Demo"
---
apiVersion: v1
kind: Service
metadata:
  name: ingress-demo
spec:
  type: ClusterIP
  ports:
    - port: 80
  selector:
    app: ingress-demo

```

Run the two demo applications using `kubectl apply`:

```

kubectl apply -f aks-helloworld.yaml --namespace ingress-basic
kubectl apply -f ingress-demo.yaml --namespace ingress-basic

```

Create an ingress route

Both applications are now running on your Kubernetes cluster, however they're configured with a service of type `ClusterIP`. As such, the applications aren't accessible from the internet. To make them publicly available, create a Kubernetes ingress resource. The ingress resource configures the rules that route traffic to one of the two applications.

In the following example, traffic to the address `https://demo.azure.com/` is routed to the service named `aks-helloworld`. Traffic to the address `https://demo.azure.com/hello-world-two` is routed to the `ingress-demo` service. For this article, you don't need to change those demo host names. For production use, provide the names specified as part of the certificate request and generation process.

TIP

If the host name specified during the certificate request process, the CN name, doesn't match the host defined in your ingress route, your ingress controller displays a *Kubernetes Ingress Controller Fake Certificate* warning. Make sure your certificate and ingress route host names match.

The `tls` section tells the ingress route to use the Secret named `aks-ingress-tls` for the host `demo.azure.com`. Again, for production use, specify your own host address.

Create a file named `hello-world-ingress.yaml` and copy in the following example YAML.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hello-world-ingress
  namespace: ingress-basic
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  tls:
  - hosts:
    - demo.azure.com
    secretName: aks-ingress-tls
  rules:
  - host: demo.azure.com
    http:
      paths:
      - path: /hello-world-one(/|$(.).*)
        pathType: Prefix
        backend:
          service:
            name: aks-helloworld
            port:
              number: 80
      - path: /hello-world-two(/|$(.).*)
        pathType: Prefix
        backend:
          service:
            name: ingress-demo
            port:
              number: 80
      - path: /(.*)
        pathType: Prefix
        backend:
          service:
            name: aks-helloworld
            port:
              number: 80
```

Create the ingress resource using the `kubectl apply -f hello-world-ingress.yaml` command.

```
kubectl apply -f hello-world-ingress.yaml
```

The example output shows the ingress resource is created.

```
$ kubectl apply -f hello-world-ingress.yaml
ingress.extensions/hello-world-ingress created
```

Test the ingress configuration

To test the certificates with our fake `demo.azure.com` host, use `curl` and specify the `--resolve` parameter. This parameter lets you map the `demo.azure.com` name to the public IP address of your ingress controller. Specify the public IP address of your own ingress controller, as shown in the following example:

```
curl -v -k --resolve demo.azure.com:443:EXTERNAL_IP https://demo.azure.com
```

No additional path was provided with the address, so the ingress controller defaults to the /route. The first demo application is returned, as shown in the following condensed example output:

```
$ curl -v -k --resolve demo.azure.com:443:EXTERNAL_IP https://demo.azure.com

[...]
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <link rel="stylesheet" type="text/css" href="/static/default.css">
  <title>Welcome to Azure Kubernetes Service (AKS)</title>
[...]
```

The `-v` parameter in our `curl` command outputs verbose information, including the TLS certificate received. Half-way through your curl output, you can verify that your own TLS certificate was used. The `-k` parameter continues loading the page even though we're using a self-signed certificate. The following example shows that the `issuer: CN=demo.azure.com; O=aks-ingress-tls` certificate was used:

```
[...]
* Server certificate:
*   subject: CN=demo.azure.com; O=aks-ingress-tls
*   start date: Oct 22 22:13:54 2018 GMT
*   expire date: Oct 22 22:13:54 2019 GMT
*   issuer: CN=demo.azure.com; O=aks-ingress-tls
*   SSL certificate verify result: self signed certificate (18), continuing anyway.
[...]
```

Now add `/hello-world-two` path to the address, such as `https://demo.azure.com/hello-world-two`. The second demo application with the custom title is returned, as shown in the following condensed example output:

```
$ curl -v -k --resolve demo.azure.com:443:EXTERNAL_IP https://demo.azure.com/hello-world-two

[...]
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <link rel="stylesheet" type="text/css" href="/static/default.css">
  <title>AKS Ingress Demo</title>
[...]
```

Clean up resources

This article used Helm to install the ingress components and sample apps. When you deploy a Helm chart, a number of Kubernetes resources are created. These resources include pods, deployments, and services. To clean up these resources, you can either delete the entire sample namespace, or the individual resources.

Delete the sample namespace and all resources

To delete the entire sample namespace, use the `kubectl delete` command and specify your namespace name. All the resources in the namespace are deleted.

```
kubectl delete namespace ingress-basic
```

Delete resources individually

Alternatively, a more granular approach is to delete the individual resources created. List the Helm releases with the `helm list` command.

```
helm list --namespace ingress-basic
```

Look for chart named *nginx-ingress* as shown in the following example output:

```
$ helm list --namespace ingress-basic
```

NAME	NAMESPACE	REVISION	UPDATED	STATUS
CHART	APP VERSION			
nginx-ingress	ingress-basic	1	2020-01-06 19:55:46.358275 -0600 CST	deployed
nginx-ingress-1.27.1	0.26.1			

Uninstall the releases with the `helm uninstall` command.

```
helm uninstall nginx-ingress --namespace ingress-basic
```

The following example uninstalls the NGINX ingress deployment.

```
$ helm uninstall nginx-ingress --namespace ingress-basic
```

```
release "nginx-ingress" uninstalled
```

Next, remove the two sample applications:

```
kubectl delete -f aks-helloworld.yaml --namespace ingress-basic
```

```
kubectl delete -f ingress-demo.yaml --namespace ingress-basic
```

Remove the ingress route that directed traffic to the sample apps:

```
kubectl delete -f hello-world-ingress.yaml
```

Delete the certificate Secret:

```
kubectl delete secret aks-ingress-tls --namespace ingress-basic
```

Finally, you can delete the itself namespace. Use the `kubectl delete` command and specify your namespace name:

```
kubectl delete namespace ingress-basic
```

Next steps

This article included some external components to AKS. To learn more about these components, see the following project pages:

- [Helm CLI](#)
- [NGINX ingress controller](#)

You can also:

- Create a basic ingress controller with external network connectivity
- Enable the HTTP application routing add-on
- Create an ingress controller that uses an internal, private network and IP address
- Create an ingress controller that uses Let's Encrypt to automatically generate TLS certificates [with a dynamic public IP address](#) or [with a static public IP address](#)

Create an HTTPS ingress controller on Azure Kubernetes Service (AKS)

4/26/2021 • 10 minutes to read • [Edit Online](#)

An ingress controller is a piece of software that provides reverse proxy, configurable traffic routing, and TLS termination for Kubernetes services. Kubernetes ingress resources are used to configure the ingress rules and routes for individual Kubernetes services. Using an ingress controller and ingress rules, a single IP address can be used to route traffic to multiple services in a Kubernetes cluster.

This article shows you how to deploy the [NGINX ingress controller](#) in an Azure Kubernetes Service (AKS) cluster. The [cert-manager](#) project is used to automatically generate and configure [Let's Encrypt](#) certificates. Finally, two applications are run in the AKS cluster, each of which is accessible over a single IP address.

You can also:

- [Create a basic ingress controller with external network connectivity](#)
- [Enable the HTTP application routing add-on](#)
- [Create an ingress controller that uses an internal, private network and IP address](#)
- [Create an ingress controller that uses your own TLS certificates](#)
- [Create an ingress controller that uses Let's Encrypt to automatically generate TLS certificates with a static public IP address](#)

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart [using the Azure CLI](#) or [using the Azure portal](#).

This article also assumes you have [a custom domain](#) with a [DNS Zone](#) in the same resource group as your AKS cluster.

This article uses [Helm 3](#) to install the NGINX ingress controller and cert-manager. Make sure that you are using the latest release of Helm and have access to the *ingress-nginx* and *jetstack* Helm repositories. For upgrade instructions, see the [Helm install docs](#). For more information on configuring and using Helm, see [Install applications with Helm in Azure Kubernetes Service \(AKS\)](#).

This article also requires that you are running the Azure CLI version 2.0.64 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Create an ingress controller

To create the ingress controller, use the `helm` command to install *nginx-ingress*. For added redundancy, two replicas of the NGINX ingress controllers are deployed with the `--set controller.replicaCount` parameter. To fully benefit from running replicas of the ingress controller, make sure there's more than one node in your AKS cluster.

The ingress controller also needs to be scheduled on a Linux node. Windows Server nodes shouldn't run the ingress controller. A node selector is specified using the `--set nodeSelector` parameter to tell the Kubernetes scheduler to run the NGINX ingress controller on a Linux-based node.

TIP

The following example creates a Kubernetes namespace for the ingress resources named *ingress-basic*. Specify a namespace for your own environment as needed.

TIP

If you would like to enable [client source IP preservation](#) for requests to containers in your cluster, add

```
--set controller.service.externalTrafficPolicy=Local
```

 to the Helm install command. The client source IP is stored in the request header under *X-Forwarded-For*. When using an ingress controller with client source IP preservation enabled, TLS pass-through will not work.

```
# Create a namespace for your ingress resources
kubectl create namespace ingress-basic

# Add the ingress-nginx repository
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

# Use Helm to deploy an NGINX ingress controller
helm install nginx-ingress ingress-nginx/ingress-nginx \
--namespace ingress-basic \
--set controller.replicaCount=2 \
--set controller.nodeSelector."beta\.kubernetes\.io/os"=linux \
--set defaultBackend.nodeSelector."beta\.kubernetes\.io/os"=linux \
--set controller.admissionWebhooks.patch.nodeSelector."beta\.kubernetes\.io/os"=linux
```

During the installation, an Azure public IP address is created for the ingress controller. This public IP address is static for the life-span of the ingress controller. If you delete the ingress controller, the public IP address assignment is lost. If you then create an additional ingress controller, a new public IP address is assigned. If you wish to retain the use of the public IP address, you can instead [create an ingress controller with a static public IP address](#).

To get the public IP address, use the `kubectl get service` command. It takes a few minutes for the IP address to be assigned to the service.

```
$ kubectl --namespace ingress-basic get services -o wide -w nginx-ingress-ingress-nginx-controller
NAME                                     TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE   SELECTOR
nginx-ingress-ingress-nginx-controller   LoadBalancer  10.0.74.133    EXTERNAL_IP
80:32486/TCP,443:30953/TCP   44s   app.kubernetes.io/component=controller,app.kubernetes.io/instance=nginx-
ingress,app.kubernetes.io/name=ingress-nginx
```

No ingress rules have been created yet. If you browse to the public IP address, the NGINX ingress controller's default 404 page is displayed.

Add an A record to your DNS zone

Add an *A* record to your DNS zone with the external IP address of the NGINX service using [az network dns record-set a add-record](#).

```
az network dns record-set a add-record \
--resource-group myResourceGroup \
--zone-name MY_CUSTOM_DOMAIN \
--record-set-name * \
--ipv4-address MY_EXTERNAL_IP
```

NOTE

Optionally, you can configure an FQDN for the ingress controller IP address instead of a custom domain. Note that this sample is for a Bash shell.

```
# Public IP address of your ingress controller
IP="MY_EXTERNAL_IP"

# Name to associate with public IP address
DNSNAME="demo-aks-ingress"

# Get the resource-id of the public ip
PUBLICIPID=$(az network public-ip list --query "[?ipAddress!=null] | [?contains(ipAddress, '$IP')].[id]" -o tsv)

# Update public ip address with DNS name
az network public-ip update --ids $PUBLICIPID --dns-name $DNSNAME

# Display the FQDN
az network public-ip show --ids $PUBLICIPID --query "[dnsSettings.fqdn]" --output tsv
```

Install cert-manager

The NGINX ingress controller supports TLS termination. There are several ways to retrieve and configure certificates for HTTPS. This article demonstrates using [cert-manager](#), which provides automatic [Lets Encrypt](#) certificate generation and management functionality.

To install the cert-manager controller:

```
# Label the ingress-basic namespace to disable resource validation
kubectl label namespace ingress-basic cert-manager.io/disable-validation=true

# Add the Jetstack Helm repository
helm repo add jetstack https://charts.jetstack.io

# Update your local Helm chart repository cache
helm repo update

# Install the cert-manager Helm chart
helm install cert-manager jetstack/cert-manager \
--namespace ingress-basic \
--set installCRDs=true \
--set nodeSelector."kubernetes\\.io/os"=linux \
--set webhook.nodeSelector."kubernetes\\.io/os"=linux \
--set cainjector.nodeSelector."kubernetes\\.io/os"=linux
```

For more information on cert-manager configuration, see the [cert-manager project](#).

Create a CA cluster issuer

Before certificates can be issued, cert-manager requires an [Issuer](#) or [ClusterIssuer](#) resource. These Kubernetes resources are identical in functionality, however [Issuer](#) works in a single namespace, and [ClusterIssuer](#) works across all namespaces. For more information, see the [cert-manager issuer](#) documentation.

Create a cluster issuer, such as `cluster-issuer.yaml`, using the following example manifest. Update the email address with a valid address from your organization:

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: MY_EMAIL_ADDRESS
    privateKeySecretRef:
      name: letsencrypt
    solvers:
      - http01:
          ingress:
            class: nginx
            podTemplate:
              spec:
                nodeSelector:
                  "kubernetes.io/os": linux
```

To create the issuer, use the `kubectl apply` command.

```
kubectl apply -f cluster-issuer.yaml
```

Run demo applications

An ingress controller and a certificate management solution have been configured. Now let's run two demo applications in your AKS cluster. In this example, Helm is used to deploy two instances of a simple *Hello world* application.

To see the ingress controller in action, run two demo applications in your AKS cluster. In this example, you use `kubectl apply` to deploy two instances of a simple *Hello world* application.

Create a `aks-helloworld-one.yaml` file and copy in the following example YAML:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: aks-helloworld-one
spec:
  replicas: 1
  selector:
    matchLabels:
      app: aks-helloworld-one
  template:
    metadata:
      labels:
        app: aks-helloworld-one
    spec:
      containers:
        - name: aks-helloworld-one
          image: mcr.microsoft.com/azuredocs/aks-helloworld:v1
          ports:
            - containerPort: 80
          env:
            - name: TITLE
              value: "Welcome to Azure Kubernetes Service (AKS)"
---
apiVersion: v1
kind: Service
metadata:
  name: aks-helloworld-one
spec:
  type: ClusterIP
  ports:
    - port: 80
  selector:
    app: aks-helloworld-one
```

Create a *aks-helloworld-two.yaml* file and copy in the following example YAML:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: aks-helloworld-two
spec:
  replicas: 1
  selector:
    matchLabels:
      app: aks-helloworld-two
  template:
    metadata:
      labels:
        app: aks-helloworld-two
    spec:
      containers:
        - name: aks-helloworld-two
          image: mcr.microsoft.com/azuredocs/aks-helloworld:v1
          ports:
            - containerPort: 80
          env:
            - name: TITLE
              value: "AKS Ingress Demo"
---
apiVersion: v1
kind: Service
metadata:
  name: aks-helloworld-two
spec:
  type: ClusterIP
  ports:
    - port: 80
  selector:
    app: aks-helloworld-two

```

Run the two demo applications using `kubectl apply`:

```

kubectl apply -f aks-helloworld-one.yaml --namespace ingress-basic
kubectl apply -f aks-helloworld-two.yaml --namespace ingress-basic

```

Create an ingress route

Both applications are now running on your Kubernetes cluster. However they're configured with a service of type `ClusterIP` and aren't accessible from the internet. To make them publicly available, create a Kubernetes ingress resource. The ingress resource configures the rules that route traffic to one of the two applications.

In the following example, traffic to the address `hello-world-ingress.MY_CUSTOM_DOMAIN` is routed to the `aks-helloworld-one` service. Traffic to the address `hello-world-ingress.MY_CUSTOM_DOMAIN/hello-world-two` is routed to the `aks-helloworld-two` service. Traffic to `hello-world-ingress.MY_CUSTOM_DOMAIN/static` is routed to the service named `aks-helloworld-one` for static assets.

NOTE

If you configured an FQDN for the ingress controller IP address instead of a custom domain, use the FQDN instead of `hello-world-ingress.MY_CUSTOM_DOMAIN`. For example if your FQDN is `demo-aks-ingress.eastus.cloudapp.azure.com`, replace `hello-world-ingress.MY_CUSTOM_DOMAIN` with `demo-aks-ingress.eastus.cloudapp.azure.com` in `hello-world-ingress.yaml`.

Create a file named `hello-world-ingress.yaml` using below example YAML. Update the `hosts` and `host` to the DNS name you created in a previous step.

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hello-world-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /$1
    nginx.ingress.kubernetes.io/use-regex: "true"
    cert-manager.io/cluster-issuer: letsencrypt
spec:
  tls:
  - hosts:
    - hello-world-ingress.MY_CUSTOM_DOMAIN
    secretName: tls-secret
  rules:
  - host: hello-world-ingress.MY_CUSTOM_DOMAIN
    http:
      paths:
        - path: /hello-world-one(/|$(.)*)
          pathType: Prefix
          backend:
            service:
              name: aks-helloworld-one
              port:
                number: 80
        - path: /hello-world-two(/|$(.)*)
          pathType: Prefix
          backend:
            service:
              name: aks-helloworld-two
              port:
                number: 80
        - path: /(.*)
          pathType: Prefix
          backend:
            service:
              name: aks-helloworld-one
              port:
                number: 80
  ---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hello-world-ingress-static
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /static/$2
    nginx.ingress.kubernetes.io/use-regex: "true"
    cert-manager.io/cluster-issuer: letsencrypt
spec:
  tls:
  - hosts:
    - hello-world-ingress.MY_CUSTOM_DOMAIN
    secretName: tls-secret
  rules:
  - host: hello-world-ingress.MY_CUSTOM_DOMAIN
    http:
      paths:
        - path:
          pathType: Prefix
          backend:
            service:
              name: aks-helloworld-one
              port:
                number: 80
        path: /static(/|$(.)*)

```

Create the ingress resource using the `kubectl apply` command.

```
kubectl apply -f hello-world-ingress.yaml --namespace ingress-basic
```

Verify a certificate object has been created

Next, a certificate resource must be created. The certificate resource defines the desired X.509 certificate. For more information, see [cert-manager certificates](#). Cert-manager has automatically created a certificate object for you using ingress-shim, which is automatically deployed with cert-manager since v0.2.2. For more information, see the [ingress-shim documentation](#).

To verify that the certificate was created successfully, use the `kubectl get certificate --namespace ingress-basic` command and verify *READY* is *True*, which may take several minutes.

```
$ kubectl get certificate --namespace ingress-basic  
  
NAME      READY   SECRET      AGE  
tls-secret  True    tls-secret  11m
```

Test the ingress configuration

Open a web browser to *hello-world-ingress.MY_CUSTOM_DOMAIN* of your Kubernetes ingress controller. Notice you are redirect to use HTTPS and the certificate is trusted and the demo application is shown in the web browser. Add the */hello-world-two* path and notice the second demo application with the custom title is shown.

Clean up resources

This article used Helm to install the ingress components, certificates, and sample apps. When you deploy a Helm chart, a number of Kubernetes resources are created. These resources includes pods, deployments, and services. To clean up these resources, you can either delete the entire sample namespace, or the individual resources.

Delete the sample namespace and all resources

To delete the entire sample namespace, use the `kubectl delete` command and specify your namespace name. All the resources in the namespace are deleted.

```
kubectl delete namespace ingress-basic
```

Delete resources individually

Alternatively, a more granular approach is to delete the individual resources created. First, remove the cluster issuer resources:

```
kubectl delete -f cluster-issuer.yaml --namespace ingress-basic
```

List the Helm releases with the `helm list` command. Look for charts named *nginx* and *cert-manager*, as shown in the following example output:

NAME	NAMESPACE	REVISION	UPDATED	STATUS
CHART	APP VERSION			
cert-manager	ingress-basic	1	2020-01-15 10:23:36.515514 -0600 CST	deployed
cert-manager-v0.13.0	v0.13.0			
nginx	ingress-basic	1	2020-01-15 10:09:45.982693 -0600 CST	deployed
nginx-ingress-1.29.1	0.27.0			

Uninstall the releases with the `helm uninstall` command. The following example uninstalls the NGINX ingress and cert-manager deployments.

```
$ helm uninstall cert-manager nginx --namespace ingress-basic  
release "cert-manager" uninstalled  
release "nginx" uninstalled
```

Next, remove the two sample applications:

```
kubectl delete -f aks-helloworld-one.yaml --namespace ingress-basic  
kubectl delete -f aks-helloworld-two.yaml --namespace ingress-basic
```

Remove the ingress route that directed traffic to the sample apps:

```
kubectl delete -f hello-world-ingress.yaml --namespace ingress-basic
```

Finally, you can delete the itself namespace. Use the `kubectl delete` command and specify your namespace name:

```
kubectl delete namespace ingress-basic
```

Next steps

This article included some external components to AKS. To learn more about these components, see the following project pages:

- [Helm CLI](#)
- [NGINX ingress controller](#)
- [cert-manager](#)

You can also:

- [Create a basic ingress controller with external network connectivity](#)
- [Enable the HTTP application routing add-on](#)
- [Create an ingress controller that uses an internal, private network and IP address](#)
- [Create an ingress controller that uses your own TLS certificates](#)
- [Create an ingress controller that uses Let's Encrypt to automatically generate TLS certificates with a static public IP address](#)

Create an ingress controller with a static public IP address in Azure Kubernetes Service (AKS)

4/26/2021 • 12 minutes to read • [Edit Online](#)

An ingress controller is a piece of software that provides reverse proxy, configurable traffic routing, and TLS termination for Kubernetes services. Kubernetes ingress resources are used to configure the ingress rules and routes for individual Kubernetes services. Using an ingress controller and ingress rules, a single IP address can be used to route traffic to multiple services in a Kubernetes cluster.

This article shows you how to deploy the [NGINX ingress controller](#) in an Azure Kubernetes Service (AKS) cluster. The ingress controller is configured with a static public IP address. The [cert-manager](#) project is used to automatically generate and configure [Let's Encrypt](#) certificates. Finally, two applications are run in the AKS cluster, each of which is accessible over a single IP address.

You can also:

- [Create a basic ingress controller with external network connectivity](#)
- [Enable the HTTP application routing add-on](#)
- [Create an ingress controller that uses your own TLS certificates](#)
- [Create an ingress controller that uses Let's Encrypt to automatically generate TLS certificates with a dynamic public IP address](#)

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart [using the Azure CLI](#) or [using the Azure portal](#).

This article uses [Helm 3](#) to install the NGINX ingress controller and cert-manager. Make sure that you are using the latest release of Helm and have access to the *ingress-nginx* and *jetstack* Helm repositories. For upgrade instructions, see the [Helm install docs](#). For more information on configuring and using Helm, see [Install applications with Helm in Azure Kubernetes Service \(AKS\)](#).

This article also requires that you are running the Azure CLI version 2.0.64 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Create an ingress controller

By default, an NGINX ingress controller is created with a new public IP address assignment. This public IP address is only static for the life-span of the ingress controller, and is lost if the controller is deleted and re-created. A common configuration requirement is to provide the NGINX ingress controller an existing static public IP address. The static public IP address remains if the ingress controller is deleted. This approach allows you to use existing DNS records and network configurations in a consistent manner throughout the lifecycle of your applications.

If you need to create a static public IP address, first get the resource group name of the AKS cluster with the `az aks show` command:

```
az aks show --resource-group myResourceGroup --name myAKSCluster --query nodeResourceGroup -o tsv
```

Next, create a public IP address with the *static* allocation method using the `az network public-ip create`

command. The following example creates a public IP address named *myAKSPublicIP* in the AKS cluster resource group obtained in the previous step:

```
az network public-ip create --resource-group MC_myResourceGroup_myAKSCluster_eastus --name myAKSPublicIP --sku Standard --allocation-method static --query publicIp.ipAddress -o tsv
```

NOTE

The above commands create an IP address that will be deleted if you delete your AKS cluster. Alternatively, you can create an IP address in a different resource group which can be managed separately from your AKS cluster. If you create an IP address in a different resource group, ensure the cluster identity used by the AKS cluster has delegated permissions to the other resource group, such as *Network Contributor*. For more information, see [Use a static public IP address and DNS label with the AKS load balancer](#).

Now deploy the *nginx-ingress* chart with Helm. For added redundancy, two replicas of the NGINX ingress controllers are deployed with the `--set controller.replicaCount` parameter. To fully benefit from running replicas of the ingress controller, make sure there's more than one node in your AKS cluster.

You must pass two additional parameters to the Helm release so the ingress controller is made aware both of the static IP address of the load balancer to be allocated to the ingress controller service, and of the DNS name label being applied to the public IP address resource. For the HTTPS certificates to work correctly, a DNS name label is used to configure an FQDN for the ingress controller IP address.

1. Add the `--set controller.service.loadBalancerIP` parameter. Specify your own public IP address that was created in the previous step.
2. Add the `--set controller.service.annotations."service\\.beta\\.kubernetes\\.io/azure-dns-label-name"` parameter. Specify a DNS name label to be applied to the public IP address that was created in the previous step.

The ingress controller also needs to be scheduled on a Linux node. Windows Server nodes shouldn't run the ingress controller. A node selector is specified using the `--set nodeSelector` parameter to tell the Kubernetes scheduler to run the NGINX ingress controller on a Linux-based node.

TIP

The following example creates a Kubernetes namespace for the ingress resources named *ingress-basic*. Specify a namespace for your own environment as needed. If your AKS cluster is not Kubernetes RBAC enabled, add `--set rbac.create=false` to the Helm commands.

TIP

If you would like to enable [client source IP preservation](#) for requests to containers in your cluster, add `--set controller.service.externalTrafficPolicy=Local` to the Helm install command. The client source IP is stored in the request header under *X-Forwarded-For*. When using an ingress controller with client source IP preservation enabled, TLS pass-through will not work.

Update the following script with the **IP address** of your ingress controller and a **unique name** that you would like to use for the FQDN prefix.

IMPORTANT

You must update replace *STATIC_IP* and *DNS_LABEL* with your own IP address and unique name when running the command.

```
# Create a namespace for your ingress resources
kubectl create namespace ingress-basic

# Add the ingress-nginx repository
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

# Use Helm to deploy an NGINX ingress controller
helm install nginx-ingress ingress-nginx/ingress-nginx \
--namespace ingress-basic \
--set controller.replicaCount=2 \
--set controller.nodeSelector."beta\.kubernetes\.io/os"=linux \
--set defaultBackend.nodeSelector."beta\.kubernetes\.io/os"=linux \
--set controller.admissionWebhooks.patch.nodeSelector."beta\.kubernetes\.io/os"=linux \
--set controller.service.loadBalancerIP="STATIC_IP" \
--set controller.service.annotations."service\.beta\.kubernetes\.io/azure-dns-label-name"="DNS_LABEL"
```

When the Kubernetes load balancer service is created for the NGINX ingress controller, your static IP address is assigned, as shown in the following example output:

```
$ kubectl --namespace ingress-basic get services -o wide -w nginx-ingress-ingress-nginx-controller
NAME                                     TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE   SELECTOR
nginx-ingress-ingress-nginx-controller   LoadBalancer  10.0.74.133    EXTERNAL_IP
80:32486/TCP,443:30953/TCP   44s   app.kubernetes.io/component=controller,app.kubernetes.io/instance=nginx-
ingress,app.kubernetes.io/name=ingress-nginx
```

No ingress rules have been created yet, so the NGINX ingress controller's default 404 page is displayed if you browse to the public IP address. Ingress rules are configured in the following steps.

You can verify that the DNS name label has been applied by querying the FQDN on the public IP address as follows:

```
az network public-ip list --resource-group MC_myResourceGroup_myAKSCluster_eastus --query "[?name=='myAKSPublicIP'].[dnsSettings.fqdn]" -o tsv
```

The ingress controller is now accessible through the IP address or the FQDN.

Install cert-manager

The NGINX ingress controller supports TLS termination. There are several ways to retrieve and configure certificates for HTTPS. This article demonstrates using [cert-manager](#), which provides automatic [Lets Encrypt](#) certificate generation and management functionality.

NOTE

This article uses the `staging` environment for Let's Encrypt. In production deployments, use `letsencrypt-prod` and <https://acme-v02.api.letsencrypt.org/directory> in the resource definitions and when installing the Helm chart.

To install the cert-manager controller in an Kubernetes RBAC-enabled cluster, use the following `helm install`

command:

```
# Label the cert-manager namespace to disable resource validation
kubectl label namespace ingress-basic cert-manager.io/disable-validation=true

# Add the Jetstack Helm repository
helm repo add jetstack https://charts.jetstack.io

# Update your local Helm chart repository cache
helm repo update

# Install the cert-manager Helm chart
helm install \
  cert-manager \
  --namespace ingress-basic \
  --version v1.3.1 \
  --set installCRDs=true \
  --set nodeSelector."beta\\.kubernetes\\.io/os"=linux \
  jetstack/cert-manager
```

For more information on cert-manager configuration, see the [cert-manager project](#).

Create a CA cluster issuer

Before certificates can be issued, cert-manager requires an `Issuer` or `ClusterIssuer` resource. These Kubernetes resources are identical in functionality, however `Issuer` works in a single namespace, and `ClusterIssuer` works across all namespaces. For more information, see the [cert-manager issuer](#) documentation.

Create a cluster issuer, such as `cluster-issuer.yaml`, using the following example manifest. Update the email address with a valid address from your organization:

```
apiVersion: cert-manager.io/v1alpha2
kind: ClusterIssuer
metadata:
  name: letsencrypt-staging
spec:
  acme:
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    email: user@contoso.com
    privateKeySecretRef:
      name: letsencrypt-staging
    solvers:
      - http01:
          ingress:
            class: nginx
            podTemplate:
              spec:
                nodeSelector:
                  "kubernetes.io/os": linux
```

To create the issuer, use the `kubectl apply` command.

```
kubectl apply -f cluster-issuer.yaml --namespace ingress-basic
```

The output should be similar to this example:

```
clusterissuer.cert-manager.io/letsencrypt-staging created
```

Run demo applications

An ingress controller and a certificate management solution have been configured. Now let's run two demo applications in your AKS cluster. In this example, Helm is used to deploy two instances of a simple 'Hello world' application.

To see the ingress controller in action, run two demo applications in your AKS cluster. In this example, you use `kubectl apply` to deploy two instances of a simple *Hello world* application.

Create a *aks-helloworld.yaml* file and copy in the following example YAML:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: aks-helloworld
spec:
  replicas: 1
  selector:
    matchLabels:
      app: aks-helloworld
  template:
    metadata:
      labels:
        app: aks-helloworld
    spec:
      containers:
        - name: aks-helloworld
          image: mcr.microsoft.com/azuredocs/aks-helloworld:v1
          ports:
            - containerPort: 80
          env:
            - name: TITLE
              value: "Welcome to Azure Kubernetes Service (AKS)"
---
apiVersion: v1
kind: Service
metadata:
  name: aks-helloworld
spec:
  type: ClusterIP
  ports:
    - port: 80
  selector:
    app: aks-helloworld
```

Create a *ingress-demo.yaml* file and copy in the following example YAML:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: ingress-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ingress-demo
  template:
    metadata:
      labels:
        app: ingress-demo
    spec:
      containers:
        - name: ingress-demo
          image: mcr.microsoft.com/azuredocs/aks-helloworld:v1
          ports:
            - containerPort: 80
          env:
            - name: TITLE
              value: "AKS Ingress Demo"
---
apiVersion: v1
kind: Service
metadata:
  name: ingress-demo
spec:
  type: ClusterIP
  ports:
    - port: 80
  selector:
    app: ingress-demo

```

Run the two demo applications using `kubectl apply`:

```

kubectl apply -f aks-helloworld.yaml --namespace ingress-basic
kubectl apply -f ingress-demo.yaml --namespace ingress-basic

```

Create an ingress route

Both applications are now running on your Kubernetes cluster, however they're configured with a service of type `ClusterIP`. As such, the applications aren't accessible from the internet. To make them publicly available, create a Kubernetes ingress resource. The ingress resource configures the rules that route traffic to one of the two applications.

In the following example, traffic to the address `https://demo-aks-ingress.eastus.cloudapp.azure.com/` is routed to the service named `aks-helloworld`. Traffic to the address

`https://demo-aks-ingress.eastus.cloudapp.azure.com/hello-world-two` is routed to the `ingress-demo` service.

Update the `hosts` and `host` to the DNS name you created in a previous step.

Create a file named `hello-world-ingress.yaml` and copy in the following example YAML.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hello-world-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    cert-manager.io/cluster-issuer: letsencrypt-staging
    nginx.ingress.kubernetes.io/rewrite-target: /$1
    nginx.ingress.kubernetes.io/use-regex: "true"
spec:
  tls:
  - hosts:
    - demo-aks-ingress.eastus.cloudapp.azure.com
    secretName: tls-secret
  rules:
  - host: demo-aks-ingress.eastus.cloudapp.azure.com
    http:
      paths:
        - path: /hello-world-one(/|$(.)*)
          pathType: Prefix
          backend:
            service:
              name: aks-helloworld
              port:
                number: 80
        - path: /hello-world-two(/|$(.)*)
          pathType: Prefix
          backend:
            service:
              name: ingress-demo
              port:
                number: 80
        - path: /(.*)
          pathType: Prefix
          backend:
            service:
              name: aks-helloworld
              port:
                number: 80
```

Create the ingress resource using the `kubectl apply` command.

```
kubectl apply -f hello-world-ingress.yaml --namespace ingress-basic
```

The output should be similar to this example:

```
ingress.extensions/hello-world-ingress created
```

Create a certificate object

Next, a certificate resource must be created. The certificate resource defines the desired X.509 certificate. For more information, see [cert-manager certificates](#).

Cert-manager has likely automatically created a certificate object for you using ingress-shim, which is automatically deployed with cert-manager since v0.2.2. For more information, see the [ingress-shim documentation](#).

To verify that the certificate was created successfully, use the

```
kubectl describe certificate tls-secret --namespace ingress-basic
```

If the certificate was issued, you will see output similar to the following:

Type	Reason	Age	From	Message
Normal	CreateOrder	11m	cert-manager	Created new ACME order, attempting validation...
Normal	DomainVerified	10m	cert-manager	Domain "demo-aks-ingress.eastus.cloudapp.azure.com" verified with "http-01" validation
Normal	IssueCert	10m	cert-manager	Issuing certificate...
Normal	CertObtained	10m	cert-manager	Obtained certificate from ACME server
Normal	CertIssued	10m	cert-manager	Certificate issued successfully

If you need to create an additional certificate resource, you can do so with the following example manifest. Update the *dnsNames* and *domains* to the DNS name you created in a previous step. If you use an internal-only ingress controller, specify the internal DNS name for your service.

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: tls-secret
  namespace: ingress-basic
spec:
  secretName: tls-secret
  dnsNames:
  - demo-aks-ingress.eastus.cloudapp.azure.com
  acme:
    config:
    - http01:
        ingressClass: nginx
        domains:
        - demo-aks-ingress.eastus.cloudapp.azure.com
  issuerRef:
    name: letsencrypt-staging
    kind: ClusterIssuer
```

To create the certificate resource, use the `kubectl apply` command.

```
$ kubectl apply -f certificates.yaml
certificate.cert-manager.io/tls-secret created
```

Test the ingress configuration

Open a web browser to the FQDN of your Kubernetes ingress controller, such as

<https://demo-aks-ingress.eastus.cloudapp.azure.com>.

As these examples use `letsencrypt-staging`, the issued TLS/SSL certificate is not trusted by the browser. Accept the warning prompt to continue to your application. The certificate information shows this *Fake LE Intermediate X1* certificate is issued by Let's Encrypt. This fake certificate indicates `cert-manager` processed the request correctly and received a certificate from the provider:



When you change Let's Encrypt to use `prod` rather than `staging`, a trusted certificate issued by Let's Encrypt is used, as shown in the following example:



The demo application is shown in the web browser:



Now add the `/hello-world-two` path to the FQDN, such as

<https://demo-aks-ingress.eastus.cloudapp.azure.com/hello-world-two>. The second demo application with the custom title is shown:



Clean up resources

This article used Helm to install the ingress components, certificates, and sample apps. When you deploy a Helm chart, a number of Kubernetes resources are created. These resources includes pods, deployments, and services. To clean up these resources, you can either delete the entire sample namespace, or the individual resources.

Delete the sample namespace and all resources

To delete the entire sample namespace, use the `kubectl delete` command and specify your namespace name. All the resources in the namespace are deleted.

```
kubectl delete namespace ingress-basic
```

Delete resources individually

Alternatively, a more granular approach is to delete the individual resources created. First, remove the certificate resources:

```
kubectl delete -f certificates.yaml  
kubectl delete -f cluster-issuer.yaml
```

Now list the Helm releases with the `helm list` command. Look for charts named *nginx-ingress* and *cert-manager* as shown in the following example output:

NAME	APP VERSION	NAMESPACE	REVISION	UPDATED	STATUS	CHART
nginx-ingress	nginx-ingress-1.28.2	ingress-basic	0.26.2	1	2020-01-11 14:51:03.454165006	deployed
cert-manager	cert-manager-v0.13.0	ingress-basic	v0.13.0	1	2020-01-06 21:19:03.866212286	deployed

Uninstall the releases with the `helm uninstall` command. The following example uninstalls the NGINX ingress

deployment and certificate manager deployments.

```
$ helm uninstall nginx-ingress cert-manager -n ingress-basic  
release "nginx-ingress" deleted  
release "cert-manager" deleted
```

Next, remove the two sample applications:

```
kubectl delete -f aks-helloworld.yaml --namespace ingress-basic  
kubectl delete -f ingress-demo.yaml --namespace ingress-basic
```

Delete the itself namespace. Use the `kubectl delete` command and specify your namespace name:

```
kubectl delete namespace ingress-basic
```

Finally, remove the static public IP address created for the ingress controller. Provide your *MC_* cluster resource group name obtained in the first step of this article, such as *MC_myResourceGroup_myAKSCluster_eastus*:

```
az network public-ip delete --resource-group MC_myResourceGroup_myAKSCluster_eastus --name myAKSPublicIP
```

Next steps

This article included some external components to AKS. To learn more about these components, see the following project pages:

- [Helm CLI](#)
- [NGINX ingress controller](#)
- [cert-manager](#)

You can also:

- [Create a basic ingress controller with external network connectivity](#)
- [Enable the HTTP application routing add-on](#)
- [Create an ingress controller that uses an internal, private network and IP address](#)
- [Create an ingress controller that uses your own TLS certificates](#)
- [Create an ingress controller with a dynamic public IP and configure Let's Encrypt to automatically generate TLS certificates](#)

Control egress traffic for cluster nodes in Azure Kubernetes Service (AKS)

5/14/2021 • 24 minutes to read • [Edit Online](#)

This article provides the necessary details that allow you to secure outbound traffic from your Azure Kubernetes Service (AKS). It contains the cluster requirements for a base AKS deployment, and additional requirements for optional addons and features. [An example will be provided at the end on how to configure these requirements with Azure Firewall](#). However, you can apply this information to any outbound restriction method or appliance.

Background

AKS clusters are deployed on a virtual network. This network can be managed (created by AKS) or custom (pre-configured by the user beforehand). In either case, the cluster has **outbound** dependencies on services outside of that virtual network (the service has no inbound dependencies).

For management and operational purposes, nodes in an AKS cluster need to access certain ports and fully qualified domain names (FQDNs). These endpoints are required for the nodes to communicate with the API server, or to download and install core Kubernetes cluster components and node security updates. For example, the cluster needs to pull base system container images from Microsoft Container Registry (MCR).

The AKS outbound dependencies are almost entirely defined with FQDNs, which don't have static addresses behind them. The lack of static addresses means that Network Security Groups can't be used to lock down the outbound traffic from an AKS cluster.

By default, AKS clusters have unrestricted outbound (egress) internet access. This level of network access allows nodes and services you run to access external resources as needed. If you wish to restrict egress traffic, a limited number of ports and addresses must be accessible to maintain healthy cluster maintenance tasks. The simplest solution to securing outbound addresses lies in use of a firewall device that can control outbound traffic based on domain names. Azure Firewall, for example, can restrict outbound HTTP and HTTPS traffic based on the FQDN of the destination. You can also configure your preferred firewall and security rules to allow these required ports and addresses.

IMPORTANT

This document covers only how to lock down the traffic leaving the AKS subnet. AKS has no ingress requirements by default. Blocking **internal subnet traffic** using network security groups (NSGs) and firewalls is not supported. To control and block the traffic within the cluster, use [Network Policies](#).

Required outbound network rules and FQDNs for AKS clusters

The following network and FQDN/application rules are required for an AKS cluster, you can use them if you wish to configure a solution other than Azure Firewall.

- IP Address dependencies are for non-HTTP/S traffic (both TCP and UDP traffic)
- FQDN HTTP/HTTPS endpoints can be placed in your firewall device.
- Wildcard HTTP/HTTPS endpoints are dependencies that can vary with your AKS cluster based on a number of qualifiers.
- AKS uses an admission controller to inject the FQDN as an environment variable to all deployments under kube-system and gatekeeper-system, that ensures all system communication between nodes and API server

uses the API server FQDN and not the API server IP.

- If you have an app or solution that needs to talk to the API server, you must add an **additional** network rule to allow *TCP communication to port 443 of your API server's IP*.
- On rare occasions, if there's a maintenance operation your API server IP might change. Planned maintenance operations that can change the API server IP are always communicated in advance.

Azure Global required network rules

The required network rules and IP address dependencies are:

DESTINATION ENDPOINT	PROTOCOL	PORT	USE
<code>*:1194</code> <i>Or</i> <code>ServiceTag -</code> <code>AzureCloud.</code> <code><Region>:1194</code> <i>Or</i> <code>Regional CIDRs -</code> <code>RegionCIDRs:1194</code> <i>Or</i> <code>APIServerPublicIP:1194</code> <code>(only known after cluster creation)</code>	UDP	1194	For tunneled secure communication between the nodes and the control plane. This is not required for private clusters
<code>*:9000</code> <i>Or</i> <code>ServiceTag -</code> <code>AzureCloud.</code> <code><Region>:9000</code> <i>Or</i> <code>Regional CIDRs -</code> <code>RegionCIDRs:9000</code> <i>Or</i> <code>APIServerPublicIP:9000</code> <code>(only known after cluster creation)</code>	TCP	9000	For tunneled secure communication between the nodes and the control plane. This is not required for private clusters
<code>*:123</code> or <code>ntp.ubuntu.com:123</code> (if using Azure Firewall network rules)	UDP	123	Required for Network Time Protocol (NTP) time synchronization on Linux nodes.
<code>CustomDNSIP:53</code> (if using custom DNS servers)	UDP	53	If you're using custom DNS servers, you must ensure they're accessible by the cluster nodes.
<code>APIServerPublicIP:443</code> (if running pods/deployments that access the API Server)	TCP	443	Required if running pods/deployments that access the API Server, those pods/deployments would use the API IP. This is not required for private clusters

Azure Global required FQDN / application rules

The following FQDN / application rules are required:

DESTINATION FQDN	PORT	USE
*.hcp.<location>.azmk8s.io	HTTPS:443	Required for Node <-> API server communication. Replace <location> with the region where your AKS cluster is deployed.
mcr.microsoft.com	HTTPS:443	Required to access images in Microsoft Container Registry (MCR). This registry contains first-party images/charts (for example, coreDNS, etc.). These images are required for the correct creation and functioning of the cluster, including scale and upgrade operations.
*.data.mcr.microsoft.com	HTTPS:443	Required for MCR storage backed by the Azure content delivery network (CDN).
management.azure.com	HTTPS:443	Required for Kubernetes operations against the Azure API.
login.microsoftonline.com	HTTPS:443	Required for Azure Active Directory authentication.
packages.microsoft.com	HTTPS:443	This address is the Microsoft packages repository used for cached <i>apt-get</i> operations. Example packages include Moby, PowerShell, and Azure CLI.
acs-mirror.azureedge.net	HTTPS:443	This address is for the repository required to download and install required binaries like kubenet and Azure CNI.

Azure China 21Vianet required network rules

The required network rules and IP address dependencies are:

DESTINATION ENDPOINT	PROTOCOL	PORT	USE
*:1194 <i>Or</i> ServiceTag - AzureCloud.Region:1194 <i>Or</i> Regional CIDRs - RegionCIDRs:1194 <i>Or</i> APIServerPublicIP:1194 (only known after cluster creation)	UDP	1194	For tunneled secure communication between the nodes and the control plane.

DESTINATION ENDPOINT	PROTOCOL	PORT	USE
<p>*:9000</p> <p>Or</p> <p>ServiceTag -</p> <p>AzureCloud.<Region>:9000</p> <p>Or</p> <p>Regional CIDRs -</p> <p>RegionCIDRs:9000</p> <p>Or</p> <p>APIServerPublicIP:9000</p> <p>(only known after cluster creation)</p>	TCP	9000	For tunneled secure communication between the nodes and the control plane.
<p>*:22</p> <p>Or</p> <p>ServiceTag -</p> <p>AzureCloud.<Region>:22</p> <p>Or</p> <p>Regional CIDRs -</p> <p>RegionCIDRs:22</p> <p>Or</p> <p>APIServerPublicIP:22</p> <p>(only known after cluster creation)</p>	TCP	22	For tunneled secure communication between the nodes and the control plane.
<p>*:123 or</p> <p>ntp.ubuntu.com:123 (if using Azure Firewall network rules)</p>	UDP	123	Required for Network Time Protocol (NTP) time synchronization on Linux nodes.
<p>CustomDNSIP:53 (if using custom DNS servers)</p>	UDP	53	If you're using custom DNS servers, you must ensure they're accessible by the cluster nodes.
<p>APIServerPublicIP:443 (if running pods/deployments that access the API Server)</p>	TCP	443	Required if running pods/deployments that access the API Server, those pod/deployments would use the API IP.

Azure China 21Vianet required FQDN / application rules

The following FQDN / application rules are required:

DESTINATION FQDN	PORT	USE
<p>*.hcp.<location>.cx.prod.service.azk8s.cn</p>	HTTPS:443	Required for Node <-> API server communication. Replace <location> with the region where your AKS cluster is deployed.
<p>*.tun.<location>.cx.prod.service.azk8s.cn</p>	HTTPS:443	Required for Node <-> API server communication. Replace <location> with the region where your AKS cluster is deployed.

DESTINATION FQDN	PORT	USE
mcr.microsoft.com	HTTPS:443	Required to access images in Microsoft Container Registry (MCR). This registry contains first-party images/charts (for example, coreDNS, etc.). These images are required for the correct creation and functioning of the cluster, including scale and upgrade operations.
.data.mcr.microsoft.com	HTTPS:443	Required for MCR storage backed by the Azure Content Delivery Network (CDN).
management.chinacloudapi.cn	HTTPS:443	Required for Kubernetes operations against the Azure API.
login.chinacloudapi.cn	HTTPS:443	Required for Azure Active Directory authentication.
packages.microsoft.com	HTTPS:443	This address is the Microsoft packages repository used for cached <i>apt-get</i> operations. Example packages include Moby, PowerShell, and Azure CLI.
*.azk8s.cn	HTTPS:443	This address is for the repository required to download and install required binaries like kubenet and Azure CNI.

Azure US Government required network rules

The required network rules and IP address dependencies are:

DESTINATION ENDPOINT	PROTOCOL	PORT	USE
*:1194 <i>Or</i> ServiceTag - AzureCloud. <Region>:1194 <i>Or</i> Regional CIDRs - RegionCIDRs:1194 <i>Or</i> APIServerPublicIP:1194 (only known after cluster creation)	UDP	1194	For tunneled secure communication between the nodes and the control plane.

DESTINATION ENDPOINT	PROTOCOL	PORT	USE
<p>*:9000</p> <p>Or</p> <p>ServiceTag -</p> <p>AzureCloud. <Region>:9000</p> <p>Or</p> <p>Regional CIDRs -</p> <p>RegionCIDRs:9000</p> <p>Or</p> <p>APIServerPublicIP:9000</p> <p>(only known after cluster creation)</p>	TCP	9000	For tunneled secure communication between the nodes and the control plane.
<p>*:123 or</p> <p>ntp.ubuntu.com:123 (if using Azure Firewall network rules)</p>	UDP	123	Required for Network Time Protocol (NTP) time synchronization on Linux nodes.
<p>CustomDNSIP:53</p> <p>(if using custom DNS servers)</p>	UDP	53	If you're using custom DNS servers, you must ensure they're accessible by the cluster nodes.
<p>APIServerPublicIP:443</p> <p>(if running pods/deployments that access the API Server)</p>	TCP	443	Required if running pods/deployments that access the API Server, those pods/deployments would use the API IP.

Azure US Government required FQDN / application rules

The following FQDN / application rules are required:

DESTINATION FQDN	PORT	USE
<p>*.hcp. <location>.cx.aks.containerservice.azure.us</p>	HTTPS:443	Required for Node <-> API server communication. Replace <location> with the region where your AKS cluster is deployed.
mcr.microsoft.com	HTTPS:443	Required to access images in Microsoft Container Registry (MCR). This registry contains first-party images/charts (for example, coreDNS, etc.). These images are required for the correct creation and functioning of the cluster, including scale and upgrade operations.
*.data.mcr.microsoft.com	HTTPS:443	Required for MCR storage backed by the Azure content delivery network (CDN).
management.usgovcloudapi.net	HTTPS:443	Required for Kubernetes operations against the Azure API.

DESTINATION FQDN	PORT	USE
login.microsoftonline.us	HTTPS:443	Required for Azure Active Directory authentication.
packages.microsoft.com	HTTPS:443	This address is the Microsoft packages repository used for cached <i>apt-get</i> operations. Example packages include Moby, PowerShell, and Azure CLI.
acs-mirror.azureedge.net	HTTPS:443	This address is for the repository required to install required binaries like kubenet and Azure CNI.

Optional recommended FQDN / application rules for AKS clusters

The following FQDN / application rules are optional but recommended for AKS clusters:

DESTINATION FQDN	PORT	USE
security.ubuntu.com , azure.archive.ubuntu.com , changelogs.ubuntu.com	HTTP:80	This address lets the Linux cluster nodes download the required security patches and updates.

If you choose to block/not allow these FQDNs, the nodes will only receive OS updates when you do a [node image upgrade](#) or [cluster upgrade](#).

GPU enabled AKS clusters

Required FQDN / application rules

The following FQDN / application rules are required for AKS clusters that have GPU enabled:

DESTINATION FQDN	PORT	USE
nvidia.github.io	HTTPS:443	This address is used for correct driver installation and operation on GPU-based nodes.
us.download.nvidia.com	HTTPS:443	This address is used for correct driver installation and operation on GPU-based nodes.
apt.dockerproject.org	HTTPS:443	This address is used for correct driver installation and operation on GPU-based nodes.

Windows Server based node pools

Required FQDN / application rules

The following FQDN / application rules are required for using Windows Server based node pools:

DESTINATION FQDN	PORT	USE
onegetcdn.azureedge.net, go.microsoft.com	HTTPS:443	To install windows-related binaries
*.mp.microsoft.com, www.msftconnecttest.com, ctld1.windowsupdate.com	HTTP:80	To install windows-related binaries

AKS addons and integrations

Azure Monitor for containers

There are two options to provide access to Azure Monitor for containers, you may allow the Azure Monitor [ServiceTag](#) or provide access to the required FQDN/Application Rules.

Required network rules

The following FQDN / application rules are required:

DESTINATION ENDPOINT	PROTOCOL	PORT	USE
ServiceTag - AzureMonitor:443	TCP	443	This endpoint is used to send metrics data and logs to Azure Monitor and Log Analytics.

Required FQDN / application rules

The following FQDN / application rules are required for AKS clusters that have the Azure Monitor for containers enabled:

FQDN	PORT	USE
dc.services.visualstudio.com	HTTPS:443	This endpoint is used for metrics and monitoring telemetry using Azure Monitor.
*.ods.opinsights.azure.com	HTTPS:443	This endpoint is used by Azure Monitor for ingesting log analytics data.
*.oms.opinsights.azure.com	HTTPS:443	This endpoint is used by omsagent, which is used to authenticate the log analytics service.
*.monitoring.azure.com	HTTPS:443	This endpoint is used to send metrics data to Azure Monitor.

Azure Policy

Required FQDN / application rules

The following FQDN / application rules are required for AKS clusters that have the Azure Policy enabled.

FQDN	PORT	USE
data.policy.core.windows.net	HTTPS:443	This address is used to pull the Kubernetes policies and to report cluster compliance status to policy service.

FQDN	PORT	USE
store.policy.core.windows.net	HTTPS:443	This address is used to pull the Gatekeeper artifacts of built-in policies.
dc.services.visualstudio.com	HTTPS:443	Azure Policy add-on that sends telemetry data to applications insights endpoint.

Azure China 21Vianet Required FQDN / application rules

The following FQDN / application rules are required for AKS clusters that have the Azure Policy enabled.

FQDN	PORT	USE
data.policy.azure.cn	HTTPS:443	This address is used to pull the Kubernetes policies and to report cluster compliance status to policy service.
store.policy.azure.cn	HTTPS:443	This address is used to pull the Gatekeeper artifacts of built-in policies.

Azure US Government Required FQDN / application rules

The following FQDN / application rules are required for AKS clusters that have the Azure Policy enabled.

FQDN	PORT	USE
data.policy.azure.us	HTTPS:443	This address is used to pull the Kubernetes policies and to report cluster compliance status to policy service.
store.policy.azure.us	HTTPS:443	This address is used to pull the Gatekeeper artifacts of built-in policies.

Restrict egress traffic using Azure firewall

Azure Firewall provides an Azure Kubernetes Service (`AzureKubernetesService`) FQDN Tag to simplify this configuration.

NOTE

The FQDN tag contains all the FQDNs listed above and is kept automatically up to date.

We recommend having a minimum of 20 Frontend IPs on the Azure Firewall for production scenarios to avoid incurring in SNAT port exhaustion issues.

Below is an example architecture of the deployment:



- Public Ingress is forced to flow through firewall filters
 - AKS agent nodes are isolated in a dedicated subnet.
 - [Azure Firewall](#) is deployed in its own subnet.
 - A DNAT rule translates the FW public IP into the LB frontend IP.
- Outbound requests start from agent nodes to the Azure Firewall internal IP using a [user-defined route](#)
 - Requests from AKS agent nodes follow a UDR that has been placed on the subnet the AKS cluster was deployed into.
 - Azure Firewall egresses out of the virtual network from a public IP frontend
 - Access to the public internet or other Azure services flows to and from the firewall frontend IP address
 - Optionally, access to the AKS control plane is protected by [API server Authorized IP ranges](#), which includes the firewall public frontend IP address.
- Internal Traffic
 - Optionally, instead or in addition to a [Public Load Balancer](#) you can use an [Internal Load Balancer](#) for internal traffic, which you could isolate on its own subnet as well.

The below steps make use of Azure Firewall's `AzureKubernetesService` FQDN tag to restrict the outbound traffic from the AKS cluster and provide an example how to configure public inbound traffic via the firewall.

Set configuration via environment variables

Define a set of environment variables to be used in resource creations.

```
PREFIX="aks-egress"
RG="${PREFIX}-rg"
LOC="eastus"
PLUGIN=azure
AKSNAME="${PREFIX}"
VNET_NAME="${PREFIX}-vnet"
AKSSUBNET_NAME="aks-subnet"
# DO NOT CHANGE FWSUBNET_NAME - This is currently a requirement for Azure Firewall.
FWSUBNET_NAME="AzureFirewallSubnet"
FWNAME="${PREFIX}-fw"
FWPUBLICIP_NAME="${PREFIX}-fwpublicip"
FWIPCONFIG_NAME="${PREFIX}-fwconfig"
FWROUTE_TABLE_NAME="${PREFIX}-fwrt"
FWROUTE_NAME="${PREFIX}-fwrn"
FWROUTE_NAME_INTERNET="${PREFIX}-fwinternet"
```

Create a virtual network with multiple subnets

Provision a virtual network with two separate subnets, one for the cluster, one for the firewall. Optionally you could also create one for internal service ingress.



Create a resource group to hold all of the resources.

```
# Create Resource Group

az group create --name $RG --location $LOC
```

Create a virtual network with two subnets to host the AKS cluster and the Azure Firewall. Each will have their own subnet. Let's start with the AKS network.

```

# Dedicated virtual network with AKS subnet

az network vnet create \
--resource-group $RG \
--name $VNET_NAME \
--location $LOC \
--address-prefixes 10.42.0.0/16 \
--subnet-name $AKSSUBNET_NAME \
--subnet-prefix 10.42.1.0/24

# Dedicated subnet for Azure Firewall (Firewall name cannot be changed)

az network vnet subnet create \
--resource-group $RG \
--vnet-name $VNET_NAME \
--name $FWSUBNET_NAME \
--address-prefix 10.42.2.0/24

```

Create and set up an Azure Firewall with a UDR

Azure Firewall inbound and outbound rules must be configured. The main purpose of the firewall is to enable organizations to configure granular ingress and egress traffic rules into and out of the AKS Cluster.



IMPORTANT

If your cluster or application creates a large number of outbound connections directed to the same or small subset of destinations, you might require more firewall frontend IPs to avoid maxing out the ports per frontend IP. For more information on how to create an Azure firewall with multiple IPs, see [here](#)

Create a standard SKU public IP resource that will be used as the Azure Firewall frontend address.

```
az network public-ip create -g $RG -n $FWPUBLICIP_NAME -l $LOC --sku "Standard"
```

Register the preview cli-extension to create an Azure Firewall.

```
# Install Azure Firewall preview CLI extension  
  
az extension add --name azure-firewall  
  
# Deploy Azure Firewall  
  
az network firewall create -g $RG -n $FWNAME -l $LOC --enable-dns-proxy true
```

The IP address created earlier can now be assigned to the firewall frontend.

NOTE

Set up of the public IP address to the Azure Firewall may take a few minutes. To leverage FQDN on network rules we need DNS proxy enabled, when enabled the firewall will listen on port 53 and will forward DNS requests to the DNS server specified above. This will allow the firewall to translate that FQDN automatically.

```
# Configure Firewall IP Config  
  
az network firewall ip-config create -g $RG -f $FWNAME -n $FWIPCONFIG_NAME --public-ip-address  
$FWPUBLICIP_NAME --vnet-name $VNET_NAME
```

When the previous command has succeeded, save the firewall frontend IP address for configuration later.

```
# Capture Firewall IP Address for Later Use  
  
FWPUBLIC_IP=$(az network public-ip show -g $RG -n $FWPUBLICIP_NAME --query "ipAddress" -o tsv)  
FWPRIVATE_IP=$(az network firewall show -g $RG -n $FWNAME --query "ipConfigurations[0].privateIpAddress" -o tsv)
```

NOTE

If you use secure access to the AKS API server with [authorized IP address ranges](#), you need to add the firewall public IP into the authorized IP range.

Create a UDR with a hop to Azure Firewall

Azure automatically routes traffic between Azure subnets, virtual networks, and on-premises networks. If you want to change any of Azure's default routing, you do so by creating a route table.

Create an empty route table to be associated with a given subnet. The route table will define the next hop as the Azure Firewall created above. Each subnet can have zero or one route table associated to it.

```
# Create UDR and add a route for Azure Firewall  
  
az network route-table create -g $RG -l $LOC --name $FWRROUTE_TABLE_NAME  
az network route-table route create -g $RG --name $FWRROUTE_NAME --route-table-name $FWRROUTE_TABLE_NAME --  
address-prefix 0.0.0.0/0 --next-hop-type VirtualAppliance --next-hop-ip-address $FWPRIVATE_IP --subscription  
$SUBID  
az network route-table route create -g $RG --name $FWRROUTE_NAME_INTERNET --route-table-name  
$FWRROUTE_TABLE_NAME --address-prefix $FWPUBLIC_IP/32 --next-hop-type Internet
```

See [virtual network route table documentation](#) about how you can override Azure's default system routes or add additional routes to a subnet's route table.

Adding firewall rules

Below are three network rules you can use to configure on your firewall, you may need to adapt these rules based on your deployment. The first rule allows access to port 9000 via TCP. The second rule allows access to port 1194 and 123 via UDP (if you're deploying to Azure China 21Vianet, you might require [more](#)). Both these rules will only allow traffic destined to the Azure Region CIDR that we're using, in this case East US. Finally, we'll add a third network rule opening port 123 to `ntp.ubuntu.com` FQDN via UDP (adding an FQDN as a network rule is one of the specific features of Azure Firewall, and you'll need to adapt it when using your own options).

After setting the network rules, we'll also add an application rule using the `AzureKubernetesService` that covers all needed FQDNs accessible through TCP port 443 and port 80.

```
# Add FW Network Rules

az network firewall network-rule create -g $RG -f $FWNAME --collection-name 'aksfwnr' -n 'apiudp' --
protocols 'UDP' --source-addresses '*' --destination-addresses "AzureCloud.$LOC" --destination-ports 1194 --
action allow --priority 100
az network firewall network-rule create -g $RG -f $FWNAME --collection-name 'aksfwnr' -n 'apitcp' --
protocols 'TCP' --source-addresses '*' --destination-addresses "AzureCloud.$LOC" --destination-ports 9000
az network firewall network-rule create -g $RG -f $FWNAME --collection-name 'aksfwnr' -n 'time' --protocols
'UDP' --source-addresses '*' --destination-fqdns 'ntp.ubuntu.com' --destination-ports 123

# Add FW Application Rules

az network firewall application-rule create -g $RG -f $FWNAME --collection-name 'aksfwar' -n 'fqdn' --
source-addresses '*' --protocols 'http=80' 'https=443' --fqdn-tags "AzureKubernetesService" --action allow -
priority 100
```

See [Azure Firewall documentation](#) to learn more about the Azure Firewall service.

Associate the route table to AKS

To associate the cluster with the firewall, the dedicated subnet for the cluster's subnet must reference the route table created above. Association can be done by issuing a command to the virtual network holding both the cluster and firewall to update the route table of the cluster's subnet.

```
# Associate route table with next hop to Firewall to the AKS subnet

az network vnet subnet update -g $RG --vnet-name $VNET_NAME --name $AKSSUBNET_NAME --route-table
$FWROUTE_TABLE_NAME
```

Deploy AKS with outbound type of UDR to the existing network

Now an AKS cluster can be deployed into the existing virtual network. We'll also use [outbound type userDefinedRouting](#), this feature ensures any outbound traffic will be forced through the firewall and no other egress paths will exist (by default the Load Balancer outbound type could be used).



Create a service principal with access to provision inside the existing virtual network

A cluster identity (managed identity or service principal) is used by AKS to create cluster resources. A service principal that is passed at create time is used to create underlying AKS resources such as Storage resources, IPs, and Load Balancers used by AKS (you may also use a [managed identity](#) instead). If not granted the appropriate permissions below, you won't be able to provision the AKS Cluster.

```
# Create SP and Assign Permission to Virtual Network
az ad sp create-for-rbac -n "${PREFIX}sp" --skip-assignment
```

Now replace the `APPID` and `PASSWORD` below with the service principal appid and service principal password autogenerated by the previous command output. We'll reference the VNET resource ID to grant the permissions to the service principal so AKS can deploy resources into it.

```
APPID=<SERVICE_PRINCIPAL_APPID_Goes_Here>
PASSWORD=<SERVICEPRINCIPAL_PASSWORD_Goes_Here>
VNETID=$(az network vnet show -g $RG --name $VNET_NAME --query id -o tsv)

# Assign SP Permission to VNET

az role assignment create --assignee $APPID --scope $VNETID --role "Network Contributor"
```

You can check the detailed permissions that are required [here](#).

NOTE

If you're using the kubenet network plugin, you'll need to give the AKS service principal or managed identity permissions to the pre-created route table, since kubenet requires a route table to add necessary routing rules.

```
RTID=$(az network route-table show -g $RG -n $FWROUTE_TABLE_NAME --query id -o tsv)
az role assignment create --assignee $APPID --scope $RTID --role "Network Contributor"
```

Deploy AKS

Finally, the AKS cluster can be deployed into the existing subnet we've dedicated for the cluster. The target

subnet to be deployed into is defined with the environment variable, `$SUBNETID`. We didn't define the `$SUBNETID` variable in the previous steps. To set the value for the subnet ID, you can use the following command:

```
SUBNETID=$(az network vnet subnet show -g $RG --vnet-name $VNET_NAME --name $AKSSUBNET_NAME --query id -o tsv)
```

You'll define the outbound type to use the UDR that already exists on the subnet. This configuration will enable AKS to skip the setup and IP provisioning for the load balancer.

IMPORTANT

For more information on outbound type UDR including limitations, see [egress outbound type UDR](#).

TIP

Additional features can be added to the cluster deployment such as [Private Cluster](#).

The AKS feature for [API server authorized IP ranges](#) can be added to limit API server access to only the firewall's public endpoint. The authorized IP ranges feature is denoted in the diagram as optional. When enabling the authorized IP range feature to limit API server access, your developer tools must use a jumpbox from the firewall's virtual network or you must add all developer endpoints to the authorized IP range.

```
az aks create -g $RG -n $AKSNAME -l $LOC \
--node-count 3 --generate-ssh-keys \
--network-plugin $PLUGIN \
--outbound-type userDefinedRouting \
--service-cidr 10.41.0.0/16 \
--dns-service-ip 10.41.0.10 \
--docker-bridge-address 172.17.0.1/16 \
--vnet-subnet-id $SUBNETID \
--service-principal $APPID \
--client-secret $PASSWORD \
--api-server-authorized-ip-ranges $FWPUBLIC_IP
```

Enable developer access to the API server

If you used authorized IP ranges for the cluster on the previous step, you must add your developer tooling IP addresses to the AKS cluster list of approved IP ranges in order to access the API server from there. Another option is to configure a jumpbox with the needed tooling inside a separate subnet in the Firewall's virtual network.

Add another IP address to the approved ranges with the following command

```
# Retrieve your IP address
CURRENT_IP=$(dig @resolver1.opendns.com ANY myip.opendns.com +short)

# Add to AKS approved list
az aks update -g $RG -n $AKSNAME --api-server-authorized-ip-ranges $CURRENT_IP/32
```

Use the [az aks get-credentials][az-aks-get-credentials] command to configure `kubectl` to connect to your newly created Kubernetes cluster.

```
az aks get-credentials -g $RG -n $AKSNAME
```

Deploy a public service

You can now start exposing services and deploying applications to this cluster. In this example, we'll expose a public service, but you may also choose to expose an internal service via [internal load balancer](#).



Deploy the Azure voting app application by copying the yaml below to a file named `example.yaml`.

```
# voting-storage-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: voting-storage
spec:
  replicas: 1
  selector:
    matchLabels:
      app: voting-storage
  template:
    metadata:
      labels:
        app: voting-storage
    spec:
      containers:
        - name: voting-storage
          image: mcr.microsoft.com/aks/samples/voting/storage:2.0
          args: ["--ignore-db-dir=lost+found"]
          resources:
            requests:
              cpu: 100m
              memory: 128Mi
            limits:
              cpu: 250m
              memory: 256Mi
        ports:
          - containerPort: 3306
            name: mysql
        volumeMounts:
          - name: mysql-persistent-storage
            mountPath: /var/lib/mysql
      env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: voting-storage-secret
              key: MYSQL_ROOT_PASSWORD
        - name: MYSQL_USER
```

```

    - -
      valueFrom:
        secretKeyRef:
          name: voting-storage-secret
          key: MYSQL_USER
    - name: MYSQL_PASSWORD
      valueFrom:
        secretKeyRef:
          name: voting-storage-secret
          key: MYSQL_PASSWORD
    - name: MYSQL_DATABASE
      valueFrom:
        secretKeyRef:
          name: voting-storage-secret
          key: MYSQL_DATABASE
  volumes:
    - name: mysql-persistent-storage
      persistentVolumeClaim:
        claimName: mysql-pv-claim
---
# voting-storage-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: voting-storage-secret
type: Opaque
data:
  MYSQL_USER: ZGJ1c2Vy
  MYSQL_PASSWORD: UGFzc3dvcmQxMg==
  MYSQL_DATABASE: YXp1cmV2b3Rl
  MYSQL_ROOT_PASSWORD: UGFzc3dvcmQxMg==
---
# voting-storage-pv-claim.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
---
# voting-storage-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: voting-storage
  labels:
    app: voting-storage
spec:
  ports:
    - port: 3306
      name: mysql
  selector:
    app: voting-storage
---
# voting-app-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: voting-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: voting-app
  template:
    metadata:

```

```

metadata:
  labels:
    app: voting-app
  spec:
    containers:
      - name: voting-app
        image: mcr.microsoft.com/aks/samples/voting/app:2.0
        imagePullPolicy: Always
      ports:
        - containerPort: 8080
          name: http
      env:
        - name: MYSQL_HOST
          value: "voting-storage"
        - name: MYSQL_USER
          valueFrom:
            secretKeyRef:
              name: voting-storage-secret
              key: MYSQL_USER
        - name: MYSQL_PASSWORD
          valueFrom:
            secretKeyRef:
              name: voting-storage-secret
              key: MYSQL_PASSWORD
        - name: MYSQL_DATABASE
          valueFrom:
            secretKeyRef:
              name: voting-storage-secret
              key: MYSQL_DATABASE
        - name: ANALYTICS_HOST
          value: "voting-analytics"
---
# voting-app-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: voting-app
  labels:
    app: voting-app
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 8080
      name: http
  selector:
    app: voting-app
---
# voting-analytics-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: voting-analytics
spec:
  replicas: 1
  selector:
    matchLabels:
      app: voting-analytics
      version: "2.0"
  template:
    metadata:
      labels:
        app: voting-analytics
        version: "2.0"
    spec:
      containers:
        - name: voting-analytics
          image: mcr.microsoft.com/aks/samples/voting/analytics:2.0
          imagePullPolicy: Always

```

```

ports:
  - containerPort: 8080
    name: http
  env:
    - name: MYSQL_HOST
      value: "voting-storage"
    - name: MYSQL_USER
      valueFrom:
        secretKeyRef:
          name: voting-storage-secret
          key: MYSQL_USER
    - name: MYSQL_PASSWORD
      valueFrom:
        secretKeyRef:
          name: voting-storage-secret
          key: MYSQL_PASSWORD
    - name: MYSQL_DATABASE
      valueFrom:
        secretKeyRef:
          name: voting-storage-secret
          key: MYSQL_DATABASE
---
# voting-analytics-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: voting-analytics
  labels:
    app: voting-analytics
spec:
  ports:
    - port: 8080
      name: http
  selector:
    app: voting-analytics

```

Deploy the service by running:

```
kubectl apply -f example.yaml
```

Add a DNAT rule to Azure Firewall

IMPORTANT

When you use Azure Firewall to restrict egress traffic and create a user-defined route (UDR) to force all egress traffic, make sure you create an appropriate DNAT rule in Firewall to correctly allow ingress traffic. Using Azure Firewall with a UDR breaks the ingress setup due to asymmetric routing. (The issue occurs if the AKS subnet has a default route that goes to the firewall's private IP address, but you're using a public load balancer - ingress or Kubernetes service of type: LoadBalancer). In this case, the incoming load balancer traffic is received via its public IP address, but the return path goes through the firewall's private IP address. Because the firewall is stateful, it drops the returning packet because the firewall isn't aware of an established session. To learn how to integrate Azure Firewall with your ingress or service load balancer, see [Integrate Azure Firewall with Azure Standard Load Balancer](#).

To configure inbound connectivity, a DNAT rule must be written to the Azure Firewall. To test connectivity to your cluster, a rule is defined for the firewall frontend public IP address to route to the internal IP exposed by the internal service.

The destination address can be customized as it's the port on the firewall to be accessed. The translated address must be the IP address of the internal load balancer. The translated port must be the exposed port for your Kubernetes service.

You'll need to specify the internal IP address assigned to the load balancer created by the Kubernetes service.

Retrieve the address by running:

```
kubectl get services
```

The IP address needed will be listed in the EXTERNAL-IP column, similar to the following.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.41.0.1	<none>	443/TCP	10h
voting-analytics	ClusterIP	10.41.88.129	<none>	8080/TCP	9m
voting-app	LoadBalancer	10.41.185.82	20.39.18.6	80:32718/TCP	9m
voting-storage	ClusterIP	10.41.221.201	<none>	3306/TCP	9m

Get the service IP by running:

```
SERVICE_IP=$(kubectl get svc voting-app -o jsonpath='{.status.loadBalancer.ingress[*].ip}')
```

Add the NAT rule by running:

```
az network firewall nat-rule create --collection-name exampleset --destination-addresses $FWPUBLIC_IP --destination-ports 80 --firewall-name $FWNAME --name inboundrule --protocols Any --resource-group $RG --source-addresses '*' --translated-port 80 --action Dnat --priority 100 --translated-address $SERVICE_IP
```

Validate connectivity

Navigate to the Azure Firewall frontend IP address in a browser to validate connectivity.

You should see the AKS voting app. In this example, the Firewall public IP was 52.253.228.132.



Clean up resources

To clean up Azure resources, delete the AKS resource group.

```
az group delete -g $RG
```

Next steps

In this article, you learned what ports and addresses to allow if you want to restrict egress traffic for the cluster. You also saw how to secure your outbound traffic using Azure Firewall.

If needed, you can generalize the steps above to forward the traffic to your preferred egress solution, following the [Outbound Type `userDefinedRoute` documentation](#).

If you want to restrict how pods communicate between themselves and East-West traffic restrictions within cluster see [Secure traffic between pods using network policies in AKS](#).

Customize cluster egress with a User-Defined Route

4/21/2021 • 3 minutes to read • [Edit Online](#)

Egress from an AKS cluster can be customized to fit specific scenarios. By default, AKS will provision a Standard SKU Load Balancer to be set up and used for egress. However, the default setup may not meet the requirements of all scenarios if public IPs are disallowed or additional hops are required for egress.

This article walks through how to customize a cluster's egress route to support custom network scenarios, such as those which disallows public IPs and requires the cluster to sit behind a network virtual appliance (NVA).

Prerequisites

- Azure CLI version 2.0.81 or greater
- API version of `2020-01-01` or greater

Limitations

- `OutboundType` can only be defined at cluster create time and can't be updated afterwards.
- Setting `outboundType` requires AKS clusters with a `vm-set-type` of `VirtualMachineScaleSets` and `load-balancer-sku` of `Standard`.
- Setting `outboundType` to a value of `UDR` requires a user-defined route with valid outbound connectivity for the cluster.
- Setting `outboundType` to a value of `UDR` implies the ingress source IP routed to the load-balancer may **not match** the cluster's outgoing egress destination address.

Overview of outbound types in AKS

An AKS cluster can be customized with a unique `outboundType` of type `loadBalancer` or `userDefinedRouting`.

IMPORTANT

Outbound type impacts only the egress traffic of your cluster. For more information, see [setting up ingress controllers](#).

NOTE

You can use your own [route table](#) with UDR and kubenet networking. Make sure your cluster identity (service principal or managed identity) has Contributor permissions to the custom route table.

Outbound type of `loadBalancer`

If `loadBalancer` is set, AKS completes the following configuration automatically. The load balancer is used for egress through an AKS assigned public IP. An outbound type of `loadBalancer` supports Kubernetes services of type `loadBalancer`, which expect egress out of the load balancer created by the AKS resource provider.

The following configuration is done by AKS.

- A public IP address is provisioned for cluster egress.
- The public IP address is assigned to the load balancer resource.
- Backend pools for the load balancer are set up for agent nodes in the cluster.

Below is a network topology deployed in AKS clusters by default, which use an `outboundType` of `loadBalancer`.



Outbound type of userDefinedRouting

NOTE

Using outbound type is an advanced networking scenario and requires proper network configuration.

If `userDefinedRouting` is set, AKS won't automatically configure egress paths. The egress setup must be done by you.

The AKS cluster must be deployed into an existing virtual network with a subnet that has been previously configured because when not using standard load balancer (SLB) architecture, you must establish explicit egress. As such, this architecture requires explicitly sending egress traffic to an appliance like a firewall, gateway, proxy or to allow the Network Address Translation (NAT) to be done by a public IP assigned to the standard load balancer or appliance.

Load balancer creation with userDefinedRouting

AKS clusters with an outbound type of UDR receive a standard load balancer (SLB) only when the first Kubernetes service of type 'loadBalancer' is deployed. The load balancer is configured with a public IP address for *inbound* requests and a backend pool for *inbound* requests. Inbound rules are configured by the Azure cloud provider, but no **outbound public IP address or outbound rules** are configured as a result of having an outbound type of UDR. Your UDR will still be the only source for egress traffic.

Azure load balancers [don't incur a charge until a rule is placed](#).

Deploy a cluster with outbound type of UDR and Azure Firewall

To illustrate the application of a cluster with outbound type using a user-defined route, a cluster can be configured on a virtual network with an Azure Firewall on its own subnet. See this example on the [restrict egress traffic with Azure firewall example](#).

IMPORTANT

Outbound type of UDR requires there is a route for 0.0.0.0/0 and next hop destination of NVA (Network Virtual Appliance) in the route table. The route table already has a default 0.0.0.0/0 to Internet, without a Public IP to SNAT just adding this route will not provide you egress. AKS will validate that you don't create a 0.0.0.0/0 route pointing to the Internet but instead to NVA or gateway, etc. When using an outbound type of UDR, a load balancer public IP address for **inbound requests** is not created unless a service of type *loadbalancer* is configured. A public IP address for **outbound requests** is never created by AKS if an outbound type of UDR is set.

Next steps

See [Azure networking UDR overview](#).

See [how to create, change, or delete a route table](#).

Use a static public IP address for egress traffic with a Basic SKU load balancer in Azure Kubernetes Service (AKS)

4/21/2021 • 4 minutes to read • [Edit Online](#)

By default, the egress IP address from an Azure Kubernetes Service (AKS) cluster is randomly assigned. This configuration is not ideal when you need to identify an IP address for access to external services, for example. Instead, you may need to assign a static IP address to be added to an allowlist for service access.

This article shows you how to create and use a static public IP address for use with egress traffic in an AKS cluster.

Before you begin

This article assumes you are using the Azure Basic Load Balancer. We recommend using the [Azure Standard Load Balancer](#), and you can use more advanced features for [controlling AKS egress traffic](#).

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart using the [Azure CLI](#) or [using the Azure portal](#).

You also need the Azure CLI version 2.0.59 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

IMPORTANT

This article uses the *Basic* SKU load balancer with a single node pool. This configuration is not available for multiple node pools since the *Basic* SKU load balancer is not supported with multiple node pools. See [Use a public Standard Load Balancer in Azure Kubernetes Service \(AKS\)](#) for more details on using the *Standard* SKU load balancer.

Egress traffic overview

Outbound traffic from an AKS cluster follows [Azure Load Balancer conventions](#). Before the first Kubernetes service of type `LoadBalancer` is created, the agent nodes in an AKS cluster are not part of any Azure Load Balancer pool. In this configuration, the nodes have no instance level Public IP address. Azure translates the outbound flow to a public source IP address that is not configurable or deterministic.

Once a Kubernetes service of type `LoadBalancer` is created, agent nodes are added to an Azure Load Balancer pool. Load Balancer Basic chooses a single frontend to be used for outbound flows when multiple (public) IP frontends are candidates for outbound flows. This selection is not configurable, and you should consider the selection algorithm to be random. This public IP address is only valid for the lifespan of that resource. If you delete the Kubernetes LoadBalancer service, the associated load balancer and IP address are also deleted. If you want to assign a specific IP address or retain an IP address for redeployed Kubernetes services, you can create and use a static public IP address.

Create a static public IP

Get the resource group name with the `az aks show` command and add the `--query nodeResourceGroup` query parameter. The following example gets the node resource group for the AKS cluster name *myAKSCluster* in the resource group name *myResourceGroup*.

```
$ az aks show --resource-group myResourceGroup --name myAKSCluster --query nodeResourceGroup -o tsv
```

```
MC_myResourceGroup_myAKSCluster_eastus
```

Now create a static public IP address with the [az network public ip create](#) command. Specify the node resource group name obtained in the previous command, and then a name for the IP address resource, such as *myAKSPublicIP*.

```
az network public-ip create \
--resource-group MC_myResourceGroup_myAKSCluster_eastus \
--name myAKSPublicIP \
--allocation-method static
```

The IP address is shown, as shown in the following condensed example output:

```
{
  "publicIp": {
    "dnsSettings": null,
    "etag": "W/\"6b6fb15c-5281-4f64-b332-8f68f46e1358\"",
    "id": "/subscriptions/<SubscriptionID>/resourceGroups/MC_myResourceGroup_myAKSCluster_eastus/providers/Microsoft.Network/publicIPAddresses/myAKSPublicIP",
    "idleTimeoutInMinutes": 4,
    "ipAddress": "40.121.183.52",
    [...]
  }
}
```

You can later get the public IP address using the [az network public-ip list](#) command. Specify the name of the node resource group, and then query for the *ipAddress* as shown in the following example:

```
$ az network public-ip list --resource-group MC_myResourceGroup_myAKSCluster_eastus --query [0].ipAddress --output tsv
```

```
40.121.183.52
```

Create a service with the static IP

To create a service with the static public IP address, add the `loadBalancerIP` property and the value of the static public IP address to the YAML manifest. Create a file named `egress-service.yaml` and copy in the following YAML. Provide your own public IP address created in the previous step.

```
apiVersion: v1
kind: Service
metadata:
  name: azure-egress
spec:
  loadBalancerIP: 40.121.183.52
  type: LoadBalancer
  ports:
  - port: 80
```

Create the service and deployment with the `kubectl apply` command.

```
kubectl apply -f egress-service.yaml
```

This service configures a new frontend IP on the Azure Load Balancer. If you do not have any other IPs configured, then all egress traffic should now use this address. When multiple addresses are configured on the Azure Load Balancer, any of these public IP addresses are a candidate for outbound flows, and one is selected at random.

Verify egress address

To verify that the static public IP address is being used, you can use DNS look-up service such as `checkip.dyndns.org`.

Start and attach to a basic *Debian* pod:

```
kubectl run -it --rm aks-ip --image=mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11
```

To access a web site from within the container, use `apt-get` to install `curl` into the container.

```
apt-get update && apt-get install curl -y
```

Now use `curl` to access the `checkip.dyndns.org` site. The egress IP address is shown, as displayed in the following example output. This IP address matches the static public IP address created and defined for the `loadBalancer` service:

```
$ curl -s checkip.dyndns.org
<html><head><title>Current IP Check</title></head><body>Current IP Address: 40.121.183.52</body></html>
```

Next steps

To avoid maintaining multiple public IP addresses on the Azure Load Balancer, you can instead use an ingress controller. Ingress controllers provide additional benefits such as SSL/TLS termination, support for URI rewrites, and upstream SSL/TLS encryption. For more information, see [Create a basic ingress controller in AKS](#).

Customize CoreDNS with Azure Kubernetes Service

3/5/2021 • 5 minutes to read • [Edit Online](#)

Azure Kubernetes Service (AKS) uses the [CoreDNS](#) project for cluster DNS management and resolution with all 1.12.x and higher clusters. Previously, the kube-dns project was used. This kube-dns project is now deprecated. For more information about CoreDNS customization and Kubernetes, see the [official upstream documentation](#).

As AKS is a managed service, you cannot modify the main configuration for CoreDNS (a *CoreFile*). Instead, you use a Kubernetes *ConfigMap* to override the default settings. To see the default AKS CoreDNS ConfigMaps, use the `kubectl get configmaps --namespace=kube-system coredns -o yaml` command.

This article shows you how to use ConfigMaps for basic customization options of CoreDNS in AKS. This approach differs from configuring CoreDNS in other contexts such as using the CoreFile. Verify the version of CoreDNS you are running as the configuration values may change between versions.

NOTE

`kube-dns` offered different [customization options](#) via a Kubernetes config map. CoreDNS is **not** backwards compatible with kube-dns. Any customizations you previously used must be updated for use with CoreDNS.

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart [using the Azure CLI](#) or [using the Azure portal](#).

When creating a configuration like the examples below, your names in the *data* section must end in either `.server` or `.override`. This naming convention is defined in the default AKS CoreDNS Configmap which you can view using the `kubectl get configmaps --namespace=kube-system coredns -o yaml` command.

What is supported/unsupported

All built-in CoreDNS plugins are supported. No add-on/third party plugins are supported.

Rewrite DNS

One scenario you have is to perform on-the-fly DNS name rewrites. In the following example, replace `<domain to be written>` with your own fully qualified domain name. Create a file named `corednsms.yaml` and paste the following example configuration:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns-custom
  namespace: kube-system
data:
  test.server: | # you may select any name here, but it must end with the .server file extension
    <domain to be rewritten>.com:53 {
      errors
      cache 30
      rewrite name substring <domain to be rewritten>.com default.svc.cluster.local
      kubernetes cluster.local in-addr.arpa ip6.arpa {
        pods insecure
        upstream
        fallthrough in-addr.arpa ip6.arpa
      }
      forward . /etc/resolv.conf # you can redirect this to a specific DNS server such as 10.0.0.10, but
      that server must be able to resolve the rewritten domain name
    }
```

IMPORTANT

If you redirect to a DNS server, such as the CoreDNS service IP, that DNS server must be able to resolve the rewritten domain name.

Create the ConfigMap using the [kubectl apply configmap](#) command and specify the name of your YAML manifest:

```
kubectl apply -f corednsms.yaml
```

To verify the customizations have been applied, use the [kubectl get configmaps](#) and specify your *coredns-custom* ConfigMap:

```
kubectl get configmaps --namespace=kube-system coredns-custom -o yaml
```

Now force CoreDNS to reload the ConfigMap. The [kubectl delete pod](#) command isn't destructive and doesn't cause down time. The `kube-dns` pods are deleted, and the Kubernetes Scheduler then recreates them. These new pods contain the change in TTL value.

```
kubectl delete pod --namespace kube-system -l k8s-app=kube-dns
```

NOTE

The command above is correct. While we're changing `coredns`, the deployment is under the `kube-dns` name.

Custom forward server

If you need to specify a forward server for your network traffic, you can create a ConfigMap to customize DNS. In the following example, update the `forward` name and address with the values for your own environment.

Create a file named `corednsms.yaml` and paste the following example configuration:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns-custom
  namespace: kube-system
data:
  test.server: | # you may select any name here, but it must end with the .server file extension
    <domain to be rewritten>.com:53 {
      forward foo.com 1.1.1.1
    }
```

As in the previous examples, create the ConfigMap using the [kubectl apply configmap](#) command and specify the name of your YAML manifest. Then, force CoreDNS to reload the ConfigMap using the [kubectl delete pod](#) for the Kubernetes Scheduler to recreate them:

```
kubectl apply -f corednsms.yaml
kubectl delete pod --namespace kube-system --selector k8s-app=kube-dns
```

Use custom domains

You may want to configure custom domains that can only be resolved internally. For example, you may want to resolve the custom domain *puglife.local*, which isn't a valid top-level domain. Without a custom domain ConfigMap, the AKS cluster can't resolve the address.

In the following example, update the custom domain and IP address to direct traffic to with the values for your own environment. Create a file named `corednsms.yaml` and paste the following example configuration:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns-custom
  namespace: kube-system
data:
  puglife.server: | # you may select any name here, but it must end with the .server file extension
    puglife.local:53 {
      errors
      cache 30
      forward . 192.11.0.1 # this is my test/dev DNS server
    }
```

As in the previous examples, create the ConfigMap using the [kubectl apply configmap](#) command and specify the name of your YAML manifest. Then, force CoreDNS to reload the ConfigMap using the [kubectl delete pod](#) for the Kubernetes Scheduler to recreate them:

```
kubectl apply -f corednsms.yaml
kubectl delete pod --namespace kube-system --selector k8s-app=kube-dns
```

Stub domains

CoreDNS can also be used to configure stub domains. In the following example, update the custom domains and IP addresses with the values for your own environment. Create a file named `corednsms.yaml` and paste the following example configuration:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns-custom
  namespace: kube-system
data:
  test.server: | # you may select any name here, but it must end with the .server file extension
    abc.com:53 {
      errors
      cache 30
      forward . 1.2.3.4
    }
    my.cluster.local:53 {
      errors
      cache 30
      forward . 2.3.4.5
    }

```

As in the previous examples, create the ConfigMap using the [kubectl apply configmap](#) command and specify the name of your YAML manifest. Then, force CoreDNS to reload the ConfigMap using the [kubectl delete pod](#) for the Kubernetes Scheduler to recreate them:

```

kubectl apply -f corednsm.yaml
kubectl delete pod --namespace kube-system --selector k8s-app=kube-dns

```

Hosts plugin

As all built-in plugins are supported this means that the CoreDNS [Hosts](#) plugin is available to customize as well:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns-custom # this is the name of the configmap you can overwrite with your changes
  namespace: kube-system
data:
  test.override: | # you may select any name here, but it must end with the .override file extension
    hosts example.hosts example.org { # example.hosts must be a file
      10.0.0.1 example.org
      fallthrough
    }

```

Enable logging for DNS query debugging

To enable DNS query logging, apply the following configuration in your coredns-custom ConfigMap:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns-custom
  namespace: kube-system
data:
  log.override: | # you may select any name here, but it must end with the .override file extension
    log

```

Next steps

This article showed some example scenarios for CoreDNS customization. For information on the CoreDNS project, see [the CoreDNS upstream project page](#).

To learn more about core network concepts, see [Network concepts for applications in AKS](#).

Service principals with Azure Kubernetes Service (AKS)

5/11/2021 • 11 minutes to read • [Edit Online](#)

To interact with Azure APIs, an AKS cluster requires either an [Azure Active Directory \(AD\) service principal](#) or a [managed identity](#). A service principal or managed identity is needed to dynamically create and manage other Azure resources such as an Azure load balancer or container registry (ACR).

This article shows how to create and use a service principal for your AKS clusters.

Before you begin

To create an Azure AD service principal, you must have permissions to register an application with your Azure AD tenant, and to assign the application to a role in your subscription. If you don't have the necessary permissions, you might need to ask your Azure AD or subscription administrator to assign the necessary permissions, or pre-create a service principal for you to use with the AKS cluster.

If you are using a service principal from a different Azure AD tenant, there are additional considerations around the permissions available when you deploy the cluster. You may not have the appropriate permissions to read and write directory information. For more information, see [What are the default user permissions in Azure Active Directory?](#)

- [Azure CLI](#)
- [Azure PowerShell](#)

You also need the Azure CLI version 2.0.59 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Automatically create and use a service principal

- [Azure CLI](#)
- [Azure PowerShell](#)

When you create an AKS cluster in the Azure portal or using the `az aks create` command, Azure can automatically generate a service principal.

In the following Azure CLI example, a service principal is not specified. In this scenario, the Azure CLI creates a service principal for the AKS cluster. To successfully complete the operation, your Azure account must have the proper rights to create a service principal.

```
az aks create --name myAKSCluster --resource-group myResourceGroup
```

Manually create a service principal

- [Azure CLI](#)
- [Azure PowerShell](#)

To manually create a service principal with the Azure CLI, use the `az ad sp create-for-rbac` command. In the following example, the `--skip-assignment` parameter prevents any additional default assignments being

assigned:

```
az ad sp create-for-rbac --skip-assignment --name myAKSClusterServicePrincipal
```

The output is similar to the following example. Make a note of your own `appId` and `password`. These values are used when you create an AKS cluster in the next section.

```
{  
    "appId": "559513bd-0c19-4c1a-87cd-851a26af5fc",  
    "displayName": "myAKSClusterServicePrincipal",  
    "name": "http://myAKSClusterServicePrincipal",  
    "password": "e763725a-5eee-40e8-a466-dc88d980f415",  
    "tenant": "72f988bf-86f1-41af-91ab-2d7cd011db48"  
}
```

Specify a service principal for an AKS cluster

- [Azure CLI](#)
- [Azure PowerShell](#)

To use an existing service principal when you create an AKS cluster using the `az aks create` command, use the `--service-principal` and `--client-secret` parameters to specify the `appId` and `password` from the output of the `az ad sp create-for-rbac` command:

```
az aks create \  
    --resource-group myResourceGroup \  
    --name myAKSCluster \  
    --service-principal <appId> \  
    --client-secret <password>
```

NOTE

If you're using an existing service principal with customized secret, ensure the secret is no longer than 190 bytes.

If you deploy an AKS cluster using the Azure portal, on the *Authentication* page of the **Create Kubernetes cluster** dialog, choose to **Configure service principal**. Select **Use existing**, and specify the following values:

- **Service principal client ID** is your `appId`
- **Service principal client secret** is the `password` value

The screenshot shows the 'Create Kubernetes cluster' wizard in the Azure portal. The current step is 'Configure service principal'. In the 'Service principal' section, the 'Use existing' radio button is selected. The 'Service principal client ID' field contains the value '559513bd-0c19-4c1a-87cd-851a26af5fc'. The 'Service principal client secret' field is partially visible. On the left, the 'Authentication' tab is selected, showing options for 'Cluster infrastructure' (using a default service principal) and 'Kubernetes authentication and authorization' (enabling RBAC). The RBAC setting is set to 'Yes'. At the bottom, there are buttons for 'Review + create', '< Previous' (Next : Networking >), and 'Ok'.

Delegate access to other Azure resources

The service principal for the AKS cluster can be used to access other resources. For example, if you want to deploy your AKS cluster into an existing Azure virtual network subnet or connect to Azure Container Registry (ACR), you need to delegate access to those resources to the service principal.

- [Azure CLI](#)
- [Azure PowerShell](#)

To delegate permissions, create a role assignment using the `az role assignment create` command. Assign the `<appId>` to a particular scope, such as a resource group or virtual network resource. A role then defines what permissions the service principal has on the resource, as shown in the following example:

```
az role assignment create --assignee <appId> --scope <resourceScope> --role Contributor
```

The `--scope` for a resource needs to be a full resource ID, such as `/subscriptions/<guid>/resourceGroups/myResourceGroup` or `/subscriptions/<guid>/resourceGroups/myResourceGroupVnet/providers/Microsoft.Network/virtualNetworks/myVnet`

NOTE

If you have removed the Contributor role assignment from the node resource group, the operations below may fail.

The following sections detail common delegations that you may need to make.

Azure Container Registry

- [Azure CLI](#)
- [Azure PowerShell](#)

If you use Azure Container Registry (ACR) as your container image store, you need to grant permissions to the service principal for your AKS cluster to read and pull images. Currently, the recommended configuration is to use the `az aks create` or `az aks update` command to integrate with a registry and assign the appropriate role for the service principal. For detailed steps, see [Authenticate with Azure Container Registry from Azure Kubernetes Service](#).

Networking

You may use advanced networking where the virtual network and subnet or public IP addresses are in another resource group. Assign the [Network Contributor](#) built-in role on the subnet within the virtual network.

Alternatively, you can create a [custom role](#) with permissions to access the network resources in that resource group. See [AKS service permissions](#) for more details.

Storage

You may need to access existing Disk resources in another resource group. Assign one of the following set of role permissions:

- Create a [custom role](#) and define the following role permissions:
 - *Microsoft.Compute/disks/read*
 - *Microsoft.Compute/disks/write*
- Or, assign the [Storage Account Contributor](#) built-in role on the resource group

Azure Container Instances

If you use Virtual Kubelet to integrate with AKS and choose to run Azure Container Instances (ACI) in resource group separate to the AKS cluster, the AKS service principal must be granted *Contributor* permissions on the ACI resource group.

Additional considerations

- [Azure CLI](#)
- [Azure PowerShell](#)

When using AKS and Azure AD service principals, keep the following considerations in mind.

- The service principal for Kubernetes is a part of the cluster configuration. However, don't use the identity to deploy the cluster.
- By default, the service principal credentials are valid for one year. You can [update or rotate the service principal credentials](#) at any time.
- Every service principal is associated with an Azure AD application. The service principal for a Kubernetes cluster can be associated with any valid Azure AD application name (for example: <https://www.contoso.org/example>). The URL for the application doesn't have to be a real endpoint.
- When you specify the service principal **Client ID**, use the value of the `appId`.
- On the agent node VMs in the Kubernetes cluster, the service principal credentials are stored in the file `/etc/kubernetes/azure.json`
- When you use the `az aks create` command to generate the service principal automatically, the service principal credentials are written to the file `~/.azure/aksServicePrincipal.json` on the machine used to run the command.
- If you do not specifically pass a service principal in additional AKS CLI commands, the default service principal located at `~/.azure/aksServicePrincipal.json` is used.
- You can also optionally remove the `aksServicePrincipal.json` file, and AKS will create a new service principal.
- When you delete an AKS cluster that was created by `az aks create`, the service principal that was created automatically is not deleted.
 - To delete the service principal, query for your cluster `servicePrincipalProfile.clientId` and then delete with `az ad sp delete`. Replace the following resource group and cluster names with your own values:

```
az ad sp delete --id $(az aks show -g myResourceGroup -n myAKScluster --query servicePrincipalProfile.clientId -o tsv)
```

Troubleshoot

- [Azure CLI](#)
- [Azure PowerShell](#)

The service principal credentials for an AKS cluster are cached by the Azure CLI. If these credentials have expired, you encounter errors deploying AKS clusters. The following error message when running `az aks create` may indicate a problem with the cached service principal credentials:

```
Operation failed with status: 'Bad Request'.
Details: The credentials in ServicePrincipalProfile were invalid. Please see https://aka.ms/aks-sp-help for
more details.
(Details: adal: Refresh request failed. Status Code = '401'.
```

Check the age of the credentials file using the following command:

```
ls -la $HOME/.azure/aksServicePrincipal.json
```

The default expiration time for the service principal credentials is one year. If your `aksServicePrincipal.json` file is older than one year, delete the file and try to deploy an AKS cluster again.

Next steps

For more information about Azure Active Directory service principals, see [Application and service principal objects](#).

For information on how to update the credentials, see [Update or rotate the credentials for a service principal in AKS](#).

Use managed identities in Azure Kubernetes Service

5/12/2021 • 8 minutes to read • [Edit Online](#)

Currently, an Azure Kubernetes Service (AKS) cluster (specifically, the Kubernetes cloud provider) requires an identity to create additional resources like load balancers and managed disks in Azure. This identity can be either a *managed identity* or a *service principal*. If you use a [service principal](#), you must either provide one or AKS creates one on your behalf. If you use managed identity, this will be created for you by AKS automatically. Clusters using service principals eventually reach a state in which the service principal must be renewed to keep the cluster working. Managing service principals adds complexity, which is why it's easier to use managed identities instead. The same permission requirements apply for both service principals and managed identities.

Managed identities are essentially a wrapper around service principals, and make their management simpler. Credential rotation for MI happens automatically every 46 days according to Azure Active Directory default. AKS uses both system-assigned and user-assigned managed identity types. These identities are currently immutable. To learn more, read about [managed identities for Azure resources](#).

Before you begin

You must have the following resource installed:

- The Azure CLI, version 2.23.0 or later

Limitations

- Tenants move / migrate of managed identity enabled clusters isn't supported.
- If the cluster has `aad-pod-identity` enabled, Node-Managed Identity (NMI) pods modify the nodes' iptables to intercept calls to the Azure Instance Metadata endpoint. This configuration means any request made to the Metadata endpoint is intercepted by NMI even if the pod doesn't use `aad-pod-identity`.

AzurePodIdentityException CRD can be configured to inform `aad-pod-identity` that any requests to the Metadata endpoint originating from a pod that matches labels defined in CRD should be proxied without any processing in NMI. The system pods with `kubernetes.azure.com/managedby: aks` label in `kube-system` namespace should be excluded in `aad-pod-identity` by configuring the AzurePodIdentityException CRD. For more information, see [Disable aad-pod-identity for a specific pod or application](#). To configure an exception, install the [mic-exception YAML](#).

Summary of managed identities

AKS uses several managed identities for built-in services and add-ons.

IDENTITY	NAME	USE CASE	DEFAULT PERMISSIONS	BRING YOUR OWN IDENTITY

IDENTITY	NAME	USE CASE	DEFAULT PERMISSIONS	BRING YOUR OWN IDENTITY
Control plane	not visible	Used by AKS control plane components to manage cluster resources including ingress load balancers and AKS managed public IPs, and Cluster Autoscaler operations	Contributor role for Node resource group	Supported
Kubelet	AKS Cluster Name-agentpool	Authentication with Azure Container Registry (ACR)	NA (for kubernetes v1.15+)	Supported (Preview)
Add-on	AzureNPM	No identity required	NA	No
Add-on	AzureCNI network monitoring	No identity required	NA	No
Add-on	azure-policy (gatekeeper)	No identity required	NA	No
Add-on	azure-policy	No identity required	NA	No
Add-on	Calico	No identity required	NA	No
Add-on	Dashboard	No identity required	NA	No
Add-on	HTTPApplicationRouting	Manages required network resources	Reader role for node resource group, contributor role for DNS zone	No
Add-on	Ingress application gateway	Manages required network resources	Contributor role for node resource group	No
Add-on	omsagent	Used to send AKS metrics to Azure Monitor	Monitoring Metrics Publisher role	No
Add-on	Virtual-Node (ACIConnector)	Manages required network resources for Azure Container Instances (ACI)	Contributor role for node resource group	No
OSS project	aad-pod-identity	Enables applications to access cloud resources securely with Azure Active Directory (AAD)	NA	Steps to grant permission at https://github.com/Azure/aad-pod-identity#role-assignment .

Create an AKS cluster with managed identities

You can now create an AKS cluster with managed identities by using the following CLI commands.

First, create an Azure resource group:

```
# Create an Azure resource group
az group create --name myResourceGroup --location westus2
```

Then, create an AKS cluster:

```
az aks create -g myResourceGroup -n myManagedCluster --enable-managed-identity
```

Once the cluster is created, you can then deploy your application workloads to the new cluster and interact with it just as you've done with service-principal-based AKS clusters.

Finally, get credentials to access the cluster:

```
az aks get-credentials --resource-group myResourceGroup --name myManagedCluster
```

Update an AKS cluster to managed identities

You can now update an AKS cluster currently working with service principals to work with managed identities by using the following CLI commands.

```
az aks update -g <RGName> -n <AKSName> --enable-managed-identity
```

NOTE

Once the system-assigned or user-assigned identities have been updated to managed identity, perform an

```
az aks nodepool upgrade --node-image-only
```

 on your nodes to complete the update to managed identity.

Obtain and use the system-assigned managed identity for your AKS cluster

Confirm your AKS cluster is using managed identity with the following CLI command:

```
az aks show -g <RGName> -n <ClusterName> --query "servicePrincipalProfile"
```

If the cluster is using managed identities, you will see a `clientId` value of "msi". A cluster using a Service Principal instead will instead show the object ID. For example:

```
{
  "clientId": "msi"
}
```

After verifying the cluster is using managed identities, you can find the control plane system-assigned identity's object ID with the following command:

```
az aks show -g <RGName> -n <ClusterName> --query "identity"
```

```
{  
    "principalId": "<object-id>",  
    "tenantId": "<tenant-id>",  
    "type": "SystemAssigned",  
    "userAssignedIdentities": null  
},
```

NOTE

For creating and using your own VNet, static IP address, or attached Azure disk where the resources are outside of the worker node resource group, use the PrincipalID of the cluster System Assigned Managed Identity to perform a role assignment. For more information on role assignment, see [Delegate access to other Azure resources](#).

Permission grants to cluster Managed Identity used by Azure Cloud provider may take up 60 minutes to populate.

Bring your own control plane MI

A custom control plane identity enables access to be granted to the existing identity prior to cluster creation. This feature enables scenarios such as using a custom VNET or outboundType of UDR with a pre-created managed identity.

You must have the Azure CLI, version 2.15.1 or later installed.

Limitations

- Azure Government isn't currently supported.
- Azure China 21Vianet isn't currently supported.

If you don't have a managed identity yet, you should go ahead and create one for example by using [az identity CLI](#).

```
az identity create --name myIdentity --resource-group myResourceGroup
```

The result should look like:

```
{  
    "clientId": "<client-id>",  
    "clientSecretUrl": "<clientSecretUrl>",  
    "id":  
        "/subscriptions/<subscriptionid>/resourcegroups/myResourceGroup/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myIdentity",  
    "location": "westus2",  
    "name": "myIdentity",  
    "principalId": "<principalId>",  
    "resourceGroup": "myResourceGroup",  
    "tags": {},  
    "tenantId": "<tenant-id>>",  
    "type": "Microsoft.ManagedIdentity/userAssignedIdentities"  
}
```

If your managed identity is part of your subscription, you can use [az identity CLI command](#) to query it.

```
az identity list --query "[].{Name:name, Id:id, Location:location}" -o table
```

Now you can use the following command to create your cluster with your existing identity:

```
az aks create \
    --resource-group myResourceGroup \
    --name myManagedCluster \
    --network-plugin azure \
    --vnet-subnet-id <subnet-id> \
    --docker-bridge-address 172.17.0.1/16 \
    --dns-service-ip 10.2.0.10 \
    --service-cidr 10.2.0.0/24 \
    --enable-managed-identity \
    --assign-identity <identity-id> \
```

A successful cluster creation using your own managed identities contains this userAssignedIdentities profile information:

```
"identity": {
    "principalId": null,
    "tenantId": null,
    "type": "UserAssigned",
    "userAssignedIdentities": {

        "/subscriptions/<subscriptionid>/resourcegroups/myResourceGroup/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myIdentity": {
            "clientId": "<client-id>",
            "principalId": "<principal-id>"
        }
    }
},
```

Bring your own kubelet MI (Preview)

IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

A Kubelet identity enables access to be granted to the existing identity prior to cluster creation. This feature enables scenarios such as connection to ACR with a pre-created managed identity.

Prerequisites

- You must have the Azure CLI, version 2.21.1 or later installed.
- You must have the aks-preview, version 0.5.10 or later installed.

Limitations

- Only works with a User-Assigned Managed cluster.
- Azure Government isn't currently supported.
- Azure China 21Vianet isn't currently supported.

First, register the feature flag for Kubelet identity:

```
az feature register --namespace Microsoft.ContainerService -n CustomKubeletIdentityPreview
```

It takes a few minutes for the status to show *Registered*. You can check on the registration status using the [az feature list](#) command:

```
az feature list -o table --query "[?contains(name, 'Microsoft.ContainerService/CustomKubeletIdentityPreview')].{Name:name,State:properties.state}"
```

When ready, refresh the registration of the *Microsoft.ContainerService* resource provider using the [az provider register](#) command:

```
az provider register --namespace Microsoft.ContainerService
```

Create or obtain managed identities

If you don't have a control plane managed identity yet, you should go ahead and create one. The following example uses the [az identity create](#) command:

```
az identity create --name myIdentity --resource-group myResourceGroup
```

The result should look like:

```
{
  "clientId": "<client-id>",
  "clientSecretUrl": "<clientSecretUrl>",
  "id": "/subscriptions/<subscriptionid>/resourcegroups/myResourceGroup/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myIdentity",
  "location": "westus2",
  "name": "myIdentity",
  "principalId": "<principalId>",
  "resourceGroup": "myResourceGroup",
  "tags": {},
  "tenantId": "<tenant-id>",
  "type": "Microsoft.ManagedIdentity/userAssignedIdentities"
}
```

If you don't have a kubelet managed identity yet, you should go ahead and create one. The following example uses the [az identity create](#) command:

```
az identity create --name myKubeletIdentity --resource-group myResourceGroup
```

The result should look like:

```
{
  "clientId": "<client-id>",
  "clientSecretUrl": "<clientSecretUrl>",
  "id": "/subscriptions/<subscriptionid>/resourcegroups/myResourceGroup/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myKubeletIdentity",
  "location": "westus2",
  "name": "myKubeletIdentity",
  "principalId": "<principalId>",
  "resourceGroup": "myResourceGroup",
  "tags": {},
  "tenantId": "<tenant-id>",
  "type": "Microsoft.ManagedIdentity/userAssignedIdentities"
}
```

If your existing managed identity is part of your subscription, you can use the [az identity list](#) command to query it:

```
az identity list --query "[].{Name:name, Id:id, Location:location}" -o table
```

Create a cluster using kubelet identity

Now you can use the following command to create your cluster with your existing identities. Provide the control plane identity id via `assign-identity` and the kubelet managed identity via `assign-kubelet-identity`:

```
az aks create \
--resource-group myResourceGroup \
--name myManagedCluster \
--network-plugin azure \
--vnet-subnet-id <subnet-id> \
--docker-bridge-address 172.17.0.1/16 \
--dns-service-ip 10.2.0.10 \
--service-cidr 10.2.0.0/24 \
--enable-managed-identity \
--assign-identity <identity-id> \
--assign-kubelet-identity <kubelet-identity-id> \
```

A successful cluster creation using your own kubelet managed identity contains the following output:

```
"identity": {
    "principalId": null,
    "tenantId": null,
    "type": "UserAssigned",
    "userAssignedIdentities": {

        "/subscriptions/<subscriptionid>/resourcegroups/resourcegroups/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myIdentity": {
            "clientId": "<client-id>",
            "principalId": "<principal-id>"
        }
    },
    "identityProfile": {
        "kubeletIdentity": {
            "clientId": "<client-id>",
            "objectId": "<object-id>",
            "resourceId": "<resource-id>"
        }
    }
},
```

Next steps

- Use [Azure Resource Manager templates](#) to create Managed Identity enabled clusters.

Use Azure Active Directory pod-managed identities in Azure Kubernetes Service (Preview)

4/21/2021 • 5 minutes to read • [Edit Online](#)

Azure Active Directory pod-managed identities uses Kubernetes primitives to associate [managed identities for Azure resources](#) and identities in Azure Active Directory (AAD) with pods. Administrators create identities and bindings as Kubernetes primitives that allow pods to access Azure resources that rely on AAD as an identity provider.

NOTE

If you have an existing installation of `AADPODIDENTITY`, you must remove the existing installation. Enabling this feature means that the MIC component isn't needed.

IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

Before you begin

You must have the following resource installed:

- The Azure CLI, version 2.20.0 or later
- The `azure-preview` extension version 0.5.5 or later

Limitations

- A maximum of 200 pod identities are allowed for a cluster.
- A maximum of 200 pod identity exceptions are allowed for a cluster.
- Pod-managed identities are available on Linux node pools only.

Register the `EnablePodIdentityPreview`

Register the `EnablePodIdentityPreview` feature:

```
az feature register --name EnablePodIdentityPreview --namespace Microsoft.ContainerService
```

Install the `aks-preview` Azure CLI

You also need the `aks-preview` Azure CLI extension version 0.4.64 or later. Install the `aks-preview` Azure CLI extension by using the `az extension add` command. Or install any available updates by using the `az extension update` command.

```
# Install the aks-preview extension  
az extension add --name aks-preview  
  
# Update the extension to make sure you have the latest version installed  
az extension update --name aks-preview
```

Create an AKS cluster with Azure CNI

NOTE

This is the default recommended configuration

Create an AKS cluster with Azure CNI and pod-managed identity enabled. The following commands use [az group create](#) to create a resource group named *myResourceGroup* and the [az aks create](#) command to create an AKS cluster named *myAKSCluster* in the *myResourceGroup* resource group.

```
az group create --name myResourceGroup --location eastus  
az aks create -g myResourceGroup -n myAKSCluster --enable-pod-identity --network-plugin azure
```

Use [az aks get-credentials](#) to sign in to your AKS cluster. This command also downloads and configures the [kubectl](#) client certificate on your development computer.

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```

Update an existing AKS cluster with Azure CNI

Update an existing AKS cluster with Azure CNI to include pod-managed identity.

```
az aks update -g $MY_RESOURCE_GROUP -n $MY_CLUSTER --enable-pod-identity --network-plugin azure
```

Using Kubenet network plugin with Azure Active Directory pod-managed identities

IMPORTANT

Running aad-pod-identity in a cluster with Kubenet is not a recommended configuration because of the security implication. Please follow the mitigation steps and configure policies before enabling aad-pod-identity in a cluster with Kubenet.

Mitigation

To mitigate the vulnerability at the cluster level, you can use OpenPolicyAgent admission controller together with Gatekeeper validating webhook. Provided you have Gatekeeper already installed in your cluster, add the ConstraintTemplate of type K8sPSPCapabilities:

```
kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper-library/master/library/pod-security-policy/capabilities/template.yaml
```

Add a template to limit the spawning of Pods with the NET_RAW capability:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sPSPCapabilities
metadata:
  name: prevent-net-raw
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
  excludedNamespaces:
    - "kube-system"
parameters:
  requiredDropCapabilities: ["NET_RAW"]
```

Create an AKS cluster with Kubenet network plugin

Create an AKS cluster with Kubenet network plugin and pod-managed identity enabled.

```
az aks create -g $MY_RESOURCE_GROUP -n $MY_CLUSTER --enable-pod-identity --enable-pod-identity-with-kubenet
```

Update an existing AKS cluster with Kubenet network plugin

Update an existing AKS cluster with Kubenet network plugin to include pod-managed identity.

```
az aks update -g $MY_RESOURCE_GROUP -n $MY_CLUSTER --enable-pod-identity --enable-pod-identity-with-kubenet
```

Create an identity

Create an identity using [az identity create](#) and set the *IDENTITY_CLIENT_ID* and *IDENTITY_RESOURCE_ID* variables.

```
az group create --name myIdentityResourceGroup --location eastus
export IDENTITY_RESOURCE_GROUP="myIdentityResourceGroup"
export IDENTITY_NAME="application-identity"
az identity create --resource-group ${IDENTITY_RESOURCE_GROUP} --name ${IDENTITY_NAME}
export IDENTITY_CLIENT_ID="$(az identity show -g ${IDENTITY_RESOURCE_GROUP} -n ${IDENTITY_NAME} --query clientId -otsv)"
export IDENTITY_RESOURCE_ID="$(az identity show -g ${IDENTITY_RESOURCE_GROUP} -n ${IDENTITY_NAME} --query id -otsv)"
```

Assign permissions for the managed identity

The *IDENTITY_CLIENT_ID* managed identity must have Reader permissions in the resource group that contains the virtual machine scale set of your AKS cluster.

```
NODE_GROUP=$(az aks show -g myResourceGroup -n myAKSCluster --query nodeResourceGroup -o tsv)
NODES_RESOURCE_ID=$(az group show -n $NODE_GROUP -o tsv --query "id")
az role assignment create --role "Reader" --assignee "$IDENTITY_CLIENT_ID" --scope $NODES_RESOURCE_ID
```

Create a pod identity

Create a pod identity for the cluster using `az aks pod-identity add`.

IMPORTANT

You must have the appropriate permissions, such as `Owner`, on your subscription to create the identity and role binding.

```
export POD_IDENTITY_NAME="my-pod-identity"
export POD_IDENTITY_NAMESPACE="my-app"
az aks pod-identity add --resource-group myResourceGroup --cluster-name myAKSCluster --namespace
${POD_IDENTITY_NAMESPACE} --name ${POD_IDENTITY_NAME} --identity-resource-id ${IDENTITY_RESOURCE_ID}
```

NOTE

When you enable pod-managed identity on your AKS cluster, an `AzurePodIdentityException` named `aks-addon-exception` is added to the `kube-system` namespace. An `AzurePodIdentityException` allows pods with certain labels to access the Azure Instance Metadata Service (IMDS) endpoint without being intercepted by the node-managed identity (NMI) server. The `aks-addon-exception` allows AKS first-party addons, such as AAD pod-managed identity, to operate without having to manually configure an `AzurePodIdentityException`. Optionally, you can add, remove, and update an `AzurePodIdentityException` using `az aks pod-identity exception add`, `az aks pod-identity exception delete`, `az aks pod-identity exception update`, or `kubectl`.

Run a sample application

For a pod to use AAD pod-managed identity, the pod needs an `aadpodidbinding` label with a value that matches a selector from a `AzureIdentityBinding`. To run a sample application using AAD pod-managed identity, create a `demo.yaml` file with the following contents. Replace `POD_IDENTITY_NAME`, `IDENTITY_CLIENT_ID`, and `IDENTITY_RESOURCE_GROUP` with the values from the previous steps. Replace `SUBSCRIPTION_ID` with your subscription id.

NOTE

In the previous steps, you created the `POD_IDENTITY_NAME`, `IDENTITY_CLIENT_ID`, and `IDENTITY_RESOURCE_GROUP` variables. You can use a command such as `echo` to display the value you set for variables, for example `echo $IDENTITY_NAME`.

```

apiVersion: v1
kind: Pod
metadata:
  name: demo
  labels:
    aadpodidbinding: POD_IDENTITY_NAME
spec:
  containers:
    - name: demo
      image: mcr.microsoft.com/oss/azure/aad-pod-identity/demo:v1.6.3
      args:
        - --subscriptionid=SUBSCRIPTION_ID
        - --clientid=IDENTITY_CLIENT_ID
        - --resourcegroup=IDENTITY_RESOURCE_GROUP
      env:
        - name: MY_POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: MY_POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        - name: MY_POD_IP
          valueFrom:
            fieldRef:
              fieldPath: status.podIP
  nodeSelector:
    kubernetes.io/os: linux

```

Notice the pod definition has an *aadpodidbinding* label with a value that matches the name of the pod identity you ran `az aks pod-identity add` in the previous step.

Deploy `demo.yaml` to the same namespace as your pod identity using `kubectl apply`:

```
kubectl apply -f demo.yaml --namespace $POD_IDENTITY_NAMESPACE
```

Verify the sample application successfully runs using `kubectl logs`.

```
kubectl logs demo --follow --namespace $POD_IDENTITY_NAMESPACE
```

Verify the logs show the a token is successfully acquired and the *GET* operation is successful.

```

...
successfully doARMOperations vm count 0
successfully acquired a token using the MSI,
msiEndpoint(http://169.254.169.254/metadata/identity/oauth2/token)
successfully acquired a token, userAssignedID MSI,
msiEndpoint(http://169.254.169.254/metadata/identity/oauth2/token) clientID(xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxx)
successfully made GET on instance metadata
...

```

Clean up

To remove AAD pod-managed identity from your cluster, remove the sample application and the pod identity from the cluster. Then remove the identity.

```
kubectl delete pod demo --namespace ${POD_IDENTITY_NAMESPACE}
az aks pod-identity delete --name ${POD_IDENTITY_NAME} --namespace ${POD_IDENTITY_NAMESPACE} --resource-group myResourceGroup --cluster-name myAKSCluster
az identity delete -g ${IDENTITY_RESOURCE_GROUP} -n ${IDENTITY_NAME}
```

Next steps

For more information on managed identities, see [Managed identities for Azure resources](#).

Use Azure role-based access control to define access to the Kubernetes configuration file in Azure Kubernetes Service (AKS)

4/21/2021 • 4 minutes to read • [Edit Online](#)

You can interact with Kubernetes clusters using the `kubectl` tool. The Azure CLI provides an easy way to get the access credentials and configuration information to connect to your AKS clusters using `kubectl`. To limit who can get that Kubernetes configuration (*kubeconfig*) information and to limit the permissions they then have, you can use Azure role-based access control (Azure RBAC).

This article shows you how to assign Azure roles that limit who can get the configuration information for an AKS cluster.

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart [using the Azure CLI](#) or [using the Azure portal](#).

This article also requires that you are running the Azure CLI version 2.0.65 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Available cluster roles permissions

When you interact with an AKS cluster using the `kubectl` tool, a configuration file is used that defines cluster connection information. This configuration file is typically stored in `~/.kube/config`. Multiple clusters can be defined in this *kubeconfig* file. You switch between clusters using the [kubectl config use-context](#) command.

The [az aks get-credentials](#) command lets you get the access credentials for an AKS cluster and merges them into the *kubeconfig* file. You can use Azure role-based access control (Azure RBAC) to control access to these credentials. These Azure roles let you define who can retrieve the *kubeconfig* file, and what permissions they then have within the cluster.

The two built-in roles are:

- **Azure Kubernetes Service Cluster Admin Role**
 - Allows access to *Microsoft.ContainerService/managedClusters/listClusterAdminCredential*/action API call. This API call [lists the cluster admin credentials](#).
 - Downloads *kubeconfig* for the *clusterAdmin* role.
- **Azure Kubernetes Service Cluster User Role**
 - Allows access to *Microsoft.ContainerService/managedClusters/listClusterUserCredential*/action API call. This API call [lists the cluster user credentials](#).
 - Downloads *kubeconfig* for *clusterUser* role.

These Azure roles can be applied to an Azure Active Directory (AD) user or group.

NOTE

On clusters that use Azure AD, users with the *clusterUser* role have an empty *kubeconfig* file that prompts a log in. Once logged in, users have access based on their Azure AD user or group settings. Users with the *clusterAdmin* role have admin access.

Clusters that do not use Azure AD only use the *clusterAdmin* role.

Assign role permissions to a user or group

To assign one of the available roles, you need to get the resource ID of the AKS cluster and the ID of the Azure AD user account or group. The following example commands:

- Get the cluster resource ID using the `az aks show` command for the cluster named *myAKSCluster* in the *myResourceGroup* resource group. Provide your own cluster and resource group name as needed.
- Use the `az account show` and `az ad user show` commands to get your user ID.
- Finally, assign a role using the `az role assignment create` command.

The following example assigns the *Azure Kubernetes Service Cluster Admin Role* to an individual user account:

```
# Get the resource ID of your AKS cluster
$AKS_CLUSTER=$(az aks show --resource-group myResourceGroup --name myAKSCluster --query id -o tsv)

# Get the account credentials for the logged in user
$ACCOUNT_UPN=$(az account show --query user.name -o tsv)
$ACCOUNT_ID=$(az ad user show --id $ACCOUNT_UPN --query objectId -o tsv)

# Assign the 'Cluster Admin' role to the user
az role assignment create \
    --assignee $ACCOUNT_ID \
    --scope $AKS_CLUSTER \
    --role "Azure Kubernetes Service Cluster Admin Role"
```

IMPORTANT

In some cases, the *user.name* in the account is different than the *userPrincipalName*, such as with Azure AD guest users:

```
$ az account show --query user.name -o tsv
user@contoso.com
$ az ad user list --query "[?contains(otherMails,'user@contoso.com')].{UPN:userPrincipalName}" -o tsv
user_contoso.com#EXT#@contoso.onmicrosoft.com
```

In this case, set the value of *ACCOUNT_UPN* to the *userPrincipalName* from the Azure AD user. For example, if your account *user.name* is *user@contoso.com*.

```
ACCOUNT_UPN=$(az ad user list --query "[?contains(otherMails,'user@contoso.com')].{UPN:userPrincipalName}" -o tsv)
```

TIP

If you want to assign permissions to an Azure AD group, update the `--assignee` parameter shown in the previous example with the object ID for the *group* rather than a *user*. To obtain the object ID for a group, use the `az ad group show` command. The following example gets the object ID for the Azure AD group named *appdev*.

```
az ad group show --group appdev --query objectId -o tsv
```

You can change the previous assignment to the *Cluster User Role* as needed.

The following example output shows the role assignment has been successfully created:

```
{  
  "canDelegate": null,  
  "id":  
    "/subscriptions/<guid>/resourcegroups/myResourceGroup/providers/Microsoft.ContainerService/managedClusters/m  
yAKScluster/providers/Microsoft.Authorization/roleAssignments/b2712174-5a41-4ecb-82c5-12b8ad43d4fb",  
  "name": "b2712174-5a41-4ecb-82c5-12b8ad43d4fb",  
  "principalId": "946016dd-9362-4183-b17d-4c416d1f8f61",  
  "resourceGroup": "myResourceGroup",  
  "roleDefinitionId": "/subscriptions/<guid>/providers/Microsoft.Authorization/roleDefinitions/0ab01a8-8aac-  
4efd-b8c2-3ee1fb270be8",  
  "scope":  
    "/subscriptions/<guid>/resourcegroups/myResourceGroup/providers/Microsoft.ContainerService/managedClusters/m  
yAKScluster",  
  "type": "Microsoft.Authorization/roleAssignments"  
}
```

Get and verify the configuration information

With Azure roles assigned, use the [az aks get-credentials](#) command to get the *kubeconfig* definition for your AKS cluster. The following example gets the *--admin* credentials, which work correctly if the user has been granted the *Cluster Admin Role*.

```
az aks get-credentials --resource-group myResourceGroup --name myAKScluster --admin
```

You can then use the [kubectl config view](#) command to verify that the *context* for the cluster shows that the admin configuration information has been applied:

```
$ kubectl config view  
  
apiVersion: v1  
clusters:  
- cluster:  
    certificate-authority-data: DATA+OMITTED  
    server: https://myaksclust-myresourcegroup-19da35-4839be06.hcp.eastus.azmk8s.io:443  
    name: myAKScluster  
contexts:  
- context:  
    cluster: myAKScluster  
    user: clusterAdmin_myResourceGroup_myAKScluster  
    name: myAKScluster-admin  
current-context: myAKScluster-admin  
kind: Config  
preferences: {}  
users:  
- name: clusterAdmin_myResourceGroup_myAKScluster  
  user:  
    client-certificate-data: REDACTED  
    client-key-data: REDACTED  
    token: e9f2f819a4496538b02cefff94e61d35
```

Remove role permissions

To remove role assignments, use the [az role assignment delete](#) command. Specify the account ID and cluster resource ID, as obtained in the previous commands. If you assigned the role to a group rather than a user, specify the appropriate group object ID rather than account object ID for the `--assignee` parameter:

```
az role assignment delete --assignee $ACCOUNT_ID --scope $AKS_CLUSTER
```

Next steps

For enhanced security on access to AKS clusters, [integrate Azure Active Directory authentication](#).

Secure traffic between pods using network policies in Azure Kubernetes Service (AKS)

4/21/2021 • 15 minutes to read • [Edit Online](#)

When you run modern, microservices-based applications in Kubernetes, you often want to control which components can communicate with each other. The principle of least privilege should be applied to how traffic can flow between pods in an Azure Kubernetes Service (AKS) cluster. Let's say you likely want to block traffic directly to back-end applications. The *Network Policy* feature in Kubernetes lets you define rules for ingress and egress traffic between pods in a cluster.

This article shows you how to install the network policy engine and create Kubernetes network policies to control the flow of traffic between pods in AKS. Network policy should only be used for Linux-based nodes and pods in AKS.

Before you begin

You need the Azure CLI version 2.0.61 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

TIP

If you used the network policy feature during preview, we recommend that you [create a new cluster](#).

If you wish to continue using existing test clusters that used network policy during preview, upgrade your cluster to a new Kubernetes versions for the latest GA release and then deploy the following YAML manifest to fix the crashing metrics server and Kubernetes dashboard. This fix is only required for clusters that used the Calico network policy engine.

As a security best practice, review the contents of this YAML manifest to understand what is deployed into the AKS cluster.

```
kubectl delete -f https://raw.githubusercontent.com/Azure/aks-engine/master/docs/topics/calico-3.3.1-cleanup-after-upgrade.yaml
```

Overview of network policy

All pods in an AKS cluster can send and receive traffic without limitations, by default. To improve security, you can define rules that control the flow of traffic. Back-end applications are often only exposed to required front-end services, for example. Or, database components are only accessible to the application tiers that connect to them.

Network Policy is a Kubernetes specification that defines access policies for communication between Pods. Using Network Policies, you define an ordered set of rules to send and receive traffic and apply them to a collection of pods that match one or more label selectors.

These network policy rules are defined as YAML manifests. Network policies can be included as part of a wider manifest that also creates a deployment or service.

Network policy options in AKS

Azure provides two ways to implement network policy. You choose a network policy option when you create an AKS cluster. The policy option can't be changed after the cluster is created:

- Azure's own implementation, called [Azure Network Policies](#).

- [Calico Network Policies](#), an open-source network and network security solution founded by [Tigera](#).

Both implementations use Linux [IPTables](#) to enforce the specified policies. Policies are translated into sets of allowed and disallowed IP pairs. These pairs are then programmed as IPTable filter rules.

Differences between Azure and Calico policies and their capabilities

CAPABILITY	AZURE	CALICO
Supported platforms	Linux	Linux, Windows Server 2019 (preview)
Supported networking options	Azure CNI	Azure CNI (Windows Server 2019 and Linux) and kubenet (Linux)
Compliance with Kubernetes specification	All policy types supported	All policy types supported
Additional features	None	Extended policy model consisting of Global Network Policy, Global Network Set, and Host Endpoint. For more information on using the calicoctl CLI to manage these extended features, see calicoctl user reference .
Support	Supported by Azure support and Engineering team	Calico community support. For more information on additional paid support, see Project Calico support options .
Logging	Rules added / deleted in IPTables are logged on every host under <code>/var/log/azure-npm.log</code>	For more information, see Calico component logs

Create an AKS cluster and enable network policy

To see network policies in action, let's create and then expand on a policy that defines traffic flow:

- Deny all traffic to pod.
- Allow traffic based on pod labels.
- Allow traffic based on namespace.

First, let's create an AKS cluster that supports network policy.

IMPORTANT

The network policy feature can only be enabled when the cluster is created. You can't enable network policy on an existing AKS cluster.

To use Azure Network Policy, you must use the [Azure CNI plug-in](#) and define your own virtual network and subnets. For more detailed information on how to plan out the required subnet ranges, see [configure advanced networking](#). Calico Network Policy could be used with either this same Azure CNI plug-in or with the [Kubenet CNI plug-in](#).

The following example script:

- Creates a virtual network and subnet.
- Creates an Azure Active Directory (Azure AD) service principal for use with the AKS cluster.

- Assigns *Contributor* permissions for the AKS cluster service principal on the virtual network.
- Creates an AKS cluster in the defined virtual network and enables network policy.
 - The *Azure Network* policy option is used. To use Calico as the network policy option instead, use the `--network-policy calico` parameter. Note: Calico could be used with either `--network-plugin azure` or `--network-plugin kubenet`.

Note that instead of using a service principal, you can use a managed identity for permissions. For more information, see [Use managed identities](#).

Provide your own secure `SP_PASSWORD`. You can replace the `RESOURCE_GROUP_NAME` and `CLUSTER_NAME` variables:

```
RESOURCE_GROUP_NAME=myResourceGroup-NP
CLUSTER_NAME=myAKSCluster
LOCATION=canadaeast

# Create a resource group
az group create --name $RESOURCE_GROUP_NAME --location $LOCATION

# Create a virtual network and subnet
az network vnet create \
  --resource-group $RESOURCE_GROUP_NAME \
  --name myVnet \
  --address-prefixes 10.0.0.0/8 \
  --subnet-name myAKSSubnet \
  --subnet-prefix 10.240.0.0/16

# Create a service principal and read in the application ID
SP=$(az ad sp create-for-rbac --output json)
SP_ID=$(echo $SP | jq -r .appId)
SP_PASSWORD=$(echo $SP | jq -r .password)

# Wait 15 seconds to make sure that service principal has propagated
echo "Waiting for service principal to propagate..."
sleep 15

# Get the virtual network resource ID
VNET_ID=$(az network vnet show --resource-group $RESOURCE_GROUP_NAME --name myVnet --query id -o tsv)

# Assign the service principal Contributor permissions to the virtual network resource
az role assignment create --assignee $SP_ID --scope $VNET_ID --role Contributor

# Get the virtual network subnet resource ID
SUBNET_ID=$(az network vnet subnet show --resource-group $RESOURCE_GROUP_NAME --vnet-name myVnet --name myAKSSubnet --query id -o tsv)
```

Create an AKS cluster for Azure network policies

Create the AKS cluster and specify the virtual network, service principal information, and `azure` for the network plugin and network policy.

```
az aks create \
    --resource-group $RESOURCE_GROUP_NAME \
    --name $CLUSTER_NAME \
    --node-count 1 \
    --generate-ssh-keys \
    --service-cidr 10.0.0.0/16 \
    --dns-service-ip 10.0.0.10 \
    --docker-bridge-address 172.17.0.1/16 \
    --vnet-subnet-id $SUBNET_ID \
    --service-principal $SP_ID \
    --client-secret $SP_PASSWORD \
    --network-plugin azure \
    --network-policy azure
```

It takes a few minutes to create the cluster. When the cluster is ready, configure `kubectl` to connect to your Kubernetes cluster by using the `az aks get-credentials` command. This command downloads credentials and configures the Kubernetes CLI to use them:

```
az aks get-credentials --resource-group $RESOURCE_GROUP_NAME --name $CLUSTER_NAME
```

Create an AKS cluster for Calico network policies

Create the AKS cluster and specify the virtual network, service principal information, `azure` for the network plugin, and `calico` for the network policy. Using `calico` as the network policy enables Calico networking on both Linux and Windows node pools.

If you plan on adding Windows node pools to your cluster, include the `windows-admin-username` and `windows-admin-password` parameters with that meet the [Windows Server password requirements](#). To use Calico with Windows node pools, you also need to register the `Microsoft.ContainerService/EnableAKSWindowsCalico`.

Register the `EnableAKSWindowsCalico` feature flag using the `az feature register` command as shown in the following example:

```
az feature register --namespace "Microsoft.ContainerService" --name "EnableAKSWindowsCalico"
```

You can check on the registration status using the `az feature list` command:

```
az feature list -o table --query "[?contains(name, 'Microsoft.ContainerService/EnableAKSWindowsCalico')].{Name:name,State:properties.state}"
```

When ready, refresh the registration of the `Microsoft.ContainerService` resource provider using the `az provider register` command:

```
az provider register --namespace Microsoft.ContainerService
```

IMPORTANT

At this time, using Calico network policies with Windows nodes is available on new clusters using Kubernetes version 1.20 or later with Calico 3.17.2 and requires using Azure CNI networking. Windows nodes on AKS clusters with Calico enabled also have [Direct Server Return \(DSR\)](#) enabled by default.

For clusters with only Linux node pools running Kubernetes 1.20 with earlier versions of Calico, the Calico version will automatically be upgraded to 3.17.2.

Calico networking policies with Windows nodes is currently in preview.

IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

Create a username to use as administrator credentials for your Windows Server containers on your cluster. The following commands prompt you for a username and set it `WINDOWS_USERNAME` for use in a later command (remember that the commands in this article are entered into a BASH shell).

```
echo "Please enter the username to use as administrator credentials for Windows Server containers on your cluster: " && read WINDOWS_USERNAME
```

```
az aks create \
--resource-group $RESOURCE_GROUP_NAME \
--name $CLUSTER_NAME \
--node-count 1 \
--generate-ssh-keys \
--service-cidr 10.0.0.0/16 \
--dns-service-ip 10.0.0.10 \
--docker-bridge-address 172.17.0.1/16 \
--vnet-subnet-id $SUBNET_ID \
--service-principal $SP_ID \
--client-secret $SP_PASSWORD \
--windows-admin-username $WINDOWS_USERNAME \
--vm-set-type VirtualMachineScaleSets \
--kubernetes-version 1.20.2 \
--network-plugin azure \
--network-policy calico
```

It takes a few minutes to create the cluster. By default, your cluster is created with only a Linux node pool. If you would like to use Windows node pools, you can add one. For example:

```
az aks nodepool add \
--resource-group $RESOURCE_GROUP_NAME \
--cluster-name $CLUSTER_NAME \
--os-type Windows \
--name npwin \
--node-count 1
```

When the cluster is ready, configure `kubectl` to connect to your Kubernetes cluster by using the `az aks get-credentials` command. This command downloads credentials and configures the Kubernetes CLI to use them:

```
az aks get-credentials --resource-group $RESOURCE_GROUP_NAME --name $CLUSTER_NAME
```

Deny all inbound traffic to a pod

Before you define rules to allow specific network traffic, first create a network policy to deny all traffic. This policy gives you a starting point to begin to create an allowlist for only the desired traffic. You can also clearly

see that traffic is dropped when the network policy is applied.

For the sample application environment and traffic rules, let's first create a namespace called *development* to run the example pods:

```
kubectl create namespace development  
kubectl label namespace/development purpose=development
```

Create an example back-end pod that runs NGINX. This back-end pod can be used to simulate a sample back-end web-based application. Create this pod in the *development* namespace, and open port *80* to serve web traffic. Label the pod with *app=webapp,role=backend* so that we can target it with a network policy in the next section:

```
kubectl run backend --image=mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine --labels app=webapp,role=backend  
--namespace development --expose --port 80
```

Create another pod and attach a terminal session to test that you can successfully reach the default NGINX webpage:

```
kubectl run --rm -it --image=mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11 network-policy --  
namespace development
```

At the shell prompt, use `wget` to confirm that you can access the default NGINX webpage:

```
wget -qO- http://backend
```

The following sample output shows that the default NGINX webpage returned:

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Welcome to nginx!</title>  
[...]
```

Exit out of the attached terminal session. The test pod is automatically deleted.

```
exit
```

Create and apply a network policy

Now that you've confirmed you can use the basic NGINX webpage on the sample back-end pod, create a network policy to deny all traffic. Create a file named `backend-policy.yaml` and paste the following YAML manifest. This manifest uses a `podSelector` to attach the policy to pods that have the `app:webapp,role:backend` label, like your sample NGINX pod. No rules are defined under `ingress`, so all inbound traffic to the pod is denied:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: backend-policy
  namespace: development
spec:
  podSelector:
    matchLabels:
      app: webapp
      role: backend
  ingress: []
```

Go to <https://shell.azure.com> to open Azure Cloud Shell in your browser.

Apply the network policy by using the `kubectl apply` command and specify the name of your YAML manifest:

```
kubectl apply -f backend-policy.yaml
```

Test the network policy

Let's see if you can use the NGINX webpage on the back-end pod again. Create another test pod and attach a terminal session:

```
kubectl run --rm -it --image=mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11 network-policy --
namespace development
```

At the shell prompt, use `wget` to see if you can access the default NGINX webpage. This time, set a timeout value to 2 seconds. The network policy now blocks all inbound traffic, so the page can't be loaded, as shown in the following example:

```
wget -qO- --timeout=2 http://backend
```

```
wget: download timed out
```

Exit out of the attached terminal session. The test pod is automatically deleted.

```
exit
```

Allow inbound traffic based on a pod label

In the previous section, a back-end NGINX pod was scheduled, and a network policy was created to deny all traffic. Let's create a front-end pod and update the network policy to allow traffic from front-end pods.

Update the network policy to allow traffic from pods with the labels `app:webapp,role:frontend` and in any namespace. Edit the previous `backend-policy.yaml` file, and add `matchLabels` ingress rules so that your manifest looks like the following example:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: backend-policy
  namespace: development
spec:
  podSelector:
    matchLabels:
      app: webapp
      role: backend
  ingress:
    - from:
      - namespaceSelector: {}
        podSelector:
          matchLabels:
            app: webapp
            role: frontend
```

NOTE

This network policy uses a `namespaceSelector` and a `podSelector` element for the ingress rule. The YAML syntax is important for the ingress rules to be additive. In this example, both elements must match for the ingress rule to be applied. Kubernetes versions prior to 1.12 might not interpret these elements correctly and restrict the network traffic as you expect. For more about this behavior, see [Behavior of to and from selectors](#).

Apply the updated network policy by using the `kubectl apply` command and specify the name of your YAML manifest:

```
kubectl apply -f backend-policy.yaml
```

Schedule a pod that is labeled as `app=webapp,role=frontend` and attach a terminal session:

```
kubectl run --rm -it frontend --image=mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11 --labels
app=webapp,role=frontend --namespace development
```

At the shell prompt, use `wget` to see if you can access the default NGINX webpage:

```
wget -qO- http://backend
```

Because the ingress rule allows traffic with pods that have the labels `app: webapp,role: frontend`, the traffic from the front-end pod is allowed. The following example output shows the default NGINX webpage returned:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

Exit out of the attached terminal session. The pod is automatically deleted.

```
exit
```

Test a pod without a matching label

The network policy allows traffic from pods labeled `app: webapp,role: frontend`, but should deny all other traffic. Let's test to see whether another pod without those labels can access the back-end NGINX pod. Create another test pod and attach a terminal session:

```
kubectl run --rm -it --image=mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11 network-policy --namespace development
```

At the shell prompt, use `wget` to see if you can access the default NGINX webpage. The network policy blocks the inbound traffic, so the page can't be loaded, as shown in the following example:

```
wget -qO- --timeout=2 http://backend
```

```
wget: download timed out
```

Exit out of the attached terminal session. The test pod is automatically deleted.

```
exit
```

Allow traffic only from within a defined namespace

In the previous examples, you created a network policy that denied all traffic, and then updated the policy to allow traffic from pods with a specific label. Another common need is to limit traffic to only within a given namespace. If the previous examples were for traffic in a `development` namespace, create a network policy that prevents traffic from another namespace, such as `production`, from reaching the pods.

First, create a new namespace to simulate a production namespace:

```
kubectl create namespace production  
kubectl label namespace/production purpose=production
```

Schedule a test pod in the `production` namespace that is labeled as `app=webapp,role=frontend`. Attach a terminal session:

```
kubectl run --rm -it frontend --image=mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11 --labels app=webapp,role=frontend --namespace production
```

At the shell prompt, use `wget` to confirm that you can access the default NGINX webpage:

```
wget -qO- http://backend.production
```

Because the labels for the pod match what is currently permitted in the network policy, the traffic is allowed. The network policy doesn't look at the namespaces, only the pod labels. The following example output shows the default NGINX webpage returned:

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Welcome to nginx!</title>  
[...]
```

Exit out of the attached terminal session. The test pod is automatically deleted.

```
exit
```

Update the network policy

Let's update the ingress rule `namespaceSelector` section to only allow traffic from within the `development` namespace. Edit the `backend-policy.yaml` manifest file as shown in the following example:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: backend-policy
  namespace: development
spec:
  podSelector:
    matchLabels:
      app: webapp
      role: backend
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          purpose: development
    podSelector:
      matchLabels:
        app: webapp
        role: frontend
```

In more complex examples, you could define multiple ingress rules, like a `namespaceSelector` and then a `podSelector`.

Apply the updated network policy by using the `kubectl apply` command and specify the name of your YAML manifest:

```
kubectl apply -f backend-policy.yaml
```

Test the updated network policy

Schedule another pod in the `production` namespace and attach a terminal session:

```
kubectl run --rm -it frontend --image=mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11 --labels
app=webapp,role=frontend --namespace production
```

At the shell prompt, use `wget` to see that the network policy now denies traffic:

```
wget -qO- --timeout=2 http://backend.development
```

```
wget: download timed out
```

Exit out of the test pod:

```
exit
```

With traffic denied from the `production` namespace, schedule a test pod back in the `development` namespace,

and attach a terminal session:

```
kubectl run --rm -it frontend --image=mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11 --labels app=webapp,role=frontend --namespace development
```

At the shell prompt, use `wget` to see that the network policy allows the traffic:

```
wget -qO- http://backend
```

Traffic is allowed because the pod is scheduled in the namespace that matches what's permitted in the network policy. The following sample output shows the default NGINX webpage returned:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

Exit out of the attached terminal session. The test pod is automatically deleted.

```
exit
```

Clean up resources

In this article, we created two namespaces and applied a network policy. To clean up these resources, use the `kubectl delete` command and specify the resource names:

```
kubectl delete namespace production
kubectl delete namespace development
```

Next steps

For more about network resources, see [Network concepts for applications in Azure Kubernetes Service \(AKS\)](#).

To learn more about policies, see [Kubernetes network policies](#).

Secure your cluster with Azure Policy

4/21/2021 • 3 minutes to read • [Edit Online](#)

To improve the security of your Azure Kubernetes Service (AKS) cluster, you can apply and enforce built-in security policies on your cluster using Azure Policy. [Azure Policy](#) helps to enforce organizational standards and to assess compliance at-scale. After installing the [Azure Policy Add-on for AKS](#), you can apply individual policy definitions or groups of policy definitions called initiatives (sometimes called policysets) to your cluster. See [Azure Policy built-in definitions for AKS](#) for a complete list of AKS policy and initiative definitions.

This article shows you how to apply policy definitions to your cluster and verify those assignments are being enforced.

Prerequisites

- An existing AKS cluster. If you need an AKS cluster, see the AKS quickstart [using the Azure CLI](#) or [using the Azure portal](#).
- The Azure Policy Add-on for AKS installed on an AKS cluster. Follow these [steps to install the Azure Policy Add-on](#).

Assign a built-in policy definition or initiative

To apply a policy definition or initiative, use the Azure portal.

1. Navigate to the Azure Policy service in Azure portal.
2. In the left pane of the Azure Policy page, select **Definitions**.
3. Under **Categories** select `Kubernetes`.
4. Choose the policy definition or initiative you want to apply. For this example, select the `Kubernetes cluster pod security baseline standards for Linux-based workloads` initiative.
5. Select **Assign**.
6. Set the **Scope** to the resource group of the AKS cluster with the Azure Policy Add-on enabled.
7. Select the **Parameters** page and update the **Effect** from `audit` to `deny` to block new deployments violating the baseline initiative. You can also add additional namespaces to exclude from evaluation. For this example, keep the default values.
8. Select **Review + create** then **Create** to submit the policy assignment.

Validate a Azure Policy is running

Confirm the policy assignments are applied to your cluster by running the following:

```
kubectl get constrainttemplates
```

NOTE

Policy assignments can take [up to 20 minutes to sync](#) into each cluster.

The output should be similar to:

```
$ kubectl get constrainttemplate
NAME                                AGE
k8sazureallowedcapabilities          23m
k8sazureallowedusersgroups          23m
k8sazureblockhostnamespace          23m
k8sazurecontainerallowedimages     23m
k8sazurecontainerallowedports      23m
k8sazurecontainerlimits            23m
k8sazurecontainernoprivilege       23m
k8sazurecontainernoprivilegeescalation 23m
k8sazureenforceapparmor           23m
k8sazurehostfilesystem             23m
k8sazurehostnetworkingports        23m
k8sazurereadonlyrootfilesystem    23m
k8sazureserviceallowedports       23m
```

Validate rejection of a privileged pod

Let's first test what happens when you schedule a pod with the security context of `privileged: true`. This security context escalates the pod's privileges. The initiative disallows privileged pods, so the request will be denied resulting in the deployment being rejected.

Create a file named `nginx-privileged.yaml` and paste the following YAML manifest:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-privileged
spec:
  containers:
    - name: nginx-privileged
      image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
      securityContext:
        privileged: true
```

Create the pod with `kubectl apply` command and specify the name of your YAML manifest:

```
kubectl apply -f nginx-privileged.yaml
```

As expected the pod fails to be scheduled, as shown in the following example output:

```
$ kubectl apply -f privileged.yaml
Error from server ([denied by azurepolicy-container-no-privilege-00edd87bf80f443fa51d10910255adbc4013d590bec3d290b4f48725d4dfbd9] Privileged container is not allowed: nginx-privileged, securityContext: {"privileged": true}): error when creating "privileged.yaml": admission webhook "validation.gatekeeper.sh" denied the request: [denied by azurepolicy-container-no-privilege-00edd87bf80f443fa51d10910255adbc4013d590bec3d290b4f48725d4dfbd9] Privileged container is not allowed: nginx-privileged, securityContext: {"privileged": true}
```

The pod doesn't reach the scheduling stage, so there are no resources to delete before you move on.

Test creation of an unprivileged pod

In the previous example, the container image automatically tried to use root to bind NGINX to port 80. This request was denied by the policy initiative, so the pod fails to start. Let's try now running that same NGINX pod without privileged access.

Create a file named `nginx-unprivileged.yaml` and paste the following YAML manifest:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-unprivileged
spec:
  containers:
    - name: nginx-unprivileged
      image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
```

Create the pod using the [kubectl apply](#) command and specify the name of your YAML manifest:

```
kubectl apply -f nginx-unprivileged.yaml
```

The pod is successfully scheduled. When you check the status of the pod using the [kubectl get pods](#) command, the pod is *Running*.

```
$ kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
nginx-unprivileged   1/1     Running   0          18s
```

This example shows the baseline initiative affecting only deployments which violate policies in the collection. Allowed deployments continue to function.

Delete the NGINX unprivileged pod using the [kubectl delete](#) command and specify the name of your YAML manifest:

```
kubectl delete -f nginx-unprivileged.yaml
```

Disable a policy or initiative

To remove the baseline initiative:

1. Navigate to the Policy pane on the Azure portal.
2. Select **Assignments** from the left pane.
3. Click the ... button next to the [Kubernetes cluster pod security baseline standards for Linux-based workloads](#) initiative.
4. Select **Delete assignment**.

Next steps

For more information about how Azure Policy works:

- [Azure Policy Overview](#)
- [Azure Policy initiatives and polices for AKS](#)
- Remove the [Azure Policy Add-on](#).

Preview - Secure your cluster using pod security policies in Azure Kubernetes Service (AKS)

4/26/2021 • 15 minutes to read • [Edit Online](#)

WARNING

The feature described in this document, pod security policy (preview), will begin deprecation with Kubernetes version 1.21, with its removal in version 1.25. As Kubernetes Upstream approaches that milestone, the Kubernetes community will be working to document viable alternatives. The previous deprecation announcement was made at the time as there was not a viable option for customers. Now that the Kubernetes community is working on an alternative, there no longer is a pressing need to deprecate ahead of Kubernetes.

After pod security policy (preview) is deprecated, you must disable the feature on any existing clusters using the deprecated feature to perform future cluster upgrades and stay within Azure support.

To improve the security of your AKS cluster, you can limit what pods can be scheduled. Pods that request resources you don't allow can't run in the AKS cluster. You define this access using pod security policies. This article shows you how to use pod security policies to limit the deployment of pods in AKS.

IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart using the [Azure CLI](#) or [using the Azure portal](#).

You need the Azure CLI version 2.0.61 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Install `aks-preview` CLI extension

To use pod security policies, you need the `aks-preview` CLI extension version 0.4.1 or higher. Install the `aks-preview` Azure CLI extension using the `az extension add` command, then check for any available updates using the `az extension update` command:

```
# Install the aks-preview extension
az extension add --name aks-preview

# Update the extension to make sure you have the latest version installed
az extension update --name aks-preview
```

Register pod security policy feature provider

This document and feature are set for deprecation on October 15th, 2020.

To create or update an AKS cluster to use pod security policies, first enable a feature flag on your subscription. To register the *PodSecurityPolicyPreview* feature flag, use the [az feature register](#) command as shown in the following example:

```
az feature register --name PodSecurityPolicyPreview --namespace Microsoft.ContainerService
```

It takes a few minutes for the status to show *Registered*. You can check on the registration status using the [az feature list](#) command:

```
az feature list -o table --query "[?contains(name, 'Microsoft.ContainerService/PodSecurityPolicyPreview')].{Name:name,State:properties.state}"
```

When ready, refresh the registration of the *Microsoft.ContainerService* resource provider using the [az provider register](#) command:

```
az provider register --namespace Microsoft.ContainerService
```

Overview of pod security policies

In a Kubernetes cluster, an admission controller is used to intercept requests to the API server when a resource is to be created. The admission controller can then *validate* the resource request against a set of rules, or *mutate* the resource to change deployment parameters.

PodSecurityPolicy is an admission controller that validates a pod specification meets your defined requirements. These requirements may limit the use of privileged containers, access to certain types of storage, or the user or group the container can run as. When you try to deploy a resource where the pod specifications don't meet the requirements outlined in the pod security policy, the request is denied. This ability to control what pods can be scheduled in the AKS cluster prevents some possible security vulnerabilities or privilege escalations.

When you enable pod security policy in an AKS cluster, some default policies are applied. These default policies provide an out-of-the-box experience to define what pods can be scheduled. However, cluster users may run into problems deploying pods until you define your own policies. The recommended approach is to:

- Create an AKS cluster
- Define your own pod security policies
- Enable the pod security policy feature

To show how the default policies limit pod deployments, in this article we first enable the pod security policies feature, then create a custom policy.

Behavior changes between pod security policy and Azure Policy

Below is a summary of behavior changes between pod security policy and Azure Policy.

SCENARIO	POD SECURITY POLICY	AZURE POLICY
Installation	Enable pod security policy feature	Enable Azure Policy Add-on
Deploy policies	Deploy pod security policy resource	Assign Azure policies to the subscription or resource group scope. The Azure Policy Add-on is required for Kubernetes resource applications.

SCENARIO	POD SECURITY POLICY	AZURE POLICY
Default policies	When pod security policy is enabled in AKS, default Privileged and Unrestricted policies are applied.	No default policies are applied by enabling the Azure Policy Add-on. You must explicitly enable policies in Azure Policy.
Who can create and assign policies	Cluster admin creates a pod security policy resource	Users must have a minimum role of 'owner' or 'Resource Policy Contributor' permissions on the AKS cluster resource group. - Through API, users can assign policies at the AKS cluster resource scope. The user should have minimum of 'owner' or 'Resource Policy Contributor' permissions on AKS cluster resource. - In the Azure portal, policies can be assigned at the Management group/subscription/resource group level.
Authorizing policies	Users and Service Accounts require explicit permissions to use pod security policies.	No additional assignment is required to authorize policies. Once policies are assigned in Azure, all cluster users can use these policies.
Policy applicability	The admin user bypasses the enforcement of pod security policies.	All users (admin & non-admin) sees the same policies. There is no special casing based on users. Policy application can be excluded at the namespace level.
Policy scope	Pod security policies are not namespaced	Constraint templates used by Azure Policy are not namespaced.
Deny/Audit/Mutation action	Pod security policies support only deny actions. Mutation can be done with default values on create requests. Validation can be done during update requests.	Azure Policy supports both audit & deny actions. Mutation is not supported yet, but planned.
Pod security policy compliance	There is no visibility on compliance of pods that existed before enabling pod security policy. Non-compliant pods created after enabling pod security policies are denied.	Non-compliant pods that existed before applying Azure policies would show up in policy violations. Non-compliant pods created after enabling Azure policies are denied if policies are set with a deny effect.
How to view policies on the cluster	<code>kubectl get psp</code>	<code>kubectl get constrainttemplate</code> - All policies are returned.
Pod security policy standard - Privileged	A privileged pod security policy resource is created by default when enabling the feature.	Privileged mode implies no restriction, as a result it is equivalent to not having any Azure Policy assignment.
Pod security policy standard - Baseline/default	User installs a pod security policy baseline resource.	Azure Policy provides a built-in baseline initiative which maps to the baseline pod security policy.

SCENARIO	POD SECURITY POLICY	AZURE POLICY
Pod security policy standard - Restricted	User installs a pod security policy restricted resource.	Azure Policy provides a built-in restricted initiative which maps to the restricted pod security policy.

Enable pod security policy on an AKS cluster

You can enable or disable pod security policy using the [az aks update](#) command. The following example enables pod security policy on the cluster name *myAKSCluster* in the resource group named *myResourceGroup*.

NOTE

For real-world use, don't enable the pod security policy until you have defined your own custom policies. In this article, you enable pod security policy as the first step to see how the default policies limit pod deployments.

```
az aks update \
--resource-group myResourceGroup \
--name myAKSCluster \
--enable-pod-security-policy
```

Default AKS policies

When you enable pod security policy, AKS creates one default policy named *privileged*. Don't edit or remove the default policy. Instead, create your own policies that define the settings you want to control. Let's first look at what these default policies are and how they impact pod deployments.

To view the policies available, use the [kubectl get psp](#) command, as shown in the following example

```
$ kubectl get psp

NAME      PRIV   CAPS   SELINUX   RUNASUSER       FSGROUP     SUPGROUP   READONLYROOTFS   VOLUMES
privileged  true    *    RunAsAny  RunAsAny      RunAsAny   RunAsAny   false          *
configMap,emptyDir,projected,secret,downwardAPI,persistentVolumeClaim
```

The *privileged* pod security policy is applied to any authenticated user in the AKS cluster. This assignment is controlled by ClusterRoles and ClusterRoleBindings. Use the [kubectl get rolebindings](#) command and search for the *default:privileged*:binding in the *kube-system* namespace:

```
kubectl get rolebindings default:privileged -n kube-system -o yaml
```

As shown in the following condensed output, the *psp:privileged* ClusterRole is assigned to any *system:authenticated* users. This ability provides a basic level of privilege without your own policies being defined.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  [...]
  name: default:privileged
  [...]
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: psp:privileged
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:masters
```

It's important to understand how these default policies interact with user requests to schedule pods before you start to create your own pod security policies. In the next few sections, let's schedule some pods to see these default policies in action.

Create a test user in an AKS cluster

By default, when you use the [az aks get-credentials](#) command, the *admin* credentials for the AKS cluster are added to your `kubectl` config. The admin user bypasses the enforcement of pod security policies. If you use Azure Active Directory integration for your AKS clusters, you could sign in with the credentials of a non-admin user to see the enforcement of policies in action. In this article, let's create a test user account in the AKS cluster that you can use.

Create a sample namespace named *psp-aks* for test resources using the [kubectl create namespace](#) command. Then, create a service account named *nonadmin-user* using the [kubectl create serviceaccount](#) command:

```
kubectl create namespace psp-aks
kubectl create serviceaccount --namespace psp-aks nonadmin-user
```

Next, create a RoleBinding for the *nonadmin-user* to perform basic actions in the namespace using the [kubectl create rolebinding](#) command:

```
kubectl create rolebinding \
  --namespace psp-aks \
  psp-aks-editor \
  --clusterrole=edit \
  --serviceaccount=psp-aks:nonadmin-user
```

Create alias commands for admin and non-admin user

To highlight the difference between the regular admin user when using `kubectl` and the non-admin user created in the previous steps, create two command-line aliases:

- The `kubectl-admin` alias is for the regular admin user, and is scoped to the *psp-aks* namespace.
- The `kubectl-nonadminuser` alias is for the *nonadmin-user* created in the previous step, and is scoped to the *psp-aks* namespace.

Create these two aliases as shown in the following commands:

```
alias kubectl-admin='kubectl --namespace psp-aks'
alias kubectl-nonadminuser='kubectl --as=system:serviceaccount:psp-aks:nonadmin-user --namespace psp-aks'
```

Test the creation of a privileged pod

Let's first test what happens when you schedule a pod with the security context of `privileged: true`. This security context escalates the pod's privileges. In the previous section that showed the default AKS pod security policies, the *privilege* policy should deny this request.

Create a file named `nginx-privileged.yaml` and paste the following YAML manifest:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-privileged
spec:
  containers:
    - name: nginx-privileged
      image: mcr.microsoft.com/oss/nginx/nginx:1.14.2-alpine
      securityContext:
        privileged: true
```

Create the pod using the [kubectl apply](#) command and specify the name of your YAML manifest:

```
kubectl-nonadminuser apply -f nginx-privileged.yaml
```

The pod fails to be scheduled, as shown in the following example output:

```
$ kubectl-nonadminuser apply -f nginx-privileged.yaml

Error from server (Forbidden): error when creating "nginx-privileged.yaml": pods "nginx-privileged" is
forbidden: unable to validate against any pod security policy: []
```

The pod doesn't reach the scheduling stage, so there are no resources to delete before you move on.

Test creation of an unprivileged pod

In the previous example, the pod specification requested privileged escalation. This request is denied by the default *privilege* pod security policy, so the pod fails to be scheduled. Let's try now running that same NGINX pod without the privilege escalation request.

Create a file named `nginx-unprivileged.yaml` and paste the following YAML manifest:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-unprivileged
spec:
  containers:
    - name: nginx-unprivileged
      image: mcr.microsoft.com/oss/nginx/nginx:1.14.2-alpine
```

Create the pod using the [kubectl apply](#) command and specify the name of your YAML manifest:

```
kubectl-nonadminuser apply -f nginx-unprivileged.yaml
```

The pod fails to be scheduled, as shown in the following example output:

```
$ kubectl-nonadminuser apply -f nginx-unprivileged.yaml

Error from server (Forbidden): error when creating "nginx-unprivileged.yaml": pods "nginx-unprivileged" is
forbidden: unable to validate against any pod security policy: []
```

The pod doesn't reach the scheduling stage, so there are no resources to delete before you move on.

Test creation of a pod with a specific user context

In the previous example, the container image automatically tried to use root to bind NGINX to port 80. This request was denied by the default *privilege* pod security policy, so the pod fails to start. Let's try now running that same NGINX pod with a specific user context, such as `runAsUser: 2000`.

Create a file named `nginx-unprivileged-nonroot.yaml` and paste the following YAML manifest:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-unprivileged-nonroot
spec:
  containers:
    - name: nginx-unprivileged
      image: mcr.microsoft.com/oss/nginx/nginx:1.14.2-alpine
      securityContext:
        runAsUser: 2000
```

Create the pod using the [kubectl apply](#) command and specify the name of your YAML manifest:

```
kubectl-nonadminuser apply -f nginx-unprivileged-nonroot.yaml
```

The pod fails to be scheduled, as shown in the following example output:

```
$ kubectl-nonadminuser apply -f nginx-unprivileged-nonroot.yaml

Error from server (Forbidden): error when creating "nginx-unprivileged-nonroot.yaml": pods "nginx-
unprivileged-nonroot" is forbidden: unable to validate against any pod security policy: []
```

The pod doesn't reach the scheduling stage, so there are no resources to delete before you move on.

Create a custom pod security policy

Now that you've seen the behavior of the default pod security policies, let's provide a way for the *nonadmin-user* to successfully schedule pods.

Let's create a policy to reject pods that request privileged access. Other options, such as `runAsUser` or allowed `volumes`, aren't explicitly restricted. This type of policy denies a request for privileged access, but otherwise lets the cluster run the requested pods.

Create a file named `psp-deny-privileged.yaml` and paste the following YAML manifest:

```

apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: psp-deny-privileged
spec:
  privileged: false
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  runAsUser:
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
  volumes:
    - '*'

```

Create the policy using the [kubectl apply](#) command and specify the name of your YAML manifest:

```
kubectl apply -f psp-deny-privileged.yaml
```

To view the policies available, use the [kubectl get psp](#) command, as shown in the following example. Compare the *psp-deny-privileged* policy with the default *privilege* policy that was enforced in the previous examples to create a pod. Only the use of *PRIV* escalation is denied by your policy. There are no restrictions on the user or group for the *psp-deny-privileged* policy.

\$ kubectl get psp							
NAME	PRIV	CAPS	SELINUX	RUNASUSER	FSGROUP	SUPGROUP	READONLYROOTFS
VOLUMES							
privileged	true	*	RunAsAny	RunAsAny	RunAsAny	RunAsAny	false
*							
psp-deny-privileged	false		RunAsAny	RunAsAny	RunAsAny	RunAsAny	false
*							

Allow user account to use the custom pod security policy

In the previous step, you created a pod security policy to reject pods that request privileged access. To allow the policy to be used, you create a *Role* or a *ClusterRole*. Then, you associate one of these roles using a *RoleBinding* or *ClusterRoleBinding*.

For this example, create a ClusterRole that allows you to *use* the *psp-deny-privileged* policy created in the previous step. Create a file named `psp-deny-privileged-clusterrole.yaml` and paste the following YAML manifest:

```

kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: psp-deny-privileged-clusterrole
rules:
- apiGroups:
  - extensions
  resources:
  - podsecuritypolicies
  resourceNames:
  - psp-deny-privileged
  verbs:
  - use

```

Create the ClusterRole using the [kubectl apply](#) command and specify the name of your YAML manifest:

```
kubectl apply -f psp-deny-privileged-clusterrole.yaml
```

Now create a ClusterRoleBinding to use the ClusterRole created in the previous step. Create a file named [psp-deny-privileged-clusterrolebinding.yaml](#) and paste the following YAML manifest:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: psp-deny-privileged-clusterrolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: psp-deny-privileged-clusterrole
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:serviceaccounts
```

Create a ClusterRoleBinding using the [kubectl apply](#) command and specify the name of your YAML manifest:

```
kubectl apply -f psp-deny-privileged-clusterrolebinding.yaml
```

NOTE

In the first step of this article, the pod security policy feature was enabled on the AKS cluster. The recommended practice was to only enable the pod security policy feature after you've defined your own policies. This is the stage where you would enable the pod security policy feature. One or more custom policies have been defined, and user accounts have been associated with those policies. Now you can safely enable the pod security policy feature and minimize problems caused by the default policies.

Test the creation of an unprivileged pod again

With your custom pod security policy applied and a binding for the user account to use the policy, let's try to create an unprivileged pod again. Use the same [nginx-privileged.yaml](#) manifest to create the pod using the [kubectl apply](#) command:

```
kubectl-nonadminuser apply -f nginx-unprivileged.yaml
```

The pod is successfully scheduled. When you check the status of the pod using the [kubectl get pods](#) command, the pod is *Running*.

```
$ kubectl-nonadminuser get pods
NAME           READY   STATUS    RESTARTS   AGE
nginx-unprivileged   1/1     Running   0          7m14s
```

This example shows how you can create custom pod security policies to define access to the AKS cluster for different users or groups. The default AKS policies provide tight controls on what pods can run, so create your own custom policies to then correctly define the restrictions you need.

Delete the NGINX unprivileged pod using the [kubectl delete](#) command and specify the name of your YAML

manifest:

```
kubectl-nonadminuser delete -f nginx-unprivileged.yaml
```

Clean up resources

To disable pod security policy, use the [az aks update](#) command again. The following example disables pod security policy on the cluster name *myAKSCluster* in the resource group named *myResourceGroup*.

```
az aks update \
--resource-group myResourceGroup \
--name myAKSCluster \
--disable-pod-security-policy
```

Next, delete the ClusterRole and ClusterRoleBinding:

```
kubectl delete -f psp-deny-privileged-clusterrolebinding.yaml
kubectl delete -f psp-deny-privileged-clusterrole.yaml
```

Delete the security policy using [kubectl delete](#) command and specify the name of your YAML manifest:

```
kubectl delete -f psp-deny-privileged.yaml
```

Finally, delete the *psp-aks* namespace:

```
kubectl delete namespace psp-aks
```

Next steps

This article showed you how to create a pod security policy to prevent the use of privileged access. There are lots of features that a policy can enforce, such as type of volume or the RunAs user. For more information on the available options, see the [Kubernetes pod security policy reference docs](#).

For more information about limiting pod network traffic, see [Secure traffic between pods using network policies in AKS](#).

Secure access to the API server using authorized IP address ranges in Azure Kubernetes Service (AKS)

5/14/2021 • 6 minutes to read • [Edit Online](#)

In Kubernetes, the API server receives requests to perform actions in the cluster such as to create resources or scale the number of nodes. The API server is the central way to interact with and manage a cluster. To improve cluster security and minimize attacks, the API server should only be accessible from a limited set of IP address ranges.

This article shows you how to use API server authorized IP address ranges to limit which IP addresses and CIDRs can access control plane.

Before you begin

This article shows you how to create an AKS cluster using the Azure CLI.

You need the Azure CLI version 2.0.76 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Limitations

The API server Authorized IP ranges feature has the following limitations:

- On clusters created after API server authorized IP address ranges moved out of preview in October 2019, API server authorized IP address ranges are only supported on the *Standard* SKU load balancer. Existing clusters with the *Basic* SKU load balancer and API server authorized IP address ranges configured will continue work as is but cannot be migrated to a *Standard* SKU load balancer. Those existing clusters will also continue to work if their Kubernetes version or control plane are upgraded. API server authorized IP address ranges are not supported for private clusters.
- This feature is not compatible with clusters that use [Public IP per Node](#).

Overview of API server authorized IP ranges

The Kubernetes API server is how the underlying Kubernetes APIs are exposed. This component provides the interaction for management tools, such as `kubectl` or the Kubernetes dashboard. AKS provides a single-tenant cluster control plane, with a dedicated API server. By default, the API server is assigned a public IP address, and you should control access using Kubernetes role-based access control (Kubernetes RBAC) or Azure RBAC.

To secure access to the otherwise publicly accessible AKS control plane / API server, you can enable and use authorized IP ranges. These authorized IP ranges only allow defined IP address ranges to communicate with the API server. A request made to the API server from an IP address that isn't part of these authorized IP ranges is blocked. Continue to use Kubernetes RBAC or Azure RBAC to authorize users and the actions they request.

For more information about the API server and other cluster components, see [Kubernetes core concepts for AKS](#).

Create an AKS cluster with API server authorized IP ranges enabled

Create a cluster using the `az aks create` and specify the `--api-server-authorized-ip-ranges` parameter to provide a list of authorized IP address ranges. These IP address ranges are usually address ranges used by your on-premises networks or public IPs. When you specify a CIDR range, start with the first IP address in the range. For

example, `137.117.106.90/29` is a valid range, but make sure you specify the first IP address in the range, such as `137.117.106.88/29`.

IMPORTANT

By default, your cluster uses the [Standard SKU load balancer](#) which you can use to configure the outbound gateway. When you enable API server authorized IP ranges during cluster creation, the public IP for your cluster is also allowed by default in addition to the ranges you specify. If you specify "" or no value for `--api-server-authorized-ip-ranges`, API server authorized IP ranges will be disabled. Note that if you're using PowerShell, use `--api-server-authorized-ip-ranges=""` (with equals sign) to avoid any parsing issues.

The following example creates a single-node cluster named `myAKSCluster` in the resource group named `myResourceGroup` with API server authorized IP ranges enabled. The IP address ranges allowed are `73.140.245.0/24`:

```
az aks create \
  --resource-group myResourceGroup \
  --name myAKSCluster \
  --node-count 1 \
  --vm-set-type VirtualMachineScaleSets \
  --load-balancer-sku standard \
  --api-server-authorized-ip-ranges 73.140.245.0/24 \
  --generate-ssh-keys
```

NOTE

You should add these ranges to an allow list:

- The firewall public IP address
- Any range that represents networks that you'll administer the cluster from

The upper limit for the number of IP ranges you can specify is 200.

The rules can take up to 2min to propagate. Please allow up to that time when testing the connection.

Specify the outbound IPs for the Standard SKU load balancer

When creating an AKS cluster, if you specify the outbound IP addresses or prefixes for the cluster, those addresses or prefixes are allowed as well. For example:

```
az aks create \
  --resource-group myResourceGroup \
  --name myAKSCluster \
  --node-count 1 \
  --vm-set-type VirtualMachineScaleSets \
  --load-balancer-sku standard \
  --api-server-authorized-ip-ranges 73.140.245.0/24 \
  --load-balancer-outbound-ips <publicIpId1>,<publicIpId2> \
  --generate-ssh-keys
```

In the above example, all IPs provided in the parameter `--Load-balancer-outbound-ip-prefixes` are allowed along with the IPs in the `--api-server-authorized-ip-ranges` parameter.

Instead, you can specify the `--Load-balancer-outbound-ip-prefixes` parameter to allow outbound load balancer IP prefixes.

Allow only the outbound public IP of the Standard SKU load balancer

When you enable API server authorized IP ranges during cluster creation, the outbound public IP for the Standard SKU load balancer for your cluster is also allowed by default in addition to the ranges you specify. To allow only the outbound public IP of the Standard SKU load balancer, use `0.0.0.0/32` when specifying the `--api-server-authorized-ip-ranges` parameter.

In the following example, only the outbound public IP of the Standard SKU load balancer is allowed, and you can only access the API server from the nodes within the cluster.

```
az aks create \
    --resource-group myResourceGroup \
    --name myAKScluster \
    --node-count 1 \
    --vm-set-type VirtualMachineScaleSets \
    --load-balancer-sku standard \
    --api-server-authorized-ip-ranges 0.0.0.0/32 \
    --generate-ssh-keys
```

Update a cluster's API server authorized IP ranges

To update the API server authorized IP ranges on an existing cluster, use `az aks update` command and use the `--api-server-authorized-ip-ranges`, `--load-balancer-outbound-ip-prefixes*`, `--Load-balancer-outbound-ips`, or `--load-balancer-outbound-ip-prefixes*` parameters.

The following example updates API server authorized IP ranges on the cluster named `myAKScluster` in the resource group named `myResourceGroup`. The IP address range to authorize is `73.140.245.0/24`:

```
az aks update \
    --resource-group myResourceGroup \
    --name myAKScluster \
    --api-server-authorized-ip-ranges 73.140.245.0/24
```

You can also use `0.0.0.0/32` when specifying the `--api-server-authorized-ip-ranges` parameter to allow only the public IP of the Standard SKU load balancer.

Disable authorized IP ranges

To disable authorized IP ranges, use `az aks update` and specify an empty range to disable API server authorized IP ranges. For example:

```
az aks update \
    --resource-group myResourceGroup \
    --name myAKScluster \
    --api-server-authorized-ip-ranges ""
```

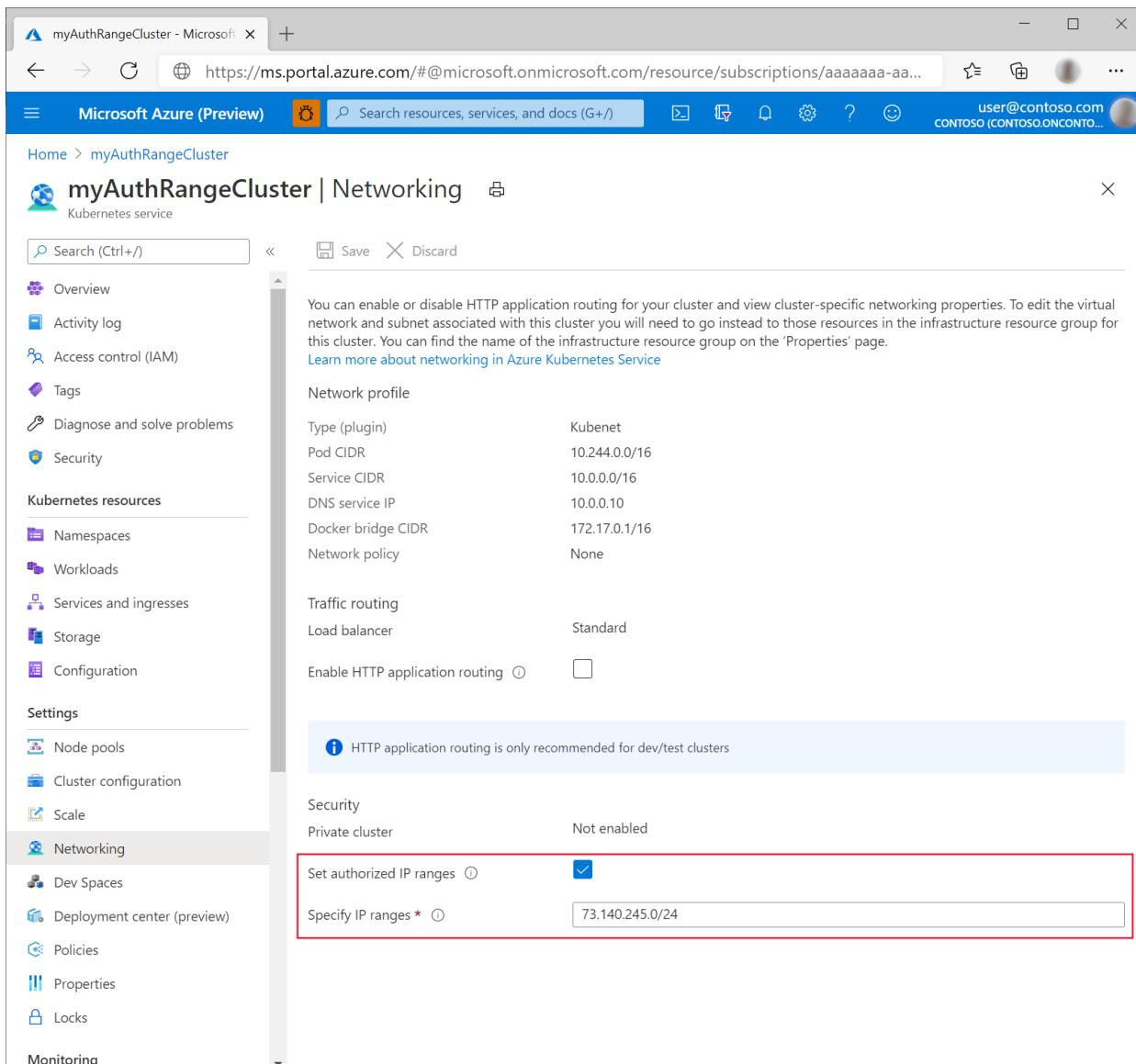
Find existing authorized IP ranges

To find IP ranges that have been authorized, use `az aks show` and specify the cluster's name and resource group. For example:

```
az aks show \
    --resource-group myResourceGroup \
    --name myAKScluster \
    --query apiServerAccessProfile.authorizedIpRanges'
```

Update, disable, and find authorized IP ranges using Azure portal

The above operations of adding, updating, finding, and disabling authorized IP ranges can also be performed in the Azure portal. To access, navigate to **Networking** under **Settings** in the menu blade of your cluster resource.



You can enable or disable HTTP application routing for your cluster and view cluster-specific networking properties. To edit the virtual network and subnet associated with this cluster you will need to go instead to those resources in the infrastructure resource group for this cluster. You can find the name of the infrastructure resource group on the 'Properties' page.
[Learn more about networking in Azure Kubernetes Service](#)

Network profile

Type (plugin)	Kubenet
Pod CIDR	10.244.0.0/16
Service CIDR	10.0.0.0/16
DNS service IP	10.0.0.10
Docker bridge CIDR	172.17.0.1/16
Network policy	None

Traffic routing

Load balancer	Standard
---------------	----------

Enable HTTP application routing

HTTP application routing is only recommended for dev/test clusters

Security

Private cluster	Not enabled
-----------------	-------------

Set authorized IP ranges

Specify IP ranges *

How to find my IP to include in `--api-server-authorized-ip-ranges` ?

You must add your development machines, tooling or automation IP addresses to the AKS cluster list of approved IP ranges in order to access the API server from there.

Another option is to configure a jumpbox with the needed tooling inside a separate subnet in the Firewall's virtual network. This assumes your environment has a Firewall with the respective network, and you have added the Firewall IPs to authorized ranges. Similarly, if you have forced tunneling from the AKS subnet to the Firewall subnet, than having the jumpbox in the cluster subnet is fine too.

Add another IP address to the approved ranges with the following command.

```
# Retrieve your IP address
CURRENT_IP=$(dig @resolver1.opendns.com ANY myip.opendns.com +short)
# Add to AKS approved list
az aks update -g $RG -n $AKSNAME --api-server-authorized-ip-ranges $CURRENT_IP/32
```

NOTE

The above example appends the API server authorized IP ranges on the cluster. To disable authorized IP ranges, use `az aks update` and specify an empty range "".

Another option is to use the below command on Windows systems to get the public IPv4 address, or you can use the steps in [Find your IP address](#).

```
Invoke-RestMethod http://ipinfo.io/json | Select -exp ip
```

You can also find this address by searching "what is my IP address" in an internet browser.

Next steps

In this article, you enabled API server authorized IP ranges. This approach is one part of how you can run a secure AKS cluster.

For more information, see [Security concepts for applications and clusters in AKS](#) and [Best practices for cluster security and upgrades in AKS](#).

Understand Azure Policy for Kubernetes clusters

5/13/2021 • 17 minutes to read • [Edit Online](#)

Azure Policy extends [Gatekeeper](#) v3, an *admission controller webhook* for [Open Policy Agent](#) (OPA), to apply at-scale enforcements and safeguards on your clusters in a centralized, consistent manner. Azure Policy makes it possible to manage and report on the compliance state of your Kubernetes clusters from one place. The add-on enacts the following functions:

- Checks with Azure Policy service for policy assignments to the cluster.
- Deploys policy definitions into the cluster as [constraint template](#) and [constraint](#) custom resources.
- Reports auditing and compliance details back to Azure Policy service.

Azure Policy for Kubernetes supports the following cluster environments:

- [Azure Kubernetes Service \(AKS\)](#)
- [Azure Arc enabled Kubernetes](#)
- [AKS Engine](#)

IMPORTANT

The add-ons for AKS Engine and Arc enabled Kubernetes are in [preview](#). Azure Policy for Kubernetes only supports Linux node pools and built-in policy definitions. Built-in policy definitions are in the **Kubernetes** category. The limited preview policy definitions with **EnforceOPAConstraint** and **EnforceRegoPolicy** effect and the related **Kubernetes Service** category are *deprecated*. Instead, use the effects *audit* and *deny* with Resource Provider mode [Microsoft.Kubernetes.Data](#).

Overview

To enable and use Azure Policy with your Kubernetes cluster, take the following actions:

1. Configure your Kubernetes cluster and install the add-on:

- [Azure Kubernetes Service \(AKS\)](#)
- [Azure Arc enabled Kubernetes](#)
- [AKS Engine](#)

NOTE

For common issues with installation, see [Troubleshoot - Azure Policy Add-on](#).

2. [Understand the Azure Policy language for Kubernetes](#)
3. [Assign a built-in definition to your Kubernetes cluster](#)
4. [Wait for validation](#)

Limitations

The following general limitations apply to the Azure Policy Add-on for Kubernetes clusters:

- Azure Policy Add-on for Kubernetes is supported on Kubernetes version **1.14** or higher.

- Azure Policy Add-on for Kubernetes can only be deployed to Linux node pools
- Only built-in policy definitions are supported
- Maximum number of Non-compliant records per policy per cluster: 500
- Maximum number of Non-compliant records per subscription: **1 million**
- Installations of Gatekeeper outside of the Azure Policy Add-on aren't supported. Uninstall any components installed by a previous Gatekeeper installation before enabling the Azure Policy Add-on.
- Reasons for non-compliance aren't available for the `Microsoft.Kubernetes.Data` Resource Provider mode. Use Component details.
- Exemptions aren't supported for Resource Provider modes.

The following limitations apply only to the Azure Policy Add-on for AKS:

- AKS Pod security policy and the Azure Policy Add-on for AKS can't both be enabled. For more information, see [AKS pod security limitation](#).
- Namespaces automatically excluded by Azure Policy Add-on for evaluation: *kube-system*, *gatekeeper-system*, and *aks-periscope*.

Recommendations

The following are general recommendations for using the Azure Policy Add-on:

- The Azure Policy Add-on requires three Gatekeeper components to run: 1 audit pod and 2 webhook pod replicas. These components consume more resources as the count of Kubernetes resources and policy assignments increases in the cluster, which requires audit and enforcement operations.
 - For fewer than 500 pods in a single cluster with a max of 20 constraints: 2 vCPUs and 350 MB memory per component.
 - For more than 500 pods in a single cluster with a max of 40 constraints: 3 vCPUs and 600 MB memory per component.
- Windows pods [don't support security contexts](#). Thus, some of the Azure Policy definitions, such as disallowing root privileges, can't be escalated in Windows pods and only apply to Linux pods.

The following recommendation applies only to AKS and the Azure Policy Add-on:

- Use system node pool with `criticalAddonsOnly` taint to schedule Gatekeeper pods. For more information, see [Using system node pools](#).
- Secure outbound traffic from your AKS clusters. For more information, see [Control egress traffic for cluster nodes](#).
- If the cluster has `aad-pod-identity` enabled, Node Managed Identity (NMI) pods modify the nodes' iptables to intercept calls to the Azure Instance Metadata endpoint. This configuration means any request made to the Metadata endpoint is intercepted by NMI even if the pod doesn't use `aad-pod-identity`.
AzurePodIdentityException CRD can be configured to inform `aad-pod-identity` that any requests to the Metadata endpoint originating from a pod that matches labels defined in CRD should be proxied without any processing in NMI. The system pods with `kubernetes.azure.com/managedby: aks` label in *kube-system* namespace should be excluded in `aad-pod-identity` by configuring the AzurePodIdentityException CRD. For more information, see [Disable aad-pod-identity for a specific pod or application](#). To configure an exception, install the [mic-exception YAML](#).

Install Azure Policy Add-on for AKS

Before installing the Azure Policy Add-on or enabling any of the service features, your subscription must enable the **Microsoft.PolicyInsights** resource providers.

1. You need the Azure CLI version 2.12.0 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install the Azure CLI](#).

2. Register the resource providers and preview features.

- Azure portal:

Register the **Microsoft.PolicyInsights** resource providers. For steps, see [Resource providers and types](#).

- Azure CLI:

```
# Log in first with az login if you're not using Cloud Shell  
  
# Provider register: Register the Azure Policy provider  
az provider register --namespace Microsoft.PolicyInsights
```

3. If limited preview policy definitions were installed, remove the add-on with the **Disable** button on your AKS cluster under the **Policies** page.

4. The AKS cluster must be version **1.14** or higher. Use the following script to validate your AKS cluster version:

```
# Log in first with az login if you're not using Cloud Shell  
  
# Look for the value in kubernetesVersion  
az aks list
```

5. Install version **2.12.0** or higher of the Azure CLI. For more information, see [Install the Azure CLI](#).

Once the above prerequisite steps are completed, install the Azure Policy Add-on in the AKS cluster you want to manage.

- Azure portal

1. Launch the AKS service in the Azure portal by selecting **All services**, then searching for and selecting **Kubernetes services**.
2. Select one of your AKS clusters.
3. Select **Policies** on the left side of the Kubernetes service page.
4. In the main page, select the **Enable add-on** button.

- Azure CLI

```
# Log in first with az login if you're not using Cloud Shell  
  
az aks enable-addons --addons azure-policy --name MyAKSCluster --resource-group MyResourceGroup
```

To validate that the add-on installation was successful and that the *azure-policy* and *gatekeeper* pods are running, run the following command:

```
# azure-policy pod is installed in kube-system namespace  
kubectl get pods -n kube-system  
  
# gatekeeper pod is installed in gatekeeper-system namespace  
kubectl get pods -n gatekeeper-system
```

Lastly, verify that the latest add-on is installed by running this Azure CLI command, replacing <rg> with your resource group name and <cluster-name> with the name of your AKS cluster:

az aks show --query addonProfiles.azurepolicy -g <rg> -n <cluster-name>. The result should look similar to the following output:

```
"addonProfiles": {  
    "azurepolicy": {  
        "enabled": true,  
        "identity": null  
    },  
}
```

Install Azure Policy Add-on for Azure Arc enabled Kubernetes (preview)

Before installing the Azure Policy Add-on or enabling any of the service features, your subscription must enable the **Microsoft.PolicyInsights** resource provider and create a role assignment for the cluster service principal.

1. You need the Azure CLI version 2.12.0 or later installed and configured. Run az --version to find the version. If you need to install or upgrade, see [Install the Azure CLI](#).
2. To enable the resource provider, follow the steps in [Resource providers and types](#) or run either the Azure CLI or Azure PowerShell command:

- Azure CLI

```
# Log in first with az login if you're not using Cloud Shell  
  
# Provider register: Register the Azure Policy provider  
az provider register --namespace 'Microsoft.PolicyInsights'
```

- Azure PowerShell

```
# Log in first with Connect-AzAccount if you're not using Cloud Shell  
  
# Provider register: Register the Azure Policy provider  
Register-AzResourceProvider -ProviderNamespace 'Microsoft.PolicyInsights'
```

3. The Kubernetes cluster must be version 1.14 or higher.
4. Install [Helm 3](#).
5. Your Kubernetes cluster enabled for Azure Arc. For more information, see [onboarding a Kubernetes cluster to Azure Arc](#).
6. Have the fully qualified Azure Resource ID of the Azure Arc enabled Kubernetes cluster.
7. Open ports for the add-on. The Azure Policy Add-on uses these domains and ports to fetch policy definitions and assignments and report compliance of the cluster back to Azure Policy.

DOMAIN	PORT
gov-prod-policy-data.trafficmanager.net	443
raw.githubusercontent.com	443

DOMAIN	PORT
login.windows.net	443
dc.services.visualstudio.com	443

8. Assign 'Policy Insights Data Writer (Preview)' role assignment to the Azure Arc enabled Kubernetes cluster. Replace <subscriptionId> with your subscription ID, <rg> with the Azure Arc enabled Kubernetes cluster's resource group, and <clusterName> with the name of the Azure Arc enabled Kubernetes cluster. Keep track of the returned values for *appId*, *password*, and *tenant* for the installation steps.

- Azure CLI

```
az ad sp create-for-rbac --role "Policy Insights Data Writer (Preview)" --scopes
"/subscriptions/<subscriptionId>/resourceGroups/<rg>/providers/Microsoft.Kubernetes/connectedC
lusters/<clusterName>"
```

- Azure PowerShell

```
$sp = New-AzADServicePrincipal -Role "Policy Insights Data Writer (Preview)" -Scope
"/subscriptions/<subscriptionId>/resourceGroups/<rg>/providers/Microsoft.Kubernetes/connectedC
lusters/<clusterName>"

@{ appId=$sp.ApplicationId;password=
[System.Runtime.InteropServices.Marshal]::PtrToStringAuto([System.Runtime.InteropServices.Mars
hal]::SecureStringToBSTR($sp.Secret));tenant=(Get-AzContext).Tenant.Id } | ConvertTo-Json
```

Sample output of the above commands:

```
{
  "appId": "aaaaaaaa-aaaa-aaaa-aaa-aaaaaaaaaa",
  "password": "bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbb",
  "tenant": "ccccccc-cccc-cccc-cccc-cccccccccc"
}
```

Once the above prerequisite steps are completed, install the Azure Policy Add-on in your Azure Arc enabled Kubernetes cluster:

1. Add the Azure Policy Add-on repo to Helm:

```
helm repo add azure-policy https://raw.githubusercontent.com/Azure/azure-
policy/master/extensions/policy-addon-kubernetes/helm-charts
```

2. Install the Azure Policy Add-on using Helm Chart:

```

# In below command, replace the following values with those gathered above.
#   <AzureArcClusterResourceId> with your Azure Arc enabled Kubernetes cluster resource Id. For example:
/subscriptions/<subscriptionId>/resourceGroups/<rg>/providers/Microsoft.Kubernetes/connectedClusters/<clusterName>
#   <ServicePrincipalAppId> with app Id of the service principal created during prerequisites.
#   <ServicePrincipalPassword> with password of the service principal created during prerequisites.
#   <ServicePrincipalTenantId> with tenant of the service principal created during prerequisites.
helm install azure-policy-addon azure-policy/azure-policy-addon-arc-clusters \
--set azureready.env.resourceid=<AzureArcClusterResourceId> \
--set azureready.env.clientid=<ServicePrincipalAppId> \
--set azureready.env.clientsecret=<ServicePrincipalPassword> \
--set azureready.env.tenantid=<ServicePrincipalTenantId>

```

For more information about what the add-on Helm Chart installs, see the [Azure Policy Add-on Helm Chart definition](#) on GitHub.

To validate that the add-on installation was successful and that the *azure-policy* and *gatekeeper* pods are running, run the following command:

```

# azure-policy pod is installed in kube-system namespace
kubectl get pods -n kube-system

# gatekeeper pod is installed in gatekeeper-system namespace
kubectl get pods -n gatekeeper-system

```

Install Azure Policy Add-on for AKS Engine (preview)

Before installing the Azure Policy Add-on or enabling any of the service features, your subscription must enable the **Microsoft.PolicyInsights** resource provider and create a role assignment for the cluster service principal.

1. You need the Azure CLI version 2.0.62 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install the Azure CLI](#).
2. To enable the resource provider, follow the steps in [Resource providers and types](#) or run either the Azure CLI or Azure PowerShell command:

- Azure CLI

```

# Log in first with az login if you're not using Cloud Shell

# Provider register: Register the Azure Policy provider
az provider register --namespace 'Microsoft.PolicyInsights'

```

- Azure PowerShell

```

# Log in first with Connect-AzAccount if you're not using Cloud Shell

# Provider register: Register the Azure Policy provider
Register-AzResourceProvider -ProviderNamespace 'Microsoft.PolicyInsights'

```

3. Create a role assignment for the cluster service principal.

- If you don't know the cluster service principal app ID, look it up with the following command.

```
# Get the kube-apiserver pod name  
kubectl get pods -n kube-system  
  
# Find the aadClientID value  
kubectl exec <kube-apiserver pod name> -n kube-system cat /etc/kubernetes/azure.json
```

- Assign 'Policy Insights Data Writer (Preview)' role assignment to the cluster service principal app ID (value *aadClientID* from previous step) with Azure CLI. Replace `<subscriptionId>` with your subscription ID and `<aks engine cluster resource group>` with the resource group the AKS Engine self-managed Kubernetes cluster is in.

```
az role assignment create --assignee <cluster service principal app ID> --scope  
"/subscriptions/<subscriptionId>/resourceGroups/<aks engine cluster resource group>" --role  
"Policy Insights Data Writer (Preview)"
```

Once the above prerequisite steps are completed, install the Azure Policy Add-on. The installation can be during the creation or update cycle of an AKS Engine or as an independent action on an existing cluster.

- Install during creation or update cycle

To enable the Azure Policy Add-on during the creation of a new self-managed cluster or as an update to an existing cluster, include the `addons` property cluster definition for AKS Engine.

```
"addons": [{}  
  "name": "azure-policy",  
  "enabled": true  
]
```

For more information about, see the external guide [AKS Engine cluster definition](#).

- Install in existing cluster with Helm Charts

Use the following steps to prepare the cluster and install the add-on:

1. Install [Helm 3](#).
2. Add the Azure Policy repo to Helm.

```
helm repo add azure-policy https://raw.githubusercontent.com/Azure/azure-  
policy/master/extensions/policy-addon-kubernetes/helm-charts
```

For more information, see [Helm Chart - Quickstart Guide](#).

3. Install the add-on with a Helm Chart. Replace `<subscriptionId>` with your subscription ID and `<aks engine cluster resource group>` with the resource group the AKS Engine self-managed Kubernetes cluster is in.

```
helm install azure-policy-addon azure-policy/azure-policy-addon-aks-engine --set  
azurepolicy.env.resourceid="/subscriptions/<subscriptionId>/resourceGroups/<aks engine cluster  
resource group>"
```

For more information about what the add-on Helm Chart installs, see the [Azure Policy Add-on Helm Chart definition](#) on GitHub.

NOTE

Because of the relationship between Azure Policy Add-on and the resource group ID, Azure Policy supports only one AKS Engine cluster for each resource group.

To validate that the add-on installation was successful and that the *azure-policy* and *gatekeeper* pods are running, run the following command:

```
# azure-policy pod is installed in kube-system namespace  
kubectl get pods -n kube-system  
  
# gatekeeper pod is installed in gatekeeper-system namespace  
kubectl get pods -n gatekeeper-system
```

Policy language

The Azure Policy language structure for managing Kubernetes follows that of existing policy definitions. With a **Resource Provider mode** of `Microsoft.Kubernetes.Data`, the effects `audit` and `deny` are used to manage your Kubernetes clusters. `Audit` and `deny` must provide `details` properties specific to working with [OPA Constraint Framework](#) and Gatekeeper v3.

As part of the `details.constraintTemplate` and `details.constraint` properties in the policy definition, Azure Policy passes the URIs of these [CustomResourceDefinitions](#) (CRD) to the add-on. Rego is the language that OPA and Gatekeeper support to validate a request to the Kubernetes cluster. By supporting an existing standard for Kubernetes management, Azure Policy makes it possible to reuse existing rules and pair them with Azure Policy for a unified cloud compliance reporting experience. For more information, see [What is Rego?](#).

Assign a built-in policy definition

To assign a policy definition to your Kubernetes cluster, you must be assigned the appropriate Azure role-based access control (Azure RBAC) policy assignment operations. The Azure built-in roles **Resource Policy Contributor** and **Owner** have these operations. To learn more, see [Azure RBAC permissions in Azure Policy](#).

Find the built-in policy definitions for managing your cluster using the Azure portal with the following steps:

1. Start the Azure Policy service in the Azure portal. Select **All services** in the left pane and then search for and select **Policy**.
2. In the left pane of the Azure Policy page, select **Definitions**.
3. From the Category dropdown list box, use **Select all** to clear the filter and then select **Kubernetes**.
4. Select the policy definition, then select the **Assign** button.
5. Set the **Scope** to the management group, subscription, or resource group of the Kubernetes cluster where the policy assignment will apply.

NOTE

When assigning the Azure Policy for Kubernetes definition, the **Scope** must include the cluster resource. For an AKS Engine cluster, the **Scope** must be the resource group of the cluster.

6. Give the policy assignment a **Name** and **Description** that you can use to identify it easily.
7. Set the **Policy enforcement** to one of the values below.

- **Enabled** - Enforce the policy on the cluster. Kubernetes admission requests with violations are denied.
- **Disabled** - Don't enforce the policy on the cluster. Kubernetes admission requests with violations aren't denied. Compliance assessment results are still available. When rolling out new policy definitions to running clusters, *Disabled* option is helpful for testing the policy definition as admission requests with violations aren't denied.

8. Select **Next**.

9. Set **parameter values**

- To exclude Kubernetes namespaces from policy evaluation, specify the list of namespaces in parameter **Namespace exclusions**. It's recommended to exclude: *kube-system*, *gatekeeper-system*, and *azure-arc*.

10. Select **Review + create**.

Alternately, use the [Assign a policy - Portal](#) quickstart to find and assign a Kubernetes policy. Search for a Kubernetes policy definition instead of the sample 'audit vms'.

IMPORTANT

Built-in policy definitions are available for Kubernetes clusters in category **Kubernetes**. For a list of built-in policy definitions, see [Kubernetes samples](#).

Policy evaluation

The add-on checks in with Azure Policy service for changes in policy assignments every 15 minutes. During this refresh cycle, the add-on checks for changes. These changes trigger creates, updates, or deletes of the constraint templates and constraints.

In a Kubernetes cluster, if a namespace has either of the following labels, the admission requests with violations aren't denied. Compliance assessment results are still available.

- `control-plane`
- `admission.policy.azure.com/ignore`

NOTE

While a cluster admin may have permission to create and update constraint templates and constraints resources install by the Azure Policy Add-on, these aren't supported scenarios as manual updates are overwritten. Gatekeeper continues to evaluate policies that existed prior to installing the add-on and assigning Azure Policy policy definitions.

Every 15 minutes, the add-on calls for a full scan of the cluster. After gathering details of the full scan and any real-time evaluations by Gatekeeper of attempted changes to the cluster, the add-on reports the results back to Azure Policy for inclusion in [compliance details](#) like any Azure Policy assignment. Only results for active policy assignments are returned during the audit cycle. Audit results can also be seen as [violations](#) listed in the status field of the failed constraint. For details on *Non-compliant* resources, see [Component details for Resource Provider modes](#).

NOTE

Each compliance report in Azure Policy for your Kubernetes clusters include all violations within the last 45 minutes. The timestamp indicates when a violation occurred.

Some other considerations:

- If the cluster subscription is registered with Azure Security Center, then Azure Security Center Kubernetes policies are applied on the cluster automatically.
- When a deny policy is applied on cluster with existing Kubernetes resources, any pre-existing resource that is not compliant with the new policy continues to run. When the non-compliant resource gets rescheduled on a different node the Gatekeeper blocks the resource creation.
- When a cluster has a deny policy that validates resources, the user will not see a rejection message when creating a deployment. For example, consider a Kubernetes deployment that contains replicsets and pods. When a user executes `kubectl describe deployment $MY_DEPLOYMENT`, it does not return a rejection message as part of events. However, `kubectl describe replicsets.apps $MY_DEPLOYMENT` returns the events associated with rejection.

Logging

As a Kubernetes controller/container, both the *azure-policy* and *gatekeeper* pods keep logs in the Kubernetes cluster. The logs can be exposed in the **Insights** page of the Kubernetes cluster. For more information, see [Monitor your Kubernetes cluster performance with Azure Monitor for containers](#).

To view the add-on logs, use `kubectl`:

```
# Get the azure-policy pod name installed in kube-system namespace
kubectl logs <azure-policy pod name> -n kube-system

# Get the gatekeeper pod name installed in gatekeeper-system namespace
kubectl logs <gatekeeper pod name> -n gatekeeper-system
```

For more information, see [Debugging Gatekeeper](#) in the Gatekeeper documentation.

Troubleshooting the add-on

For more information about troubleshooting the Add-on for Kubernetes, see the [Kubernetes section](#) of the Azure Policy troubleshooting article.

Remove the add-on

Remove the add-on from AKS

To remove the Azure Policy Add-on from your AKS cluster, use either the Azure portal or Azure CLI:

- Azure portal
 1. Launch the AKS service in the Azure portal by selecting **All services**, then searching for and selecting **Kubernetes services**.
 2. Select your AKS cluster where you want to disable the Azure Policy Add-on.
 3. Select **Policies** on the left side of the Kubernetes service page.
 4. In the main page, select the **Disable add-on** button.
- Azure CLI

```
# Log in first with az login if you're not using Cloud Shell

az aks disable-addons --addons azure-policy --name MyAKSCluster --resource-group MyResourceGroup
```

Remove the add-on from Azure Arc enabled Kubernetes

To remove the Azure Policy Add-on and Gatekeeper from your Azure Arc enabled Kubernetes cluster, run the following Helm command:

```
helm uninstall azure-policy-addon
```

Remove the add-on from AKS Engine

To remove the Azure Policy Add-on and Gatekeeper from your AKS Engine cluster, use the method that aligns with how the add-on was installed:

- If installed by setting the **addons** property in the cluster definition for AKS Engine:

Redeploy the cluster definition to AKS Engine after changing the **addons** property for *azure-policy* to false:

```
"addons": [{}  
  "name": "azure-policy",  
  "enabled": false  
]
```

For more information, see [AKS Engine - Disable Azure Policy Add-on](#).

- If installed with Helm Charts, run the following Helm command:

```
helm uninstall azure-policy-addon
```

Diagnostic data collected by Azure Policy Add-on

The Azure Policy Add-on for Kubernetes collects limited cluster diagnostic data. This diagnostic data is vital technical data related to software and performance. It's used in the following ways:

- Keep Azure Policy Add-on up to date
- Keep Azure Policy Add-on secure, reliable, performant
- Improve Azure Policy Add-on - through the aggregate analysis of the use of the add-on

The information collected by the add-on isn't personal data. The following details are currently collected:

- Azure Policy Add-on agent version
- Cluster type
- Cluster region
- Cluster resource group
- Cluster resource ID
- Cluster subscription ID
- Cluster OS (Example: Linux)
- Cluster city (Example: Seattle)
- Cluster state or province (Example: Washington)
- Cluster country or region (Example: United States)
- Exceptions/errors encountered by Azure Policy Add-on during agent installation on policy evaluation
- Number of Gatekeeper policy definitions not installed by Azure Policy Add-on

Next steps

- Review examples at [Azure Policy samples](#).
- Review the [Policy definition structure](#).
- Review [Understanding policy effects](#).
- Understand how to [programmatically create policies](#).
- Learn how to [get compliance data](#).
- Learn how to [remediate non-compliant resources](#).
- Review what a management group is with [Organize your resources with Azure management groups](#).

Update or rotate the credentials for Azure Kubernetes Service (AKS)

4/23/2021 • 4 minutes to read • [Edit Online](#)

AKS clusters created with a service principal have a one-year expiration time. As you near the expiration date, you can reset the credentials to extend the service principal for an additional period of time. You may also want to update, or rotate, the credentials as part of a defined security policy. This article details how to update these credentials for an AKS cluster.

You may also have [integrated your AKS cluster with Azure Active Directory](#), and use it as an authentication provider for your cluster. In that case you will have 2 more identities created for your cluster, the AAD Server App and the AAD Client App, you may also reset those credentials.

Alternatively, you can use a managed identity for permissions instead of a service principal. Managed identities are easier to manage than service principals and do not require updates or rotations. For more information, see [Use managed identities](#).

Before you begin

You need the Azure CLI version 2.0.65 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Update or create a new service principal for your AKS cluster

When you want to update the credentials for an AKS cluster, you can choose to either:

- Update the credentials for the existing service principal.
- Create a new service principal and update the cluster to use these new credentials.

WARNING

If you choose to create a *new* service principal, wait around 30 minutes for the service principal permission to propagate across all regions. Updating a large AKS cluster to use these credentials may take a long time to complete.

Check the expiration date of your service principal

To check the expiration date of your service principal, use the `az ad sp credential list` command. The following example gets the service principal ID for the cluster named *myAKSCluster* in the *myResourceGroup* resource group using the `az aks show` command. The service principal ID is set as a variable named *SP_ID* for use with the `az ad sp credential list` command.

```
SP_ID=$(az aks show --resource-group myResourceGroup --name myAKSCluster \
    --query servicePrincipalProfile.clientId -o tsv)
az ad sp credential list --id $SP_ID --query "[].endDate" -o tsv
```

Reset the existing service principal credential

To update the credentials for the existing service principal, get the service principal ID of your cluster using the `az aks show` command. The following example gets the ID for the cluster named *myAKSCluster* in the *myResourceGroup* resource group. The service principal ID is set as a variable named *SP_ID* for use in additional command. These commands use Bash syntax.

WARNING

When you reset your cluster credentials on an AKS cluster that uses Azure Virtual Machine Scale Sets, a [node image upgrade](#) is performed to update your nodes with the new credential information.

```
SP_ID=$(az aks show --resource-group myResourceGroup --name myAKSCluster \
--query servicePrincipalProfile.clientId -o tsv)
```

With a variable set that contains the service principal ID, now reset the credentials using [az ad sp credential reset](#). The following example lets the Azure platform generate a new secure secret for the service principal. This new secure secret is also stored as a variable.

```
SP_SECRET=$(az ad sp credential reset --name $SP_ID --query password -o tsv)
```

Now continue on to [update AKS cluster with new service principal credentials](#). This step is necessary for the Service Principal changes to reflect on the AKS cluster.

Create a new service principal

If you chose to update the existing service principal credentials in the previous section, skip this step. Continue to [update AKS cluster with new service principal credentials](#).

To create a service principal and then update the AKS cluster to use these new credentials, use the [az ad sp create-for-rbac](#) command. In the following example, the `--skip-assignment` parameter prevents any additional default assignments being assigned:

```
az ad sp create-for-rbac --skip-assignment
```

The output is similar to the following example. Make a note of your own `appId` and `password`. These values are used in the next step.

```
{
  "appId": "7d837646-b1f3-443d-874c-fd83c7c739c5",
  "name": "7d837646-b1f3-443d-874c-fd83c7c739c",
  "password": "a5ce83c9-9186-426d-9183-614597c7f2f7",
  "tenant": "a4342dc8-cd0e-4742-a467-3129c469d0e5"
}
```

Now define variables for the service principal ID and client secret using the output from your own [az ad sp create-for-rbac](#) command, as shown in the following example. The `SP_ID` is your `appId`, and the `SP_SECRET` is your `password`.

```
SP_ID=7d837646-b1f3-443d-874c-fd83c7c739c5
SP_SECRET=a5ce83c9-9186-426d-9183-614597c7f2f7
```

Now continue on to [update AKS cluster with new service principal credentials](#). This step is necessary for the Service Principal changes to reflect on the AKS cluster.

Update AKS cluster with new service principal credentials

IMPORTANT

For large clusters, updating the AKS cluster with a new service principal may take a long time to complete.

Regardless of whether you chose to update the credentials for the existing service principal or create a service principal, you now update the AKS cluster with your new credentials using the [az aks update-credentials](#) command. The variables for the *--service-principal* and *--client-secret* are used:

```
az aks update-credentials \
--resource-group myResourceGroup \
--name myAKScluster \
--reset-service-principal \
--service-principal $SP_ID \
--client-secret $SP_SECRET
```

For small and midsize clusters, it takes a few moments for the service principal credentials to be updated in the AKS.

Update AKS Cluster with new AAD Application credentials

You may create new AAD Server and Client applications by following the [AAD integration steps](#). Or reset your existing AAD Applications following the [same method as for service principal reset](#). After that you just need to update your cluster AAD Application credentials using the same [az aks update-credentials](#) command but using the *--reset-aad* variables.

```
az aks update-credentials \
--resource-group myResourceGroup \
--name myAKScluster \
--reset-aad \
--aad-server-app-id <SERVER APPLICATION ID> \
--aad-server-app-secret <SERVER APPLICATION SECRET> \
--aad-client-app-id <CLIENT APPLICATION ID>
```

Next steps

In this article, the service principal for the AKS cluster itself and the AAD Integration Applications were updated. For more information on how to manage identity for workloads within a cluster, see [Best practices for authentication and authorization in AKS](#).

AKS-managed Azure Active Directory integration

5/19/2021 • 11 minutes to read • [Edit Online](#)

AKS-managed Azure AD integration is designed to simplify the Azure AD integration experience, where users were previously required to create a client app, a server app, and required the Azure AD tenant to grant Directory Read permissions. In the new version, the AKS resource provider manages the client and server apps for you.

Azure AD authentication overview

Cluster administrators can configure Kubernetes role-based access control (Kubernetes RBAC) based on a user's identity or directory group membership. Azure AD authentication is provided to AKS clusters with OpenID Connect. OpenID Connect is an identity layer built on top of the OAuth 2.0 protocol. For more information on OpenID Connect, see the [Open ID connect documentation](#).

Learn more about the Azure AD integration flow on the [Azure Active Directory integration concepts documentation](#).

Limitations

- AKS-managed Azure AD integration can't be disabled
- Changing a AKS-managed Azure AD integrated cluster to legacy AAD is not supported
- non-Kubernetes RBAC enabled clusters aren't supported for AKS-managed Azure AD integration
- Changing the Azure AD tenant associated with AKS-managed Azure AD integration isn't supported

Prerequisites

- The Azure CLI version 2.11.0 or later
- Kubectl with a minimum version of [1.18.1](#) or [kubelogin](#)
- If you are using [helm](#), minimum version of helm 3.3.

IMPORTANT

You must use Kubectl with a minimum version of 1.18.1 or kubelogin. The difference between the minor versions of Kubernetes and kubectl should not be more than 1 version. If you don't use the correct version, you will notice authentication issues.

To install kubectl and kubelogin, use the following commands:

```
sudo az aks install-cli  
kubectl version --client  
kubelogin --version
```

Use [these instructions](#) for other operating systems.

Before you begin

For your cluster, you need an Azure AD group. This group is needed as admin group for the cluster to grant cluster admin permissions. You can use an existing Azure AD group, or create a new one. Record the object ID of

your Azure AD group.

```
# List existing groups in the directory
az ad group list --filter "displayname eq '<group-name>'" -o table
```

To create a new Azure AD group for your cluster administrators, use the following command:

```
# Create an Azure AD group
az ad group create --display-name myAKSAdminGroup --mail-nickname myAKSAdminGroup
```

Create an AKS cluster with Azure AD enabled

Create an AKS cluster by using the following CLI commands.

Create an Azure resource group:

```
# Create an Azure resource group
az group create --name myResourceGroup --location centralus
```

Create an AKS cluster, and enable administration access for your Azure AD group

```
# Create an AKS-managed Azure AD cluster
az aks create -g myResourceGroup -n myManagedCluster --enable-aad --aad-admin-group-object-ids <id> [--aad-tenant-id <id>]
```

A successful creation of an AKS-managed Azure AD cluster has the following section in the response body

```
"AADProfile": {
    "adminGroupObjectIds": [
        "5d24****-****-****-****-****afa27aed"
    ],
    "clientAppId": null,
    "managed": true,
    "serverAppId": null,
    "serverAppSecret": null,
    "tenantId": "72f9****-****-****-****-****d011db47"
}
```

Once the cluster is created, you can start accessing it.

Access an Azure AD enabled cluster

You'll need the [Azure Kubernetes Service Cluster User](#) built-in role to do the following steps.

Get the user credentials to access the cluster:

```
az aks get-credentials --resource-group myResourceGroup --name myManagedCluster
```

Follow the instructions to sign in.

Use the kubectl get nodes command to view nodes in the cluster:

```
kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
aks-nodepool1-15306047-0	Ready	agent	102m	v1.15.10
aks-nodepool1-15306047-1	Ready	agent	102m	v1.15.10
aks-nodepool1-15306047-2	Ready	agent	102m	v1.15.10

Configure [Azure role-based access control \(Azure RBAC\)](#) to configure additional security groups for your clusters.

Troubleshooting access issues with Azure AD

IMPORTANT

The steps described below are bypassing the normal Azure AD group authentication. Use them only in an emergency.

If you're permanently blocked by not having access to a valid Azure AD group with access to your cluster, you can still obtain the admin credentials to access the cluster directly.

To do these steps, you'll need to have access to the [Azure Kubernetes Service Cluster Admin](#) built-in role.

```
az aks get-credentials --resource-group myResourceGroup --name myManagedCluster --admin
```

Enable AKS-managed Azure AD Integration on your existing cluster

You can enable AKS-managed Azure AD Integration on your existing Kubernetes RBAC enabled cluster. Ensure to set your admin group to keep access on your cluster.

```
az aks update -g MyResourceGroup -n MyManagedCluster --enable-aad --aad-admin-group-object-ids <id-1> [--aad-tenant-id <id>]
```

A successful activation of an AKS-managed Azure AD cluster has the following section in the response body

```
"AADProfile": {  
    "adminGroupObjectIds": [  
        "5d24****-****-****-****-****afa27aed"  
    ],  
    "clientAppId": null,  
    "managed": true,  
    "serverAppId": null,  
    "serverAppSecret": null,  
    "tenantId": "72f9****-****-****-****-****d011db47"  
}
```

Download user credentials again to access your cluster by following the steps [here](#).

Upgrading to AKS-managed Azure AD Integration

If your cluster uses legacy Azure AD integration, you can upgrade to AKS-managed Azure AD Integration.

```
az aks update -g myResourceGroup -n myManagedCluster --enable-aad --aad-admin-group-object-ids <id> [--aad-tenant-id <id>]
```

A successful migration of an AKS-managed Azure AD cluster has the following section in the response body

```
"AADProfile": {  
    "adminGroupObjectIds": [  
        "5d24****-****-****-****-****afa27aed"  
    ],  
    "clientAppId": null,  
    "managed": true,  
    "serverAppId": null,  
    "serverAppSecret": null,  
    "tenantId": "72f9****-****-****-****-****d011db47"  
}
```

If you want to access the cluster, follow the steps [here](#).

Non-interactive sign in with kubelogin

There are some non-interactive scenarios, such as continuous integration pipelines, that aren't currently available with `kubectl`. You can use `kubelogin` to access the cluster with non-interactive service principal sign-in.

Disable local accounts (preview)

When deploying an AKS Cluster, local accounts are enabled by default. Even when enabling RBAC or Azure Active Directory integration, `--admin` access still exists, essentially as a non-auditable backdoor option. With this in mind, AKS offers users the ability to disable local accounts via a flag, `disable-local`. A field, `properties.disableLocalAccounts`, has also been added to the managed cluster API to indicate whether the feature has been enabled on the cluster.

NOTE

On clusters with Azure AD integration enabled, users belonging to a group specified by `aad-admin-group-object-ids` will still be able to gain access via non-admin credentials. On clusters without Azure AD integration enabled and `properties.disableLocalAccounts` set to true, obtaining both user and admin credentials will fail.

Register the `DisableLocalAccountsPreview` preview feature

IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

To use an AKS cluster without local accounts, you must enable the `DisableLocalAccountsPreview` feature flag on your subscription. Ensure you are using the latest version of the Azure CLI and the `aks-preview` extension.

Register the `DisableLocalAccountsPreview` feature flag using the `az feature register` command as shown in the following example:

```
az feature register --namespace "Microsoft.ContainerService" --name "DisableLocalAccountsPreview"
```

It takes a few minutes for the status to show *Registered*. You can check on the registration status using the `az`

[feature list](#) command:

```
az feature list -o table --query "[?contains(name, 'Microsoft.ContainerService/DisableLocalAccountsPreview')].{Name:name,State:properties.state}"
```

When ready, refresh the registration of the *Microsoft.ContainerService* resource provider using the [az provider register](#) command:

```
az provider register --namespace Microsoft.ContainerService
```

Create a new cluster without local accounts

To create a new AKS cluster without any local accounts, use the [az aks create](#) command with the `--disable-local` flag:

```
az aks create -g <resource-group> -n <cluster-name> --enable-aad --aad-admin-group-object-ids <aad-group-id> --disable-local
```

In the output, confirm local accounts have been disabled by checking the field `properties.disableLocalAccounts` is set to true:

```
"properties": {  
    ...  
    "disableLocalAccounts": true,  
    ...  
}
```

Attempting to get admin credentials will fail with an error message indicating the feature is preventing access:

```
az aks get-credentials --resource-group <resource-group> --name <cluster-name> --admin  
  
Operation failed with status: 'Bad Request'. Details: Getting static credential is not allowed because this cluster is set to disable local accounts.
```

Disable local accounts on an existing cluster

To disable local accounts on an existing AKS cluster, use the [az aks update](#) command with the `--disable-local` flag:

```
az aks update -g <resource-group> -n <cluster-name> --enable-aad --aad-admin-group-object-ids <aad-group-id> --disable-local
```

In the output, confirm local accounts have been disabled by checking the field `properties.disableLocalAccounts` is set to true:

```
"properties": {  
    ...  
    "disableLocalAccounts": true,  
    ...  
}
```

Attempting to get admin credentials will fail with an error message indicating the feature is preventing access:

```
az aks get-credentials --resource-group <resource-group> --name <cluster-name> --admin

Operation failed with status: 'Bad Request'. Details: Getting static credential is not allowed because this
cluster is set to disable local accounts.
```

Re-enable local accounts on an existing cluster

AKS also offers the ability to re-enable local accounts on an existing cluster with the `enable-local` flag:

```
az aks update -g <resource-group> -n <cluster-name> --enable-aad --aad-admin-group-object-ids <aad-group-id>
--enable-local
```

In the output, confirm local accounts have been re-enabled by checking the field `properties.disableLocalAccounts` is set to false:

```
"properties": {
    ...
    "disableLocalAccounts": false,
    ...
}
```

Attempting to get admin credentials will succeed:

```
az aks get-credentials --resource-group <resource-group> --name <cluster-name> --admin

Merged "<cluster-name>-admin" as current context in C:\Users\<username>\.kube\config
```

Use Conditional Access with Azure AD and AKS

When integrating Azure AD with your AKS cluster, you can also use [Conditional Access](#) to control access to your cluster.

NOTE

Azure AD Conditional Access is an Azure AD Premium capability.

To create an example Conditional Access policy to use with AKS, complete the following steps:

1. At the top of the Azure portal, search for and select Azure Active Directory.
2. In the menu for Azure Active Directory on the left-hand side, select *Enterprise applications*.
3. In the menu for Enterprise applications on the left-hand side, select *Conditional Access*.
4. In the menu for Conditional Access on the left-hand side, select *Policies* then *New policy*.

The screenshot shows the Microsoft Azure Conditional Access Policies page. At the top, there's a navigation bar with icons for home, back, forward, and search, followed by the URL 'portal.azure.com'. Below the navigation is a header bar with the Microsoft Azure logo, a search bar containing 'Search resources, services, and docs (G+/)', and a user profile icon.

The main title is 'Conditional Access | Policies' under 'Azure Active Directory'. Below the title, there are several buttons: '+ New policy' (which is highlighted with a red box), 'What If', 'Refresh', and 'Got feedback?'. To the left, there's a sidebar with sections like 'Policies' (selected), 'Insights and reporting', and 'Diagnose and solve problems'. Under 'Manage', there are links for 'Named locations', 'Custom controls (Preview)', 'Terms of use', 'VPN connectivity', and 'Classic policies'. Under 'Troubleshooting + Support', there are links for 'Virtual assistant (Preview)' and 'New support request'.

5. Enter a name for the policy such as *aks-policy*.
6. Select *Users and groups*, then under *Include* select *Select users and groups*. Choose the users and groups where you want to apply the policy. For this example, choose the same Azure AD group that has administration access to your cluster.

New

Conditional access policy

Control user access based on conditional access policy to bring signals together, to make decisions, and enforce organizational policies. [Learn more](#)

Name *

Example: 'Device compliance app policy'

Assignments

Users and groups ⓘ



Specific users included

Cloud apps or actions ⓘ



No cloud apps or actions selected

Conditions ⓘ



0 conditions selected

Access controls

Grant ⓘ



0 controls selected

Session ⓘ



0 controls selected

Control user access based on users and groups assignment for all users, specific groups of users, directory roles, or external guest users

[Learn more](#)

Include **Exclude**

None

All users

Select users and groups

All guest and external users ⓘ

Directory roles ⓘ

Users and groups

Select



1 group

MY

myAKSAdminGroup

...

7. Select *Cloud apps or actions*, then under *Include* select *Select apps*. Search for *Azure Kubernetes Service* and select *Azure Kubernetes Service AAD Server*.

New

Conditional access policy

Control user access based on conditional access policy to bring signals together, to make decisions, and enforce organizational policies. [Learn more](#)

Name *

Example: 'Device compliance app policy'

Assignments

Users and groups ⓘ



Specific users included

Cloud apps or actions ⓘ



1 app included

Conditions ⓘ



0 conditions selected

Access controls

Grant ⓘ



0 controls selected

Session ⓘ



0 controls selected

Control user access based on all or specific cloud apps or actions. [Learn more](#)

Select what this policy applies to

[Cloud apps](#)[User actions](#)
[Include](#) [Exclude](#)
 None All cloud apps Select apps

Select



Azure Kubernetes Service AAD S...

AK

Azure Kubernetes Service AAD ! ...
6dae42f8-4368-4678-94ff-3960e28e36...

8. Under *Access controls*, select *Grant*. Select *Grant access* then *Require device to be marked as compliant*.

New

Conditional access policy

Control user access based on conditional access policy to bring signals together, to make decisions, and enforce organizational policies. [Learn more](#)

Name *

Example: 'Device compliance app policy'

Assignments

Users and groups ⓘ



Specific users included

Cloud apps or actions ⓘ



1 app included

Conditions ⓘ



0 conditions selected

Access controls

Grant ⓘ



0 controls selected

Session ⓘ



0 controls selected

Grant



Control user access enforcement to block or grant access. [Learn more](#)

 Block access Grant access Require multi-factor authentication ⓘ Require device to be marked as compliant ⓘ Require Hybrid Azure AD joined device ⓘ Require approved client app ⓘ
[See list of approved client apps](#) Require app protection policy ⓘ
[See list of policy protected client apps](#) Require password change ⓘ

For multiple controls

 Require all the selected controls Require one of the selected controls
 Don't lock yourself out! Make sure that your device is compliant.

9. Under *Enable policy*, select *On* then *Create*.

Home > Default Directory > Enterprise applications > Conditional Access >

New

Conditional access policy

Control user access based on conditional access policy to bring signals together, to make decisions, and enforce organizational policies. [Learn more](#)

Name *

Example: 'Device compliance app policy'

Assignments

Users and groups ⓘ >

Specific users included

Cloud apps or actions ⓘ >

1 app included

Conditions ⓘ >

0 conditions selected

Access controls

Grant ⓘ >

1 control selected

Session ⓘ >

0 controls selected

Enable policy

Report-only On Off

Create

Get the user credentials to access the cluster, for example:

```
az aks get-credentials --resource-group myResourceGroup --name myManagedCluster
```

Follow the instructions to sign in.

Use the `kubectl get nodes` command to view nodes in the cluster:

```
kubectl get nodes
```

Follow the instructions to sign in again. Notice there is an error message stating you are successfully logged in, but your admin requires the device requesting access to be managed by your Azure AD to access the resource.

In the Azure portal, navigate to Azure Active Directory, select *Enterprise applications* then under *Activity* select *Sign-ins*. Notice an entry at the top with a *Status* of *Failed* and a *Conditional Access* of *Success*. Select the entry then select *Conditional Access* in *Details*. Notice your Conditional Access policy is listed.

Details							
Basic info	Location	Device info	Authentication Details	Conditional Access	Report-only	Additional Details	Troubleshooting and support
Policy Name ↑↓	Grant Controls ↑↓		Session Controls ↑↓		Result ↑↓		
aks-policy	require compliant device				Failure		
A sign-in can also be interrupted (e.g. blocked, MFA challenged) because of a user risk policy or sign-in risk policy. Currently, this tab only lists Conditional Access policies.							

Configure just-in-time cluster access with Azure AD and AKS

Another option for cluster access control is to use Privileged Identity Management (PIM) for just-in-time requests.

NOTE

PIM is an Azure AD Premium capability requiring a Premium P2 SKU. For more on Azure AD SKUs, see the [pricing guide](#).

To integrate just-in-time access requests with an AKS cluster using AKS-managed Azure AD integration, complete the following steps:

1. At the top of the Azure portal, search for and select Azure Active Directory.
2. Take note of the Tenant ID, referred to for the rest of these instructions as `<tenant-id>`

The screenshot shows the Azure Active Directory Overview page. On the left, there's a navigation menu with sections like Overview, Getting started, Preview hub, and Diagnose and solve problems. Under Manage, there are links for Users, Groups, External Identities, Roles and administrators, Administrative units, Enterprise applications, Devices, App registrations, Identity Governance, Application proxy, Licenses, Azure AD Connect, Custom domain names, and Mobility (MDM and MAM). The main content area has a section titled 'Default Directory' with a search bar. Below it are two boxes: 'Tenant information' (containing Your role: Global administrator, License: Azure AD Premium P2, Tenant ID: 4ee91c09-89de-4e00-8214-515f..., Primary domain: user.contoso.com) and 'Azure AD Connect' (Status: Not enabled, Last sync: Sync has never run). A red box highlights the Tenant ID field.

3. In the menu for Azure Active Directory on the left-hand side, under *Manage* select *Groups* then *New Group*.

The screenshot shows the 'Groups | All groups (Preview)' page. The left sidebar includes links for All groups (Preview), Deleted groups, and Diagnose and solve problems. Under Settings, there are General, Expiration, and Naming policy options. Activity sections include Privileged access groups (Preview), Access reviews, Audit logs, and Bulk operation results. Troubleshooting + Support includes a New support request link. At the top, there's a toolbar with a 'New group' button (highlighted with a red box), Download groups, Delete, Refresh, Columns, Preview features, and more. A message indicates preview features are available. The main area shows a table with columns: Name, Object Id, Group Type, Membership Ty..., Email, and Source. The table displays '0 groups found' and 'No groups found'.

4. Make sure a Group Type of *Security* is selected and enter a group name, such as *myJITGroup*. Under *Azure AD Roles can be assigned to this group (Preview)*, select *Yes*. Finally, select *Create*.

Microsoft Azure Search resources, services, and docs (G+)

Home > Default Directory > Groups > New Group

Group type * ①
Security

Group name * ①
myJITGroup

Group description ①
Enter a description for the group

Azure AD roles can be assigned to the group (Preview) ①
 Yes No

Membership type ①
Assigned

Owners
No owners selected

Members
No members selected

Roles
No roles selected

Create

5. You will be brought back to the *Groups* page. Select your newly created group and take note of the Object ID, referred to for the rest of these instructions as `<object-id>`.

Microsoft Azure Search resources, services, and docs (G+)

Home > myJITGroup

myJITGroup Group

Overview (Preview)

Diagnose and solve problems

Manage

- Properties
- Members (Preview)
- Owners (Preview)
- Administrative units
- Group memberships (Preview)
- Assigned roles (Preview)
- Applications
- Licenses
- Azure role assignments

Activity

- Privileged access (Preview)
- Access reviews
- Audit logs
- Bulk operation results

Troubleshooting + Support

New support request

MY myJITGroup

Membership type: Assigned

Source: Cloud

Type: Security

Object Id: b91afa54-4717-41ef-912a-a0ace8e42e76

Creation date: 2/17/2021, 2:10:34 PM

Direct members: 0 Total, 0 User(s), 0 Group(s), 0 Device(s), 0 Other(s)

Group memberships: 0

Owners: 0

Total members: 0

6. Deploy an AKS cluster with AKS-managed Azure AD integration by using the `<tenant-id>` and `<object-id>` values from earlier:

```
az aks create -g myResourceGroup -n myManagedCluster --enable-aad --aad-admin-group-object-ids
<object-id> --aad-tenant-id <tenant-id>
```

7. Back in the Azure portal, in the menu for *Activity* on the left-hand side, select *Privileged Access (Preview)* and

select *Enable Privileged Access*.

The screenshot shows the Microsoft Azure Groups page for a group named "myJITGroup". On the left, there's a sidebar with navigation links like "Properties", "Members (Preview)", "Owners (Preview)", etc. The main area has a large "Enable Privileged Access" button with a clock icon. Below it, there's a section titled "Enable Privileged Access" with a brief description: "Privileged Access Groups enable just-in-time (JIT) access to the Owner or Member role of this group. JIT access by Azure AD PIM provides enhanced security for owners with delegated administrative tasks." There are also two callout boxes: one for "Learn more about Privileged Access Groups" and another for "Activate more in less time".

8. Select *Add Assignments* to begin granting access.

The screenshot shows the Microsoft Azure Groups page for "myJITGroup". The "Add assignments" button is highlighted with a red box. Below it, there are tabs for "Eligible assignments", "Active assignments" (which is selected), and "Expired assignments". A search bar allows filtering by member name or principal name. The table below shows no results found.

Name	Principal name	Type	Membership	State	Start time
No results					

9. Select a role of *member*, and select the users and groups to whom you wish to grant cluster access. These assignments can be modified at any time by a group admin. When you're ready to move on, select *Next*.

Microsoft Azure Search resources, services, and docs (G+) Home > Default Directory > Groups > myJITGroup >

Add assignments

Privileged Identity Management | Privileged access groups (Preview)

Membership **Setting**

Resource
myJITGroup

Resource type
Security

Select role ⓘ
Member

Select member(s) * ⓘ
1 Member(s) selected

Selected member(s) ⓘ

 Test User
user_contoso.com#EXT#@user.contoso.com Remove

Next > **Cancel**

10. Choose an assignment type of *Active*, the desired duration, and provide a justification. When you're ready to proceed, select *Assign*. For more on assignment types, see [Assign eligibility for a privileged access group \(preview\)](#) in Privileged Identity Management.

Microsoft Azure Search resources, services, and docs (G+) Home > Default Directory > Groups > myJITGroup >

Add assignments

Privileged Identity Management | Privileged access groups (Preview)

Membership **Setting**

Assignment type ⓘ
 Eligible
 Active

Maximum allowed assignment duration is 6 month(s).

Assignment starts *
02/17/2021 3:34:59 PM

Assignment ends *
08/16/2021 4:34:59 PM

Enter justification *
AKS cluster access

Assign **< Prev** **Cancel**

Once the assignments have been made, verify just-in-time access is working by accessing the cluster. For example:

```
az aks get-credentials --resource-group myResourceGroup --name myManagedCluster
```

Follow the steps to sign in.

Use the `kubectl get nodes` command to view nodes in the cluster:

```
kubectl get nodes
```

Note the authentication requirement and follow the steps to authenticate. If successful, you should see output similar to the following:

```
To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code  
AAAAAAA to authenticate.
```

NAME	STATUS	ROLES	AGE	VERSION
aks-nodepool1-61156405-vmss000000	Ready	agent	6m36s	v1.18.14
aks-nodepool1-61156405-vmss000001	Ready	agent	6m42s	v1.18.14
aks-nodepool1-61156405-vmss000002	Ready	agent	6m33s	v1.18.14

Troubleshooting

If `kubectl get nodes` returns an error similar to the following:

```
Error from server (Forbidden): nodes is forbidden: User "aaaa11111-11aa-aa11-a1a1-111111aaaa" cannot list  
resource "nodes" in API group "" at the cluster scope
```

Make sure the admin of the security group has given your account an *Active* assignment.

Next steps

- Learn about [Azure RBAC integration for Kubernetes Authorization](#)
- Learn about [Azure AD integration with Kubernetes RBAC](#).
- Use [kubelogin](#) to access features for Azure authentication that aren't available in kubectl.
- Learn more about [AKS and Kubernetes identity concepts](#).
- Use [Azure Resource Manager \(ARM\) templates](#) to create AKS-managed Azure AD enabled clusters.

Integrate Azure Active Directory with Azure Kubernetes Service using the Azure CLI (legacy)

4/21/2021 • 8 minutes to read • [Edit Online](#)

Azure Kubernetes Service (AKS) can be configured to use Azure Active Directory (AD) for user authentication. In this configuration, you can log into an AKS cluster using an Azure AD authentication token. Cluster operators can also configure Kubernetes role-based access control (Kubernetes RBAC) based on a user's identity or directory group membership.

This article shows you how to create the required Azure AD components, then deploy an Azure AD-enabled cluster and create a basic Kubernetes role in the AKS cluster.

For the complete sample script used in this article, see [Azure CLI samples - AKS integration with Azure AD](#).

IMPORTANT

AKS has a new improved [AKS-managed Azure AD](#) experience that doesn't require you to manage server or client application. If you want to migrate follow the instructions [here](#).

The following limitations apply:

- Azure AD can only be enabled on Kubernetes RBAC-enabled cluster.
- Azure AD legacy integration can only be enabled during cluster creation.

Before you begin

You need the Azure CLI version 2.0.61 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Go to <https://shell.azure.com> to open Cloud Shell in your browser.

For consistency and to help run the commands in this article, create a variable for your desired AKS cluster name. The following example uses the name *myakscluster*.

```
aksname="myakscluster"
```

Azure AD authentication overview

Azure AD authentication is provided to AKS clusters with OpenID Connect. OpenID Connect is an identity layer built on top of the OAuth 2.0 protocol. For more information on OpenID Connect, see the [Open ID connect documentation](#).

From inside of the Kubernetes cluster, Webhook Token Authentication is used to verify authentication tokens. Webhook token authentication is configured and managed as part of the AKS cluster. For more information on Webhook token authentication, see the [webhook authentication documentation](#).

NOTE

When configuring Azure AD for AKS authentication, two Azure AD applications are configured. This operation must be completed by an Azure tenant administrator.

Create Azure AD server component

To integrate with AKS, you create and use an Azure AD application that acts as an endpoint for the identity requests. The first Azure AD application you need gets Azure AD group membership for a user.

Create the server application component using the [az ad app create](#) command, then update the group membership claims using the [az ad app update](#) command. The following example uses the *aksname* variable defined in the [Before you begin](#) section, and creates a variable

```
# Create the Azure AD application
serverApplicationId=$(az ad app create \
    --display-name "${aksname}Server" \
    --identifier-uris "https://${aksname}Server" \
    --query appId -o tsv)

# Update the application group membership claims
az ad app update --id $serverApplicationId --set groupMembershipClaims=All
```

Now create a service principal for the server app using the [az ad sp create](#) command. This service principal is used to authenticate itself within the Azure platform. Then, get the service principal secret using the [az ad sp credential reset](#) command and assign to the variable named *serverApplicationSecret* for use in one of the following steps:

```
# Create a service principal for the Azure AD application
az ad sp create --id $serverApplicationId

# Get the service principal secret
serverApplicationSecret=$(az ad sp credential reset \
    --name $serverApplicationId \
    --credential-description "AKSPassword" \
    --query password -o tsv)
```

The Azure AD service principal needs permissions to perform the following actions:

- Read directory data
- Sign in and read user profile

Assign these permissions using the [az ad app permission add](#) command:

```
az ad app permission add \
    --id $serverApplicationId \
    --api 00000003-0000-0000-000000000000 \
    --api-permissions e1fe6dd8-ba31-4d61-89e7-88639da4683d=Scope 06da0dbc-49e2-44d2-8312-53f166ab848a=Scope
7ab1d382-f21e-4acd-a863-ba3e13f7da61=Role
```

Finally, grant the permissions assigned in the previous step for the server application using the [az ad app permission grant](#) command. This step fails if the current account is not a tenant admin. You also need to add permissions for Azure AD application to request information that may otherwise require administrative consent using the [az ad app permission admin-consent](#):

```
az ad app permission grant --id $serverApplicationId --api 00000003-0000-0000-c000-000000000000
az ad app permission admin-consent --id $serverApplicationId
```

Create Azure AD client component

The second Azure AD application is used when a user logs to the AKS cluster with the Kubernetes CLI (`kubectl`). This client application takes the authentication request from the user and verifies their credentials and permissions. Create the Azure AD app for the client component using the [az ad app create](#) command:

```
clientApplicationId=$(az ad app create \
--display-name "${aksname}Client" \
--native-app \
--reply-urls "https://${aksname}Client" \
--query appId -o tsv)
```

Create a service principal for the client application using the [az ad sp create](#) command:

```
az ad sp create --id $clientApplicationId
```

Get the oAuth2 ID for the server app to allow the authentication flow between the two app components using the [az ad app show](#) command. This oAuth2 ID is used in the next step.

```
oAuthPermissionId=$(az ad app show --id $serverApplicationId --query "oauth2Permissions[0].id" -o tsv)
```

Add the permissions for the client application and server application components to use the oAuth2 communication flow using the [az ad app permission add](#) command. Then, grant permissions for the client application to communicate with the server application using the [az ad app permission grant](#) command:

```
az ad app permission add --id $clientApplicationId --api $serverApplicationId --api-permissions
${oAuthPermissionId}=Scope
az ad app permission grant --id $clientApplicationId --api $serverApplicationId
```

Deploy the cluster

With the two Azure AD applications created, now create the AKS cluster itself. First, create a resource group using the [az group create](#) command. The following example creates the resource group in the *EastUS* region:

Create a resource group for the cluster:

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

Get the tenant ID of your Azure subscription using the [az account show](#) command. Then, create the AKS cluster using the [az aks create](#) command. The command to create the AKS cluster provides the server and client application IDs, the server application service principal secret, and your tenant ID:

```
tenantId=$(az account show --query tenantId -o tsv)

az aks create \
    --resource-group myResourceGroup \
    --name $aksname \
    --node-count 1 \
    --generate-ssh-keys \
    --aad-server-app-id $serverApplicationId \
    --aad-server-app-secret $serverApplicationSecret \
    --aad-client-app-id $clientApplicationId \
    --aad-tenant-id $tenantId
```

Finally, get the cluster admin credentials using the [az aks get-credentials](#) command. In one of the following steps, you get the regular *user* cluster credentials to see the Azure AD authentication flow in action.

```
az aks get-credentials --resource-group myResourceGroup --name $aksname --admin
```

Create Kubernetes RBAC binding

Before an Azure Active Directory account can be used with the AKS cluster, a role binding or cluster role binding needs to be created. *Roles* define the permissions to grant, and *bindings* apply them to desired users. These assignments can be applied to a given namespace, or across the entire cluster. For more information, see [Using Kubernetes RBAC authorization](#).

Get the user principal name (UPN) for the user currently logged in using the [az ad signed-in-user show](#) command. This user account is enabled for Azure AD integration in the next step.

```
az ad signed-in-user show --query userPrincipalName -o tsv
```

IMPORTANT

If the user you grant the Kubernetes RBAC binding for is in the same Azure AD tenant, assign permissions based on the *userPrincipalName*. If the user is in a different Azure AD tenant, query for and use the *objectId* property instead.

Create a YAML manifest named `basic-azure-ad-binding.yaml` and paste the following contents. On the last line, replace *userPrincipalName_or_objectId* with the UPN or object ID output from the previous command:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: contoso-cluster-admins
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: userPrincipalName_or_objectId
```

Create the ClusterRoleBinding using the [kubectl apply](#) command and specify the filename of your YAML manifest:

```
kubectl apply -f basic-azure-ad-binding.yaml
```

Access cluster with Azure AD

Now let's test the integration of Azure AD authentication for the AKS cluster. Set the `kubectl` config context to use regular user credentials. This context passes all authentication requests back through Azure AD.

```
az aks get-credentials --resource-group myResourceGroup --name $aksname --overwrite-existing
```

Now use the `kubectl get pods` command to view pods across all namespaces:

```
kubectl get pods --all-namespaces
```

You receive a sign in prompt to authenticate using Azure AD credentials using a web browser. After you've successfully authenticated, the `kubectl` command displays the pods in the AKS cluster, as shown in the following example output:

```
kubectl get pods --all-namespaces
```

```
To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code  
BYMK7UXVD to authenticate.
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	coredns-754f947b4-2v75r	1/1	Running	0	23h
kube-system	coredns-754f947b4-tghwh	1/1	Running	0	23h
kube-system	coredns-autoscaler-6fcdb7d64-4wkvp	1/1	Running	0	23h
kube-system	heapster-5fb7488d97-t5wzk	2/2	Running	0	23h
kube-system	kube-proxy-2nd5m	1/1	Running	0	23h
kube-system	kube-svc-redirect-swp9r	2/2	Running	0	23h
kube-system	kubernetes-dashboard-847bb4ddc6-trt7m	1/1	Running	0	23h
kube-system	metrics-server-7b97f9cd9-btxzz	1/1	Running	0	23h
kube-system	tunnelfront-6ff887cffb-xkfmq	1/1	Running	0	23h

The authentication token received for `kubectl` is cached. You are only repromted to sign in when the token has expired or the Kubernetes config file is re-created.

If you see an authorization error message after you've successfully signed in using a web browser as in the following example output, check the following possible issues:

```
error: You must be logged in to the server (Unauthorized)
```

- You defined the appropriate object ID or UPN, depending on if the user account is in the same Azure AD tenant or not.
- The user is not a member of more than 200 groups.
- Secret defined in the application registration for server matches the value configured using `--aad-server-app-secret`

Next steps

For the complete script that contains the commands shown in this article, see the [Azure AD integration script in the AKS samples repo](#).

To use Azure AD users and groups to control access to cluster resources, see [Control access to cluster resources using Kubernetes role-based access control and Azure AD identities in AKS](#).

For more information about how to secure Kubernetes clusters, see [Access and identity options for AKS](#).

For best practices on identity and resource control, see [Best practices for authentication and authorization in AKS](#).

Use Azure RBAC for Kubernetes Authorization

5/14/2021 • 6 minutes to read • [Edit Online](#)

Today you can already leverage [integrated authentication between Azure Active Directory \(Azure AD\) and AKS](#). When enabled, this integration allows customers to use Azure AD users, groups, or service principals as subjects in Kubernetes RBAC, see more [here](#). This feature frees you from having to separately manage user identities and credentials for Kubernetes. However, you still have to set up and manage Azure RBAC and Kubernetes RBAC separately. For more details on authentication and authorization with RBAC on AKS, see [here](#).

This document covers a new approach that allows for the unified management and access control across Azure Resources, AKS, and Kubernetes resources.

Before you begin

The ability to manage RBAC for Kubernetes resources from Azure gives you the choice to manage RBAC for the cluster resources either using Azure or native Kubernetes mechanisms. When enabled, Azure AD principals will be validated exclusively by Azure RBAC while regular Kubernetes users and service accounts are exclusively validated by Kubernetes RBAC. For more details on authentication and authorization with RBAC on AKS, see [here](#).

Prerequisites

- Ensure you have the Azure CLI version 2.24.0 or later
- Ensure you have installed [kubectl v1.18.3+](#).

Limitations

- Requires [Managed Azure AD integration](#).
- Use [kubectl v1.18.3+](#).
- If you have CRDs and are making custom role definitions, the only way to cover CRDs today is to provide `Microsoft.ContainerService/managedClusters/*/read`. AKS is working on providing more granular permissions for CRDs. For the remaining objects you can use the specific API Groups, for example: `Microsoft.ContainerService/apps/deployments/read`.
- New role assignments can take up to 5min to propagate and be updated by the authorization server.
- Requires the Azure AD tenant configured for authentication to be the same as the tenant for the subscription that holds the AKS cluster.

Create a new cluster using Azure RBAC and managed Azure AD integration

Create an AKS cluster by using the following CLI commands.

Create an Azure resource group:

```
# Create an Azure resource group
az group create --name myResourceGroup --location westus2
```

Create the AKS cluster with managed Azure AD integration and Azure RBAC for Kubernetes Authorization.

```
# Create an AKS-managed Azure AD cluster
az aks create -g MyResourceGroup -n MyManagedCluster --enable-aad --enable-azure-rbac
```

A successful creation of a cluster with Azure AD integration and Azure RBAC for Kubernetes Authorization has the following section in the response body:

```
"AADProfile": {
    "adminGroupObjectIds": null,
    "clientAppId": null,
    "enableAzureRbac": true,
    "managed": true,
    "serverAppId": null,
    "serverAppSecret": null,
    "tenantId": "*****-****-****-****-*****"
}
```

Integrate Azure RBAC into an existing cluster

NOTE

To use Azure RBAC for Kubernetes Authorization, Azure Active Directory integration must be enabled on your cluster. For more, see [Azure Active Directory integration](#).

To add Azure RBAC for Kubernetes Authorization into an existing AKS cluster, use the `az aks update` command with the flag `--enable-azure-rbac`.

```
az aks update -g myResourceGroup -n myAKScluster --enable-azure-rbac
```

Create role assignments for users to access cluster

AKS provides the following four built-in roles:

ROLE	DESCRIPTION
Azure Kubernetes Service RBAC Reader	Allows read-only access to see most objects in a namespace. It doesn't allow viewing roles or role bindings. This role doesn't allow viewing Secrets, since reading the contents of Secrets enables access to ServiceAccount credentials in the namespace, which would allow API access as any ServiceAccount in the namespace (a form of privilege escalation)
Azure Kubernetes Service RBAC Writer	Allows read/write access to most objects in a namespace. This role doesn't allow viewing or modifying roles or role bindings. However, this role allows accessing Secrets and running Pods as any ServiceAccount in the namespace, so it can be used to gain the API access levels of any ServiceAccount in the namespace.

ROLE	DESCRIPTION
Azure Kubernetes Service RBAC Admin	Allows admin access, intended to be granted within a namespace. Allows read/write access to most resources in a namespace (or cluster scope), including the ability to create roles and role bindings within the namespace. This role doesn't allow write access to resource quota or to the namespace itself.
Azure Kubernetes Service RBAC Cluster Admin	Allows super-user access to perform any action on any resource. It gives full control over every resource in the cluster and in all namespaces.

Roles assignments scoped to the entire AKS cluster can be done either on the Access Control (IAM) blade of the cluster resource on Azure portal or by using Azure CLI commands as shown below:

```
# Get your AKS Resource ID
AKS_ID=$(az aks show -g MyResourceGroup -n MyManagedCluster --query id -o tsv)
```

```
az role assignment create --role "Azure Kubernetes Service RBAC Admin" --assignee <AAD-ENTITY-ID> --scope $AKS_ID
```

where <AAD-ENTITY-ID> could be a username (for example, user@contoso.com) or even the ClientID of a service principal.

You can also create role assignments scoped to a specific namespace within the cluster:

```
az role assignment create --role "Azure Kubernetes Service RBAC Reader" --assignee <AAD-ENTITY-ID> --scope $AKS_ID/namespaces/<namespace-name>
```

Today, role assignments scoped to namespaces need to be configured via Azure CLI.

Create custom roles definitions

Optionally you may choose to create your own role definition and then assign as above.

Below is an example of a role definition that allows a user to only read deployments and nothing else. You can check the full list of possible actions [here](#).

Copy the below json into a file called `deploy-view.json`.

```
{
  "Name": "AKS Deployment Reader",
  "Description": "Lets you view all deployments in cluster/namespace.",
  "Actions": [],
  "NotActions": [],
  "DataActions": [
    "Microsoft.ContainerService/managedClusters/apps/deployments/read"
  ],
  "NotDataActions": [],
  "AssignableScopes": [
    "/subscriptions/<YOUR SUBSCRIPTION ID>"
  ]
}
```

Replace <YOUR SUBSCRIPTION ID> by the ID from your subscription, which you can get by running:

```
az account show --query id -o tsv
```

Now we can create the role definition by running the below command from the folder where you saved `deploy-view.json`:

```
az role definition create --role-definition @deploy-view.json
```

Now that you have your role definition, you can assign it to a user or other identity by running:

```
az role assignment create --role "AKS Deployment Reader" --assignee <AAD-ENTITY-ID> --scope $AKS_ID
```

Use Azure RBAC for Kubernetes Authorization with `kubectl`

NOTE

Ensure you have the latest `kubectl` by running the below command:

```
az aks install-cli
```

You might need to run it with `sudo` privileges.

Now that you have assigned your desired role and permissions. You can start calling the Kubernetes API, for example, from `kubectl`.

For this purpose, let's first get the cluster's kubeconfig using the below command:

```
az aks get-credentials -g MyResourceGroup -n MyManagedCluster
```

IMPORTANT

You'll need the [Azure Kubernetes Service Cluster User](#) built-in role to perform the step above.

Now, you can use `kubectl` to, for example, list the nodes in the cluster. The first time you run it you'll need to sign in, and subsequent commands will use the respective access token.

```
kubectl get nodes
To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code
AAAAAAA to authenticate.
```

NAME	STATUS	ROLES	AGE	VERSION
aks-nodepool1-93451573-vms000000	Ready	agent	3h6m	v1.15.11
aks-nodepool1-93451573-vms000001	Ready	agent	3h6m	v1.15.11
aks-nodepool1-93451573-vms000002	Ready	agent	3h6m	v1.15.11

Use Azure RBAC for Kubernetes Authorization with `kubelogin`

To unblock additional scenarios like non-interactive logins, older `kubectl` versions or leveraging SSO across multiple clusters without the need to sign in to new cluster, granted that your token is still valid, AKS created an exec plugin called `kubelogin`.

You can use it by running:

```
export KUBECONFIG=/path/to/kubeconfig  
kubelogin convert-kubeconfig
```

The first time, you'll have to sign in interactively like with regular kubectl, but afterwards you'll no longer need to, even for new Azure AD clusters (as long as your token is still valid).

```
kubectl get nodes  
To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code  
AAAAAAA to authenticate.
```

NAME	STATUS	ROLES	AGE	VERSION
aks-nodepool1-93451573-vmss000000	Ready	agent	3h6m	v1.15.11
aks-nodepool1-93451573-vmss000001	Ready	agent	3h6m	v1.15.11
aks-nodepool1-93451573-vmss000002	Ready	agent	3h6m	v1.15.11

Clean up

Clean Role assignment

```
az role assignment list --scope $AKS_ID --query [].id -o tsv
```

Copy the ID or IDs from all the assignments you did and then.

```
az role assignment delete --ids <LIST OF ASSIGNMENT IDS>
```

Clean up role definition

```
az role definition delete -n "AKS Deployment Reader"
```

Delete cluster and resource group

```
az group delete -n MyResourceGroup
```

Next steps

- Read more about AKS Authentication, Authorization, Kubernetes RBAC, and Azure RBAC [here](#).
- Read more about Azure RBAC [here](#).
- Read more about the all the actions you can use to granularly define custom Azure roles for Kubernetes authorization [here](#).

Control access to cluster resources using Kubernetes role-based access control and Azure Active Directory identities in Azure Kubernetes Service

4/21/2021 • 11 minutes to read • [Edit Online](#)

Azure Kubernetes Service (AKS) can be configured to use Azure Active Directory (AD) for user authentication. In this configuration, you sign in to an AKS cluster using an Azure AD authentication token. You can also configure Kubernetes role-based access control (Kubernetes RBAC) to limit access to cluster resources based a user's identity or group membership.

This article shows you how to use Azure AD group membership to control access to namespaces and cluster resources using Kubernetes RBAC in an AKS cluster. Example groups and users are created in Azure AD, then Roles and RoleBindings are created in the AKS cluster to grant the appropriate permissions to create and view resources.

Before you begin

This article assumes that you have an existing AKS cluster enabled with Azure AD integration. If you need an AKS cluster, see [Integrate Azure Active Directory with AKS](#).

You need the Azure CLI version 2.0.61 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Create demo groups in Azure AD

In this article, let's create two user roles that can be used to show how Kubernetes RBAC and Azure AD control access to cluster resources. The following two example roles are used:

- **Application developer**
 - A user named *aksdev* that is part of the *appdev* group.
- **Site reliability engineer**
 - A user named *akssre* that is part of the *opssre* group.

In production environments, you can use existing users and groups within an Azure AD tenant.

First, get the resource ID of your AKS cluster using the `az aks show` command. Assign the resource ID to a variable named *AKS_ID* so that it can be referenced in additional commands.

```
AKS_ID=$(az aks show \
    --resource-group myResourceGroup \
    --name myAKScluster \
    --query id -o tsv)
```

Create the first example group in Azure AD for the application developers using the `az ad group create` command. The following example creates a group named *appdev*.

```
APPDEV_ID=$(az ad group create --display-name appdev --mail-nickname appdev --query objectId -o tsv)
```

Now, create an Azure role assignment for the *appdev* group using the `az role assignment create` command. This

assignment lets any member of the group use `kubectl` to interact with an AKS cluster by granting them the *Azure Kubernetes Service Cluster User Role*.

```
az role assignment create \
--assignee $APPDEV_ID \
--role "Azure Kubernetes Service Cluster User Role" \
--scope $AKS_ID
```

TIP

If you receive an error such as

```
Principal 35bfec9328bd4d8d9b54dea6dac57b82 does not exist in the directory a5443dcd-cd0e-494d-a387-3039b419f0d5.
```

, wait a few seconds for the Azure AD group object ID to propagate through the directory then try the

```
az role assignment create
```

Create a second example group, this one for SREs named *opssre*.

```
OPSSRE_ID=$(az ad group create --display-name opssre --mail-nickname opssre --query objectId -o tsv)
```

Again, create an Azure role assignment to grant members of the group the *Azure Kubernetes Service Cluster User Role*.

```
az role assignment create \
--assignee $OPSSRE_ID \
--role "Azure Kubernetes Service Cluster User Role" \
--scope $AKS_ID
```

Create demo users in Azure AD

With two example groups created in Azure AD for our application developers and SREs, now lets create two example users. To test the Kubernetes RBAC integration at the end of the article, you sign in to the AKS cluster with these accounts.

Set the user principal name (UPN) and password for the application developers. The following command prompts you for the UPN and sets it to *AAD_DEV_UPN* for use in a later command (remember that the commands in this article are entered into a BASH shell). The UPN must include the verified domain name of your tenant, for example `aksdev@contoso.com`.

```
echo "Please enter the UPN for application developers: " && read AAD_DEV_UPN
```

The following command prompts you for the password and sets it to *AAD_DEV_PW* for use in a later command.

```
echo "Please enter the secure password for application developers: " && read AAD_DEV_PW
```

Create the first user account in Azure AD using the `az ad user create` command.

The following example creates a user with the display name *AKS Dev* and the UPN and secure password using the values in *AAD_DEV_UPN* and *AAD_DEV_PW*.

```
AKSDEV_ID=$(az ad user create \
--display-name "AKS Dev" \
--user-principal-name $AAD_DEV_UPN \
--password $AAD_DEV_PW \
--query objectId -o tsv)
```

Now add the user to the *appdev* group created in the previous section using the [az ad group member add](#) command:

```
az ad group member add --group appdev --member-id $AKSDEV_ID
```

Set the UPN and password for SREs. The following command prompts you for the UPN and sets it to *AAD_SRE_UPN* for use in a later command (remember that the commands in this article are entered into a BASH shell). The UPN must include the verified domain name of your tenant, for example `akssre@contoso.com`.

```
echo "Please enter the UPN for SREs: " && read AAD_SRE_UPN
```

The following command prompts you for the password and sets it to *AAD_SRE_PW* for use in a later command.

```
echo "Please enter the secure password for SREs: " && read AAD_SRE_PW
```

Create a second user account. The following example creates a user with the display name *AKS SRE* and the UPN and secure password using the values in *AAD_SRE_UPN* and *AAD_SRE_PW*.

```
# Create a user for the SRE role
AKSSRE_ID=$(az ad user create \
--display-name "AKS SRE" \
--user-principal-name $AAD_SRE_UPN \
--password $AAD_SRE_PW \
--query objectId -o tsv)

# Add the user to the opssre Azure AD group
az ad group member add --group opssre --member-id $AKSSRE_ID
```

Create the AKS cluster resources for app devs

The Azure AD groups and users are now created. Azure role assignments were created for the group members to connect to an AKS cluster as a regular user. Now, let's configure the AKS cluster to allow these different groups access to specific resources.

First, get the cluster admin credentials using the [az aks get-credentials](#) command. In one of the following sections, you get the regular *user* cluster credentials to see the Azure AD authentication flow in action.

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster --admin
```

Create a namespace in the AKS cluster using the [kubectl create namespace](#) command. The following example creates a namespace name *dev*.

```
kubectl create namespace dev
```

In Kubernetes, *Roles* define the permissions to grant, and *RoleBindings* apply them to desired users or groups. These assignments can be applied to a given namespace, or across the entire cluster. For more information, see

Using Kubernetes RBAC authorization.

First, create a Role for the *dev* namespace. This role grants full permissions to the namespace. In production environments, you can specify more granular permissions for different users or groups.

Create a file named `role-dev-namespace.yaml` and paste the following YAML manifest:

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: dev-user-full-access
  namespace: dev
rules:
- apiGroups: [ "", "extensions", "apps" ]
  resources: [ "*" ]
  verbs: [ "*" ]
- apiGroups: [ "batch" ]
  resources:
  - jobs
  - cronjobs
  verbs: [ "*" ]
```

Create the Role using the [kubectl apply](#) command and specify the filename of your YAML manifest:

```
kubectl apply -f role-dev-namespace.yaml
```

Next, get the resource ID for the *appdev* group using the [az ad group show](#) command. This group is set as the subject of a RoleBinding in the next step.

```
az ad group show --group appdev --query objectId -o tsv
```

Now, create a RoleBinding for the *appdev* group to use the previously created Role for namespace access. Create a file named `rolebinding-dev-namespace.yaml` and paste the following YAML manifest. On the last line, replace `groupObjectId` with the group object ID output from the previous command:

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: dev-user-access
  namespace: dev
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: dev-user-full-access
subjects:
- kind: Group
  namespace: dev
  name: groupObjectId
```

Create the RoleBinding using the [kubectl apply](#) command and specify the filename of your YAML manifest:

```
kubectl apply -f rolebinding-dev-namespace.yaml
```

Create the AKS cluster resources for SREs

Now, repeat the previous steps to create a namespace, Role, and RoleBinding for the SREs.

First, create a namespace for *sre* using the [kubectl create namespace](#) command:

```
kubectl create namespace sre
```

Create a file named `role-sre-namespace.yaml` and paste the following YAML manifest:

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: sre-user-full-access
  namespace: sre
rules:
- apiGroups: [ "", "extensions", "apps"]
  resources: [ "*" ]
  verbs: [ "*" ]
- apiGroups: [ "batch" ]
  resources:
  - jobs
  - cronjobs
  verbs: [ "*" ]
```

Create the Role using the [kubectl apply](#) command and specify the filename of your YAML manifest:

```
kubectl apply -f role-sre-namespace.yaml
```

Get the resource ID for the *opssre* group using the [az ad group show](#) command:

```
az ad group show --group opssre --query objectId -o tsv
```

Create a RoleBinding for the *opssre* group to use the previously created Role for namespace access. Create a file named `rolebinding-sre-namespace.yaml` and paste the following YAML manifest. On the last line, replace `groupObjectId` with the group object ID output from the previous command:

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: sre-user-access
  namespace: sre
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: sre-user-full-access
subjects:
- kind: Group
  namespace: sre
  name: groupObjectId
```

Create the RoleBinding using the [kubectl apply](#) command and specify the filename of your YAML manifest:

```
kubectl apply -f rolebinding-sre-namespace.yaml
```

Interact with cluster resources using Azure AD identities

Now, let's test the expected permissions work when you create and manage resources in an AKS cluster. In these examples, you schedule and view pods in the user's assigned namespace. Then, you try to schedule and view

pods outside of the assigned namespace.

First, reset the `kubeconfig` context using the `az aks get-credentials` command. In a previous section, you set the context using the cluster admin credentials. The admin user bypasses Azure AD sign in prompts. Without the `--admin` parameter, the user context is applied that requires all requests to be authenticated using Azure AD.

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster --overwrite-existing
```

Schedule a basic NGINX pod using the `kubectl run` command in the `dev` namespace:

```
kubectl run nginx-dev --image=mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine --namespace dev
```

As the sign in prompt, enter the credentials for your own `appdev@contoso.com` account created at the start of the article. Once you are successfully signed in, the account token is cached for future `kubectl` commands. The NGINX is successfully scheduled, as shown in the following example output:

```
$ kubectl run nginx-dev --image=mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine --namespace dev  
To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code  
B24ZD6FP8 to authenticate.  
pod/nginx-dev created
```

Now use the `kubectl get pods` command to view pods in the `dev` namespace.

```
kubectl get pods --namespace dev
```

As shown in the following example output, the NGINX pod is successfully *Running*.

```
$ kubectl get pods --namespace dev  
NAME      READY   STATUS    RESTARTS   AGE  
nginx-dev  1/1     Running   0          4m
```

Create and view cluster resources outside of the assigned namespace

Now try to view pods outside of the `dev` namespace. Use the `kubectl get pods` command again, this time to see `--all-namespaces` as follows:

```
kubectl get pods --all-namespaces
```

The user's group membership does not have a Kubernetes Role that allows this action, as shown in the following example output:

```
$ kubectl get pods --all-namespaces  
Error from server (Forbidden): pods is forbidden: User "aksdev@contoso.com" cannot list resource "pods" in  
API group "" at the cluster scope
```

In the same way, try to schedule a pod in different namespace, such as the `sre` namespace. The user's group membership does not align with a Kubernetes Role and RoleBinding to grant these permissions, as shown in the following example output:

```
$ kubectl run nginx-dev --image=mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine --namespace sre  
  
Error from server (Forbidden): pods is forbidden: User "aksdev@contoso.com" cannot create resource "pods" in  
API group "" in the namespace "sre"
```

Test the SRE access to the AKS cluster resources

To confirm that our Azure AD group membership and Kubernetes RBAC work correctly between different users and groups, try the previous commands when signed in as the *opssre* user.

Reset the *kubeconfig* context using the [az aks get-credentials](#) command that clears the previously cached authentication token for the *aksdev* user:

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster --overwrite-existing
```

Try to schedule and view pods in the assigned *sre* namespace. When prompted, sign in with your own [opssre@contoso.com](#) credentials created at the start of the article:

```
kubectl run nginx-sre --image=mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine --namespace sre  
kubectl get pods --namespace sre
```

As shown in the following example output, you can successfully create and view the pods:

```
$ kubectl run nginx-sre --image=mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine --namespace sre  
  
To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code  
BM4RHP3FD to authenticate.  
  
pod/nginx-sre created  
  
$ kubectl get pods --namespace sre  
  
NAME      READY   STATUS    RESTARTS   AGE  
nginx-sre  1/1     Running   0          1m
```

Now, try to view or schedule pods outside of assigned SRE namespace:

```
kubectl get pods --all-namespaces  
kubectl run nginx-sre --image=mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine --namespace dev
```

These `kubectl` commands fail, as shown in the following example output. The user's group membership and Kubernetes Role and RoleBindings don't grant permissions to create or manager resources in other namespaces:

```
$ kubectl get pods --all-namespaces  
Error from server (Forbidden): pods is forbidden: User "akssre@contoso.com" cannot list pods at the cluster  
scope  
  
$ kubectl run nginx-sre --image=mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine --namespace dev  
Error from server (Forbidden): pods is forbidden: User "akssre@contoso.com" cannot create pods in the  
namespace "dev"
```

Clean up resources

In this article, you created resources in the AKS cluster and users and groups in Azure AD. To clean up all these resources, run the following commands:

```
# Get the admin kubeconfig context to delete the necessary cluster resources
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster --admin

# Delete the dev and sre namespaces. This also deletes the pods, Roles, and RoleBindings
kubectl delete namespace dev
kubectl delete namespace sre

# Delete the Azure AD user accounts for aksdev and akssre
az ad user delete --upn-or-object-id $AKSDEV_ID
az ad user delete --upn-or-object-id $AKSSRE_ID

# Delete the Azure AD groups for appdev and opssre. This also deletes the Azure role assignments.
az ad group delete --group appdev
az ad group delete --group opssre
```

Next steps

For more information about how to secure Kubernetes clusters, see [Access and identity options for AKS](#).

For best practices on identity and resource control, see [Best practices for authentication and authorization in AKS](#).

Rotate certificates in Azure Kubernetes Service (AKS)

5/14/2021 • 3 minutes to read • [Edit Online](#)

Azure Kubernetes Service (AKS) uses certificates for authentication with many of its components. Periodically, you may need to rotate those certificates for security or policy reasons. For example, you may have a policy to rotate all your certificates every 90 days.

This article shows you how to rotate the certificates in your AKS cluster.

Before you begin

This article requires that you are running the Azure CLI version 2.0.77 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

AKS certificates, Certificate Authorities, and Service Accounts

AKS generates and uses the following certificates, Certificate Authorities, and Service Accounts:

- The AKS API server creates a Certificate Authority (CA) called the Cluster CA.
- The API server has a Cluster CA, which signs certificates for one-way communication from the API server to kubelets.
- Each kubelet also creates a Certificate Signing Request (CSR), which is signed by the Cluster CA, for communication from the kubelet to the API server.
- The API aggregator uses the Cluster CA to issue certificates for communication with other APIs. The API aggregator can also have its own CA for issuing those certificates, but it currently uses the Cluster CA.
- Each node uses a Service Account (SA) token, which is signed by the Cluster CA.
- The `kubectl` client has a certificate for communicating with the AKS cluster.

NOTE

AKS clusters created prior to March 2019 have certificates that expire after two years. Any cluster created after March 2019 or any cluster that has its certificates rotated have Cluster CA certificates that expire after 30 years. All other certificates expire after two years. To verify when your cluster was created, use `kubectl get nodes` to see the *Age* of your node pools.

Additionally, you can check the expiration date of your cluster's certificate. For example, the following Bash command displays the certificate details for the *myAKSCluster* cluster.

```
kubectl config view --raw -o jsonpath=".clusters[?(@.name == 'myAKSCluster')].cluster.certificate-authority-data" | base64 -d | openssl x509 -text | grep -A2 Validity
```

Rotate your cluster certificates

WARNING

Rotating your certificates using `az aks rotate-certs` can cause up to 30 minutes of downtime for your AKS cluster.

Use `az aks get-credentials` to sign in to your AKS cluster. This command also downloads and configures the `kubectl` client certificate on your local machine.

```
az aks get-credentials -g $RESOURCE_GROUP_NAME -n $CLUSTER_NAME
```

Use `az aks rotate-certs` to rotate all certificates, CAs, and SAs on your cluster.

```
az aks rotate-certs -g $RESOURCE_GROUP_NAME -n $CLUSTER_NAME
```

IMPORTANT

It may take up to 30 minutes for `az aks rotate-certs` to complete. If the command fails before completing, use `az aks show` to verify the status of the cluster is *Certificate Rotating*. If the cluster is in a failed state, rerun `az aks rotate-certs` to rotate your certificates again.

Verify that the old certificates are no longer valid by running a `kubectl` command. Since you have not updated the certificates used by `kubectl`, you will see an error. For example:

```
$ kubectl get no
Unable to connect to the server: x509: certificate signed by unknown authority (possibly because of
"crypto/rsa: verification error" while trying to verify candidate authority certificate "ca")
```

Update the certificate used by `kubectl` by running `az aks get-credentials`.

```
az aks get-credentials -g $RESOURCE_GROUP_NAME -n $CLUSTER_NAME --overwrite-existing
```

Verify the certificates have been updated by running a `kubectl` command, which will now succeed. For example:

```
kubectl get no
```

NOTE

If you have any services that run on top of AKS, you may need to update certificates related to those services as well.

Next steps

This article showed you how to automatically rotate your cluster's certificates, CAs, and SAs. You can see [Best practices for cluster security and upgrades in Azure Kubernetes Service \(AKS\)](#) for more information on AKS security best practices.

Create a private Azure Kubernetes Service cluster

5/5/2021 • 8 minutes to read • [Edit Online](#)

In a private cluster, the control plane or API server has internal IP addresses that are defined in the [RFC1918 - Address Allocation for Private Internet](#) document. By using a private cluster, you can ensure network traffic between your API server and your node pools remains on the private network only.

The control plane or API server is in an Azure Kubernetes Service (AKS)-managed Azure subscription. A customer's cluster or node pool is in the customer's subscription. The server and the cluster or node pool can communicate with each other through the [Azure Private Link service](#) in the API server virtual network and a private endpoint that's exposed in the subnet of the customer's AKS cluster.

Region availability

Private cluster is available in public regions, Azure Government, and Azure China 21Vianet regions where [AKS is supported](#).

NOTE

Azure Government sites are supported, however US Gov Texas isn't currently supported because of missing Private Link support.

Prerequisites

- The Azure CLI version 2.2.0 or later
- The Private Link service is supported on Standard Azure Load Balancer only. Basic Azure Load Balancer isn't supported.
- To use a custom DNS server, add the Azure DNS IP 168.63.129.16 as the upstream DNS server in the custom DNS server.

Create a private AKS cluster

Create a resource group

Create a resource group or use an existing resource group for your AKS cluster.

```
az group create -l westus -n MyResourceGroup
```

Default basic networking

```
az aks create -n <private-cluster-name> -g <private-cluster-resource-group> --load-balancer-sku standard --enable-private-cluster
```

Where `--enable-private-cluster` is a mandatory flag for a private cluster.

Advanced networking

```
az aks create \
--resource-group <private-cluster-resource-group> \
--name <private-cluster-name> \
--load-balancer-sku standard \
--enable-private-cluster \
--network-plugin azure \
--vnet-subnet-id <subnet-id> \
--docker-bridge-address 172.17.0.1/16 \
--dns-service-ip 10.2.0.10 \
--service-cidr 10.2.0.0/24
```

Where `--enable-private-cluster` is a mandatory flag for a private cluster.

NOTE

If the Docker bridge address CIDR (172.17.0.1/16) clashes with the subnet CIDR, change the Docker bridge address appropriately.

Configure Private DNS Zone

The following parameters can be leveraged to configure Private DNS Zone.

- "System" is the default value. If the `--private-dns-zone` argument is omitted, AKS will create a Private DNS Zone in the Node Resource Group.
- If the Private DNS Zone is in a different subscription than the AKS cluster, you need to register `Microsoft.ContainerServices` in both the subscriptions.
- "None" means AKS will not create a Private DNS Zone. This requires you to Bring Your Own DNS Server and configure the DNS resolution for the Private FQDN. If you don't configure DNS resolution, DNS is only resolvable within the agent nodes and will cause cluster issues after deployment.
- "CUSTOM_PRIVATE_DNS_ZONE_RESOURCE_ID" requires you to create a Private DNS Zone in this format for azure global cloud: `privatelink.<region>.azmk8s.io`. You will need the Resource Id of that Private DNS Zone going forward. Additionally, you will need a user assigned identity or service principal with at least the `private dns zone contributor` and `vnet contributor` roles.
- "fqdn-subdomain" can be utilized with "CUSTOM_PRIVATE_DNS_ZONE_RESOURCE_ID" only to provide subdomain capabilities to `privatelink.<region>.azmk8s.io`

Prerequisites

- The AKS Preview version 0.5.7 or later
- The api version 2020-11-01 or later

Create a private AKS cluster with Private DNS Zone

```
az aks create -n <private-cluster-name> -g <private-cluster-resource-group> --load-balancer-sku standard --enable-private-cluster --enable-managed-identity --assign-identity <ResourceId> --private-dns-zone [system|none]
```

Create a private AKS cluster with a Custom Private DNS Zone

```
az aks create -n <private-cluster-name> -g <private-cluster-resource-group> --load-balancer-sku standard --enable-private-cluster --enable-managed-identity --assign-identity <ResourceId> --private-dns-zone <custom private dns zone ResourceId> --fqdn-subdomain <subdomain-name>
```

Options for connecting to the private cluster

The API server endpoint has no public IP address. To manage the API server, you'll need to use a VM that has access to the AKS cluster's Azure Virtual Network (VNet). There are several options for establishing network connectivity to the private cluster.

- Create a VM in the same Azure Virtual Network (VNet) as the AKS cluster.
- Use a VM in a separate network and set up [Virtual network peering](#). See the section below for more information on this option.
- Use an [Express Route or VPN](#) connection.
- Use the [AKS Run Command feature](#).

Creating a VM in the same VNET as the AKS cluster is the easiest option. Express Route and VPNs add costs and require additional networking complexity. Virtual network peering requires you to plan your network CIDR ranges to ensure there are no overlapping ranges.

AKS Run Command (Preview)

Today when you need to access a private cluster, you must do so within the cluster virtual network or a peered network or client machine. This usually requires your machine to be connected via VPN or Express Route to the cluster virtual network or a jumpbox to be created in the cluster virtual network. AKS run command allows you to remotely invoke commands in an AKS cluster through the AKS API. This feature provides an API that allows you to, for example, execute just-in-time commands from a remote laptop for a private cluster. This can greatly assist with quick just-in-time access to a private cluster when the client machine is not on the cluster private network while still retaining and enforcing the same RBAC controls and private API server.

Register the `RunCommandPreview` preview feature

To use the new Run Command API, you must enable the `RunCommandPreview` feature flag on your subscription.

Register the `RunCommandPreview` feature flag by using the `[az feature register][az-feature-register]` command, as shown in the following example:

```
az feature register --namespace "Microsoft.ContainerService" --name "RunCommandPreview"
```

It takes a few minutes for the status to show *Registered*. Verify the registration status by using the [az feature list](#) command:

```
az feature list -o table --query "[?contains(name, 'Microsoft.ContainerService/RunCommandPreview')].{Name:name,State:properties.state}"
```

When ready, refresh the registration of the *Microsoft.ContainerService* resource provider by using the [az provider register](#) command:

```
az provider register --namespace Microsoft.ContainerService
```

Use AKS Run Command

Simple command

```
az aks command invoke -g <resourceGroup> -n <clusterName> -c "kubectl get pods -n kube-system"
```

Deploy a manifest by attaching the specific file

```
az aks command invoke -g <resourceGroup> -n <clusterName> -c "kubectl apply -f deployment.yaml -n default" -f deployment.yaml
```

Deploy a manifest by attaching a whole folder

```
az aks command invoke -g <resourceGroup> -n <clusterName> -c "kubectl apply -f deployment.yaml -n default" -f .
```

Perform a Helm install and pass the specific values manifest

```
az aks command invoke -g <resourceGroup> -n <clusterName> -c "helm repo add bitnami https://charts.bitnami.com/bitnami && helm repo update && helm install my-release -f values.yaml bitnami/nginx" -f values.yaml
```

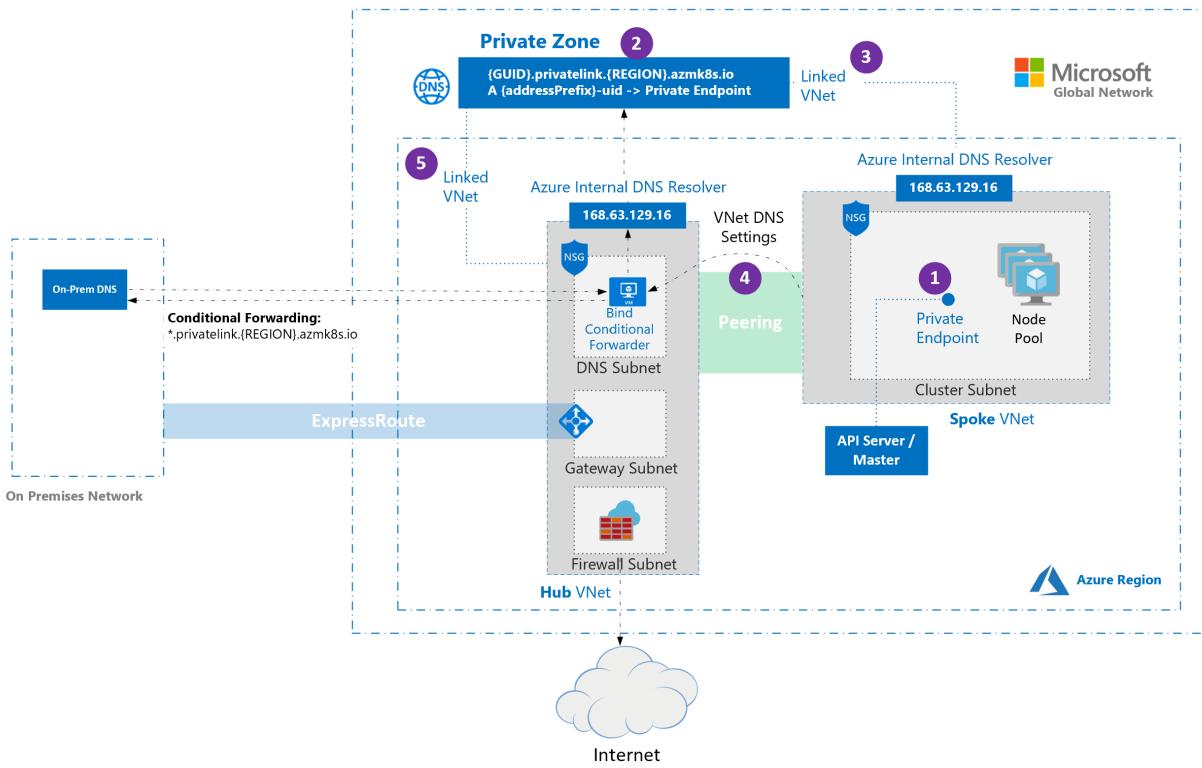
Virtual network peering

As mentioned, virtual network peering is one way to access your private cluster. To use virtual network peering, you need to set up a link between virtual network and the private DNS zone.

1. Go to the node resource group in the Azure portal.
2. Select the private DNS zone.
3. In the left pane, select the **Virtual network** link.
4. Create a new link to add the virtual network of the VM to the private DNS zone. It takes a few minutes for the DNS zone link to become available.
5. In the Azure portal, navigate to the resource group that contains your cluster's virtual network.
6. In the right pane, select the virtual network. The virtual network name is in the form *aks-vnet-**.
7. In the left pane, select **Peerings**.
8. Select **Add**, add the virtual network of the VM, and then create the peering.
9. Go to the virtual network where you have the VM, select **Peerings**, select the AKS virtual network, and then create the peering. If the address ranges on the AKS virtual network and the VM's virtual network clash, peering fails. For more information, see [Virtual network peering](#).

Hub and spoke with custom DNS

[Hub and spoke architectures](#) are commonly used to deploy networks in Azure. In many of these deployments, DNS settings in the spoke VNets are configured to reference a central DNS forwarder to allow for on-premises and Azure-based DNS resolution. When deploying an AKS cluster into such a networking environment, there are some special considerations that must be taken into account.



1. By default, when a private cluster is provisioned, a private endpoint (1) and a private DNS zone (2) are created in the cluster-managed resource group. The cluster uses an A record in the private zone to resolve the IP of the private endpoint for communication to the API server.
2. The private DNS zone is linked only to the VNet that the cluster nodes are attached to (3). This means that the private endpoint can only be resolved by hosts in that linked VNet. In scenarios where no custom DNS is configured on the VNet (default), this works without issue as hosts point at 168.63.129.16 for DNS that can resolve records in the private DNS zone because of the link.
3. In scenarios where the VNet containing your cluster has custom DNS settings (4), cluster deployment fails unless the private DNS zone is linked to the VNet that contains the custom DNS resolvers (5). This link can be created manually after the private zone is created during cluster provisioning or via automation upon detection of creation of the zone using event-based deployment mechanisms (for example, Azure Event Grid and Azure Functions).

NOTE

If you are using [Bring Your Own Route Table with kubenet](#) and [Bring Your Own DNS with Private Cluster](#), the cluster creation will fail. You will need to associate the [RouteTable](#) in the node resource group to the subnet after the cluster creation failed, in order to make the creation successful.

Limitations

- AKS-RunCommand does not work on clusters with AKS managed AAD and Private link enabled.
- IP authorized ranges can't be applied to the private api server endpoint, they only apply to the public API server
- [Azure Private Link service limitations](#) apply to private clusters.
- No support for Azure DevOps Microsoft-hosted Agents with private clusters. Consider to use [Self-hosted Agents](#).
- For customers that need to enable Azure Container Registry to work with private AKS, the Container Registry virtual network must be peered with the agent cluster virtual network.
- No support for converting existing AKS clusters into private clusters

- Deleting or modifying the private endpoint in the customer subnet will cause the cluster to stop functioning.
- After customers have updated the A record on their own DNS servers, those Pods would still resolve apiserver FQDN to the older IP after migration until they're restarted. Customers need to restart hostNetwork Pods and default-DNSPolicy Pods after control plane migration.
- In the case of maintenance on the control plane, your [AKS IP](#) might change. In this case you must update the A record pointing to the API server private IP on your custom DNS server and restart any custom pods or deployments using hostNetwork.

Bring your own keys (BYOK) with Azure disks in Azure Kubernetes Service (AKS)

4/21/2021 • 3 minutes to read • [Edit Online](#)

Azure Storage encrypts all data in a storage account at rest. By default, data is encrypted with Microsoft-managed keys. For additional control over encryption keys, you can supply customer-managed keys to use for encryption at rest for both the OS and data disks for your AKS clusters. Learn more about customer-managed keys on [Linux](#) and [Windows](#).

Limitations

- Data disk encryption support is limited to AKS clusters running Kubernetes version 1.17 and above.
- Encryption of OS and data disk with customer-managed keys can only be enabled when creating an AKS cluster.

Prerequisites

- You must enable soft delete and purge protection for *Azure Key Vault* when using Key Vault to encrypt managed disks.
- You need the Azure CLI version 2.11.1 or later.

Create an Azure Key Vault instance

Use an Azure Key Vault instance to store your keys. You can optionally use the Azure portal to [Configure customer-managed keys with Azure Key Vault](#)

Create a new *resource group*, then create a new *Key Vault* instance and enable soft delete and purge protection. Ensure you use the same region and resource group names for each command.

```
# Optionally retrieve Azure region short names for use on upcoming commands
az account list-locations
```

```
# Create new resource group in a supported Azure region
az group create -l myAzureRegionName -n myResourceGroup

# Create an Azure Key Vault resource in a supported Azure region
az keyvault create -n myKeyVaultName -g myResourceGroup -l myAzureRegionName --enable-purge-protection true
--enable-soft-delete true
```

Create an instance of a DiskEncryptionSet

Replace *myKeyVaultName* with the name of your key vault. You will also need a *key* stored in Azure Key Vault to complete the following steps. Either store your existing Key in the Key Vault you created on the previous steps, or [generate a new key](#) and replace *myKeyName* below with the name of your key.

```

# Retrieve the Key Vault Id and store it in a variable
keyVaultId=$(az keyvault show --name myKeyVaultName --query "[id]" -o tsv)

# Retrieve the Key Vault key URL and store it in a variable
keyVaultKeyUrl=$(az keyvault key show --vault-name myKeyVaultName --name myKeyName --query "[key.kid]" -o tsv)

# Create a DiskEncryptionSet
az disk-encryption-set create -n myDiskEncryptionSetName -l myAzureRegionName -g myResourceGroup --source-vault $keyVaultId --key-url $keyVaultKeyUrl

```

Grant the DiskEncryptionSet access to key vault

Use the DiskEncryptionSet and resource groups you created on the prior steps, and grant the DiskEncryptionSet resource access to the Azure Key Vault.

```

# Retrieve the DiskEncryptionSet value and set a variable
desIdentity=$(az disk-encryption-set show -n myDiskEncryptionSetName -g myResourceGroup --query "[identity.principalId]" -o tsv)

# Update security policy settings
az keyvault set-policy -n myKeyVaultName -g myResourceGroup --object-id $desIdentity --key-permissions wrapkey unwrapkey get

```

Create a new AKS cluster and encrypt the OS disk

Create a **new resource group** and AKS cluster, then use your key to encrypt the OS disk. Customer-managed keys are only supported in Kubernetes versions greater than 1.17.

IMPORTANT

Ensure you create a new resource group for your AKS cluster

```

# Retrieve the DiskEncryptionSet value and set a variable
diskEncryptionSetId=$(az disk-encryption-set show -n myDiskEncryptionSetName -g myResourceGroup --query "[id]" -o tsv)

# Create a resource group for the AKS cluster
az group create -n myResourceGroup -l myAzureRegionName

# Create the AKS cluster
az aks create -n myAKSCluster -g myResourceGroup --node-osdisk-diskencryptionset-id $diskEncryptionSetId --kubernetes-version KUBERNETES_VERSION --generate-ssh-keys

```

When new node pools are added to the cluster created above, the customer-managed key provided during the create is used to encrypt the OS disk.

Encrypt your AKS cluster data disk(optional)

OS disk encryption key will be used to encrypt data disk if key is not provided for data disk from v1.17.2, and you can also encrypt AKS data disks with your other keys.

IMPORTANT

Ensure you have the proper AKS credentials. The managed identity will need to have contributor access to the resource group where the diskencryptionset is deployed. Otherwise, you will get an error suggesting that the managed identity does not have permissions.

```
# Retrieve your Azure Subscription Id from id property as shown below  
az account list
```

```
someuser@Azure:~$ az account list  
[  
 {  
   "cloudName": "AzureCloud",  
   "id": "666e66d8-1e43-4136-be25-f25bb5de5893",  
   "isDefault": true,  
   "name": "MyAzureSubscription",  
   "state": "Enabled",  
   "tenantId": "3ebcdf90-2069-4529-a1ab-7bdcb24df7cd",  
   "user": {  
     "cloudShellID": true,  
     "name": "someuser@azure.com",  
     "type": "user"  
   }  
 }  
]
```

Create a file called **byok-azure-disk.yaml** that contains the following information. Replace myAzureSubscriptionId, myResourceGroup, and myDiskEncryptionSetName with your values, and apply the yaml. Make sure to use the resource group where your DiskEncryptionSet is deployed. If you use the Azure Cloud Shell, this file can be created using vi or nano as if working on a virtual or physical system:

```
kind: StorageClass  
apiVersion: storage.k8s.io/v1  
metadata:  
  name: hdd  
provisioner: kubernetes.io/azure-disk  
parameters:  
  skuName: Standard_LRS  
  kind: managed  
  diskEncryptionSetID:  
  "/subscriptions/{myAzureSubscriptionId}/resourceGroups/{myResourceGroup}/providers/Microsoft.Compute/diskEncryptionSets/{myDiskEncryptionSetName}"
```

Next, run this deployment in your AKS cluster:

```
# Get credentials  
az aks get-credentials --name myAksCluster --resource-group myResourceGroup --output table  
  
# Update cluster  
kubectl apply -f byok-azure-disk.yaml
```

Next steps

Review [best practices for AKS cluster security](#)

Host-based encryption on Azure Kubernetes Service (AKS)

5/18/2021 • 2 minutes to read • [Edit Online](#)

With host-based encryption, the data stored on the VM host of your AKS agent nodes' VMs is encrypted at rest and flows encrypted to the Storage service. This means the temp disks are encrypted at rest with platform-managed keys. The cache of OS and data disks is encrypted at rest with either platform-managed keys or customer-managed keys depending on the encryption type set on those disks. By default, when using AKS, OS and data disks are encrypted at rest with platform-managed keys, meaning that the caches for these disks are also by default encrypted at rest with platform-managed keys. You can specify your own managed keys following [Bring your own keys \(BYOK\) with Azure disks in Azure Kubernetes Service](#). The cache for these disks will then also be encrypted using the key that you specify in this step.

Before you begin

This feature can only be set at cluster creation or node pool creation time.

NOTE

Host-based encryption is available in [Azure regions](#) that support server side encryption of Azure managed disks and only with specific [supported VM sizes](#).

Prerequisites

- Ensure you have the CLI extension v2.23 or higher version installed.

Limitations

- Can only be enabled on new node pools.
- Can only be enabled in [Azure regions](#) that support server-side encryption of Azure managed disks and only with specific [supported VM sizes](#).
- Requires an AKS cluster and node pool based on Virtual Machine Scale Sets(VMSS) as *VM set type*.

Use host-based encryption on new clusters

Configure the cluster agent nodes to use host-based encryption when the cluster is created.

```
az aks create --name myAKSCluster --resource-group myResourceGroup -s Standard_DS2_v2 -l westus2 --enable-encryption-at-host
```

If you want to create clusters without host-based encryption, you can do so by omitting the `--enable-encryption-at-host` parameter.

Use host-based encryption on existing clusters

You can enable host-based encryption on existing clusters by adding a new node pool to your cluster. Configure a new node pool to use host-based encryption by using the `--enable-encryption-at-host` parameter.

```
az aks nodepool add --name hostencrypt --cluster-name myAKSCluster --resource-group myResourceGroup -s Standard_DS2_v2 -l westus2 --enable-encryption-at-host
```

If you want to create new node pools without the host-based encryption feature, you can do so by omitting the `--enable-encryption-at-host` parameter.

Next steps

Review [best practices for AKS cluster security](#) Read more about [host-based encryption](#).

Use the Secrets Store CSI Driver for Kubernetes in an Azure Kubernetes Service (AKS) cluster (preview)

5/11/2021 • 6 minutes to read • [Edit Online](#)

The Secrets Store CSI Driver for Kubernetes allows for the integration of Azure Key Vault as a secrets store with a Kubernetes cluster via a [CSI volume](#).

Prerequisites

- If you don't have an Azure subscription, create a [free account](#) before you begin.
- Before you start, install the latest version of the [Azure CLI](#) and the *aks-preview* extension.

Features

- Mount secrets, keys, and/or certs to a pod using a CSI volume
- Supports CSI Inline volumes (Kubernetes version v1.15+)
- Supports mounting multiple secrets store objects as a single volume
- Supports pod portability with the SecretProviderClass CRD
- Supports windows containers (Kubernetes version v1.18+)
- Sync with Kubernetes Secrets (Secrets Store CSI Driver v0.0.10+)
- Supports auto rotation of mounted contents and synced Kubernetes secrets (Secrets Store CSI Driver v0.0.15+)

Register the `AKS-AzureKeyVaultSecretsProvider` preview feature

IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

To create an AKS cluster that can use the Secrets Store CSI Driver, you must enable the `AKS-AzureKeyVaultSecretsProvider` feature flag on your subscription.

Register the `AKS-AzureKeyVaultSecretsProvider` feature flag by using the [az feature register](#) command, as shown in the following example:

```
az feature register --namespace "Microsoft.ContainerService" --name "AKS-AzureKeyVaultSecretsProvider"
```

It takes a few minutes for the status to show *Registered*. Verify the registration status by using the [az feature list](#) command:

```
az feature list -o table --query "[?contains(name, 'Microsoft.ContainerService/AKS-AzureKeyVaultSecretsProvider')].{Name:name,State:properties.state}"
```

When ready, refresh the registration of the *Microsoft.ContainerService* resource provider by using the [az provider register](#) command:

```
az provider register --namespace Microsoft.ContainerService
```

Install the aks-preview CLI extension

You also need the *aks-preview* Azure CLI extension version 0.5.9 or later. Install the *aks-preview* Azure CLI extension by using the [az extension add](#) command. If you already have the extension installed, update to the latest available version by using the [az extension update](#) command.

```
# Install the aks-preview extension
az extension add --name aks-preview

# Update the extension to make sure you have the latest version installed
az extension update --name aks-preview
```

Create an AKS cluster with Secrets Store CSI Driver support

NOTE

If you plan to provide access to the cluster via a user-assigned or system-assigned managed identity, enable Azure Active Directory on your cluster with the flag `enable-managed-identity`. See [Use managed identities in Azure Kubernetes Service](#) for more.

First, create an Azure resource group:

```
az group create -n myResourceGroup -l westus
```

To create an AKS cluster with Secrets Store CSI Driver capability, use the [az aks create](#) command with the addon `azure-keyvault-secrets-provider`:

```
az aks create -n myAKScluster -g myResourceGroup --enable-addons azure-keyvault-secrets-provider
```

Upgrade an existing AKS cluster with Secrets Store CSI Driver support

NOTE

If you plan to provide access to the cluster via a user-assigned or system-assigned managed identity, enable Azure Active Directory on your cluster with the flag `enable-managed-identity`. See [Use managed identities in Azure Kubernetes Service](#) for more.

To upgrade an existing AKS cluster with Secrets Store CSI Driver capability, use the [az aks enable-addons](#) command with the addon `azure-keyvault-secrets-provider`:

```
az aks enable-addons --addons azure-keyvault-secrets-provider --name myAKSCluster --resource-group myResourceGroup
```

Verify Secrets Store CSI Driver installation

These commands will install the Secrets Store CSI Driver and the Azure Key Vault provider on your nodes. Verify by listing all pods with the secrets-store-csi-driver and secrets-store-provider-azure labels in the kube-system namespace and ensuring your output looks similar to the following:

```
kubectl get pods -n kube-system -l 'app in (secrets-store-csi-driver, secrets-store-provider-azure)'

NAMESPACE     NAME                               READY   STATUS    RESTARTS   AGE
kube-system   aks-secrets-store-csi-driver-4vpkj   3/3    Running   2          4m25s
kube-system   aks-secrets-store-csi-driver-ctjq6    3/3    Running   2          4m21s
kube-system   aks-secrets-store-csi-driver-tlvlq    3/3    Running   2          4m24s
kube-system   aks-secrets-store-provider-azure-5p4nb 1/1    Running   0          4m21s
kube-system   aks-secrets-store-provider-azure-6pqmv 1/1    Running   0          4m24s
kube-system   aks-secrets-store-provider-azure-f5qlm  1/1    Running   0          4m25s
```

Enabling and disabling autorotation

NOTE

When enabled, the Secrets Store CSI Driver will update the pod mount and the Kubernetes Secret defined in secretObjects of the SecretProviderClass by polling for changes every two minutes.

To enable autorotation of secrets, use the flag `enable-secret-rotation` when creating your cluster:

```
az aks create -n myAKSCluster2 -g myResourceGroup --enable-addons azure-keyvault-secrets-provider --enable-secret-rotation
```

Or update an existing cluster with the addon enabled:

```
az aks update -g myResourceGroup -n myAKSCluster2 --enable-secret-rotation
```

To disable, use the flag `disable-secret-rotation`:

```
az aks update -g myResourceGroup -n myAKSCluster2 --disable-secret-rotation
```

Create or use an existing Azure Key Vault

In addition to an AKS cluster, you will need an Azure Key Vault resource containing the secret content. To deploy an Azure Key Vault instance, follow these steps:

1. [Create a key vault](#)
2. [Set a secret in a key vault](#)

Take note of the following properties for use in the next section:

- Name of secret object in Key Vault
- Secret content type (secret, key, cert)

- Name of Key Vault resource
- Azure Tenant ID the Subscription belongs to

Provide identity to access Azure Key Vault

The example in this article uses a Service Principal, but the Azure Key Vault provider offers four methods of access. Review them and choose the one that best fits your use case. Be aware additional steps may be required depending on the chosen method, such as granting the Service Principal permissions to get secrets from key vault.

- [Service Principal](#)
- [Pod Identity](#)
- [User-assigned Managed Identity](#)
- [System-assigned Managed Identity](#)

Create and apply your own SecretProviderClass object

To use and configure the Secrets Store CSI driver for your AKS cluster, create a SecretProviderClass custom resource.

Here is an example making use of a Service Principal to access the key vault:

```
apiVersion: secrets-store.csi.x-k8s.io/v1alpha1
kind: SecretProviderClass
metadata:
  name: azure-kvname
spec:
  provider: azure
  parameters:
    usePodIdentity: "false"          # [OPTIONAL] if not provided, will default to "false"
    keyvaultName: "kvname"           # the name of the KeyVault
    cloudName: ""                   # [OPTIONAL for Azure] if not provided, azure environment will default
    to AzurePublicCloud
    objects: |
      array:
        - |
          objectName: secret1
          objectType: secret      # object types: secret, key or cert
          objectVersion: ""        # [OPTIONAL] object versions, default to latest if empty
        - |
          objectName: key1
          objectType: key
          objectVersion: ""
    tenantId: "<tenant-id>"          # the tenant ID of the KeyVault
```

For more information, see [Create your own SecretProviderClass Object](#). Be sure to use the values you took note of above.

Apply the SecretProviderClass to your cluster

Next, deploy the SecretProviderClass you created. For example:

```
kubectl apply -f ./new-secretproviderclass.yaml
```

Update and apply your cluster's deployment YAML

To ensure your cluster is using the new custom resource, update the deployment YAML. For a more comprehensive example, take a look at a [sample deployment](#) using Service Principal to access Azure Key Vault.

Be sure to follow any additional steps from your chosen method of key vault access.

```
kind: Pod
apiVersion: v1
metadata:
  name: busybox-secrets-store-inline
spec:
  containers:
    - name: busybox
      image: k8s.gcr.io/e2e-test-images/busybox:1.29
      command:
        - "/bin/sleep"
        - "10000"
      volumeMounts:
        - name: secrets-store-inline
          mountPath: "/mnt/secrets-store"
          readOnly: true
      volumes:
        - name: secrets-store-inline
          csi:
            driver: secrets-store.csi.k8s.io
            readOnly: true
            volumeAttributes:
              secretProviderClass: "azure-kvname"
            nodePublishSecretRef:           # Only required when using service principal mode
              name: secrets-store-creds     # Only required when using service principal mode. The
                                            name of the Kubernetes secret that contains the service principal credentials to access keyvault.
```

Apply the updated deployment to the cluster:

```
kubectl apply -f ./my-deployment.yaml
```

Validate the secrets

After the pod starts, the mounted content at the volume path specified in your deployment YAML is available.

```
## show secrets held in secrets-store
kubectl exec busybox-secrets-store-inline -- ls /mnt/secrets-store/

## print a test secret 'secret1' held in secrets-store
kubectl exec busybox-secrets-store-inline -- cat /mnt/secrets-store/secret1
```

Disable Secrets Store CSI Driver on an existing AKS Cluster

To disable the Secrets Store CSI Driver capability in an existing cluster, use the [az aks disable-addons](#) command with the `azure-keyvault-secrets-provider` flag:

```
az aks disable-addons --addons azure-keyvault-secrets-provider -g myResourceGroup -n myAKSCluster
```

Next steps

After learning how to use the CSI Secrets Store Driver with an AKS Cluster, see the following resources:

- [Enable CSI drivers for Azure Disks and Azure Files on AKS](#)

View cluster metrics for Azure Kubernetes Service (AKS)

3/30/2021 • 2 minutes to read • [Edit Online](#)

AKS provides a set of metrics for the control plane, including the API Server and cluster autoscaler, and cluster nodes. These metrics allow you to monitor the health of your cluster and troubleshoot issues. You can view the metrics for your cluster using the Azure portal.

NOTE

These AKS cluster metrics overlap with a subset of the [metrics provided by Kubernetes](#).

View metrics for your AKS cluster using the Azure portal

To view the metrics for your AKS cluster:

1. Sign in to the [Azure portal](#) and navigate to your AKS cluster.
2. On the left side under *Monitoring*, select *Metrics*.
3. Create a chart for the metrics you want to view. For example, create a chart:
 - a. For *Scope*, choose your cluster.
 - b. For *Metric Namespace*, choose *Container service (managed) standard metrics*.
 - c. For *Metric*, under *Pods* choose *Number of Pods by phase*.
 - d. For *Aggregation* choose *Avg*.



The above example shows the metrics for the average number of pods for the *myAKSCluster*.

Available metrics

The following cluster metrics are available:

NAME	GROUP	ID	DESCRIPTION
Inflight Requests	API Server (preview)	apiserver_current_inflight_requests	Maximum number of currently active inflight requests on the API Server per request kind.
Cluster Health	Cluster Autoscaler (preview)	cluster_autoscaler_cluster_safe_to_autoscale	Determines whether or not cluster autoscaler will take action on the cluster.
Scale Down Cooldown	Cluster Autoscaler (preview)	cluster_autoscaler_scale_down_in_coldown	Determines if the scale down is in cooldown - No nodes will be removed during this timeframe.
Unneeded Nodes	Cluster Autoscaler (preview)	cluster_autoscaler_unneeded_nodes_count	Cluster autoscaler marks those nodes as candidates for deletion and are eventually deleted.
Unschedulable Pods	Cluster Autoscaler (preview)	cluster_autoscaler_unschedulable_pods_count	Number of pods that are currently unschedulable in the cluster.

NAME	GROUP	ID	DESCRIPTION
Total number of available cpu cores in a managed cluster	Nodes	kube_node_status_allocatable_cpu_cores	Total number of available CPU cores in a managed cluster.
Total amount of available memory in a managed cluster	Nodes	kube_node_status_allocatable_memory_bytes	Total amount of available memory in a managed cluster.
Statuses for various node conditions	Nodes	kube_node_status_condition	Statuses for various node conditions
CPU Usage Millicores	Nodes (preview)	node_cpu_usage_millicores	Aggregated measurement of CPU utilization in millicores across the cluster.
CPU Usage Percentage	Nodes (preview)	node_cpu_usage_percentage	Aggregated average CPU utilization measured in percentage across the cluster.
Memory RSS Bytes	Nodes (preview)	node_memory_rss_bytes	Container RSS memory used in bytes.
Memory RSS Percentage	Nodes (preview)	node_memory_rss_percentage	Container RSS memory used in percent.
Memory Working Set Bytes	Nodes (preview)	node_memory_working_set_bytes	Container working set memory used in bytes.
Memory Working Set Percentage	Nodes (preview)	node_memory_working_set_percentage	Container working set memory used in percent.
Disk Used Bytes	Nodes (preview)	node_disk_usage_bytes	Disk space used in bytes by device.
Disk Used Percentage	Nodes (preview)	node_disk_usage_percentage	Disk space used in percent by device.
Network In Bytes	Nodes (preview)	node_network_in_bytes	Network received bytes.
Network Out Bytes	Nodes (preview)	node_network_out_bytes	Network transmitted bytes.
Number of pods in Ready state	Pods	kube_pod_status_ready	Number of pods in <i>Ready</i> state.
Number of pods by phase	Pods	kube_pod_status_phase	Number of pods by phase.

IMPORTANT

Metrics in preview can be updated or changed, including their names and descriptions, while in preview.

Next steps

In addition to the cluster metrics for AKS, you can also use Azure Monitor with your AKS cluster. For more information on using Azure Monitor with AKS, see [Azure Monitor for containers](#).

Container insights overview

4/28/2021 • 3 minutes to read • [Edit Online](#)

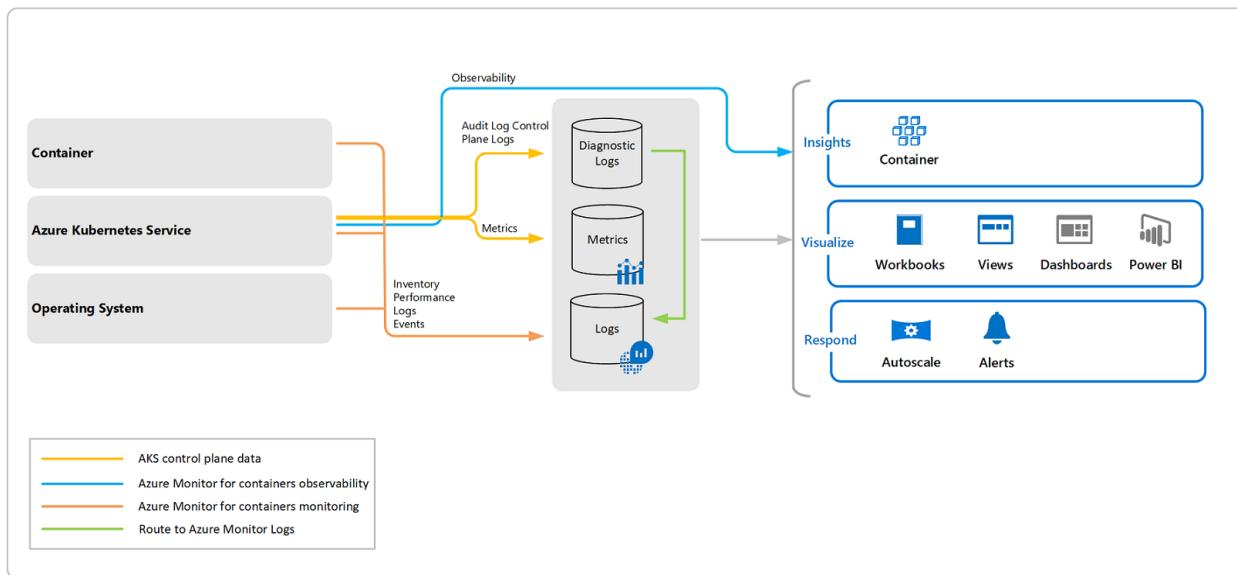
Container insights is a feature designed to monitor the performance of container workloads deployed to:

- Managed Kubernetes clusters hosted on [Azure Kubernetes Service \(AKS\)](#)
- Self-managed Kubernetes clusters hosted on Azure using [AKS Engine](#)
- [Azure Container Instances](#)
- Self-managed Kubernetes clusters hosted on [Azure Stack](#) or on-premises
- [Azure Red Hat OpenShift](#)
- [Azure Arc enabled Kubernetes](#) (preview)

Container insights supports clusters running the Linux and Windows Server 2019 operating system. The container runtimes it supports are Docker, Moby, and any CRI compatible runtime such as CRI-O and ContainerD.

Monitoring your containers is critical, especially when you're running a production cluster, at scale, with multiple applications.

Container insights gives you performance visibility by collecting memory and processor metrics from controllers, nodes, and containers that are available in Kubernetes through the Metrics API. Container logs are also collected. After you enable monitoring from Kubernetes clusters, metrics and logs are automatically collected for you through a containerized version of the Log Analytics agent for Linux. Metrics are written to the metrics store and log data is written to the logs store associated with your [Log Analytics](#) workspace.



What does Container insights provide?

Container insights delivers a comprehensive monitoring experience using different features of Azure Monitor. These features enable you to understand the performance and health of your Kubernetes cluster running Linux and Windows Server 2019 operating system, and the container workloads. With Container insights you can:

- Identify AKS containers that are running on the node and their average processor and memory utilization. This knowledge can help you identify resource bottlenecks.
- Identify processor and memory utilization of container groups and their containers hosted in Azure

Container Instances.

- Identify where the container resides in a controller or a pod. This knowledge can help you view the controller's or pod's overall performance.
- Review the resource utilization of workloads running on the host that are unrelated to the standard processes that support the pod.
- Understand the behavior of the cluster under average and heaviest loads. This knowledge can help you identify capacity needs and determine the maximum load that the cluster can sustain.
- Configure alerts to proactively notify you or record it when CPU and memory utilization on nodes or containers exceed your thresholds, or when a health state change occurs in the cluster at the infrastructure or nodes health rollup.
- Integrate with [Prometheus](#) to view application and workload metrics it collects from nodes and Kubernetes using [queries](#) to create custom alerts, dashboards, and perform detailed analysis.
- Monitor container workloads [deployed to AKS Engine on-premises](#) and [AKS Engine on Azure Stack](#).
- Monitor container workloads [deployed to Azure Red Hat OpenShift](#).

NOTE

Support for Azure Red Hat OpenShift is a feature in public preview at this time.

- Monitor container workloads [deployed to Azure Arc enabled Kubernetes \(preview\)](#).

The main differences in monitoring a Windows Server cluster compared to a Linux cluster are the following:

- Windows doesn't have a Memory RSS metric, and as a result it isn't available for Windows node and containers. The [Working Set](#) metric is available.
- Disk storage capacity information isn't available for Windows nodes.
- Only pod environments are monitored, not Docker environments.
- With the preview release, a maximum of 30 Windows Server containers are supported. This limitation doesn't apply to Linux containers.

Check out the following video providing an intermediate level deep dive to help you learn about monitoring your AKS cluster with Container insights.

How do I access this feature?

You can access Container insights two ways, from Azure Monitor or directly from the selected AKS cluster. From Azure Monitor, you have a global perspective of all the containers deployed, which are monitored and which are not, allowing you to search and filter across your subscriptions and resource groups, and then drill into Container insights from the selected container. Otherwise, you can access the feature directly from a selected AKS container from the AKS page.

The screenshot displays two main sections of the Azure Monitor Container Insights interface. The top section shows a 'Cluster Status Summary' with metrics: Total 56, Critical 5, Warning 1, Unknown 13, Healthy 25, and Unmonitored 12. Below this is a table of monitored clusters, each with columns for Cluster Name, Cluster Type, Version, Status, Nodes, User Pods, and System Pods. The bottom section is titled 'CH-ContosoSH360Cluster | Insights' and shows a detailed view of nodes. It includes tabs for What's new, Cluster, Health (Preview), Nodes (selected), Controllers, Containers, and Deployments (Preview). The 'Nodes' tab displays a table with columns: NAME, STATUS, 95TH %, CONTAINERS, UPTIME, CONTROLLER, and TREND 95TH % (1 BAR = 15M). Three nodes are listed: aks-nodepool1-2360558..., aks-nodepool1-2360558..., and akswindow00000.

If you are interested in monitoring and managing your Docker and Windows container hosts running outside of AKS to view configuration, audit, and resource utilization, see the [Container Monitoring solution](#).

Next steps

To begin monitoring your Kubernetes cluster, review [How to enable Container insights](#) to understand the requirements and available methods to enable monitoring.

Enable and review Kubernetes control plane logs in Azure Kubernetes Service (AKS)

4/21/2021 • 5 minutes to read • [Edit Online](#)

With Azure Kubernetes Service (AKS), the control plane components such as the *kube-apiserver* and *kube-controller-manager* are provided as a managed service. You create and manage the nodes that run the *kubelet* and container runtime, and deploy your applications through the managed Kubernetes API server. To help troubleshoot your application and services, you may need to view the logs generated by these control plane components. This article shows you how to use Azure Monitor logs to enable and query the logs from the Kubernetes control plane components.

Before you begin

This article requires an existing AKS cluster running in your Azure account. If you do not already have an AKS cluster, create one using the [Azure CLI](#) or [Azure portal](#). Azure Monitor logs works with both Kubernetes RBAC, Azure RBAC, and non-RBAC enabled AKS clusters.

Enable resource logs

To help collect and review data from multiple sources, Azure Monitor logs provides a query language and analytics engine that provides insights to your environment. A workspace is used to collate and analyze the data, and can integrate with other Azure services such as Application Insights and Security Center. To use a different platform to analyze the logs, you can instead choose to send resource logs to an Azure storage account or event hub. For more information, see [What is Azure Monitor logs?](#).

Azure Monitor logs are enabled and managed in the Azure portal. To enable log collection for the Kubernetes control plane components in your AKS cluster, open the Azure portal in a web browser and complete the following steps:

1. Select the resource group for your AKS cluster, such as *myResourceGroup*. Don't select the resource group that contains your individual AKS cluster resources, such as *MC_myResourceGroup_myAKSCluster_eastus*.
2. On the left-hand side, choose **Diagnostic settings**.
3. Select your AKS cluster, such as *myAKSCluster*, then choose to **Add diagnostic setting**.

Diagnostic settings are used to configure streaming export of platform logs and metrics for a resource to the destination of your choice. You may create up to five different diagnostic settings to send different logs and metrics to independent destinations. [Learn more about diagnostic settings](#)

Name	Storage account	Event hub	Log Analytics workspace	Edit setting
No diagnostic settings defined				

+ Add diagnostic setting

Click 'Add Diagnostic setting' above to configure the collection of the following data:

- kube-apiserver
- kube-audit
- kube-audit-admin
- kube-controller-manager
- kube-scheduler
- cluster-autoscaler
- guard
- AllMetrics

4. Enter a name, such as *myAKSClusterLogs*, then select the option to **Send to Log Analytics workspace**.
5. Select an existing workspace or create a new one. If you create a workspace, provide a workspace name, a resource group, and a location.
6. In the list of available logs, select the logs you wish to enable. For this example, enable the *kube-audit* and *kube-audit-admin* logs. Common logs include the *kube-apiserver*, *kube-controller-manager*, and *kube-scheduler*. You can return and change the collected logs once Log Analytics workspaces are enabled.
7. When ready, select **Save** to enable collection of the selected logs.

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic setting name *

myAKSCluster Logs

Category details

log

kube-apiserver

kube-audit

kube-audit-admin

kube-controller-manager

kube-scheduler

cluster-autoscaler

guard

metric

AllMetrics

Destination details

Send to Log Analytics workspace

Subscription

Contoso User Subscription

Log Analytics workspace

DefaultWorkspace-aaaaaaaaaaaa-aaaa-aaaa-EUS (eastus)

Archive to a storage account

Stream to an event hub

Log categories

In addition to entries written by Kubernetes, your project's audit logs also have entries from AKS.

Audit logs are recorded into three categories: *kube-audit*, *kube-audit-admin*, and *guard*.

- The *kube-audit* category contains all audit log data for every audit event, including *get*, *list*, *create*, *update*, *delete*, *patch*, and *post*.
- The *kube-audit-admin* category is a subset of the *kube-audit* log category. *kube-audit-admin* reduces the number of logs significantly by excluding the *get* and *list* audit events from the log.
- The *guard* category is managed Azure AD and Azure RBAC audits. For managed Azure AD: token in, user info out. For Azure RBAC: access reviews in and out.

Schedule a test pod on the AKS cluster

To generate some logs, create a new pod in your AKS cluster. The following example YAML manifest can be used to create a basic NGINX instance. Create a file named `nginx.yaml` in an editor of your choice and paste the following content:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  nodeSelector:
    "beta.kubernetes.io/os": linux
  containers:
  - name: mypod
    image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
    resources:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 250m
        memory: 256Mi
    ports:
    - containerPort: 80
```

Create the pod with the [kubectl create](#) command and specify your YAML file, as shown in the following example:

```
$ kubectl create -f nginx.yaml
pod/nginx created
```

View collected logs

It may take up to 10 minutes for the diagnostics logs to be enabled and appear.

NOTE

If you need all audit log data for compliance or other purposes, collect and store it in inexpensive storage such as blob storage. Use the *kube-audit-admin* log category to collect and save a meaningful set of audit log data for monitoring and alerting purposes.

In the Azure portal, navigate to your AKS cluster, and select **Logs** on the left-hand side. Close the *Example Queries* window if it appears.

On the left-hand side, choose **Logs**. To view the *kube-audit* logs, enter the following query in the text box:

```
AzureDiagnostics
| where Category == "kube-audit"
| project log_s
```

Many logs are likely returned. To scope down the query to view the logs about the NGINX pod created in the previous step, add an additional *where* statement to search for *nginx* as shown in the following example query:

```
AzureDiagnostics
| where Category == "kube-audit"
| where log_s contains "nginx"
| project log_s
```

To view the *kube-audit-admin* logs, enter the following query in the text box:

```
AzureDiagnostics  
| where Category == "kube-audit-admin"  
| project log_s
```

In this example, the query shows all create jobs in *kube-audit-admin*. There are likely many results returned, to scope down the query to view the logs about the NGINX pod created in the previous step, add an additional *where* statement to search for *nginx* as shown in the following example query.

```
AzureDiagnostics  
| where Category == "kube-audit-admin"  
| where log_s contains "nginx"  
| project log_s
```

For more information on how to query and filter your log data, see [View or analyze data collected with log analytics log search](#).

Log event schema

AKS logs the following events:

- [AzureActivity](#)
- [AzureDiagnostics](#)
- [AzureMetrics](#)
- [ContainerImageInventory](#)
- [ContainerInventory](#)
- [ContainerLog](#)
- [ContainerNodeInventory](#)
- [ContainerServiceLog](#)
- [Heartbeat](#)
- [InsightsMetrics](#)
- [KubeEvents](#)
- [KubeHealth](#)
- [KubeMonAgentEvents](#)
- [KubeNodeInventory](#)
- [KubePodInventory](#)
- [KubeServices](#)
- [Perf](#)

Log Roles

ROLE	DESCRIPTION
<i>aksService</i>	The display name in audit log for the control plane operation (from the hcpService)
<i>masterclient</i>	The display name in audit log for MasterClientCertificate, the certificate you get from az aks get-credentials
<i>nodeclient</i>	The display name for ClientCertificate, which is used by agent nodes

Next steps

In this article, you learned how to enable and review the logs for the Kubernetes control plane components in your AKS cluster. To monitor and troubleshoot further, you can also [view the Kubelet logs](#) and [enable SSH node access](#).

Get kubelet logs from Azure Kubernetes Service (AKS) cluster nodes

3/5/2021 • 2 minutes to read • [Edit Online](#)

As part of operating an AKS cluster, you may need to review logs to troubleshoot a problem. Built-in to the Azure portal is the ability to view logs for the [AKS master components](#) or [containers in an AKS cluster](#). Occasionally, you may need to get *kubelet* logs from an AKS node for troubleshooting purposes.

This article shows you how you can use `journalctl` to view the *kubelet* logs on an AKS node.

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart [using the Azure CLI](#) or [using the Azure portal](#).

Create an SSH connection

First, create an SSH connection with the node on which you need to view *kubelet* logs. This operation is detailed in the [SSH into Azure Kubernetes Service \(AKS\) cluster nodes](#) document.

Get kubelet logs

Once you have connected to the node, run the following command to pull the *kubelet* logs:

```
sudo journalctl -u kubelet -o cat
```

NOTE

For Windows nodes, the log data is in `C:\k` and can be viewed using the *more* command:

```
more C:\k\kubelet.log
```

The following sample output shows the *kubelet* log data:

```
I0508 12:26:17.905042          8672 kubelet_node_status.go:497] Using Node Hostname from cloudprovider: "aks-
agentpool-11482510-0"
I0508 12:26:27.943494          8672 kubelet_node_status.go:497] Using Node Hostname from cloudprovider: "aks-
agentpool-11482510-0"
I0508 12:26:28.920125          8672 server.go:796] GET /stats/summary: (10.370874ms) 200 [[Ruby] 10.244.0.2:52292]
I0508 12:26:37.964650          8672 kubelet_node_status.go:497] Using Node Hostname from cloudprovider: "aks-
agentpool-11482510-0"
I0508 12:26:47.996449          8672 kubelet_node_status.go:497] Using Node Hostname from cloudprovider: "aks-
agentpool-11482510-0"
I0508 12:26:58.019746          8672 kubelet_node_status.go:497] Using Node Hostname from cloudprovider: "aks-
agentpool-11482510-0"
I0508 12:27:05.107680          8672 server.go:796] GET /stats/summary/: (24.853838ms) 200 [[Go-http-client/1.1]
10.244.0.3:44660]
I0508 12:27:08.041736          8672 kubelet_node_status.go:497] Using Node Hostname from cloudprovider: "aks-
agentpool-11482510-0"
I0508 12:27:18.068505          8672 kubelet_node_status.go:497] Using Node Hostname from cloudprovider: "aks-
agentpool-11482510-0"
I0508 12:27:28.094889          8672 kubelet_node_status.go:497] Using Node Hostname from cloudprovider: "aks-
agentpool-11482510-0"
I0508 12:27:38.121346          8672 kubelet_node_status.go:497] Using Node Hostname from cloudprovider: "aks-
agentpool-11482510-0"
I0508 12:27:44.015205          8672 server.go:796] GET /stats/summary: (30.236824ms) 200 [[Ruby] 10.244.0.2:52588]
I0508 12:27:48.145640          8672 kubelet_node_status.go:497] Using Node Hostname from cloudprovider: "aks-
agentpool-11482510-0"
I0508 12:27:58.178534          8672 kubelet_node_status.go:497] Using Node Hostname from cloudprovider: "aks-
agentpool-11482510-0"
I0508 12:28:05.040375          8672 server.go:796] GET /stats/summary/: (27.78503ms) 200 [[Go-http-client/1.1]
10.244.0.3:44660]
I0508 12:28:08.214158          8672 kubelet_node_status.go:497] Using Node Hostname from cloudprovider: "aks-
agentpool-11482510-0"
I0508 12:28:18.242160          8672 kubelet_node_status.go:497] Using Node Hostname from cloudprovider: "aks-
agentpool-11482510-0"
I0508 12:28:28.274408          8672 kubelet_node_status.go:497] Using Node Hostname from cloudprovider: "aks-
agentpool-11482510-0"
I0508 12:28:38.296074          8672 kubelet_node_status.go:497] Using Node Hostname from cloudprovider: "aks-
agentpool-11482510-0"
I0508 12:28:48.321952          8672 kubelet_node_status.go:497] Using Node Hostname from cloudprovider: "aks-
agentpool-11482510-0"
I0508 12:28:58.344656          8672 kubelet_node_status.go:497] Using Node Hostname from cloudprovider: "aks-
agentpool-11482510-0"
```

Next steps

If you need additional troubleshooting information from the Kubernetes master, see [view Kubernetes master node logs in AKS](#).

How to view Kubernetes logs, events, and pod metrics in real-time

3/5/2021 • 6 minutes to read • [Edit Online](#)

Container insights includes the Live Data feature, which is an advanced diagnostic feature allowing you direct access to your Azure Kubernetes Service (AKS) container logs (stdout/stderror), events, and pod metrics. It exposes direct access to `kubectl logs -c`, `kubectl get` events, and `kubectl top pods`. A console pane shows the logs, events, and metrics generated by the container engine to further assist in troubleshooting issues in real-time.

This article provides a detailed overview and helps you understand how to use this feature.

For help setting up or troubleshooting the Live Data feature, review our [setup guide](#). This feature directly access the Kubernetes API, and additional information about the authentication model can be found [here](#).

View AKS resource live logs

Use the following procedure to view the live logs for pods, deployments, and replica sets with or without Container insights from the AKS resource view.

1. In the Azure portal, browse to the AKS cluster resource group and select your AKS resource.
 2. Select **Workloads** in the **Kubernetes resources** section of the menu.
 3. Select a pod, deployment, replica-set from the respective tab.
 4. Select **Live Logs** from the resource's menu.
 5. Select a pod to start collection of the live data.

View logs

You can view real-time log data as they are generated by the container engine from the **Nodes**, **Controllers**, and **Containers** view. To view log data, perform the following steps.

1. In the Azure portal, browse to the AKS cluster resource group and select your AKS resource.

2. On the AKS cluster dashboard, under **Monitoring** on the left-hand side, choose **Insights**.
3. Select either the **Nodes**, **Controllers**, or **Containers** tab.
4. Select an object from the performance grid, and on the properties pane found on the right side, select **View live data** option. If the AKS cluster is configured with single sign-on using Azure AD, you are prompted to authenticate on first use during that browser session. Select your account and complete authentication with Azure.

NOTE

When viewing the data from your Log Analytics workspace by selecting the **View in analytics** option from the properties pane, the log search results will potentially show **Nodes**, **Daemon Sets**, **Replica Sets**, **Jobs**, **Cron Jobs**, **Pods**, and **Containers** which may no longer exist. Attempting to search logs for a container which isn't available in `kubectl` will also fail here. Review the **View in analytics** feature to learn more about viewing historical logs, events and metrics.

After successfully authenticating, the Live Data console pane will appear below the performance data grid where you can view log data in a continuous stream. If the fetch status indicator shows a green check mark, which is on the far right of the pane, it means data can be retrieved and it begins streaming to your console.



The pane title shows the name of the pod the container is grouped with.

View events

You can view real-time event data as they are generated by the container engine from the **Nodes**, **Controllers**, **Containers**, and **Deployments** view when a container, pod, node, ReplicaSet, DaemonSet, job, CronJob or Deployment is selected. To view events, perform the following steps.

1. In the Azure portal, browse to the AKS cluster resource group and select your AKS resource.
2. On the AKS cluster dashboard, under **Monitoring** on the left-hand side, choose **Insights**.
3. Select either the **Nodes**, **Controllers**, **Containers**, or **Deployments** tab.
4. Select an object from the performance grid, and on the properties pane found on the right side, select

View live data option. If the AKS cluster is configured with single sign-on using Azure AD, you are prompted to authenticate on first use during that browser session. Select your account and complete authentication with Azure.

NOTE

When viewing the data from your Log Analytics workspace by selecting the **View in analytics** option from the properties pane, the log search results will potentially show **Nodes**, **Daemon Sets**, **Replica Sets**, **Jobs**, **Cron Jobs**, **Pods**, and **Containers** which may no longer exist. Attempting to search logs for a container which isn't available in `kubectl` will also fail here. Review the **View in analytics** feature to learn more about viewing historical logs, events and metrics.

After successfully authenticating, the Live Data console pane will appear below the performance data grid. If the fetch status indicator shows a green check mark, which is on the far right of the pane, it means data can be retrieved and it begins streaming to your console.

If the object you selected was a container, select the **Events** option in the pane. If you selected a Node, Pod, or controller, viewing events is automatically selected.



The pane title shows the name of the Pod the container is grouped with.

Filter events

While viewing events, you can additionally limit the results using the **Filter** pill found to the right of the search bar. Depending on what resource you have selected, the pill lists a Pod, Namespace, or cluster to choose from.

View metrics

You can view real-time metric data as they are generated by the container engine from the **Nodes** or **Controllers** view only when a **Pod** is selected. To view metrics, perform the following steps.

1. In the Azure portal, browse to the AKS cluster resource group and select your AKS resource.
2. On the AKS cluster dashboard, under **Monitoring** on the left-hand side, choose **Insights**.
3. Select either the **Nodes** or **Controllers** tab.

- Select a **Pod** object from the performance grid, and on the properties pane found on the right side, select **View live data** option. If the AKS cluster is configured with single sign-on using Azure AD, you are prompted to authenticate on first use during that browser session. Select your account and complete authentication with Azure.

NOTE

When viewing the data from your Log Analytics workspace by selecting the **View in analytics** option from the properties pane, the log search results will potentially show **Nodes**, **Daemon Sets**, **Replica Sets**, **Jobs**, **Cron Jobs**, **Pods**, and **Containers** which may no longer exist. Attempting to search logs for a container which isn't available in `kubectl` will also fail here. Review the [View in analytics](#) feature to learn more about viewing historical logs, events and metrics.

After successfully authenticating, the Live Data console pane will appear below the performance data grid. Metric data is retrieved and begins streaming to your console for presentation in the two charts. The pane title shows the name of the pod the container is grouped with.



Using live data views

The following sections describe functionality that you can use in the different live data views.

Search

The Live Data feature includes search functionality. In the **Search** field, you can filter results by typing a key word or term and any matching results are highlighted to allow quick review. While viewing events, you can additionally limit the results using the **Filter** pill found to the right of the search bar. Depending on what resource you have selected, the pill lists a Pod, Namespace, or cluster to chose from.

Pod name: kube-svc-redirect-t8b9l (redirector)

Logs Events (No New Data)

destination | 1/16 X ^ v

Scrub | Scroll || Pause Clear

```
2019-10-21T16:54:06.137209651Z target prot opt source destination
2019-10-21T16:54:06.137214751Z DNAT tcp -- anywhere 10.0.0.1 to:127.0.0.1:14612
2019-10-21T16:54:06.137218452Z RETURN all -- anywhere anywhere
2019-10-21T16:54:06.141669262Z [ Mon Oct 21 16:54:06 UTC 2019 ] INF: found expected rule count in chain:aks-hcp-custom-svc
2019-10-21T16:54:06.141682766Z [ Mon Oct 21 16:54:06 UTC 2019 ] INF: Will validate the following rules:
2019-10-21T16:54:06.141687066Z ++++++
2019-10-21T16:54:06.14534646Z Chain aks-hcp-custom-svc (2 references)
2019-10-21T16:54:06.14535986Z num target prot opt source destination
2019-10-21T16:54:06.14536416Z 1 DNAT tcp -- anywhere 10.0.0.1 to:127.0.0.1:14612
2019-10-21T16:54:06.14536776Z 2 RETURN all -- anywhere anywhere
2019-10-21T16:54:06.14537116Z ++++++
2019-10-21T16:54:06.150201984Z [ Mon Oct 21 16:54:06 UTC 2019 ] INF: Rule #1 match expectation with [1 DNAT tcp -- anywhere 10.0.0.1 to:127.0.0.1:14612]
2019-10-21T16:54:06.156271639Z [ Mon Oct 21 16:54:06 UTC 2019 ] INF: Rule #2 match expectation with [2 RETURN all -- anywhere anywhere ]
2019-10-21T16:54:06.157287965Z [ Mon Oct 21 16:54:06 UTC 2019 ] INF: chain aks-hcp-custom-svc is valid
2019-10-21T16:54:06.16411234Z [ Mon Oct 21 16:54:06 UTC 2019 ] INF: Jump rule aks-hcp-custom-svc was found at position #1 in OUTPUT
2019-10-21T16:54:06.165113766Z [ Mon Oct 21 16:54:06 UTC 2019 ] INF: Rules in OUTPUT are:
2019-10-21T16:54:06.16526637Z ++++++
2019-10-21T16:54:06.168171844Z Chain OUTPUT (policy ACCEPT)
2019-10-21T16:54:06.168183344Z num target prot opt source destination
2019-10-21T16:54:06.168187145Z 1 aks-hcp-custom-svc all -- anywhere anywhere
```

Scroll Lock and Pause

To suspend autoscroll and control the behavior of the pane, allowing you to manually scroll through the new data read, you can use the **Scroll** option. To re-enable autoscroll, simply select the **Scroll** option again. You can also pause retrieval of log or event data by selecting the the **Pause** option, and when you are ready to resume, simply select **Play**.

Pod name: kube-svc-redirect-4w7qq (redirector)

Logs Events (Paused)

Search... 0/0

File Scroll

```
2019-10-21T17:09:27.324781595Z [ Mon Oct 21 17:09:27 UTC 2019 ] INF: Rule #2 match expectation with [2 KEYWORD all -- anywhere anywhere ]  
2019-10-21T17:09:27.328073512Z [ Mon Oct 21 17:09:27 UTC 2019 ] INF: chain aks-hcp-custom-svc is valid  
2019-10-21T17:09:27.335665584Z [ Mon Oct 21 17:09:27 UTC 2019 ] INF: Jump rule aks-hcp-custom-svc was found at position #1 in OUTPUT  
2019-10-21T17:09:27.33693453Z [ Mon Oct 21 17:09:27 UTC 2019 ] INF: Rules in OUTPUT are:  
2019-10-21T17:09:27.33694353Z ++++++  
2019-10-21T17:09:27.342360624Z Chain OUTPUT (policy ACCEPT)  
2019-10-21T17:09:27.342371725Z num target prot opt source destination  
2019-10-21T17:09:27.342375125Z 1 aks-hcp-custom-svc all -- anywhere anywhere  
2019-10-21T17:09:27.342377925Z 2 KUBE-SERVICES all -- anywhere anywhere /* kubernetes service portals */  
2019-10-21T17:09:27.342380825Z 3 DOCKER all -- anywhere !127.0.0.0/8 ADDRTYPE match dst-type LOCAL  
2019-10-21T17:09:27.34251453Z ++++++  
2019-10-21T17:09:27.350940932Z [ Mon Oct 21 17:09:27 UTC 2019 ] INF: Jump rule aks-hcp-custom-svc was found at position #1 in PREROUTING  
2019-10-21T17:09:27.352168276Z [ Mon Oct 21 17:09:27 UTC 2019 ] INF: Rules in PREROUTING are:  
2019-10-21T17:09:27.352211677Z ++++++  
2019-10-21T17:09:27.353607627Z Chain PREROUTING (policy ACCEPT)  
2019-10-21T17:09:27.353618727Z num target prot opt source destination  
2019-10-21T17:09:27.353622128Z 1 aks-hcp-custom-svc all -- anywhere anywhere  
2019-10-21T17:09:27.353625428Z 2 KUBE-SERVICES all -- anywhere anywhere /* kubernetes service portals */  
2019-10-21T17:09:27.353628828Z 3 DOCKER all -- anywhere anywhere ADDRTYPE match dst-type LOCAL  
2019-10-21T17:09:27.353742332Z ++++++  
2019-10-21T17:09:27.35492974Z [ Mon Oct 21 17:09:27 UTC 2019 ] INF: Done - going to sleep for 10
```

Home > my-service > my-deployment

my-deployment | Live Logs (preview)

Deployment

Search (Ctrl+F)

Refresh View in Log Analytics

Overview

YAML

Events

Insights

Live Logs (preview)

Filter grid data

Filter by Pod

my-pod-0000000000-00000

Looking for historical logs? View in Log Analytics

12 item(s). Streaming is paused

Play

Scroll

IMPORTANT

We recommend only suspending or pausing autoscroll for a short period of time while troubleshooting an issue. These requests may impact the availability and throttling of the Kubernetes API on your cluster.

IMPORTANT

No data is stored permanently during operation of this feature. All information captured during the session is deleted when you close your browser or navigate away from it. Data only remains present for visualization inside the five minute window of the metrics feature; any metrics older than five minutes are also deleted. The Live Data buffer queries within reasonable memory usage limits.

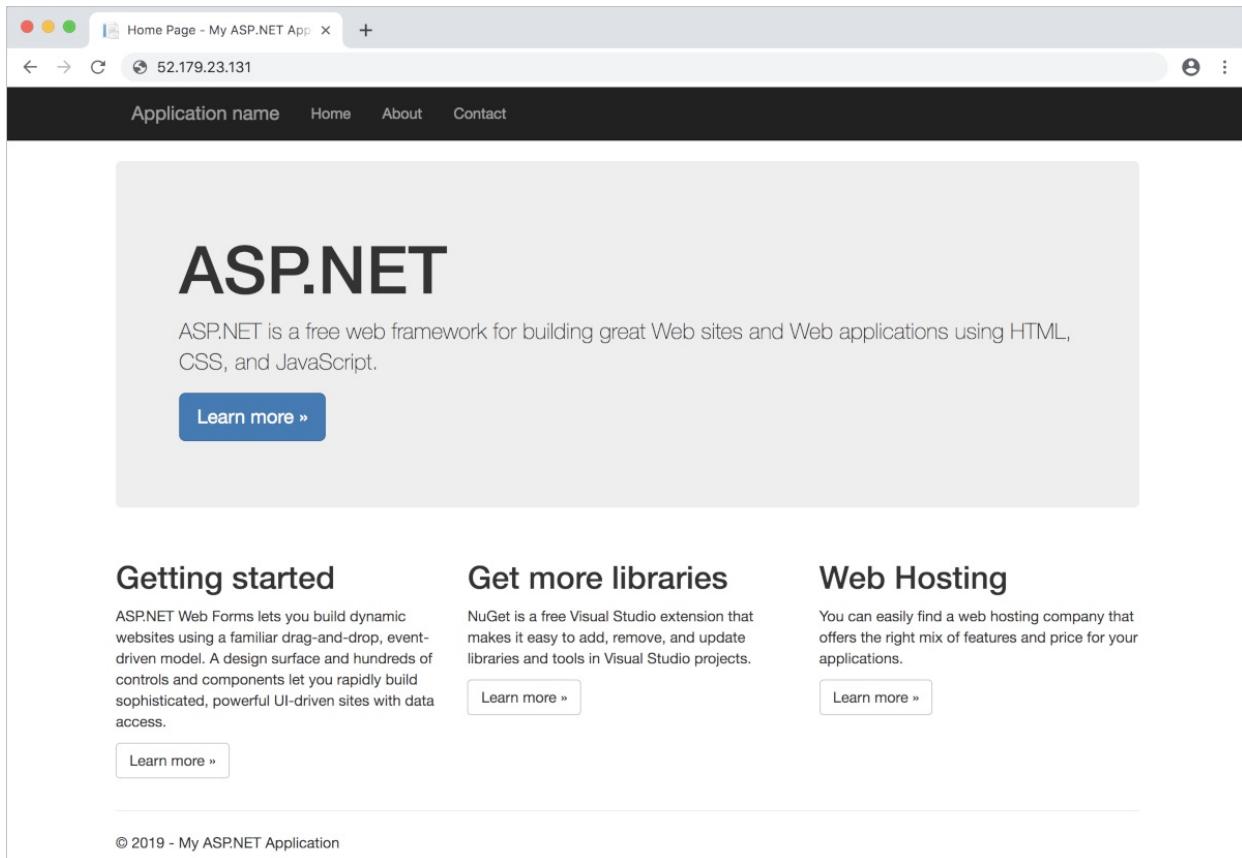
Next steps

- To continue learning how to use Azure Monitor and monitor other aspects of your AKS cluster, see [View Azure Kubernetes Service health](#).
- View [log query examples](#) to see predefined queries and examples to create alerts, visualizations, or perform further analysis of your clusters.

Create a Windows Server container on an Azure Kubernetes Service (AKS) cluster using the Azure CLI

5/11/2021 • 8 minutes to read • [Edit Online](#)

Azure Kubernetes Service (AKS) is a managed Kubernetes service that lets you quickly deploy and manage clusters. In this article, you deploy an AKS cluster using the Azure CLI. You also deploy an ASP.NET sample application in a Windows Server container to the cluster.



This article assumes a basic understanding of Kubernetes concepts. For more information, see [Kubernetes core concepts for Azure Kubernetes Service \(AKS\)](#).

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

Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#).

 [Launch Cloud Shell](#)

- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
 - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
 - When you're prompted, install Azure CLI extensions on first use. For more information about

extensions, see [Use extensions with the Azure CLI](#).

- Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.

Limitations

The following limitations apply when you create and manage AKS clusters that support multiple node pools:

- You can't delete the first node pool.

The following additional limitations apply to Windows Server node pools:

- The AKS cluster can have a maximum of 10 node pools.
- The AKS cluster can have a maximum of 100 nodes in each node pool.
- The Windows Server node pool name has a limit of 6 characters.

Create a resource group

An Azure resource group is a logical group in which Azure resources are deployed and managed. When you create a resource group, you are asked to specify a location. This location is where resource group metadata is stored, it is also where your resources run in Azure if you don't specify another region during resource creation. Create a resource group using the `az group create` command.

The following example creates a resource group named `myResourceGroup` in the `eastus` location.

NOTE

This article uses Bash syntax for the commands in this tutorial. If you are using Azure Cloud Shell, ensure that the dropdown in the upper-left of the Cloud Shell window is set to **Bash**.

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

The following example output shows the resource group created successfully:

```
{
  "id": "/subscriptions/<guid>/resourceGroups/myResourceGroup",
  "location": "eastus",
  "managedBy": null,
  "name": "myResourceGroup",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": null
}
```

Create an AKS cluster

To run an AKS cluster that supports node pools for Windows Server containers, your cluster needs to use a network policy that uses [Azure CNI](#) (advanced) network plugin. For more detailed information to help plan out the required subnet ranges and network considerations, see [configure Azure CNI networking](#). Use the `az aks create` command to create an AKS cluster named `myAKSCluster`. This command will create the necessary network resources if they don't exist.

- The cluster is configured with two nodes.

- The `--windows-admin-password` and `--windows-admin-username` parameters set the administrator credentials for any Windows Server nodes on the cluster and must meet [Windows Server password requirements](#). If you don't specify the `windows-admin-password` parameter, you will be prompted to provide a value.
- The node pool uses `VirtualMachineScaleSets`.

NOTE

To ensure your cluster to operate reliably, you should run at least 2 (two) nodes in the default node pool.

Create a username to use as administrator credentials for the Windows Server nodes on your cluster. The following commands prompt you for a username and set it `WINDOWS_USERNAME` for use in a later command (remember that the commands in this article are entered into a BASH shell).

```
echo "Please enter the username to use as administrator credentials for Windows Server nodes on your cluster: " && read WINDOWS_USERNAME
```

Create your cluster ensuring you specify `--windows-admin-username` parameter. The following example command creates a cluster using the value from `WINDOWS_USERNAME` you set in the previous command. Alternatively you can provide a different username directly in the parameter instead of using `WINDOWS_USERNAME`. The following command will also prompt you to create a password for the administrator credentials for the Windows Server nodes on your cluster. Alternatively, you can use the `windows-admin-password` parameter and specify your own value there.

```
az aks create \
  --resource-group myResourceGroup \
  --name myAKSCluster \
  --node-count 2 \
  --enable-addons monitoring \
  --generate-ssh-keys \
  --windows-admin-username $WINDOWS_USERNAME \
  --vm-set-type VirtualMachineScaleSets \
  --network-plugin azure
```

NOTE

If you get a password validation error, verify the password you set meets the [Windows Server password requirements](#). If your password meets the requirements, try creating your resource group in another region. Then try creating the cluster with the new resource group.

If you do not specify an administrator username and password when setting `--vm-set-type VirtualMachineScaleSets` and `--network-plugin azure`, the username is set to `azureuser` and the password is set to a random value.

The administrator username can't be changed, but you can change the administrator password your AKS cluster uses for Windows Server nodes using `az aks update`. For more details, see [Windows Server node pools FAQ](#).

After a few minutes, the command completes and returns JSON-formatted information about the cluster. Occasionally the cluster can take longer than a few minutes to provision. Allow up to 10 minutes in these cases.

Add a Windows Server node pool

By default, an AKS cluster is created with a node pool that can run Linux containers. Use `az aks nodepool add` command to add an additional node pool that can run Windows Server containers alongside the Linux node pool.

```
az aks nodepool add \
    --resource-group myResourceGroup \
    --cluster-name myAKSCluster \
    --os-type Windows \
    --name npwin \
    --node-count 1
```

The above command creates a new node pool named *npwin* and adds it to the *myAKSCluster*. When creating a node pool to run Windows Server containers, the default value for *node-vm-size* is *Standard_D2s_v3*. If you choose to set the *node-vm-size* parameter, please check the list of [restricted VM sizes](#). The minimum recommended size is *Standard_D2s_v3*. The above command also uses the default subnet in the default vnet created when running `az aks create`.

Connect to the cluster

To manage a Kubernetes cluster, you use [kubectl](#), the Kubernetes command-line client. If you use Azure Cloud Shell, `kubectl` is already installed. To install `kubectl` locally, use the [az aks install-cli](#) command:

```
az aks install-cli
```

To configure `kubectl` to connect to your Kubernetes cluster, use the [az aks get-credentials](#) command. This command downloads credentials and configures the Kubernetes CLI to use them.

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```

To verify the connection to your cluster, use the [kubectl get](#) command to return a list of the cluster nodes.

```
kubectl get nodes
```

The following example output shows the all the nodes in the cluster. Make sure that the status of all nodes is *Ready*:

NAME	STATUS	ROLES	AGE	VERSION
aks-nodepool1-12345678-vmssfedcba	Ready	agent	13m	v1.16.9
aksnpwin987654	Ready	agent	108s	v1.16.9

Run the application

A Kubernetes manifest file defines a desired state for the cluster, such as what container images to run. In this article, a manifest is used to create all objects needed to run the ASP.NET sample application in a Windows Server container. This manifest includes a [Kubernetes deployment](#) for the ASP.NET sample application and an external [Kubernetes service](#) to access the application from the internet.

The ASP.NET sample application is provided as part of the [.NET Framework Samples](#) and runs in a Windows Server container. AKS requires Windows Server containers to be based on images of *Windows Server 2019* or greater. The Kubernetes manifest file must also define a [node selector](#) to tell your AKS cluster to run your ASP.NET sample application's pod on a node that can run Windows Server containers.

Create a file named `sample.yaml` and copy in the following YAML definition. If you use the Azure Cloud Shell, this file can be created using `vi` or `nano` as if working on a virtual or physical system:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample
  labels:
    app: sample
spec:
  replicas: 1
  template:
    metadata:
      name: sample
      labels:
        app: sample
    spec:
      nodeSelector:
        "beta.kubernetes.io/os": windows
      containers:
        - name: sample
          image: mcr.microsoft.com/dotnet/framework/samples:aspnetapp
          resources:
            limits:
              cpu: 1
              memory: 800M
            requests:
              cpu: .1
              memory: 300M
          ports:
            - containerPort: 80
      selector:
        matchLabels:
          app: sample
---
apiVersion: v1
kind: Service
metadata:
  name: sample
spec:
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 80
  selector:
    app: sample
```

Deploy the application using the [kubectl apply](#) command and specify the name of your YAML manifest:

```
kubectl apply -f sample.yaml
```

The following example output shows the Deployment and Service created successfully:

```
deployment.apps/sample created
service/sample created
```

Test the application

When the application runs, a Kubernetes service exposes the application front end to the internet. This process can take a few minutes to complete. Occasionally the service can take longer than a few minutes to provision. Allow up to 10 minutes in these cases.

To monitor progress, use the [kubectl get service](#) command with the `--watch` argument.

```
kubectl get service sample --watch
```

Initially the *EXTERNAL-IP* for the *sample* service is shown as *pending*.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
sample	LoadBalancer	10.0.37.27	<pending>	80:30572/TCP	6s

When the *EXTERNAL-IP* address changes from *pending* to an actual public IP address, use `CTRL-C` to stop the `kubectl` watch process. The following example output shows a valid public IP address assigned to the service:

```
sample  LoadBalancer  10.0.37.27  52.179.23.131  80:30572/TCP  2m
```

To see the sample app in action, open a web browser to the external IP address of your service.

The screenshot shows a web browser window with the following details:

- Title Bar:** Home Page - My ASP.NET App
- Address Bar:** 52.179.23.131
- Page Content:**
 - ASP.NET Logo:** ASP.NET
 - Description:** ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS, and JavaScript.
 - Learn more > button:** A blue button located below the description.
 - Getting started:** ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.
 - Get more libraries:** NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.
 - Web Hosting:** You can easily find a web hosting company that offers the right mix of features and price for your applications.
- Footer:** © 2019 - My ASPNET Application

NOTE

If you receive a connection timeout when trying to load the page then you should verify the sample app is ready with the following command [kubectl get pods --watch]. Sometimes the windows container will not be started by the time your external IP address is available.

Delete cluster

When the cluster is no longer needed, use the `az group delete` command to remove the resource group, container service, and all related resources.

```
az group delete --name myResourceGroup --yes --no-wait
```

NOTE

When you delete the cluster, the Azure Active Directory service principal used by the AKS cluster is not removed. For steps on how to remove the service principal, see [AKS service principal considerations and deletion](#). If you used a managed identity, the identity is managed by the platform and does not require removal.

Next steps

In this article, you deployed a Kubernetes cluster and deployed an ASP.NET sample application in a Windows Server container to it. [Access the Kubernetes web dashboard](#) for the cluster you just created.

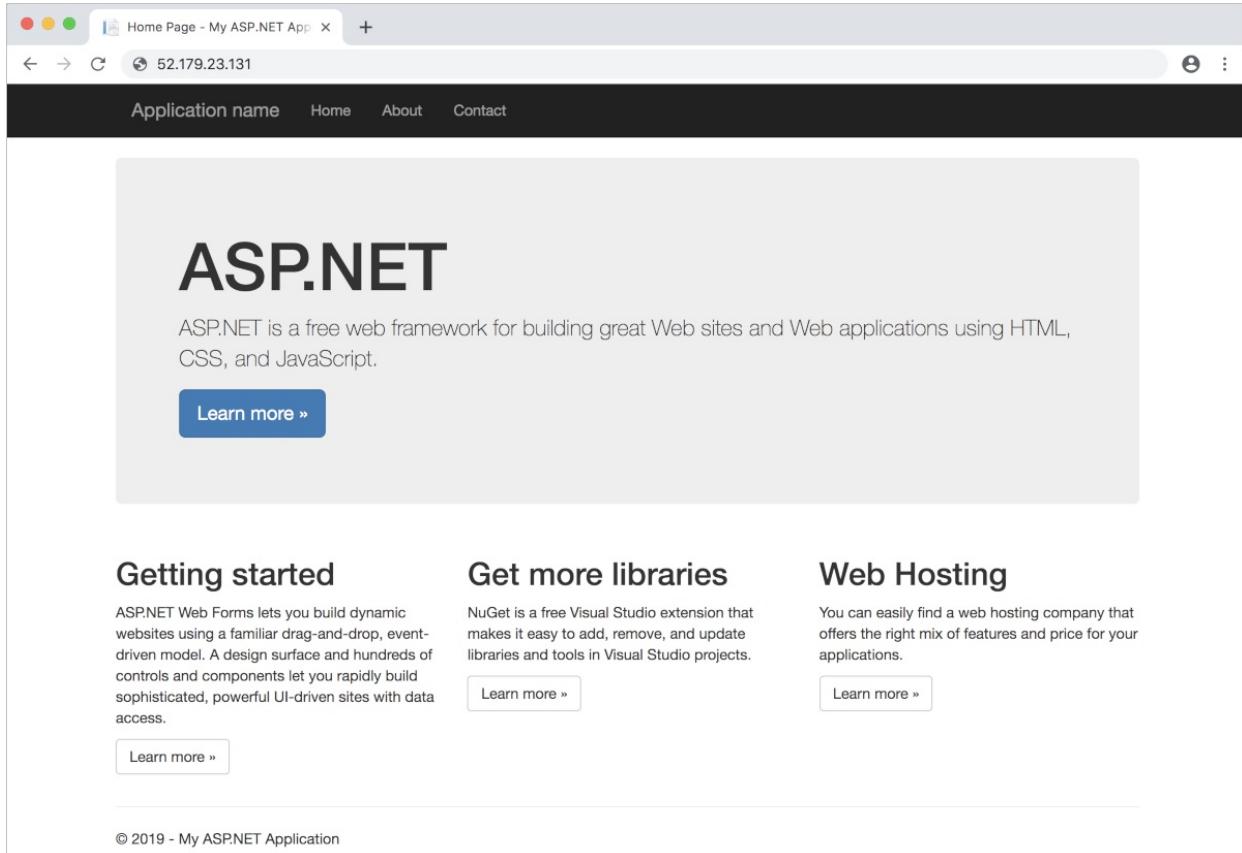
To learn more about AKS, and walk through a complete code to deployment example, continue to the Kubernetes cluster tutorial.

[AKS tutorial](#)

Create a Windows Server container on an Azure Kubernetes Service (AKS) cluster using PowerShell

3/12/2021 • 8 minutes to read • [Edit Online](#)

Azure Kubernetes Service (AKS) is a managed Kubernetes service that lets you quickly deploy and manage clusters. In this article, you deploy an AKS cluster using PowerShell. You also deploy an [ASP.NET](#) sample application in a Windows Server container to the cluster.



This article assumes a basic understanding of Kubernetes concepts. For more information, see [Kubernetes core concepts for Azure Kubernetes Service \(AKS\)](#).

Prerequisites

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

If you choose to use PowerShell locally, this article requires that you install the Az PowerShell module and connect to your Azure account using the [Connect-AzAccount](#) cmdlet. For more information about installing the Az PowerShell module, see [Install Azure PowerShell](#). You also must install the Az.Aks PowerShell module:

```
Install-Module Az.Aks
```

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local

environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu bar at the upper right in the Azure portal .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

If you have multiple Azure subscriptions, choose the appropriate subscription in which the resources should be billed. Select a specific subscription ID using the [Set-AzContext](#) cmdlet.

```
Set-AzContext -SubscriptionId 00000000-0000-0000-0000-000000000000
```

Limitations

The following limitations apply when you create and manage AKS clusters that support multiple node pools:

- You can't delete the first node pool.

The following additional limitations apply to Windows Server node pools:

- The AKS cluster can have a maximum of 10 node pools.
- The AKS cluster can have a maximum of 100 nodes in each node pool.
- The Windows Server node pool name has a limit of 6 characters.

Create a resource group

An [Azure resource group](#) is a logical group in which Azure resources are deployed and managed. When you create a resource group, you are asked to specify a location. This location is where resource group metadata is stored, it is also where your resources run in Azure if you don't specify another region during resource creation. Create a resource group using the [New-AzResourceGroup](#) cmdlet.

The following example creates a resource group named **myResourceGroup** in the **eastus** location.

NOTE

This article uses PowerShell syntax for the commands in this tutorial. If you are using Azure Cloud Shell, ensure that the dropdown in the upper-left of the Cloud Shell window is set to **PowerShell**.

```
New-AzResourceGroup -Name myResourceGroup -Location eastus
```

The following example output shows the resource group created successfully:

```
ResourceGroupName : myResourceGroup
Location         : eastus
ProvisioningState : Succeeded
Tags             :
ResourceId       : /subscriptions/00000000-0000-0000-0000-000000000000/resourceGroups/myResourceGroup
```

Create an AKS cluster

Use the `ssh-keygen` command-line utility to generate an SSH key pair. For more details, see [Quick steps: Create and use an SSH public-private key pair for Linux VMs in Azure](#).

To run an AKS cluster that supports node pools for Windows Server containers, your cluster needs to use a network policy that uses [Azure CNI](#) (advanced) network plugin. For more detailed information to help plan out the required subnet ranges and network considerations, see [configure Azure CNI networking](#). Use the `New-AzAks` cmdlet below to create an AKS cluster named **myAKSCluster**. The following example creates the necessary network resources if they don't exist.

NOTE

To ensure your cluster operates reliably, you should run at least 2 (two) nodes in the default node pool.

```
$Username = Read-Host -Prompt 'Please create a username for the administrator credentials on your Windows Server containers: '
$Password = Read-Host -Prompt 'Please create a password for the administrator credentials on your Windows Server containers: ' -AsSecureString
New-AzAksCluster -ResourceGroupName myResourceGroup -Name myAKSCluster -NodeCount 2 -NetworkPlugin azure -
NodeVmSetType VirtualMachineScaleSets -WindowsProfileAdminUserName $Username -
WindowsProfileAdminUserPassword $Password
```

NOTE

If you are unable to create the AKS cluster because the version is not supported in this region then you can use the `Get-AzAksVersion -Location eastus` command to find the supported version list for this region.

After a few minutes, the command completes and returns information about the cluster. Occasionally the cluster can take longer than a few minutes to provision. Allow up to 10 minutes in these cases.

Add a Windows Server node pool

By default, an AKS cluster is created with a node pool that can run Linux containers. Use `New-AzAksNodePool` cmdlet to add a node pool that can run Windows Server containers alongside the Linux node pool.

```
New-AzAksNodePool -ResourceGroupName myResourceGroup -ClusterName myAKScluster -VmSetType  
VirtualMachineScaleSets -OsType Windows -Name npwin
```

The above command creates a new node pool named `npwin` and adds it to the `myAKScluster`. When creating a node pool to run Windows Server containers, the default value for `VmSize` is `Standard_D2s_v3`. If you choose to set the `VmSize` parameter, check the list of [restricted VM sizes](#). The minimum recommended size is `Standard_D2s_v3`. The previous command also uses the default subnet in the default vnet created when running `New-AzAks`.

Connect to the cluster

To manage a Kubernetes cluster, you use [kubectl](#), the Kubernetes command-line client. If you use Azure Cloud Shell, `kubectl` is already installed. To install `kubectl` locally, use the `Install-AzAksKubectl` cmdlet:

```
Install-AzAksKubectl
```

To configure `kubectl` to connect to your Kubernetes cluster, use the `Import-AzAksCredential` cmdlet. This command downloads credentials and configures the Kubernetes CLI to use them.

```
Import-AzAksCredential -ResourceGroupName myResourceGroup -Name myAKScluster
```

To verify the connection to your cluster, use the `kubectl get` command to return a list of the cluster nodes.

```
kubectl get nodes
```

The following example output shows all the nodes in the cluster. Make sure that the status of all nodes is **Ready**:

NAME	STATUS	ROLES	AGE	VERSION
aks-nodepool1-12345678-vmssfedcba	Ready	agent	13m	v1.16.7
aksnpwin987654	Ready	agent	108s	v1.16.7

Run the application

A Kubernetes manifest file defines a desired state for the cluster, such as what container images to run. In this article, a manifest is used to create all objects needed to run the ASP.NET sample application in a Windows Server container. This manifest includes a [Kubernetes deployment](#) for the ASP.NET sample application and an external [Kubernetes service](#) to access the application from the internet.

The ASP.NET sample application is provided as part of the [.NET Framework Samples](#) and runs in a Windows Server container. AKS requires Windows Server containers to be based on images of [Windows Server 2019](#) or greater. The Kubernetes manifest file must also define a [node selector](#) to tell your AKS cluster to run your ASP.NET sample application's pod on a node that can run Windows Server containers.

Create a file named `sample.yaml` and copy in the following YAML definition. If you use the Azure Cloud Shell, this file can be created using `vi` or `nano` as if working on a virtual or physical system:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample
  labels:
    app: sample
spec:
  replicas: 1
  template:
    metadata:
      name: sample
      labels:
        app: sample
    spec:
      nodeSelector:
        "beta.kubernetes.io/os": windows
      containers:
        - name: sample
          image: mcr.microsoft.com/dotnet/framework/samples:aspnetapp
          resources:
            limits:
              cpu: 1
              memory: 800M
            requests:
              cpu: .1
              memory: 300M
          ports:
            - containerPort: 80
      selector:
        matchLabels:
          app: sample
---
apiVersion: v1
kind: Service
metadata:
  name: sample
spec:
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 80
  selector:
    app: sample
```

Deploy the application using the [kubectl apply](#) command and specify the name of your YAML manifest:

```
kubectl apply -f sample.yaml
```

The following example output shows the Deployment and Service created successfully:

```
deployment.apps/sample created
service/sample created
```

Test the application

When the application runs, a Kubernetes service exposes the application frontend to the internet. This process can take a few minutes to complete. Occasionally the service can take longer than a few minutes to provision. Allow up to 10 minutes in these cases.

To monitor progress, use the [kubectl get service](#) command with the `--watch` argument.

```
kubectl get service sample --watch
```

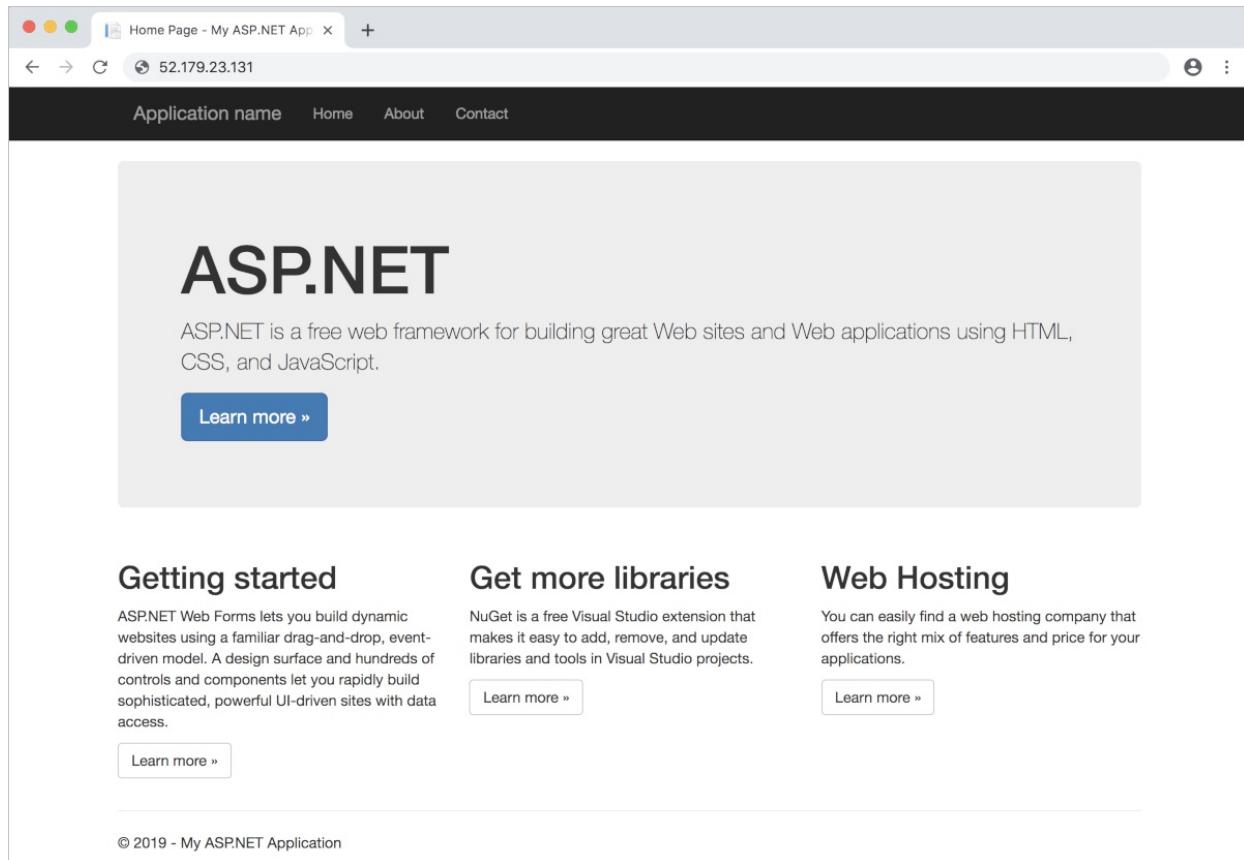
Initially the **EXTERNAL-IP** for the **sample** service is shown as **pending**.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
sample	LoadBalancer	10.0.37.27	<pending>	80:30572/TCP	6s

When the **EXTERNAL-IP** address changes from **pending** to an actual public IP address, use **CTRL-C** to stop the **kubectl** watch process. The following example output shows a valid public IP address assigned to the service:

```
sample  LoadBalancer  10.0.37.27  52.179.23.131  80:30572/TCP  2m
```

To see the sample app in action, open a web browser to the external IP address of your service.



NOTE

If you receive a connection timeout when trying to load the page then you should verify the sample app is ready with the following command **kubectl get pods --watch**. Sometimes the windows container will not be started by the time your external IP address is available.

Delete cluster

When the cluster is no longer needed, use the **Remove-AzResourceGroup** cmdlet to remove the resource group, container service, and all related resources.

```
Remove-AzResourceGroup -Name myResourceGroup
```

NOTE

When you delete the cluster, the Azure Active Directory service principal used by the AKS cluster is not removed. For steps on how to remove the service principal, see [AKS service principal considerations and deletion](#). If you used a managed identity, the identity is managed by the platform and does not require removal.

Next steps

In this article, you deployed a Kubernetes cluster and deployed an [ASP.NET](#) sample application in a Windows Server container to it. [Access the Kubernetes web dashboard](#) for the cluster you created.

To learn more about AKS, and walk through a complete code to deployment example, continue to the Kubernetes cluster tutorial.

[AKS tutorial](#)

Connect with RDP to Azure Kubernetes Service (AKS) cluster Windows Server nodes for maintenance or troubleshooting

4/21/2021 • 5 minutes to read • [Edit Online](#)

Throughout the lifecycle of your Azure Kubernetes Service (AKS) cluster, you may need to access an AKS Windows Server node. This access could be for maintenance, log collection, or other troubleshooting operations. You can access the AKS Windows Server nodes using RDP. Alternatively, if you want to use SSH to access the AKS Windows Server nodes and you have access to the same keypair that was used during cluster creation, you can follow the steps in [SSH into Azure Kubernetes Service \(AKS\) cluster nodes](#). For security purposes, the AKS nodes are not exposed to the internet.

This article shows you how to create an RDP connection with an AKS node using their private IP addresses.

Before you begin

This article assumes that you have an existing AKS cluster with a Windows Server node. If you need an AKS cluster, see the article on [creating an AKS cluster with a Windows container using the Azure CLI](#). You need the Windows administrator username and password for the Windows Server node you want to troubleshoot. If you don't know them, you can reset them by following [Reset Remote Desktop Services or its administrator password in a Windows VM](#). You also need an RDP client such as [Microsoft Remote Desktop](#).

You also need the Azure CLI version 2.0.61 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Deploy a virtual machine to the same subnet as your cluster

The Windows Server nodes of your AKS cluster don't have externally accessible IP addresses. To make an RDP connection, you can deploy a virtual machine with a publicly accessible IP address to the same subnet as your Windows Server nodes.

The following example creates a virtual machine named *myVM* in the *myResourceGroup* resource group.

First, get the subnet used by your Windows Server node pool. To get the subnet id, you need the name of the subnet. To get the name of the subnet, you need the name of the vnet. Get the vnet name by querying your cluster for its list of networks. To query the cluster, you need its name. You can get all of these by running the following in the Azure Cloud Shell:

```
CLUSTER_RG=$(az aks show -g myResourceGroup -n myAKSCluster --query nodeResourceGroup -o tsv)
VNET_NAME=$(az network vnet list -g $CLUSTER_RG --query [0].name -o tsv)
SUBNET_NAME=$(az network vnet subnet list -g $CLUSTER_RG --vnet-name $VNET_NAME --query [0].name -o tsv)
SUBNET_ID=$(az network vnet subnet show -g $CLUSTER_RG --vnet-name $VNET_NAME --name $SUBNET_NAME --query id -o tsv)
```

Now that you have the SUBNET_ID, run the following command in the same Azure Cloud Shell window to create the VM:

```
az vm create \
--resource-group myResourceGroup \
--name myVM \
--image win2019datacenter \
--admin-username azureuser \
--admin-password myP@ssw0rd12 \
--subnet $SUBNET_ID \
--query publicIpAddress -o tsv
```

The following example output shows the VM has been successfully created and displays the public IP address of the virtual machine.

```
13.62.204.18
```

Record the public IP address of the virtual machine. You will use this address in a later step.

Allow access to the virtual machine

AKS node pool subnets are protected with NSGs (Network Security Groups) by default. To get access to the virtual machine, you'll have to enable access in the NSG.

NOTE

The NSGs are controlled by the AKS service. Any change you make to the NSG will be overwritten at any time by the control plane.

First, get the resource group and nsg name of the nsg to add the rule to:

```
CLUSTER_RG=$(az aks show -g myResourceGroup -n myAKSCluster --query nodeResourceGroup -o tsv)
NSG_NAME=$(az network nsg list -g $CLUSTER_RG --query [].name -o tsv)
```

Then, create the NSG rule:

```
az network nsg rule create --name tempRDPAccess --resource-group $CLUSTER_RG --nsg-name $NSG_NAME --priority
100 --destination-port-range 3389 --protocol Tcp --description "Temporary RDP access to Windows nodes"
```

Get the node address

To manage a Kubernetes cluster, you use [kubectl](#), the Kubernetes command-line client. If you use Azure Cloud Shell, `kubectl` is already installed. To install `kubectl` locally, use the [az aks install-cli](#) command:

```
az aks install-cli
```

To configure `kubectl` to connect to your Kubernetes cluster, use the [az aks get-credentials](#) command. This command downloads credentials and configures the Kubernetes CLI to use them.

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```

List the internal IP address of the Windows Server nodes using the [kubectl get](#) command:

```
kubectl get nodes -o wide
```

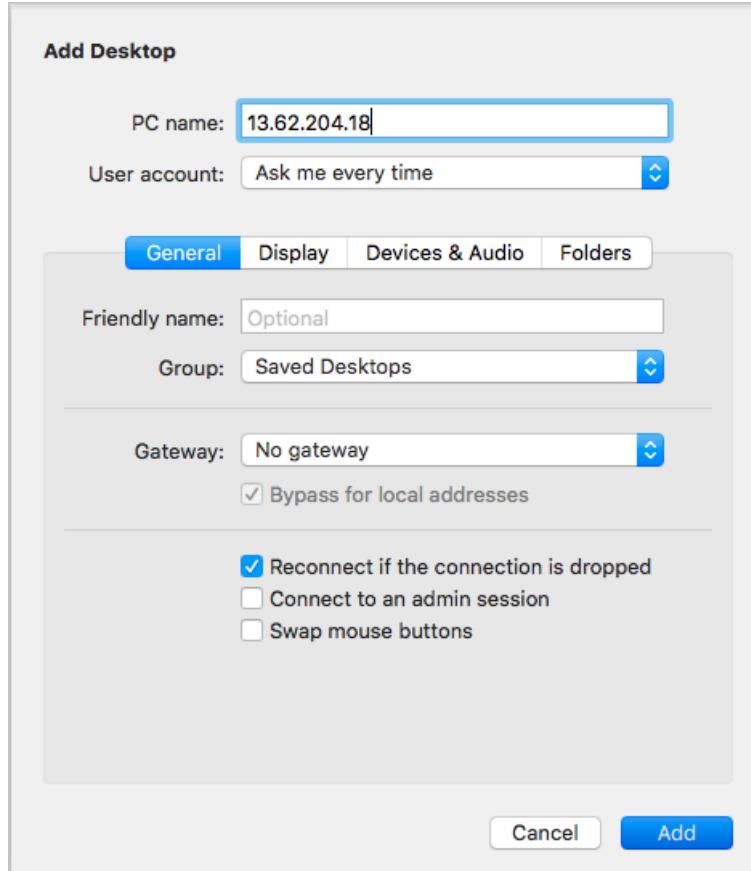
The follow example output shows the internal IP addresses of all the nodes in the cluster, including the Windows Server nodes.

```
$ kubectl get nodes -o wide
NAME                  STATUS  ROLES   AGE    VERSION  INTERNAL-IP  EXTERNAL-IP  OS-IMAGE
KERNEL-VERSION      CONTAINER-RUNTIME
aks-nodepool1-42485177-vmss000000  Ready    agent   18h    v1.12.7  10.240.0.4  <none>       Ubuntu
16.04.6 LTS          4.15.0-1040-azure  docker://3.0.4
aksnspwin000000      Ready    agent   13h    v1.12.7  10.240.0.67 <none>       Windows
Server Datacenter    10.0.17763.437
```

Record the internal IP address of the Windows Server node you wish to troubleshoot. You will use this address in a later step.

Connect to the virtual machine and node

Connect to the public IP address of the virtual machine you created earlier using an RDP client such as [Microsoft Remote Desktop](#).



After you've connected to your virtual machine, connect to the *internal IP address* of the Windows Server node you want to troubleshoot using an RDP client from within your virtual machine.



You are now connected to your Windows Server node.



You can now run any troubleshooting commands in the *cmd* window. Since Windows Server nodes use Windows Server Core, there's not a full GUI or other GUI tools when you connect to a Windows Server node over RDP.

Remove RDP access

When done, exit the RDP connection to the Windows Server node then exit the RDP session to the virtual machine. After you exit both RDP sessions, delete the virtual machine with the `az vm delete` command:

```
az vm delete --resource-group myResourceGroup --name myVM
```

And the NSG rule:

```
CLUSTER_RG=$(az aks show -g myResourceGroup -n myAKSCluster --query nodeResourceGroup -o tsv)
NSG_NAME=$(az network nsg list -g $CLUSTER_RG --query [].name -o tsv)
```

```
az network nsg rule delete --resource-group $CLUSTER_RG --nsg-name $NSG_NAME --name tempRDPAccess
```

Next steps

If you need additional troubleshooting data, you can [view the Kubernetes master node logs](#) or [Azure Monitor](#).

Frequently asked questions for Windows Server node pools in AKS

5/14/2021 • 8 minutes to read • [Edit Online](#)

In Azure Kubernetes Service (AKS), you can create a node pool that runs Windows Server as the guest OS on the nodes. These nodes can run native Windows container applications, such as those built on the .NET Framework. There are differences in how the Linux and Windows OS provides container support. Some common Linux Kubernetes and pod-related features are not currently available for Windows node pools.

This article outlines some of the frequently asked questions and OS concepts for Windows Server nodes in AKS.

Which Windows operating systems are supported?

AKS uses Windows Server 2019 as the host OS version and only supports process isolation. Container images built using other Windows Server versions are not supported. For more information, see [Windows container version compatibility](#).

Is Kubernetes different on Windows and Linux?

Window Server node pool support includes some limitations that are part of the upstream Windows Server in Kubernetes project. These limitations are not specific to AKS. For more information on this upstream support for Windows Server in Kubernetes, see the [Supported Functionality and Limitations](#) section of the [Intro to Windows support in Kubernetes](#) document, from the Kubernetes project.

Kubernetes is historically Linux-focused. Many examples used in the upstream [Kubernetes.io](#) website are intended for use on Linux nodes. When you create deployments that use Windows Server containers, the following considerations at the OS-level apply:

- **Identity** - Linux identifies a user by an integer user identifier (UID). A user also has an alphanumeric user name for logging on, which Linux translates to the user's UID. Similarly Linux identifies a user group by an integer group identifier (GID) and translates a group name to its corresponding GID.
 - Windows Server uses a larger binary security identifier (SID) which is stored in the Windows Security Access Manager (SAM) database. This database is not shared between the host and containers, or between containers.
- **File permissions** - Windows Server uses an access control list based on SIDs, rather than a bitmask of permissions and UID+GID
- **File paths** - convention on Windows Server is to use \ instead of /.
 - In pod specs that mount volumes, specify the path correctly for Windows Server containers. For example, rather than a mount point of `/mnt/volume` in a Linux container, specify a drive letter and location such as `/K/Volume` to mount as the K: drive.

What kind of disks are supported for Windows?

Azure Disks and Azure Files are the supported volume types. These are accessed as NTFS volumes in the Windows Server container.

Can I run Windows only clusters in AKS?

The master nodes (the control plane) in an AKS cluster are hosted by AKS the service, you will not be exposed to

the operating system of the nodes hosting the master components. All AKS clusters are created with a default first node pool, which is Linux-based. This node pool contains system services, which are needed for the cluster to function. It's recommended to run at least two nodes in the first node pool to ensure reliability of your cluster and the ability to do cluster operations. The first Linux-based node pool can't be deleted unless the AKS cluster itself is deleted.

How do I patch my Windows nodes?

To get the latest patches for Windows nodes, you can either [upgrade the node pool](#) or [upgrade the node image](#). Windows Updates are not enabled on nodes in AKS. AKS releases new node pool images as soon as patches are available, and it's the user's responsibility to upgrade node pools to stay current on patches and hotfixes. This is also true for the Kubernetes version being used. [AKS release notes](#) indicate when new versions are available. For more information on upgrading the entire Windows Server node pool, see [Upgrade a node pool in AKS](#). If you're only interested in updating the node image, see [AKS node image upgrades](#).

NOTE

The updated Windows Server image will only be used if a cluster upgrade (control plane upgrade) has been performed prior to upgrading the node pool.

What network plug-ins are supported?

AKS clusters with Windows node pools must use the Azure CNI (advanced) networking model. Kubenet (basic) networking is not supported. For more information on the differences in network models, see [Network concepts for applications in AKS](#). The Azure CNI network model requires additional planning and considerations for IP address management. For more information on how to plan and implement Azure CNI, see [Configure Azure CNI networking in AKS](#).

Windows nodes on AKS clusters also have [Direct Server Return \(DSR\)](#) enabled by default when Calico is enabled.

Is preserving the client source IP supported?

At this time, [client source IP preservation](#) is not supported with Windows nodes.

Can I change the max. # of pods per node?

Yes. For the implications and options that are available, see [Maximum number of pods](#).

Why am I seeing an error when I try to create a new Windows agent pool?

If you created your cluster before February 2020 and have never done any cluster upgrade operations, the cluster still uses an old Windows image. You may have seen an error that resembles:

"The following list of images referenced from the deployment template is not found: Publisher: MicrosoftWindowsServer, Offer: WindowsServer, Sku: 2019-datacenter-core-smalldisk-2004, Version: latest. Please refer to <https://docs.microsoft.com/azure/virtual-machines/windows/cli-ps-findimage> for instructions on finding available images."

To fix this error:

1. Upgrade the [cluster control plane](#) to update the image offer and publisher.
2. Create new Windows agent pools.

3. Move Windows pods from existing Windows agent pools to new Windows agent pools.
4. Delete old Windows agent pools.

How do I rotate the service principal for my Windows node pool?

Windows node pools do not support service principal rotation. To update the service principal, create a new Windows node pool and migrate your pods from the older pool to the new one. Once this is complete, delete the older node pool.

Instead, use managed identities, which are essentially wrappers around service principals. For more information, see [Use managed identities in Azure Kubernetes Service](#).

How do I change the administrator password for Windows Server nodes on my cluster?

When you create your AKS cluster, you specify the `--windows-admin-password` and `--windows-admin-username` parameters to set the administrator credentials for any Windows Server nodes on the cluster. If you did not specify administrator credentials, such as when creating a cluster using the Azure Portal or when setting `--vm-set-type VirtualMachineScaleSets` and `--network-plugin azure` using the Azure CLI, the username defaults to `azureuser` and a randomized password.

To change the administrator password, use the `az aks update` command:

```
az aks update \
--resource-group $RESOURCE_GROUP \
--name $CLUSTER_NAME \
--windows-admin-password $NEW_PW
```

IMPORTANT

Performing this operation upgrades all Windows Server node pools. Linux node pools are not affected.

When changing `--windows-admin-password`, the new password must be at least 14 characters and meet [Windows Server password requirements](#).

How many node pools can I create?

The AKS cluster can have a maximum of 10 node pools. You can have a maximum of 1000 nodes across those node pools. [Node pool limitations](#).

What can I name my Windows node pools?

You have to keep the name to a maximum of 6 (six) characters. This is a current limitation of AKS.

Are all features supported with Windows nodes?

Kubenet is currently not supported with Windows nodes.

Can I run ingress controllers on Windows nodes?

Yes, an ingress-controller that supports Windows Server containers can run on Windows nodes in AKS.

Can my Windows Server containers use gMSA?

Group managed service accounts (gMSA) support is not currently available in AKS.

Can I use Azure Monitor for containers with Windows nodes and containers?

Yes you can, however Azure Monitor is in public preview for gathering logs (stdout, stderr) and metrics from Windows containers. You can also attach to the live stream of stdout logs from a Windows container.

Are there any limitations on the number of services on a cluster with Windows nodes?

A cluster with Windows nodes can have approximately 500 services before it encounters port exhaustion.

Can I use Azure Hybrid Benefit with Windows nodes?

Yes. Azure Hybrid Benefit for Windows Server reduces operating costs by letting you bring your on-premises Windows Server license to AKS Windows nodes.

Azure Hybrid Benefit can be used on your entire AKS cluster or on individual nodes. For individual nodes, you need to navigate to the [node resource group](#) and apply the Azure Hybrid Benefit to the nodes directly. For more information on applying Azure Hybrid Benefit to individual nodes, see [Azure Hybrid Benefit for Windows Server](#).

To use Azure Hybrid Benefit on a new AKS cluster, use the `--enable-ahub` argument.

```
az aks create \
  --resource-group myResourceGroup \
  --name myAKScluster \
  --load-balancer-sku Standard \
  --windows-admin-password 'Password1234$' \
  --windows-admin-username azure \
  --network-plugin azure
  --enable-ahub
```

To use Azure Hybrid Benefit on an existing AKS cluster, update the cluster using the `--enable-ahub` argument.

```
az aks update \
  --resource-group myResourceGroup
  --name myAKScluster
  --enable-ahub
```

To check if Azure Hybrid Benefit is set on the cluster, use the following command:

```
az vmss show --name myAKScluster --resource-group MC_CLUSTERNAME
```

If the cluster has Azure Hybrid Benefit enabled, the output of `az vmss show` will be similar to the following:

```
"platformFaultDomainCount": 1,
  "provisioningState": "Succeeded",
  "proximityPlacementGroup": null,
  "resourceGroup": "MC_CLUSTERNAME"
```

Can I use the Kubernetes Web Dashboard with Windows containers?

Yes, you can use the [Kubernetes Web Dashboard](#) to access information about Windows containers, but at this

time you can't run `kubectl exec` into a running Windows container directly from the Kubernetes Web Dashboard. For more details on connecting to your running Windows container, see [Connect with RDP to Azure Kubernetes Service \(AKS\) cluster Windows Server nodes for maintenance or troubleshooting](#).

What if I need a feature that's not supported?

We work hard to bring all the features you need to Windows in AKS, but if you do encounter gaps, the open-source, upstream [aks-engine](#) project provides an easy and fully customizable way of running Kubernetes in Azure, including Windows support. Be sure to check out our roadmap of features coming [AKS roadmap](#).

Next steps

To get started with Windows Server containers in AKS, [create a node pool that runs Windows Server in AKS](#).

Access the Kubernetes web dashboard in Azure Kubernetes Service (AKS)

4/21/2021 • 5 minutes to read • [Edit Online](#)

Kubernetes includes a web dashboard that can be used for basic management operations. This dashboard lets you view basic health status and metrics for your applications, create and deploy services, and edit existing applications. This article shows you how to access the Kubernetes dashboard using the Azure CLI, then guides you through some basic dashboard operations.

For more information on the Kubernetes dashboard, see [Kubernetes Web UI Dashboard](#). AKS uses version 2.0 and greater of the open-source dashboard.

WARNING

The AKS dashboard add-on is set for deprecation. Use the [Kubernetes resource view in the Azure portal \(preview\)](#) instead.

- The Kubernetes dashboard is enabled by default for clusters running a Kubernetes version less than 1.18.
- The dashboard add-on will be disabled by default for all new clusters created on Kubernetes 1.18 or greater.
- Starting with Kubernetes 1.19 in preview, AKS will no longer support installation of the managed kube-dashboard addon.
- Existing clusters with the add-on enabled will not be impacted. Users will continue to be able to manually install the open-source dashboard as user-installed software.

Before you begin

The steps detailed in this document assume that you've created an AKS cluster and have established a `kubectl` connection with the cluster. If you need to create an AKS cluster, see [Quickstart: Deploy an Azure Kubernetes Service cluster using the Azure CLI](#).

You also need the Azure CLI version 2.6.0 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Disable the Kubernetes dashboard

The kube-dashboard addon is **enabled by default on clusters older than K8s 1.18**. The addon can be disabled by running the following command.

```
az aks disable-addons -g myRG -n myAKScluster -a kube-dashboard
```

Start the Kubernetes dashboard

WARNING

The AKS dashboard add-on is deprecated for versions 1.19+. Please use the [Kubernetes resource view in the Azure portal \(preview\)](#) instead.

- The following command will now open the Azure Portal resource view instead of the kubernetes dashboard for versions 1.19 and above.

To start the Kubernetes dashboard on a cluster, use the `az aks browse` command. This command requires the installation of the kube-dashboard addon on the cluster, which is included by default on clusters running any version older than Kubernetes 1.18.

The following example opens the dashboard for the cluster named *myAKSCluster* in the resource group named *myResourceGroup*.

```
az aks browse --resource-group myResourceGroup --name myAKSCluster
```

This command creates a proxy between your development system and the Kubernetes API, and opens a web browser to the Kubernetes dashboard. If a web browser doesn't open to the Kubernetes dashboard, copy and paste the URL address noted in the Azure CLI, typically `http://127.0.0.1:8001`.

NOTE

If you do not see the dashboard at `http://127.0.0.1:8001` you can manually route to the following addresses. Clusters on 1.16 or greater use https and require a separate endpoint.

- K8s 1.16 or greater:

```
http://127.0.0.1:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy
```

- K8s 1.15 and below:

```
http://127.0.0.1:8001/api/v1/namespaces/kube-system/services/kubernetes-dashboard:/proxy
```

Sign in to the dashboard (kubernetes 1.16+)

IMPORTANT

As of [v1.10.1 of the Kubernetes dashboard](#) or kubernetes v1.16+ the service account "kubernetes-dashboard" can no longer be used to retrieve resources due to a [security fix in that release](#). As a result, requests without auth info return a [401 unauthorized error](#). A bearer token retrieved from a service account can still be used as in this [Kubernetes Dashboard example](#), but this impacts the login flow of the dashboard add-on compared to older versions.

If you still run a version prior to 1.16 you can still give permissions to the "kubernetes-dashboard" service account, but this is **not recommended**:

```
kubectl create clusterrolebinding kubernetes-dashboard --clusterrole=cluster-admin --serviceaccount=kube-system:kubernetes-dashboard
```

The initial screen presented requires a kubeconfig or token. Both options require resource permissions to display those resources in the dashboard.

Kubernetes Dashboard

Kubeconfig

Please select the kubeconfig file that you have created to configure access to the cluster. To find out more about how to configure and use kubeconfig file, please refer to the [Configure Access to Multiple Clusters](#) section.

Token

Every Service Account has a Secret with valid Bearer Token that can be used to log in to Dashboard. To find out more about how to configure and use Bearer Tokens, please refer to the [Authentication](#) section.

Choose kubeconfig file

...

[Sign in](#)

Use a kubeconfig

For both Azure AD enabled and non-Azure AD enabled clusters, a kubeconfig can be passed in. Ensure access tokens are valid, if your tokens are expired you can refresh tokens via kubectl.

1. Set the admin kubeconfig with `az aks get-credentials -a --resource-group <RG_NAME> --name <CLUSTER_NAME>`
2. Select `Kubeconfig` and click `Choose kubeconfig file` to open file selector
3. Select your kubeconfig file (defaults to \$HOME/.kube/config)
4. Click `Sign In`

Use a token

1. For **non-Azure AD enabled cluster**, run `kubectl config view` and copy the token associated with the user account of your cluster.
2. Paste into the token option at sign in.
3. Click `Sign In`

For Azure AD enabled clusters, retrieve your AAD token with the following command. Validate you've replaced the resource group and cluster name in the command.

```
## Update <RESOURCE_GROUP> and <AKS_NAME> with your input.  
  
kubectl config view -o jsonpath='{.users[?(@.name == "clusterUser_<RESOURCE_GROUP>_<AKS_NAME>")].user.auth-provider.config.access-token}'
```

Once successful, a page similar to the below will be displayed.



Create an application

The following steps require the user to have permissions to the respective resources.

To see how the Kubernetes dashboard can reduce the complexity of management tasks, let's create an application. You can create an application from the Kubernetes dashboard by providing text input, a YAML file, or through a graphical wizard.

To create an application, complete the following steps:

1. Select the **Create** button in the upper right window.
2. To use the graphical wizard, choose to **Create an app**.
3. Provide a name for the deployment, such as *nginx*
4. Enter the name for the container image to use, such as *nginx:1.15.5*
5. To expose port 80 for web traffic, you create a Kubernetes service. Under **Service**, select **External**, then enter **80** for both the port and target port.
6. When ready, select **Deploy** to create the app.

Cluster

- Namespaces
- Nodes
- Persistent Volumes
- Roles
- Storage Classes

Namespace

- default

Overview

Workloads

- Cron Jobs
- Daemon Sets
- Deployments
- Jobs
- Pods
- Replica Sets
- Replication Controllers
- Stateful Sets

Discovery and Load Balancing

CREATE FROM TEXT INPUT **CREATE FROM FILE** **CREATE AN APP**

App name * **nginx** 5 / 24

Container image * **nginx:1.15.5**

Number of pods * **1**

Service * **External**

Port *	Target port *	Protocol *
80	80	TCP

Port Target port Protocol *

SHOW ADVANCED OPTIONS

DEPLOY **CANCEL**

It takes a minute or two for a public external IP address to be assigned to the Kubernetes service. On the left-hand side, under **Discovery and Load Balancing** select **Services**. Your application's service is listed, including the *External endpoints*, as shown in the following example:

Discovery and load balancing > Services

Overview

Workloads

- Cron Jobs
- Daemon Sets
- Deployments
- Jobs
- Pods
- Replica Sets
- Replication Controllers
- Stateful Sets

Discovery and Load Balancing

Ingresses

Services

Services

Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
nginx	k8s-app: nginx	10.0.180.34	nginx:80 TCP nginx:30344 TCP	23.96.99.120:80	41 minutes
kubernetes	component: ap... provider: kuber...	10.0.0.1	kubernetes:443 T kubernetes:0 TCP	-	7 days

Select the endpoint address to open a web browser window to the default NGINX page:



View pod information

The Kubernetes dashboard can provide basic monitoring metrics and troubleshooting information such as logs.

To see more information about your application pods, select **Pods** in the left-hand menu. The list of available pods is shown. Choose your *nginx* pod to view information, such as resource consumption:



Edit the application

In addition to creating and viewing applications, the Kubernetes dashboard can be used to edit and update application deployments. To provide additional redundancy for the application, let's increase the number of NGINX replicas.

To edit a deployment:

1. Select **Deployments** in the left-hand menu, and then choose your *nginx* deployment.
2. Select **Edit** in the upper right-hand navigation bar.
3. Locate the `spec.replicas` value, at around line 20. To increase the number of replicas for the application, change this value from `1` to `3`.
4. Select **Update** when ready.

```

15    "annotations": {
16      "deployment.kubernetes.io/revision": "1"
17    }
18  },
19  "spec": {
20    "replicas": 3,
21    "selector": {
22      "matchLabels": {
23        "k8s-app": "nginx"
24      }
25    },
26    "template": {
27      "metadata": {
28        "name": "nginx",
29        "creationTimestamp": null,
30        "labels": {

```

CANCEL COPY UPDATE

It takes a few moments for the new pods to be created inside a replica set. On the left-hand menu, choose **Replica Sets**, and then choose your *nginx* replica set. The list of pods now reflects the updated replica count, as shown in the following example output:

Name	Node	Status	Restarts	Age	CPU (cores)	Memory (bytes)
nginx-f597dd9c-aks-nodepool1-79590246-0		Running	0	56 minutes	0	2.098 Mi
nginx-f597dd9c-aks-nodepool1-79590246-0		Running	0	10 seconds	-	2.059 Mi
nginx-f597dd9c-aks-nodepool1-79590246-0		Running	0	10 seconds	-	1.980 Mi

Next steps

For more information about the Kubernetes dashboard, see the [Kubernetes Web UI Dashboard](#).

Install existing applications with Helm in Azure Kubernetes Service (AKS)

3/5/2021 • 3 minutes to read • [Edit Online](#)

[Helm](#) is an open-source packaging tool that helps you install and manage the lifecycle of Kubernetes applications. Similar to Linux package managers such as *APT* and *Yum*, Helm is used to manage Kubernetes charts, which are packages of preconfigured Kubernetes resources.

This article shows you how to configure and use Helm in a Kubernetes cluster on AKS.

Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the [AKS quickstart using the Azure CLI](#) or [using the Azure portal](#).

You also need the Helm CLI installed, which is the client that runs on your development system. It allows you to start, stop, and manage applications with Helm. If you use the Azure Cloud Shell, the Helm CLI is already installed. For installation instructions on your local platform, see [Installing Helm](#).

IMPORTANT

Helm is intended to run on Linux nodes. If you have Windows Server nodes in your cluster, you must ensure that Helm pods are only scheduled to run on Linux nodes. You also need to ensure that any Helm charts you install are also scheduled to run on the correct nodes. The commands in this article use [node-selectors][k8s-node-selector] to make sure pods are scheduled to the correct nodes, but not all Helm charts may expose a node selector. You can also consider using other options on your cluster, such as [taints](#).

Verify your version of Helm

Use the `helm version` command to verify you have Helm 3 installed:

```
helm version
```

The following example shows Helm version 3.0.0 installed:

```
$ helm version  
  
version.BuildInfo{Version:"v3.0.0", GitCommit:"e29ce2a54e96cd02ccfce88bee4f58bb6e2a28b6",  
GitTreeState:"clean", GoVersion:"go1.13.4"}
```

Install an application with Helm v3

Add Helm repositories

Use the [helm repo](#) command to add the *ingress-nginx* repository.

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
```

Find Helm charts

Helm charts are used to deploy applications into a Kubernetes cluster. To search for pre-created Helm charts, use the [helm search](#) command:

```
helm search repo ingress-nginx
```

The following condensed example output shows some of the Helm charts available for use:

```
$ helm search repo ingress-nginx

NAME          CHART VERSION   APP VERSION   DESCRIPTION
ingress-nginx/ingress-nginx    2.12.0        0.34.1        Ingress controller for Kubernetes using
NGINX a...
```

To update the list of charts, use the [helm repo update](#) command.

```
helm repo update
```

The following example shows a successful repo update:

```
$ helm repo update

Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "ingress-nginx" chart repository
Update Complete.  Happy Helming!
```

Run Helm charts

To install charts with Helm, use the [helm install](#) command and specify a release name and the name of the chart to install. To see installing a Helm chart in action, let's install a basic nginx deployment using a Helm chart.

```
helm install my-nginx-ingress ingress-nginx/ingress-nginx \
--set controller.nodeSelector."beta\.kubernetes\.io/os"=linux \
--set defaultBackend.nodeSelector."beta\.kubernetes\.io/os"=linux
```

The following condensed example output shows the deployment status of the Kubernetes resources created by the Helm chart:

```
$ helm install my-nginx-ingress ingress-nginx/ingress-nginx \
>     --set controller.nodeSelector."beta\.kubernetes\.io/os"=linux \
>     --set defaultBackend.nodeSelector."beta\.kubernetes\.io/os"=linux

NAME: my-nginx-ingress
LAST DEPLOYED: Fri Nov 22 10:08:06 2019
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The nginx-ingress controller has been installed.
It may take a few minutes for the LoadBalancer IP to be available.
You can watch the status by running 'kubectl --namespace default get services -o wide -w my-nginx-ingress-ingress-ingress-controller'
...
```

Use the `kubectl get services` command to get the *EXTERNAL-IP* of your service.

```
kubectl --namespace default get services -o wide -w my-nginx-ingress-ingress-nginx-controller
```

For example, the below command shows the *EXTERNAL-IP* for the *my-nginx-ingress-ingress-nginx-controller* service:

```
$ kubectl --namespace default get services -o wide -w my-nginx-ingress-ingress-nginx-controller  
  
NAME                                     TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)  
AGE   SELECTOR  
my-nginx-ingress-ingress-nginx-controller  LoadBalancer  10.0.2.237    <EXTERNAL-IP>  
80:31380/TCP,443:32239/TCP  72s  app.kubernetes.io/component=controller,app.kubernetes.io/instance=my-  
nginx-ingress,app.kubernetes.io/name=ingress-nginx
```

List releases

To see a list of releases installed on your cluster, use the `helm list` command.

```
helm list
```

The following example shows the *my-nginx-ingress* release deployed in the previous step:

```
$ helm list  
  
NAME          NAMESPACE  REVISION UPDATED           STATUS    CHART          APP  
VERSION  
my-nginx-ingress  default   1       2019-11-22 10:08:06.048477 -0600 CST  deployed  nginx-ingress-1.25.0  
0.26.1
```

Clean up resources

When you deploy a Helm chart, a number of Kubernetes resources are created. These resources include pods, deployments, and services. To clean up these resources, use the `helm uninstall` command and specify your release name, as found in the previous `helm list` command.

```
helm uninstall my-nginx-ingress
```

The following example shows the release named *my-nginx-ingress* has been uninstalled:

```
$ helm uninstall my-nginx-ingress  
  
release "my-nginx-ingress" uninstalled
```

Next steps

For more information about managing Kubernetes application deployments with Helm, see the Helm documentation.

[Helm documentation](#)

Using OpenFaaS on AKS

11/2/2020 • 4 minutes to read • [Edit Online](#)

[OpenFaaS](#) is a framework for building serverless functions through the use of containers. As an open source project, it has gained large-scale adoption within the community. This document details installing and using OpenFaaS on an Azure Kubernetes Service (AKS) cluster.

Prerequisites

In order to complete the steps within this article, you need the following.

- Basic understanding of Kubernetes.
- An Azure Kubernetes Service (AKS) cluster and AKS credentials configured on your development system.
- Azure CLI installed on your development system.
- Git command-line tools installed on your system.

Add the OpenFaaS helm chart repo

Go to <https://shell.azure.com> to open Azure Cloud Shell in your browser.

OpenFaaS maintains its own helm charts to keep up to date with all the latest changes.

```
helm repo add openfaas https://openfaas.github.io/faas-netes/
helm repo update
```

Deploy OpenFaaS

As a good practice, OpenFaaS and OpenFaaS functions should be stored in their own Kubernetes namespace.

Create a namespace for the OpenFaaS system and functions:

```
kubectl apply -f https://raw.githubusercontent.com/openfaas/faas-netes/master/namespaces.yml
```

Generate a password for the OpenFaaS UI Portal and REST API:

```
# generate a random password
PASSWORD=$(head -c 12 /dev/urandom | shasum| cut -d' ' -f1)

kubectl -n openfaas create secret generic basic-auth \
--from-literal=basic-auth-user=admin \
--from-literal=basic-auth-password="$PASSWORD"
```

You can get the value of the secret with `echo $PASSWORD`.

The password we create here will be used by the helm chart to enable basic authentication on the OpenFaaS Gateway, which is exposed to the Internet through a cloud LoadBalancer.

A Helm chart for OpenFaaS is included in the cloned repository. Use this chart to deploy OpenFaaS into your AKS cluster.

```
helm upgrade openfaas --install openfaas/openfaas \
--namespace openfaas \
--set basic_auth=true \
--set functionNamespace=openfaas-fn \
--set serviceType=LoadBalancer
```

Output:

```
NAME:    openfaas
LAST DEPLOYED: Wed Feb 28 08:26:11 2018
NAMESPACE: openfaas
STATUS:  DEPLOYED

RESOURCES:
==> v1/ConfigMap
NAME          DATA   AGE
prometheus-config   2      20s
alertmanager-config 1      20s

{snip}

NOTES:
To verify that openfaas has started, run:

```console
kubectl --namespace=openfaas get deployments -l "release=openfaas, app=openfaas"
```

A public IP address is created for accessing the OpenFaaS gateway. To retrieve this IP address, use the [kubectl get service](#) command. It may take a minute for the IP address to be assigned to the service.

```
kubectl get service -l component=gateway --namespace openfaas
```

Output:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
gateway	ClusterIP	10.0.156.194	<none>	8080/TCP	7m
gateway-external	LoadBalancer	10.0.28.18	52.186.64.52	8080:30800/TCP	7m

To test the OpenFaaS system, browse to the external IP address on port 8080, <http://52.186.64.52:8080> in this example. You will be prompted to log in. To fetch your password, enter `echo $PASSWORD`.



Finally, install the OpenFaaS CLI. This example used brew, see the [OpenFaaS CLI documentation](#) for more options.

```
brew install faas-cli
```

Set `$OPENFAAS_URL` to the public IP found above.

Log in with the Azure CLI:

```
export OPENFAAS_URL=http://52.186.64.52:8080
echo -n $PASSWORD | ./faas-cli login -g $OPENFAAS_URL -u admin --password-stdin
```

## Create first function

Now that OpenFaaS is operational, create a function using the OpenFaaS portal.

Click on **Deploy New Function** and search for **Figlet**. Select the Figlet function, and click **Deploy**.

## Deploy A New Function

X

FROM STORE      MANUALLY

---

Search for Function

figlet

**F** Figlet  
OpenFaaS Figlet image. This repository comes with the blog post <http://jmkhael.io/create-a-serverless-ascii-banner-with-faas/>

[CLOSE DIALOG](#)      [DEPLOY](#)

Use curl to invoke the function. Replace the IP address in the following example with that of your OpenFaas gateway.

```
curl -X POST http://52.186.64.52:8080/function/figlet -d "Hello Azure"
```

## Output:

## Create second function

Now create a second function. This example will be deployed using the OpenFaaS CLI and includes a custom container image and retrieving data from a Cosmos DB. Several items need to be configured before creating the function.

First, create a new resource group for the Cosmos DB.

```
az group create --name serverless-backing --location eastus
```

Deploy a CosmosDB instance of kind `MongoDB`. The instance needs a unique name, update `openfaas-cosmos` to something unique to your environment.

```
az cosmosdb create --resource-group serverless-backing --name openfaas-cosmos --kind MongoDB
```

Get the Cosmos database connection string and store it in a variable.

Update the value for the `--resource-group` argument to the name of your resource group, and the `--name` argument to the name of your Cosmos DB.

```
COSMOS=$(az cosmosdb list-connection-strings \
--resource-group serverless-backing \
--name openfaas-cosmos \
--query connectionStrings[0].connectionString \
--output tsv)
```

Now populate the Cosmos DB with test data. Create a file named `plans.json` and copy in the following json.

```
{
 "name" : "two_person",
 "friendlyName" : "Two Person Plan",
 "portionSize" : "1-2 Person",
 "mealsPerWeek" : "3 Unique meals per week",
 "price" : 72,
 "description" : "Our basic plan, delivering 3 meals per week, which will feed 1-2 people.",
 "__v" : 0
}
```

Use the `mongoimport` tool to load the CosmosDB instance with data.

If needed, install the MongoDB tools. The following example installs these tools using brew, see the [MongoDB documentation](#) for other options.

```
brew install mongodb
```

Load the data into the database.

```
mongoimport --uri=$COSMOS -c plans < plans.json
```

Output:

```
2018-02-19T14:42:14.313+0000 connected to: localhost
2018-02-19T14:42:14.918+0000 imported 1 document
```

Run the following command to create the function. Update the value of the `-g` argument with your OpenFaaS gateway address.

```
faas-cli deploy -g http://52.186.64.52:8080 --image=shanepeckham/openfaascosmos --name=cosmos-query --
env=NODE_ENV=$COSMOS
```

Once deployed, you should see your newly created OpenFaaS endpoint for the function.

```
Deployed. 202 Accepted.
URL: http://52.186.64.52:8080/function/cosmos-query
```

Test the function using curl. Update the IP address with the OpenFaaS gateway address.

```
curl -s http://52.186.64.52:8080/function/cosmos-query
```

Output:

```
[{"ID": "", "Name": "two_person", "FriendlyName": "", "PortionSize": "", "MealsPerWeek": "", "Price": 72, "Description": "Our basic plan, delivering 3 meals per week, which will feed 1-2 people."}]
```

You can also test the function within the OpenFaaS UI.

The screenshot shows a browser window with the URL `52.186.64.52`. The title bar says "Invoke function". Below it is a button labeled "INVOKE". Underneath are three radio buttons: "Text" (selected), "JSON", and "Download". A "Request body" input field is empty. The "Response status" section shows "200". The "Round-trip (s)" section shows "0.721". The "Response body" section displays the JSON output from the previous code block.

```
[{"ID": "", "Name": "two_person", "FriendlyName": "", "PortionSize": "", "MealsPerWeek": "", "Price": 72, "Description": "Our basic plan, delivering 3 meals per week, which will feed 1-2 people."}]
```

## Next Steps

You can continue to learn with the OpenFaaS workshop through a set of hands-on labs that cover topics such as how to create your own GitHub bot, consuming secrets, viewing metrics, and auto-scaling.

# Running Apache Spark jobs on AKS

3/5/2021 • 6 minutes to read • [Edit Online](#)

Apache Spark is a fast engine for large-scale data processing. As of the [Spark 2.3.0 release](#), Apache Spark supports native integration with Kubernetes clusters. Azure Kubernetes Service (AKS) is a managed Kubernetes environment running in Azure. This document details preparing and running Apache Spark jobs on an Azure Kubernetes Service (AKS) cluster.

## Prerequisites

In order to complete the steps within this article, you need the following.

- Basic understanding of Kubernetes and [Apache Spark](#).
- [Docker Hub](#) account, or an [Azure Container Registry](#).
- Azure CLI [installed](#) on your development system.
- [JDK 8](#) installed on your system.
- [Apache Maven](#) installed on your system.
- [SBT \(Scala Build Tool\)](#) installed on your system.
- Git command-line tools installed on your system.

## Create an AKS cluster

Spark is used for large-scale data processing and requires that Kubernetes nodes are sized to meet the Spark resources requirements. We recommend a minimum size of `Standard_D3_v2` for your Azure Kubernetes Service (AKS) nodes.

If you need an AKS cluster that meets this minimum recommendation, run the following commands.

Create a resource group for the cluster.

```
az group create --name mySparkCluster --location eastus
```

Create a Service Principal for the cluster. After it is created, you will need the Service Principal appId and password for the next command.

```
az ad sp create-for-rbac --name SparkSP
```

Create the AKS cluster with nodes that are of size `Standard_D3_v2`, and values of appId and password passed as service-principal and client-secret parameters.

```
az aks create --resource-group mySparkCluster --name mySparkCluster --node-vm-size Standard_D3_v2 --generate-ssh-keys --service-principal <APPID> --client-secret <PASSWORD>
```

Connect to the AKS cluster.

```
az aks get-credentials --resource-group mySparkCluster --name mySparkCluster
```

If you are using Azure Container Registry (ACR) to store container images, configure authentication between

AKS and ACR. See the [ACR authentication documentation](#) for these steps.

## Build the Spark source

Before running Spark jobs on an AKS cluster, you need to build the Spark source code and package it into a container image. The Spark source includes scripts that can be used to complete this process.

Clone the Spark project repository to your development system.

```
git clone -b branch-2.4 https://github.com/apache/spark
```

Change into the directory of the cloned repository and save the path of the Spark source to a variable.

```
cd spark
sparkdir=$(pwd)
```

If you have multiple JDK versions installed, set `JAVA_HOME` to use version 8 for the current session.

```
export JAVA_HOME=`/usr/libexec/java_home -d 64 -v "1.8*`
```

Run the following command to build the Spark source code with Kubernetes support.

```
./build/mvn -Pkubernetes -DskipTests clean package
```

The following commands create the Spark container image and push it to a container image registry. Replace `registry.example.com` with the name of your container registry and `v1` with the tag you prefer to use. If using Docker Hub, this value is the registry name. If using Azure Container Registry (ACR), this value is the ACR login server name.

```
REGISTRY_NAME=registry.example.com
REGISTRY_TAG=v1
```

```
./bin/docker-image-tool.sh -r $REGISTRY_NAME -t $REGISTRY_TAG build
```

Push the container image to your container image registry.

```
./bin/docker-image-tool.sh -r $REGISTRY_NAME -t $REGISTRY_TAG push
```

## Prepare a Spark job

Next, prepare a Spark job. A jar file is used to hold the Spark job and is needed when running the `spark-submit` command. The jar can be made accessible through a public URL or pre-packaged within a container image. In this example, a sample jar is created to calculate the value of Pi. This jar is then uploaded to Azure storage. If you have an existing jar, feel free to substitute.

Create a directory where you would like to create the project for a Spark job.

```
mkdir myprojects
cd myprojects
```

Create a new Scala project from a template.

```
sbt new sbt/scala-seed.g8
```

When prompted, enter `SparkPi` for the project name.

```
name [Scala Seed Project]: SparkPi
```

Navigate to the newly created project directory.

```
cd sparkpi
```

Run the following commands to add an SBT plugin, which allows packaging the project as a jar file.

```
touch project/assembly.sbt
echo 'addSbtPlugin("com.eed3si9n" % "sbt-assembly" % "0.14.10")' >> project/assembly.sbt
```

Run these commands to copy the sample code into the newly created project and add all necessary dependencies.

```
EXAMPLESDIR="src/main/scala/org/apache/spark/examples"
mkdir -p $EXAMPLESDIR
cp $sparkdir/examples/$EXAMPLESDIR/SparkPi.scala $EXAMPLESDIR/SparkPi.scala

cat <<EOT >> build.sbt
// https://mvnrepository.com/artifact/org.apache.spark/spark-sql
libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.3.0" % "provided"
EOT

sed -ie 's/scalaVersion.*/scalaVersion := "2.11.11"/' build.sbt
sed -ie 's/name.*/name := "SparkPi",/' build.sbt
```

To package the project into a jar, run the following command.

```
sbt assembly
```

After successful packaging, you should see output similar to the following.

```
[info] Packaging /Users/me/myprojects/sparkpi/target/scala-2.11/SparkPi-assembly-0.1.0-SNAPSHOT.jar ...
[info] Done packaging.
[success] Total time: 10 s, completed Mar 6, 2018 11:07:54 AM
```

## Copy job to storage

Create an Azure storage account and container to hold the jar file.

```
RESOURCE_GROUP=sparkdemo
STORAGE_ACCT=sparkdemo$RANDOM
az group create --name $RESOURCE_GROUP --location eastus
az storage account create --resource-group $RESOURCE_GROUP --name $STORAGE_ACCT --sku Standard_LRS
export AZURE_STORAGE_CONNECTION_STRING=`az storage account show-connection-string --resource-group
$RESOURCE_GROUP --name $STORAGE_ACCT -o tsv`
```

Upload the jar file to the Azure storage account with the following commands.

```
CONTAINER_NAME=jars
BLOB_NAME=SparkPi-assembly-0.1.0-SNAPSHOT.jar
FILE_TO_UPLOAD=target/scala-2.11/SparkPi-assembly-0.1.0-SNAPSHOT.jar

echo "Creating the container..."
az storage container create --name $CONTAINER_NAME
az storage container set-permission --name $CONTAINER_NAME --public-access blob

echo "Uploading the file..."
az storage blob upload --container-name $CONTAINER_NAME --file $FILE_TO_UPLOAD --name $BLOB_NAME

jarUrl=$(az storage blob url --container-name $CONTAINER_NAME --name $BLOB_NAME | tr -d '')
```

Variable `jarUrl` now contains the publicly accessible path to the jar file.

## Submit a Spark job

Start kube-proxy in a separate command-line with the following code.

```
kubectl proxy
```

Navigate back to the root of Spark repository.

```
cd $sparkdir
```

Create a service account that has sufficient permissions for running a job.

```
kubectl create serviceaccount spark
kubectl create clusterrolebinding spark-role --clusterrole=edit --serviceaccount=default:spark --
namespace=default
```

Submit the job using `spark-submit`.

```
./bin/spark-submit \
--master k8s://http://127.0.0.1:8001 \
--deploy-mode cluster \
--name spark-pi \
--class org.apache.spark.examples.SparkPi \
--conf spark.executor.instances=3 \
--conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
--conf spark.kubernetes.container.image=$REGISTRY_NAME/spark:$REGISTRY_TAG \
$jarUrl
```

This operation starts the Spark job, which streams job status to your shell session. While the job is running, you can see Spark driver pod and executor pods using the `kubectl get pods` command. Open a second terminal session to run these commands.

```
kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
spark-pi-2232778d0f663768ab27edc35cb73040-driver	1/1	Running	0	16s
spark-pi-2232778d0f663768ab27edc35cb73040-exec-1	0/1	Init:0/1	0	4s
spark-pi-2232778d0f663768ab27edc35cb73040-exec-2	0/1	Init:0/1	0	4s
spark-pi-2232778d0f663768ab27edc35cb73040-exec-3	0/1	Init:0/1	0	4s

While the job is running, you can also access the Spark UI. In the second terminal session, use the `kubectl port-forward` command provide access to Spark UI.

```
kubectl port-forward spark-pi-2232778d0f663768ab27edc35cb73040-driver 4040:4040
```

To access Spark UI, open the address `127.0.0.1:4040` in a browser.



## Get job results and logs

After the job has finished, the driver pod will be in a "Completed" state. Get the name of the pod with the following command.

```
kubectl get pods --show-all
```

Output:

NAME	READY	STATUS	RESTARTS	AGE
spark-pi-2232778d0f663768ab27edc35cb73040-driver	0/1	Completed	0	1m

Use the `kubectl logs` command to get logs from the spark driver pod. Replace the pod name with your driver pod's name.

```
kubectl logs spark-pi-2232778d0f663768ab27edc35cb73040-driver
```

Within these logs, you can see the result of the Spark job, which is the value of Pi.

```
Pi is roughly 3.152155760778804
```

## Package jar with container image

In the above example, the Spark jar file was uploaded to Azure storage. Another option is to package the jar file into custom-built Docker images.

To do so, find the `dockerfile` for the Spark image located at

`$sparkdir/resource-managers/kubernetes/docker/src/main/dockerfiles/spark/` directory. Add an `ADD` statement for the Spark job `jar` somewhere between `WORKDIR` and `ENTRYPOINT` declarations.

Update the jar path to the location of the `SparkPi-assembly-0.1.0-SNAPSHOT.jar` file on your development system. You can also use your own custom jar file.

```
WORKDIR /opt/spark/work-dir

ADD /path/to/SparkPi-assembly-0.1.0-SNAPSHOT.jar SparkPi-assembly-0.1.0-SNAPSHOT.jar

ENTRYPOINT ["/opt/entrypoint.sh"]
```

Build and push the image with the included Spark scripts.

```
./bin/docker-image-tool.sh -r <your container repository name> -t <tag> build
./bin/docker-image-tool.sh -r <your container repository name> -t <tag> push
```

When running the job, instead of indicating a remote jar URL, the `local://` scheme can be used with the path to the jar file in the Docker image.

```
./bin/spark-submit \
--master k8s://https://<k8s-apiserver-host>:<k8s-apiserver-port> \
--deploy-mode cluster \
--name spark-pi \
--class org.apache.spark.examples.SparkPi \
--conf spark.executor.instances=3 \
--conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
--conf spark.kubernetes.container.image=<spark-image> \
local:///opt/spark/work-dir/<your-jar-name>.jar
```

### WARNING

From Spark [documentation](#): "The Kubernetes scheduler is currently experimental. In future versions, there may be behavioral changes around configuration, container images and entrypoints".

## Next steps

Check out Spark documentation for more details.

[Spark documentation](#)

# Use GPUs for compute-intensive workloads on Azure Kubernetes Service (AKS)

4/21/2021 • 9 minutes to read • [Edit Online](#)

Graphical processing units (GPUs) are often used for compute-intensive workloads such as graphics and visualization workloads. AKS supports the creation of GPU-enabled node pools to run these compute-intensive workloads in Kubernetes. For more information on available GPU-enabled VMs, see [GPU optimized VM sizes in Azure](#). For AKS nodes, we recommend a minimum size of *Standard\_NC6*.

## NOTE

GPU-enabled VMs contain specialized hardware that is subject to higher pricing and region availability. For more information, see the [pricing](#) tool and [region availability](#).

Currently, using GPU-enabled node pools is only available for Linux node pools.

## Before you begin

This article assumes that you have an existing AKS cluster with nodes that support GPUs. Your AKS cluster must run Kubernetes 1.10 or later. If you need an AKS cluster that meets these requirements, see the first section of this article to [create an AKS cluster](#).

You also need the Azure CLI version 2.0.64 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

## Create an AKS cluster

If you need an AKS cluster that meets the minimum requirements (GPU-enabled node and Kubernetes version 1.10 or later), complete the following steps. If you already have an AKS cluster that meets these requirements, [skip to the next section](#).

First, create a resource group for the cluster using the `az group create` command. The following example creates a resource group name *myResourceGroup* in the *eastus* region:

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

Now create an AKS cluster using the `az aks create` command. The following example creates a cluster with a single node of size `Standard_NC6`:

```
az aks create \
--resource-group myResourceGroup \
--name myAKSCluster \
--node-vm-size Standard_NC6 \
--node-count 1
```

Get the credentials for your AKS cluster using the `az aks get-credentials` command:

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```

# Install NVIDIA device plugin

Before the GPUs in the nodes can be used, you must deploy a DaemonSet for the NVIDIA device plugin. This DaemonSet runs a pod on each node to provide the required drivers for the GPUs.

First, create a namespace using the [kubectl create namespace](#) command, such as *gpu-resources*:

```
kubectl create namespace gpu-resources
```

Create a file named *nvidia-device-plugin-ds.yaml* and paste the following YAML manifest. This manifest is provided as part of the [NVIDIA device plugin for Kubernetes project](#).

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
 name: nvidia-device-plugin-daemonset
 namespace: gpu-resources
spec:
 selector:
 matchLabels:
 name: nvidia-device-plugin-ds
 updateStrategy:
 type: RollingUpdate
 template:
 metadata:
 # Mark this pod as a critical add-on; when enabled, the critical add-on scheduler
 # reserves resources for critical add-on pods so that they can be rescheduled after
 # a failure. This annotation works in tandem with the toleration below.
 annotations:
 scheduler.alpha.kubernetes.io/critical-pod: ""
 labels:
 name: nvidia-device-plugin-ds
 spec:
 tolerations:
 # Allow this pod to be rescheduled while the node is in "critical add-ons only" mode.
 # This, along with the annotation above marks this pod as a critical add-on.
 - key: CriticalAddonsOnly
 operator: Exists
 - key: nvidia.com/gpu
 operator: Exists
 effect: NoSchedule
 containers:
 - image: mcr.microsoft.com/oss/nvidia/k8s-device-plugin:1.11
 name: nvidia-device-plugin-ctr
 securityContext:
 allowPrivilegeEscalation: false
 capabilities:
 drop: ["ALL"]
 volumeMounts:
 - name: device-plugin
 mountPath: /var/lib/kubelet/device-plugins
 volumes:
 - name: device-plugin
 hostPath:
 path: /var/lib/kubelet/device-plugins
```

Now use the [kubectl apply](#) command to create the DaemonSet and confirm the NVIDIA device plugin is created successfully, as shown in the following example output:

```
$ kubectl apply -f nvidia-device-plugin-ds.yaml
daemonset "nvidia-device-plugin" created
```

## Use the AKS specialized GPU image (preview)

As alternative to these steps, AKS is providing a fully configured AKS image that already contains the [NVIDIA device plugin for Kubernetes](#).

### WARNING

You should not manually install the NVIDIA device plugin daemon set for clusters using the new AKS specialized GPU image.

Register the `GPUDedicatedVHDPreview` feature:

```
az feature register --name GPUDedicatedVHDPreview --namespace Microsoft.ContainerService
```

It might take several minutes for the status to show as **Registered**. You can check the registration status by using the `az feature list` command:

```
az feature list -o table --query "[?contains(name, 'Microsoft.ContainerService/GPUDedicatedVHDPreview')].{Name:name,State:properties.state}"
```

When the status shows as registered, refresh the registration of the `Microsoft.ContainerService` resource provider by using the `az provider register` command:

```
az provider register --namespace Microsoft.ContainerService
```

To install the `aks-preview` CLI extension, use the following Azure CLI commands:

```
az extension add --name aks-preview
```

To update the `aks-preview` CLI extension, use the following Azure CLI commands:

```
az extension update --name aks-preview
```

## Use the AKS specialized GPU image on new clusters (preview)

Configure the cluster to use the AKS specialized GPU image when the cluster is created. Use the `--aks-custom-headers` flag for the GPU agent nodes on your new cluster to use the AKS specialized GPU image.

```
az aks create --name myAKSCluster --resource-group myResourceGroup --node-vm-size Standard_NC6 --node-count 1 --aks-custom-headers UseGPUDedicatedVHD=true
```

If you want to create a cluster using the regular AKS images, you can do so by omitting the custom `--aks-custom-headers` tag. You can also choose to add more specialized GPU node pools as per below.

## Use the AKS specialized GPU image on existing clusters (preview)

Configure a new node pool to use the AKS specialized GPU image. Use the `--aks-custom-headers` flag for

the GPU agent nodes on your new node pool to use the AKS specialized GPU image.

```
az aks nodepool add --name gpu --cluster-name myAKSCluster --resource-group myResourceGroup --node-vm-size Standard_NC6 --node-count 1 --aks-custom-headers UseGPUDedicatedVHD=true
```

If you want to create a node pool using the regular AKS images, you can do so by omitting the custom `--aks-custom-headers` tag.

#### NOTE

If your GPU sku requires generation 2 virtual machines, you can create doing:

```
az aks nodepool add --name gpu --cluster-name myAKSCluster --resource-group myResourceGroup --node-vm-size Standard_NC6s_v2 --node-count 1 --aks-custom-headers UseGPUDedicatedVHD=true,usegen2vm=true
```

## Confirm that GPUs are schedutable

With your AKS cluster created, confirm that GPUs are schedutable in Kubernetes. First, list the nodes in your cluster using the [kubectl get nodes](#) command:

```
$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
aks-nodepool1-28993262-0	Ready	agent	13m	v1.12.7

Now use the [kubectl describe node](#) command to confirm that the GPUs are schedutable. Under the *Capacity* section, the GPU should list as `nvidia.com/gpu: 1`.

The following condensed example shows that a GPU is available on the node named `aks-nodepool1-18821093-0`:

```

$ kubectl describe node aks-nodepool1-28993262-0

Name: aks-nodepool1-28993262-0
Roles: agent
Labels: accelerator=nvidia

[...]

Capacity:
 attachable-volumes-azure-disk: 24
 cpu: 6
 ephemeral-storage: 101584140Ki
 hugepages-1Gi: 0
 hugepages-2Mi: 0
 memory: 57713784Ki
 nvidia.com/gpu: 1
 pods: 110

Allocatable:
 attachable-volumes-azure-disk: 24
 cpu: 5916m
 ephemeral-storage: 93619943269
 hugepages-1Gi: 0
 hugepages-2Mi: 0
 memory: 51702904Ki
 nvidia.com/gpu: 1
 pods: 110

System Info:
 Machine ID: b0cd6fb49ffe4900b56ac8df2eaa0376
 System UUID: 486A1C08-C459-6F43-AD6B-E9CD0F8AEC17
 Boot ID: f134525f-385d-4b4e-89b8-989f3abb490b
 Kernel Version: 4.15.0-1040-azure
 OS Image: Ubuntu 16.04.6 LTS
 Operating System: linux
 Architecture: amd64
 Container Runtime Version: docker://1.13.1
 Kubelet Version: v1.12.7
 Kube-Proxy Version: v1.12.7
 PodCIDR: 10.244.0.0/24
 ProviderID:
 azure://subscriptions/<guid>/resourceGroups/MC_myResourceGroup_myAKSCluster_eastus/providers/Microsoft.Compute/virtualMachines/aks-nodepool1-28993262-0
 Non-terminated Pods: (9 in total)
 Namespace Name CPU Requests CPU Limits Memory
 Requests Memory Limits AGE
 ----- ----- ----- -----
 --- ----- --- ---
 kube-system nvidia-device-plugin-daemonset-bbjlq 0 (0%) 0 (0%) 0 (0%)
 0 (0%) 2m39s

[...]

```

## Run a GPU-enabled workload

To see the GPU in action, schedule a GPU-enabled workload with the appropriate resource request. In this example, let's run a [Tensorflow](#) job against the [MNIST dataset](#).

Create a file named `samples-tf-mnist-demo.yaml` and paste the following YAML manifest. The following job manifest includes a resource limit of `nvidia.com/gpu: 1`:

#### NOTE

If you receive a version mismatch error when calling into drivers, such as, CUDA driver version is insufficient for CUDA runtime version, review the NVIDIA driver matrix compatibility chart - <https://docs.nvidia.com/Deploy/CUDA-Compatibility/index.html>

```
apiVersion: batch/v1
kind: Job
metadata:
 labels:
 app: samples-tf-mnist-demo
 name: samples-tf-mnist-demo
spec:
 template:
 metadata:
 labels:
 app: samples-tf-mnist-demo
 spec:
 containers:
 - name: samples-tf-mnist-demo
 image: mcr.microsoft.com/azuredocs/samples-tf-mnist-demo:gpu
 args: ["--max_steps", "500"]
 imagePullPolicy: IfNotPresent
 resources:
 limits:
 nvidia.com/gpu: 1
 restartPolicy: OnFailure
```

Use the `kubectl apply` command to run the job. This command parses the manifest file and creates the defined Kubernetes objects:

```
kubectl apply -f samples-tf-mnist-demo.yaml
```

## View the status and output of the GPU-enabled workload

Monitor the progress of the job using the `kubectl get jobs` command with the `--watch` argument. It may take a few minutes to first pull the image and process the dataset. When the *COMPLETIONS* column shows *1/1*, the job has successfully finished. Exit the `kubectl --watch` command with *Ctrl-C*.

```
$ kubectl get jobs samples-tf-mnist-demo --watch
NAME COMPLETIONS DURATION AGE
samples-tf-mnist-demo 0/1 3m29s 3m29s
samples-tf-mnist-demo 1/1 3m10s 3m36s
```

To look at the output of the GPU-enabled workload, first get the name of the pod with the `kubectl get pods` command:

```
$ kubectl get pods --selector app=samples-tf-mnist-demo
NAME READY STATUS RESTARTS AGE
samples-tf-mnist-demo-mtd44 0/1 Completed 0 4m39s
```

Now use the `kubectl logs` command to view the pod logs. The following example pod logs confirm that the appropriate GPU device has been discovered, `Tesla K80`. Provide the name for your own pod:

```
$ kubectl logs samples-tf-mnist-demo-smnr6

2019-05-16 16:08:31.258328: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports
instructions that this TensorFlow binary was not compiled to use: SSE4.1 SSE4.2 AVX AVX2 FMA
2019-05-16 16:08:31.396846: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1030] Found device 0 with
properties:
name: Tesla K80 major: 3 minor: 7 memoryClockRate(GHz): 0.8235
pciBusID: 2fd7:00:00.0
totalMemory: 11.17GiB freeMemory: 11.10GiB
2019-05-16 16:08:31.396886: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1120] Creating TensorFlow
device (/device:GPU:0) -> (device: 0, name: Tesla K80, pci bus id: 2fd7:00:00.0, compute capability: 3.7)
2019-05-16 16:08:36.076962: I tensorflow/stream_executor/dso_loader.cc:139] successfully opened CUDA library
libcupti.so.8.0 locally
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Extracting /tmp/tensorflow/input_data/train-images-idx3-ubyte.gz
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Extracting /tmp/tensorflow/input_data/train-labels-idx1-ubyte.gz
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting /tmp/tensorflow/input_data/t10k-images-idx3-ubyte.gz
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting /tmp/tensorflow/input_data/t10k-labels-idx1-ubyte.gz
Accuracy at step 0: 0.1081
Accuracy at step 10: 0.7457
Accuracy at step 20: 0.8233
Accuracy at step 30: 0.8644
Accuracy at step 40: 0.8848
Accuracy at step 50: 0.8889
Accuracy at step 60: 0.8898
Accuracy at step 70: 0.8979
Accuracy at step 80: 0.9087
Accuracy at step 90: 0.9099
Adding run metadata for 99
Accuracy at step 100: 0.9125
Accuracy at step 110: 0.9184
Accuracy at step 120: 0.922
Accuracy at step 130: 0.9161
Accuracy at step 140: 0.9219
Accuracy at step 150: 0.9151
Accuracy at step 160: 0.9199
Accuracy at step 170: 0.9305
Accuracy at step 180: 0.9251
Accuracy at step 190: 0.9258
Adding run metadata for 199
Accuracy at step 200: 0.9315
Accuracy at step 210: 0.9361
Accuracy at step 220: 0.9357
Accuracy at step 230: 0.9392
Accuracy at step 240: 0.9387
Accuracy at step 250: 0.9401
Accuracy at step 260: 0.9398
Accuracy at step 270: 0.9407
Accuracy at step 280: 0.9434
Accuracy at step 290: 0.9447
Adding run metadata for 299
Accuracy at step 300: 0.9463
Accuracy at step 310: 0.943
Accuracy at step 320: 0.9439
Accuracy at step 330: 0.943
Accuracy at step 340: 0.9457
Accuracy at step 350: 0.9497
Accuracy at step 360: 0.9481
Accuracy at step 370: 0.9466
Accuracy at step 380: 0.9514
Accuracy at step 390: 0.948
Adding run metadata for 399
Accuracy at step 400: 0.9469
Accuracy at step 410: 0.9489
Accuracy at step 420: 0.9529
```

```
Accuracy at step 430: 0.9507
Accuracy at step 440: 0.9504
Accuracy at step 450: 0.951
Accuracy at step 460: 0.9512
Accuracy at step 470: 0.9539
Accuracy at step 480: 0.9533
Accuracy at step 490: 0.9494
Adding run metadata for 499
```

## Clean up resources

To remove the associated Kubernetes objects created in this article, use the [kubectl delete job](#) command as follows:

```
kubectl delete jobs samples-tf-mnist-demo
```

## Next steps

To run Apache Spark jobs, see [Run Apache Spark jobs on AKS](#).

For more information about running machine learning (ML) workloads on Kubernetes, see [Kubeflow Labs](#).

# Tutorial: Deploy Django app on AKS with Azure Database for PostgreSQL - Flexible Server

4/21/2021 • 9 minutes to read • [Edit Online](#)

In this quickstart, you deploy a Django application on Azure Kubernetes Service (AKS) cluster with Azure Database for PostgreSQL - Flexible Server (Preview) using the Azure CLI.

AKS is a managed Kubernetes service that lets you quickly deploy and manage clusters. [Azure Database for PostgreSQL - Flexible Server \(Preview\)](#) is a fully managed database service designed to provide more granular control and flexibility over database management functions and configuration settings.

## NOTE

- Azure Database for PostgreSQL Flexible Server is currently in public preview
- This quickstart assumes a basic understanding of Kubernetes concepts, Django and PostgreSQL.

## Pre-requisites

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

- Use [Azure Cloud Shell](#) using the bash environment.



- If you prefer, [install](#) Azure CLI to run CLI reference commands.
  - If you're using a local install, sign in with Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. See [Sign in with Azure CLI](#) for additional sign-in options.
  - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with Azure CLI](#).
  - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#). This article requires the latest version of Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

## NOTE

If running the commands in this quickstart locally (instead of Azure Cloud Shell), ensure you run the commands as administrator.

## Create a resource group

An Azure resource group is a logical group in which Azure resources are deployed and managed. Let's create a resource group, *django-project* using the [az group create][az-group-create] command in the *eastus* location.

```
az group create --name django-project --location eastus
```

#### NOTE

The location for the resource group is where resource group metadata is stored. It is also where your resources run in Azure if you don't specify another region during resource creation.

The following example output shows the resource group created successfully:

```
{
 "id": "/subscriptions/<guid>/resourceGroups/django-project",
 "location": "eastus",
 "managedBy": null,

 "name": "django-project",
 "properties": {
 "provisioningState": "Succeeded"
 },
 "tags": null
}
```

## Create AKS cluster

Use the [az aks create](#) command to create an AKS cluster. The following example creates a cluster named *myAKSCluster* with one node. This will take several minutes to complete.

```
az aks create --resource-group django-project --name djangoappcluster --node-count 1 --generate-ssh-keys
```

After a few minutes, the command completes and returns JSON-formatted information about the cluster.

#### NOTE

When creating an AKS cluster a second resource group is automatically created to store the AKS resources. See [Why are two resource groups created with AKS?](#)

## Connect to the cluster

To manage a Kubernetes cluster, you use [kubectl](#), the Kubernetes command-line client. If you use Azure Cloud Shell, `kubectl` is already installed. To install `kubectl` locally, use the [az aks install-cli](#) command:

```
az aks install-cli
```

To configure `kubectl` to connect to your Kubernetes cluster, use the [az aks get-credentials](#) command. This command downloads credentials and configures the Kubernetes CLI to use them.

```
az aks get-credentials --resource-group django-project --name djangoappcluster
```

#### NOTE

The above command uses the default location for the [Kubernetes configuration file](#), which is `~/.kube/config`. You can specify a different location for your Kubernetes configuration file using `--file`.

To verify the connection to your cluster, use the [kubectl get](#) command to return a list of the cluster nodes.

```
kubectl get nodes
```

The following example output shows the single node created in the previous steps. Make sure that the status of the node is *Ready*:

NAME	STATUS	ROLES	AGE	VERSION
aks-nodepool1-31718369-0	Ready	agent	6m44s	v1.12.8

## Create an Azure Database for PostgreSQL - Flexible Server

Create a flexible server with the [az postgres flexible-server create](#) command. The following command creates a server using service defaults and values from your Azure CLI's local context:

```
az postgres flexible-server create --public-access <YOUR-IP-ADDRESS>
```

The server created has the below attributes:

- A new empty database, `postgres` is created when the server is first provisioned. In this quickstart we will use this database.
- Autogenerated server name, admin username, admin password, resource group name (if not already specified in local context), and in the same location as your resource group
- Service defaults for remaining server configurations: compute tier (General Purpose), compute size/SKU (Standard\_D2s\_v3 which uses 2vCores), backup retention period (7 days), and PostgreSQL version (12)
- Using public-access argument allow you to create a server with public access protected by firewall rules. By providing your IP address to add the firewall rule to allow access from your client machine.
- Since the command is using Local context it will create the server in the resource group `django-project` and in the region `eastus`.

## Build your Django docker image

Create a new [Django application](#) or use your existing Django project. Make sure your code is in this folder structure.

```
└── my-djangoapp
 └── views.py
 └── models.py
 └── forms.py
 └── templates
 . . .
 └── static
 . . .
└── my-django-project
 └── settings.py
 └── urls.py
 └── wsgi.py
 . . .
 └── Dockerfile
 └── requirements.txt
 └── manage.py
```

Update `ALLOWED_HOSTS` in `settings.py` to make sure the Django application uses the external IP that gets assigned to kubernetes app.

```
ALLOWED_HOSTS = ['*']
```

Update `DATABASES={ }` section in the `settings.py` file. The code snippet below is reading the database host, username and password from the Kubernetes manifest file.

```
DATABASES={
 'default':{
 'ENGINE':'django.db.backends.postgresql_psycopg2',
 'NAME':os.getenv('DATABASE_NAME'),
 'USER':os.getenv('DATABASE_USER'),
 'PASSWORD':os.getenv('DATABASE_PASSWORD'),
 'HOST':os.getenv('DATABASE_HOST'),
 'PORT':'5432',
 'OPTIONS': {'sslmode': 'require'}
 }
}
```

## Generate a requirements.txt file

Create a `requirements.txt` file to list out the dependencies for the Django Application. Here is an example `requirements.txt` file. You can use `pip freeze > requirements.txt` to generate a requirements.txt file for your existing application.

```
Django==2.2.17
postgres==3.0.0
psycopg2-binary==2.8.6
psycopg2-pool==1.1
pytz==2020.4
```

## Create a Dockerfile

Create a new file named `Dockerfile` and copy the code snippet below. This Dockerfile in setting up Python 3.8 and installing all the requirements listed in requirements.txt file.

```
Use the official Python image from the Docker Hub
FROM python:3.8.2

Make a new directory to put our code in.
RUN mkdir /code

Change the working directory.
WORKDIR /code

Copy to code folder
COPY . /code/

Install the requirements.
RUN pip install -r requirements.txt

Run the application:
CMD python manage.py runserver 0.0.0.0:8000
```

## Build your image

Make sure you're in the directory `my-django-app` in a terminal using the `cd` command. Run the following command to build your bulletin board image:

```
docker build --tag myblog:latest .
```

Deploy your image to [Docker hub](#) or [Azure Container registry](#).

#### IMPORTANT

If you are using Azure container regdistry (ACR), then run the `az aks update` command to attach ACR account with the AKS cluster.

```
az aks update -n myAKSCluster -g django-project --attach-acr <your-acr-name>
```

## Create Kubernetes manifest file

A Kubernetes manifest file defines a desired state for the cluster, such as what container images to run. Let's create a manifest file named `djangoapp.yaml` and copy in the following YAML definition.

#### IMPORTANT

- Replace `[DOCKER-HUB-USER/ACR ACCOUNT]/[YOUR-IMAGE-NAME]:[TAG]` with your actual Django docker image name and tag, for example `docker-hub-user/myblog:latest`.
- Update `env` section below with your `USERNAME`, `YOUR-DATABASE-USERNAME`, `YOUR-DATABASE-PASSWORD` of your postgres flexible server.

```

apiVersion: apps/v1
kind: Deployment
metadata:
 name: django-app
spec:
 replicas: 1
 selector:
 matchLabels:
 app: django-app
 template:
 metadata:
 labels:
 app: django-app
 spec:
 containers:
 - name: django-app
 image: [DOCKER-HUB-USER-OR-ACR-ACCOUNT]/[YOUR-IMAGE-NAME]:[TAG]
 ports:
 - containerPort: 80
 env:
 - name: DATABASE_HOST
 value: "SERVERNAME.postgres.database.azure.com"
 - name: DATABASE_USERNAME
 value: "YOUR-DATABASE-USERNAME"
 - name: DATABASE_PASSWORD
 value: "YOUR-DATABASE-PASSWORD"
 - name: DATABASE_NAME
 value: "postgres"
 affinity:
 podAntiAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 - labelSelector:
 matchExpressions:
 - key: "app"
 operator: In
 values:
 - django-app
 topologyKey: "kubernetes.io/hostname"

apiVersion: v1
kind: Service
metadata:
 name: python-svc
spec:
 type: LoadBalancer
 ports:
 - port: 8000
 selector:
 app: django-app

```

## Deploy Django to AKS cluster

Deploy the application using the [kubectl apply](#) command and specify the name of your YAML manifest:

```
kubectl apply -f djangoapp.yaml
```

The following example output shows the Deployments and Services created successfully:

```
deployment "django-app" created
service "python-svc" created
```

A deployment `django-app` allows you to describes details on of your deployment such as which images to use

for the app, the number of pods and pod configuration. A service `python-svc` is created to expose the application through an external IP.

## Test the application

When the application runs, a Kubernetes service exposes the application front end to the internet. This process can take a few minutes to complete.

To monitor progress, use the `kubectl get service` command with the `--watch` argument.

```
kubectl get service django-app --watch
```

Initially the *EXTERNAL-IP* for the *django-app* service is shown as *pending*.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
django-app	LoadBalancer	10.0.37.27	<pending>	80:30572/TCP	6s

When the *EXTERNAL-IP* address changes from *pending* to an actual public IP address, use `CTRL-C` to stop the `kubectl` watch process. The following example output shows a valid public IP address assigned to the service:

```
django-app LoadBalancer 10.0.37.27 52.179.23.131 80:30572/TCP 2m
```

Now open a web browser to the external IP address of your service view the Django application.

### NOTE

- Currently the Django site is not using HTTPS. It is recommended to [ENABLE TLS with your own certificates](#).
- You can enable [HTTP routing](#) for your cluster. When http routing is enabled, it configures an Ingress controller in your AKS cluster. As applications are deployed, the solution also creates publicly accessible DNS names for application endpoints.

## Run database migrations

For any django application, you would need to run database migration or collect static files. You can run these django shell commands using `$ kubectl exec <pod-name> -- [COMMAND]`. Before running the command you need to find the pod name using `kubectl get pods`.

```
$ kubectl get pods
```

You will see an output like this

NAME	READY	STATUS	RESTARTS	AGE
django-app-5d9cd6cd8-16x4b	1/1	Running	0	2m

Once the pod name has been found you can run django database migrations with the command

```
$ kubectl exec <pod-name> -- [COMMAND]. Note /code/ is the working directory for the project define in Dockerfile above.
```

```
$ kubectl exec django-app-5d9cd6cd8-16x4b -- python /code/manage.py migrate
```

The output would look like

```
Operations to perform:
 Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
 Applying contenttypes.0001_initial... OK
 Applying auth.0001_initial... OK
 Applying admin.0001_initial... OK
 Applying admin.0002_logentry_remove_auto_add... OK
 Applying admin.0003_logentry_add_action_flag_choices... OK

```

If you run into issues, please run `kubectl logs <pod-name>` to see what exception is thrown by your application. If the application is working successfully you would see an output like this when running `kubectl logs`.

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 17 unapplied migration(s). Your project may not work properly until you apply the migrations for
app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
December 08, 2020 - 23:24:14
Django version 2.2.17, using settings 'django_postgres_app.settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with CONTROL-C.
```

## Clean up the resources

To avoid Azure charges, you should clean up unneeded resources. When the cluster is no longer needed, use the [az group delete](#) command to remove the resource group, container service, and all related resources.

```
az group delete --name django-project --yes --no-wait
```

### NOTE

When you delete the cluster, the Azure Active Directory service principal used by the AKS cluster is not removed. For steps on how to remove the service principal, see [AKS service principal considerations and deletion](#). If you used a managed identity, the identity is managed by the platform and does not require removal.

## Next steps

- Learn how to [access the Kubernetes web dashboard](#) for your AKS cluster
- Learn how to [enable continuous deployment](#)
- Learn how to [scale your cluster](#)
- Learn how to manage your [postgres flexible server](#)
- Learn how to [configure server parameters](#) for your database server.

# Tutorial: Deploy WordPress app on AKS with Azure Database for MySQL - Flexible Server

4/21/2021 • 7 minutes to read • [Edit Online](#)

In this quickstart, you deploy a WordPress application on Azure Kubernetes Service (AKS) cluster with Azure Database for MySQL - Flexible Server (Preview) using the Azure CLI. [AKS](#) is a managed Kubernetes service that lets you quickly deploy and manage clusters. [Azure Database for MySQL - Flexible Server \(Preview\)](#) is a fully managed database service designed to provide more granular control and flexibility over database management functions and configuration settings. Currently Flexible server is in Preview.

## NOTE

- Azure Database for MySQL Flexible Server is currently in public preview
- This quickstart assumes a basic understanding of Kubernetes concepts, WordPress and MySQL.

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

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#).  
[A](#) Launch Cloud Shell
- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
  - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).
- This article requires the latest version of Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

## NOTE

If running the commands in this quickstart locally (instead of Azure Cloud Shell), ensure you run the commands as administrator.

## Create a resource group

An Azure resource group is a logical group in which Azure resources are deployed and managed. Let's create a resource group, *wordpress-project* using the [az group create][az-group-create] command in the *eastus* location.

```
az group create --name wordpress-project --location eastus
```

#### NOTE

The location for the resource group is where resource group metadata is stored. It is also where your resources run in Azure if you don't specify another region during resource creation.

The following example output shows the resource group created successfully:

```
{
 "id": "/subscriptions/<guid>/resourceGroups/wordpress-project",
 "location": "eastus",
 "managedBy": null,
 "name": "wordpress-project",
 "properties": {
 "provisioningState": "Succeeded"
 },
 "tags": null
}
```

## Create AKS cluster

Use the [az aks create](#) command to create an AKS cluster. The following example creates a cluster named *myAKSCluster* with one node. This will take several minutes to complete.

```
az aks create --resource-group wordpress-project --name myAKScluster --node-count 1 --generate-ssh-keys
```

After a few minutes, the command completes and returns JSON-formatted information about the cluster.

#### NOTE

When creating an AKS cluster a second resource group is automatically created to store the AKS resources. See [Why are two resource groups created with AKS?](#)

## Connect to the cluster

To manage a Kubernetes cluster, you use [kubectl](#), the Kubernetes command-line client. If you use Azure Cloud Shell, `kubectl` is already installed. To install `kubectl` locally, use the [az aks install-cli](#) command:

```
az aks install-cli
```

To configure `kubectl` to connect to your Kubernetes cluster, use the [az aks get-credentials](#) command. This command downloads credentials and configures the Kubernetes CLI to use them.

```
az aks get-credentials --resource-group wordpress-project --name myAKScluster
```

#### NOTE

The above command uses the default location for the [Kubernetes configuration file](#), which is `~/.kube/config`. You can specify a different location for your Kubernetes configuration file using `--file`.

To verify the connection to your cluster, use the `kubectl get` command to return a list of the cluster nodes.

```
kubectl get nodes
```

The following example output shows the single node created in the previous steps. Make sure that the status of the node is *Ready*:

NAME	STATUS	ROLES	AGE	VERSION
aks-nodepool1-31718369-0	Ready	agent	6m44s	v1.12.8

## Create an Azure Database for MySQL - Flexible Server

Create a flexible server with the `az mysql flexible-server create` command. The following command creates a server using service defaults and values from your Azure CLI's local context:

```
az mysql flexible-server create --public-access <YOUR-IP-ADDRESS>
```

The server created has the below attributes:

- A new empty database, `flexibleserverdb` is created when the server is first provisioned. In this quickstart we will use this database.
- Autogenerated server name, admin username, admin password, resource group name (if not already specified in local context), and in the same location as your resource group
- Service defaults for remaining server configurations: compute tier (Burstable), compute size/SKU (B1MS), backup retention period (7 days), and MySQL version (5.7)
- Using public-access argument allow you to create a server with public access protected by firewall rules. By providing your IP address to add the firewall rule to allow access from your client machine.
- Since the command is using Local context it will create the server in the resource group `wordpress-project` and in the region `eastus`.

## Build your WordPress docker image

Download the [latest WordPress](#) version. Create new directory `my-wordpress-app` for your project and use this simple folder structure

```
└── my-wordpress-app
 └── public
 ├── wp-admin
 | └── css
 | ...
 | └── wp-content
 | └── plugins
 | ...
 | └── wp-includes
 | ...
 | └── wp-config-sample.php
 | └── index.php
 | ...
 └── Dockerfile
```

Rename `wp-config-sample.php` to `wp-config.php` and replace lines 21 to 32 with this code snippet. The code snippet below is reading the database host, username and password from the Kubernetes manifest file.

```

//Using environment variables for DB connection information

// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */

$connectstr_dbhost = getenv('DATABASE_HOST');
$connectstr_dbusername = getenv('DATABASE_USERNAME');
$connectstr_dbpassword = getenv('DATABASE_PASSWORD');

/** MySQL database name */
define('DB_NAME', 'flexibleserverdb');

/** MySQL database username */
define('DB_USER', $connectstr_dbusername);

/** MySQL database password */
define('DB_PASSWORD', $connectstr_dbpassword);

/** MySQL hostname */
define('DB_HOST', $connectstr_dbhost);

/** Database Charset to use in creating database tables. */
define('DB_CHARSET', 'utf8');

/** The Database Collate type. Don't change this if in doubt. */
define('DB_COLLATE', '');

/** SSL*/
define('MYSQL_CLIENT_FLAGS', MYSQLI_CLIENT_SSL);

```

## Create a Dockerfile

Create a new Dockerfile and copy this code snippet. This Dockerfile is setting up Apache web server with PHP and enabling mysqli extension.

```

FROM php:7.2-apache
COPY public/ /var/www/html/
RUN docker-php-ext-install mysqli
RUN docker-php-ext-enable mysqli

```

## Build your docker image

Make sure you're in the directory `my-wordpress-app` in a terminal using the `cd` command. Run the following command to build the image:

```
docker build --tag myblog:latest .
```

Deploy your image to [Docker hub](#) or [Azure Container registry](#).

### IMPORTANT

If you are using Azure container registry (ACR), then run the `az aks update` command to attach ACR account with the AKS cluster.

```
az aks update -n myAKSCluster -g wordpress-project --attach-acr <your-acr-name>
```

# Create Kubernetes manifest file

A Kubernetes manifest file defines a desired state for the cluster, such as what container images to run. Let's create a manifest file named `mywordpress.yaml` and copy in the following YAML definition.

## IMPORTANT

- Replace `[DOCKER-HUB-USER/ACR ACCOUNT]/[YOUR-IMAGE-NAME]:[TAG]` with your actual WordPress docker image name and tag, for example `docker-hub-user/myblog:latest`.
- Update `env` section below with your `SERVERNAME`, `YOUR-DATABASE-USERNAME`, `YOUR-DATABASE-PASSWORD` of your MySQL flexible server.

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: wordpress-blog
spec:
 replicas: 1
 selector:
 matchLabels:
 app: wordpress-blog
 template:
 metadata:
 labels:
 app: wordpress-blog
 spec:
 containers:
 - name: wordpress-blog
 image: [DOCKER-HUB-USER-OR-ACR-ACCOUNT]/[YOUR-IMAGE-NAME]:[TAG]
 ports:
 - containerPort: 80
 env:
 - name: DATABASE_HOST
 value: "SERVERNAME.mysql.database.azure.com"
 - name: DATABASE_USERNAME
 value: "YOUR-DATABASE-USERNAME"
 - name: DATABASE_PASSWORD
 value: "YOUR-DATABASE-PASSWORD"
 - name: DATABASE_NAME
 value: "flexibleserverdb"
 affinity:
 podAntiAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 - labelSelector:
 matchExpressions:
 - key: "app"
 operator: In
 values:
 - wordpress-blog
 topologyKey: "kubernetes.io/hostname"

apiVersion: v1
kind: Service
metadata:
 name: php-svc
spec:
 type: LoadBalancer
 ports:
 - port: 80
 selector:
 app: wordpress-blog
```

# Deploy WordPress to AKS cluster

Deploy the application using the [kubectl apply](#) command and specify the name of your YAML manifest:

```
kubectl apply -f mywordpress.yaml
```

The following example output shows the Deployments and Services created successfully:

```
deployment "wordpress-blog" created
service "php-svc" created
```

## Test the application

When the application runs, a Kubernetes service exposes the application front end to the internet. This process can take a few minutes to complete.

To monitor progress, use the [kubectl get service](#) command with the `--watch` argument.

```
kubectl get service wordpress-blog --watch
```

Initially the *EXTERNAL-IP* for the *wordpress-blog* service is shown as *pending*.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
wordpress-blog	LoadBalancer	10.0.37.27	<pending>	80:30572/TCP	6s

When the *EXTERNAL-IP* address changes from *pending* to an actual public IP address, use `CTRL-C` to stop the [kubectl](#) watch process. The following example output shows a valid public IP address assigned to the service:

```
wordpress-blog LoadBalancer 10.0.37.27 52.179.23.131 80:30572/TCP 2m
```

## Browse WordPress

Open a web browser to the external IP address of your service to see your WordPress installation page.



## Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

### Information needed

Please provide the following information. Don't worry, you can always change these settings later.

**Site Title**

**Username**

Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.

**Password**

  
Strong

Hide

**Important:** You will need this password to log in. Please store it in a secure location.

**Your Email**

Double-check your email address before continuing.

**Search Engine Visibility**

Discourage search engines from indexing this site  
It is up to search engines to honor this request.

[Install WordPress](#)

#### NOTE

- Currently the WordPress site is not using HTTPS. It is recommended to [ENABLE TLS with your own certificates](#).
- You can enable [HTTP routing](#) for your cluster.

## Clean up the resources

To avoid Azure charges, you should clean up unneeded resources. When the cluster is no longer needed, use the [az group delete](#) command to remove the resource group, container service, and all related resources.

```
az group delete --name wordpress-project --yes --no-wait
```

#### **NOTE**

When you delete the cluster, the Azure Active Directory service principal used by the AKS cluster is not removed. For steps on how to remove the service principal, see [AKS service principal considerations and deletion](#). If you used a managed identity, the identity is managed by the platform and does not require removal.

## Next steps

- Learn how to [access the Kubernetes web dashboard](#) for your AKS cluster
- Learn how to [scale your cluster](#)
- Learn how to manage your [MySQL flexible server](#)
- Learn how to [configure server parameters](#) for your database server.

# Deploy a Java application with Open Liberty or WebSphere Liberty on an Azure Kubernetes Service (AKS) cluster

3/5/2021 • 7 minutes to read • [Edit Online](#)

This article demonstrates how to:

- Run your Java, Java EE, Jakarta EE, or MicroProfile application on the Open Liberty or WebSphere Liberty runtime.
- Build the application Docker image using Open Liberty container images.
- Deploy the containerized application to an AKS cluster using the Open Liberty Operator.

The Open Liberty Operator simplifies the deployment and management of applications running on Kubernetes clusters. With Open Liberty Operator, you can also perform more advanced operations, such as gathering traces and dumps.

For more details on Open Liberty, see [the Open Liberty project page](#). For more details on IBM WebSphere Liberty, see [the WebSphere Liberty product page](#).

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

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#).  
[A Launch Cloud Shell](#)
- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
  - If you're using a local installation, sign in to the Azure CLI by using the `az login` command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.
- This article requires the latest version of Azure CLI. If using Azure Cloud Shell, the latest version is already installed.
- If running the commands in this guide locally (instead of Azure Cloud Shell):
  - Prepare a local machine with Unix-like operating system installed (for example, Ubuntu, macOS, Windows Subsystem for Linux).
  - Install a Java SE implementation (for example, [AdoptOpenJDK OpenJDK 8 LTS/OpenJ9](#)).
  - Install [Maven](#) 3.5.0 or higher.
  - Install [Docker](#) for your OS.

## Create a resource group

An Azure resource group is a logical group in which Azure resources are deployed and managed.

Create a resource group called *java-liberty-project* using the [az group create](#) command in the *eastus* location. This resource group will be used later for creating the Azure Container Registry (ACR) instance and the AKS cluster.

```
RESOURCE_GROUP_NAME=java-liberty-project
az group create --name $RESOURCE_GROUP_NAME --location eastus
```

## Create an ACR instance

Use the [az acr create](#) command to create the ACR instance. The following example creates an ACR instance named *youruniqueacrname*. Make sure *youruniqueacrname* is unique within Azure.

```
REGISTRY_NAME=youruniqueacrname
az acr create --resource-group $RESOURCE_GROUP_NAME --name $REGISTRY_NAME --sku Basic --admin-enabled
```

After a short time, you should see a JSON output that contains:

```
"provisioningState": "Succeeded",
"publicNetworkAccess": "Enabled",
"resourceGroup": "java-liberty-project",
```

### Connect to the ACR instance

You will need to sign in to the ACR instance before you can push an image to it. Run the following commands to verify the connection:

```
LOGIN_SERVER=$(az acr show -n $REGISTRY_NAME --query 'loginServer' -o tsv)
USER_NAME=$(az acr credential show -n $REGISTRY_NAME --query 'username' -o tsv)
PASSWORD=$(az acr credential show -n $REGISTRY_NAME --query 'passwords[0].value' -o tsv)

docker login $LOGIN_SERVER -u $USER_NAME -p $PASSWORD
```

You should see `Login Succeeded` at the end of command output if you have logged into the ACR instance successfully.

## Create an AKS cluster

Use the [az aks create](#) command to create an AKS cluster. The following example creates a cluster named *myAKSCluster* with one node. This will take several minutes to complete.

```
CLUSTER_NAME=myAKSCluster
az aks create --resource-group $RESOURCE_GROUP_NAME --name $CLUSTER_NAME --node-count 1 --generate-ssh-keys
--enable-managed-identity
```

After a few minutes, the command completes and returns JSON-formatted information about the cluster, including the following:

```
"nodeResourceGroup": "MC_java-liberty-project_myAKSCluster_eastus",
"privateFqdn": null,
"provisioningState": "Succeeded",
"resourceGroup": "java-liberty-project",
```

## Connect to the AKS cluster

To manage a Kubernetes cluster, you use [kubectl](#), the Kubernetes command-line client. If you use Azure Cloud Shell, `kubectl` is already installed. To install `kubectl` locally, use the [az aks install-cli](#) command:

```
az aks install-cli
```

To configure `kubectl` to connect to your Kubernetes cluster, use the [az aks get-credentials](#) command. This command downloads credentials and configures the Kubernetes CLI to use them.

```
az aks get-credentials --resource-group $RESOURCE_GROUP_NAME --name $CLUSTER_NAME --overwrite-existing
```

### NOTE

The above command uses the default location for the [Kubernetes configuration file](#), which is `~/.kube/config`. You can specify a different location for your Kubernetes configuration file using `--file`.

To verify the connection to your cluster, use the `kubectl get` command to return a list of the cluster nodes.

```
kubectl get nodes
```

The following example output shows the single node created in the previous steps. Make sure that the status of the node is *Ready*:

NAME	STATUS	ROLES	AGE	VERSION
aks-nodepool1-xxxxxxx-yyyyyyyyyy	Ready	agent	76s	v1.18.10

## Install Open Liberty Operator

After creating and connecting to the cluster, install the [Open Liberty Operator](#) by running the following commands.

```
OPERATOR_NAMESPACE=default
WATCH_NAMESPACE=''

Install Custom Resource Definitions (CRDs) for OpenLibertyApplication
kubectl apply -f https://raw.githubusercontent.com/OpenLiberty/open-liberty-operator/master/deploy/releases/0.7.0/openliberty-app-crd.yaml

Install cluster-level role-based access
curl -L https://raw.githubusercontent.com/OpenLiberty/open-liberty-operator/master/deploy/releases/0.7.0/openliberty-app-cluster-rbac.yaml \
| sed -e "s/OPEN_LIBERTY_OPERATOR_NAMESPACE/${OPERATOR_NAMESPACE}/" \
| kubectl apply -f -

Install the operator
curl -L https://raw.githubusercontent.com/OpenLiberty/open-liberty-operator/master/deploy/releases/0.7.0/openliberty-app-operator.yaml \
| sed -e "s/OPEN_LIBERTY_WATCH_NAMESPACE/${WATCH_NAMESPACE}/" \
| kubectl apply -n ${OPERATOR_NAMESPACE} -f -
```

## Build application image

To deploy and run your Liberty application on the AKS cluster, containerize your application as a Docker image

using Open Liberty container images or WebSphere Liberty container images.

1. Clone the sample code for this guide. The sample is on [GitHub](#).
2. Change directory to `javaee-app-simple-cluster` of your local clone.
3. Run `mvn clean package` to package the application.
4. Run `mvn liberty:dev` to test the application. You should see  
`The defaultServer server is ready to run a smarter planet.` in the command output if successful. Use `CTRL-C` to stop the application.
5. Run one of the following commands to build the application image and push it to the ACR instance.
  - Build with Open Liberty base image if you prefer to use Open Liberty as a lightweight open source Java™ runtime:

```
Build and tag application image. This will cause the ACR instance to pull the necessary Open Liberty base images.
az acr build -t javaee-cafe-simple:1.0.0 -r $REGISTRY_NAME .
```
  - Build with WebSphere Liberty base image if you prefer to use a commercial version of Open Liberty:

```
Build and tag application image. This will cause the ACR instance to pull the necessary WebSphere Liberty base images.
az acr build -t javaee-cafe-simple:1.0.0 -r $REGISTRY_NAME --file=Dockerfile-wlp .
```

## Deploy application on the AKS cluster

Follow steps below to deploy the Liberty application on the AKS cluster.

1. Create a pull secret so that the AKS cluster is authenticated to pull image from the ACR instance.

```
kubectl create secret docker-registry acr-secret \
--docker-server=${LOGIN_SERVER} \
--docker-username=${USER_NAME} \
--docker-password=${PASSWORD}
```

2. Verify the current working directory is `javaee-app-simple-cluster` of your local clone.

3. Run the following commands to deploy your Liberty application with 3 replicas to the AKS cluster.

Command output is also shown inline.

```

Create OpenLibertyApplication "javaee-app-simple-cluster"
cat openlibertyapplication.yaml | sed -e "s/\${Container_Registry_URL}/\$${LOGIN_SERVER}/g" | sed -e
"s/\${REPLICAS}/3/g" | kubectl apply -f -

openlibertyapplication.openliberty.io/javaee-app-simple-cluster created

Check if OpenLibertyApplication instance is created
kubectl get openlibertyapplication javaee-app-simple-cluster

NAME IMAGE EXPOSED
RECONCILED AGE
javaee-app-simple-cluster youruniqueacrname.azurecr.io/javaee-cafe-simple:1.0.0 True
59s

Check if deployment created by Operator is ready
kubectl get deployment javaee-app-simple-cluster --watch

NAME READY UP-TO-DATE AVAILABLE AGE
javaee-app-simple-cluster 0/3 3 0 20s

```

4. Wait until you see `3/3` under the `READY` column and `3` under the `AVAILABLE` column, use `CTRL-C` to stop the `kubectl` watch process.

## Test the application

When the application runs, a Kubernetes load balancer service exposes the application front end to the internet. This process can take a while to complete.

To monitor progress, use the `kubectl get service` command with the `--watch` argument.

```

kubectl get service javaee-app-simple-cluster --watch

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
javaee-app-simple-cluster LoadBalancer 10.0.251.169 52.152.189.57 80:31732/TCP 68s

```

Once the `EXTERNAL-IP` address changes from *pending* to an actual public IP address, use `CTRL-C` to stop the `kubectl` watch process.

Open a web browser to the external IP address of your service (`52.152.189.57` for the above example) to see the application home page. You should see the pod name of your application replicas displayed at the top-left of the page. Wait for a few minutes and refresh the page to see a different pod name displayed due to load balancing provided by the AKS cluster.

Pod name: javaee-app-simple-cluster-7c9448d69-2f9vh

Hi. My name is Duke. This is my cafe, Would you like to create a new coffee today?

### Our coffees

Id	Name	Price	
1	Coffee 1	10.0	<a href="#">Delete</a>
2	Coffee 2	20.0	<a href="#">Delete</a>

New coffee

Name

Price

[Submit](#)

#### NOTE

- Currently the application is not using HTTPS. It is recommended to [ENABLE TLS with your own certificates](#).

## Clean up the resources

To avoid Azure charges, you should clean up unnecessary resources. When the cluster is no longer needed, use the [az group delete](#) command to remove the resource group, container service, container registry, and all related resources.

```
az group delete --name $RESOURCE_GROUP_NAME --yes --no-wait
```

## Next steps

You can learn more from references used in this guide:

- [Azure Kubernetes Service](#)
- [Open Liberty](#)
- [Open Liberty Operator](#)
- [Open Liberty Server Configuration](#)
- [Liberty Maven Plugin](#)
- [Open Liberty Container Images](#)
- [WebSphere Liberty Container Images](#)

# Use Azure API Management with microservices deployed in Azure Kubernetes Service

3/24/2021 • 7 minutes to read • [Edit Online](#)

Microservices are perfect for building APIs. With [Azure Kubernetes Service](#) (AKS), you can quickly deploy and operate a [microservices-based architecture](#) in the cloud. You can then leverage [Azure API Management](#) (API Management) to publish your microservices as APIs for internal and external consumption. This article describes the options of deploying API Management with AKS. It assumes basic knowledge of Kubernetes, API Management, and Azure networking.

## Background

When publishing microservices as APIs for consumption, it can be challenging to manage the communication between the microservices and the clients that consume them. There is a multitude of cross-cutting concerns such as authentication, authorization, throttling, caching, transformation, and monitoring. These concerns are valid regardless of whether the microservices are exposed to internal or external clients.

The [API Gateway](#) pattern addresses these concerns. An API gateway serves as a front door to the microservices, decouples clients from your microservices, adds an additional layer of security, and decreases the complexity of your microservices by removing the burden of handling cross cutting concerns.

[Azure API Management](#) is a turnkey solution to solve your API gateway needs. You can quickly create a consistent and modern gateway for your microservices and publish them as APIs. As a full-lifecycle API management solution, it also provides additional capabilities including a self-service developer portal for API discovery, API lifecycle management, and API analytics.

When used together, AKS and API Management provide a platform for deploying, publishing, securing, monitoring, and managing your microservices-based APIs. In this article, we will go through a few options of deploying AKS in conjunction with API Management.

## Kubernetes Services and APIs

In a Kubernetes cluster, containers are deployed in [Pods](#), which are ephemeral and have a lifecycle. When a worker node dies, the Pods running on the node are lost. Therefore, the IP address of a Pod can change anytime. We cannot rely on it to communicate with the pod.

To solve this problem, Kubernetes introduced the concept of [Services](#). A Kubernetes Service is an abstraction layer which defines a logic group of Pods and enables external traffic exposure, load balancing and service discovery for those Pods.

When we are ready to publish our microservices as APIs through API Management, we need to think about how to map our Services in Kubernetes to APIs in API Management. There are no set rules. It depends on how you designed and partitioned your business capabilities or domains into microservices at the beginning. For instance, if the pods behind a Service are responsible for all operations on a given resource (e.g., Customer), the Service may be mapped to one API. If operations on a resource are partitioned into multiple microservices (e.g., GetOrder, PlaceOrder), then multiple Services may be logically aggregated into one single API in API management (See Fig. 1).

The mappings can also evolve. Since API Management creates a façade in front of the microservices, it allows us to refactor and right-size our microservices over time.



## Deploy API Management in front of AKS

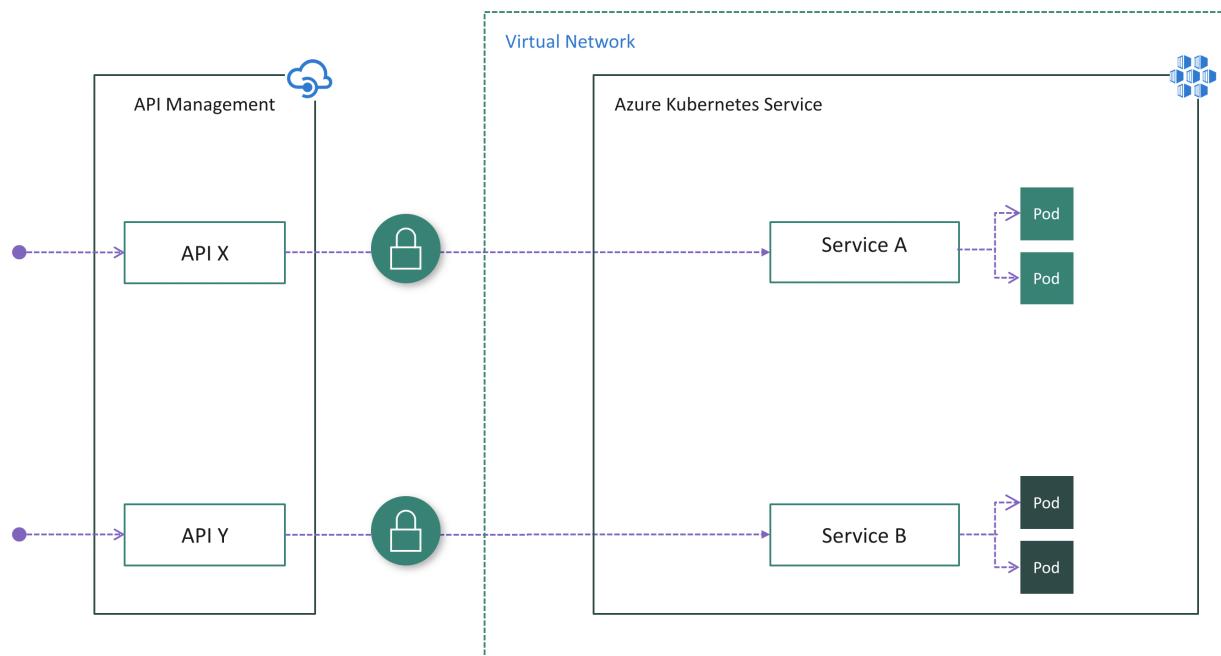
There are a few options of deploying API Management in front of an AKS cluster.

While an AKS cluster is always deployed in a virtual network (VNet), an API Management instance is not required to be deployed in a VNet. When API Management does not reside within the cluster VNet, the AKS cluster has to publish public endpoints for API Management to connect to. In that case, there is a need to secure the connection between API Management and AKS. In other words, we need to ensure the cluster can only be accessed exclusively through API Management. Let's go through the options.

### Option 1: Expose Services publicly

Services in an AKS cluster can be exposed publicly using [Service types](#) of NodePort, LoadBalancer, or ExternalName. In this case, Services are accessible directly from public internet. After deploying API Management in front of the cluster, we need to ensure all inbound traffic goes through API Management by applying authentication in the microservices. For instance, API Management can include an access token in each request made to the cluster. Each microservice is responsible for validating the token before processing the request.

This might be the easiest option to deploy API Management in front of AKS, especially if you already have authentication logic implemented in your microservices.



Pros:

- Easy configuration on the API Management side because it does not need to be injected into the cluster VNet
- No change on the AKS side if Services are already exposed publicly and authentication logic already exists in microservices

Cons:

- Potential security risk due to public visibility of Service endpoints
- No single-entry point for inbound cluster traffic
- Complicates microservices with duplicate authentication logic

### Option 2: Install an Ingress Controller

Although Option 1 might be easier, it has notable drawbacks as mentioned above. If an API Management instance does not reside in the cluster VNet, Mutual TLS authentication (mTLS) is a robust way of ensuring the traffic is secure and trusted in both directions between an API Management instance and an AKS cluster.

Mutual TLS authentication is [natively supported](#) by API Management and can be enabled in Kubernetes by [installing an Ingress Controller](#) (Fig. 3). As a result, authentication will be performed in the Ingress Controller, which simplifies the microservices. Additionally, you can add the IP addresses of API Management to the allowed list by Ingress to make sure only API Management has access to the cluster.



Pros:

- Easy configuration on the API Management side because it does not need to be injected into the cluster VNet and mTLS is natively supported
- Centralizes protection for inbound cluster traffic at the Ingress Controller layer
- Reduces security risk by minimizing publicly visible cluster endpoints

Cons:

- Increases complexity of cluster configuration due to extra work to install, configure and maintain the Ingress Controller and manage certificates used for mTLS
- Security risk due to public visibility of Ingress Controller endpoint(s)

When you publish APIs through API Management, it's easy and common to secure access to those APIs by using subscription keys. Developers who need to consume the published APIs must include a valid subscription key in HTTP requests when they make calls to those APIs. Otherwise, the calls are rejected immediately by the API

Management gateway. They aren't forwarded to the back-end services.

To get a subscription key for accessing APIs, a subscription is required. A subscription is essentially a named container for a pair of subscription keys. Developers who need to consume the published APIs can get subscriptions. And they don't need approval from API publishers. API publishers can also create subscriptions directly for API consumers.

### Option 3: Deploy APIM inside the cluster VNet

In some cases, customers with regulatory constraints or strict security requirements may find Option 1 and 2 not viable solutions due to publicly exposed endpoints. In others, the AKS cluster and the applications that consume the microservices might reside within the same VNet, hence there is no reason to expose the cluster publicly as all API traffic will remain within the VNet. For these scenarios, you can deploy API Management into the cluster VNet. [API Management Developer and Premium tiers](#) support VNet deployment.

There are two modes of [deploying API Management into a VNet](#) – External and Internal.

If API consumers do not reside in the cluster VNet, the External mode (Fig. 4) should be used. In this mode, the API Management gateway is injected into the cluster VNet but accessible from public internet via an external load balancer. It helps to hide the cluster completely while still allowing external clients to consume the microservices. Additionally, you can use Azure networking capabilities such as Network Security Groups (NSG) to restrict network traffic.



If all API consumers reside within the cluster VNet, then the Internal mode (Fig. 5) could be used. In this mode, the API Management gateway is injected into the cluster VNET and accessible only from within this VNet via an internal load balancer. There is no way to reach the API Management gateway or the AKS cluster from public internet.



In both cases, the AKS cluster is not publicly visible. Compared to Option 2, the Ingress Controller may not be necessary. Depending on your scenario and configuration, authentication might still be required between API Management and your microservices. For instance, if a Service Mesh is adopted, it always requires mutual TLS authentication.

Pros:

- The most secure option because the AKS cluster has no public endpoint
- Simplifies cluster configuration since it has no public endpoint
- Ability to hide both API Management and AKS inside the VNet using the Internal mode
- Ability to control network traffic using Azure networking capabilities such as Network Security Groups (NSG)

Cons:

- Increases complexity of deploying and configuring API Management to work inside the VNet

## Next steps

- Learn more about [Network concepts for applications in AKS](#)
- Learn more about [How to use API Management with virtual networks](#)

# About service meshes

2/25/2020 • 4 minutes to read • [Edit Online](#)

A service mesh provides capabilities like traffic management, resiliency, policy, security, strong identity, and observability to your workloads. Your application is decoupled from these operational capabilities and the service mesh moves them out of the application layer, and down to the infrastructure layer.

## Scenarios

These are some of the scenarios that can be enabled for your workloads when you use a service mesh:

- **Encrypt all traffic in cluster** - Enable mutual TLS between specified services in the cluster. This can be extended to ingress and egress at the network perimeter. Provides a secure by default option with no changes needed for application code and infrastructure.
- **Canary and phased rollouts** - Specify conditions for a subset of traffic to be routed to a set of new services in the cluster. On successful test of canary release, remove conditional routing and phase gradually increasing % of all traffic to new service. Eventually all traffic will be directed to new service.
- **Traffic management and manipulation** - Create a policy on a service that will rate limit all traffic to a version of a service from a specific origin. Or a policy that applies a retry strategy to classes of failures between specified services. Mirror live traffic to new versions of services during a migration or to debug issues. Inject faults between services in a test environment to test resiliency.
- **Observability** - Gain insight into how your services are connected the traffic that flows between them. Obtain metrics, logs, and traces for all traffic in cluster, and ingress/egress. Add distributed tracing abilities to your applications.

## Architecture

A service mesh is typically composed of a control plane and the data plane.

The **control plane** has a number of components that support managing the service mesh. This will typically include a management interface which could be a UI or an API. There will also typically be components that manage the rule and policy definitions that define how the service mesh should implement specific capabilities. There are also components that manage aspects of security like strong identity and certificates for mTLS. Service meshes will also typically have a metrics or observability component that collects and aggregates metrics and telemetry from the workloads.

The **data plane** typically consists of a proxy that is transparently injected as a sidecar to your workloads. This proxy is configured to control all network traffic in and out of the pod containing your workload. This allows the proxy to be configured to secure traffic via mTLS, dynamically route traffic, apply policies to traffic and to collect metrics and tracing information.



## Capabilities

Each of the service meshes have a natural fit and focus on supporting specific scenarios, but you'll typically find that most will implement a number of, if not all, of the following capabilities.

### Traffic management

- **Protocol** – layer 7 (http, grpc)
- **Dynamic Routing** – conditional, weighting, mirroring
- **Resiliency** – timeouts, retries, circuit breakers
- **Policy** – access control, rate limits, quotas
- **Testing** - fault injection

### Security

- **Encryption** – mTLS, certificate management, external CA
- **Strong Identity** – SPIFFE or similar
- **Auth** – authentication, authorisation

### Observability

- **Metrics** – golden metrics, prometheus, grafana
- **Tracing** - traces across workloads
- **Traffic** – cluster, ingress/egress

### Mesh

- **Supported Compute** - Kubernetes, virtual machines
- **Multi-cluster** - gateways, federation

## Selection criteria

Before you select a service mesh, ensure that you understand your requirements and the reasons for installing a service mesh. Try asking the following questions.

- **Is an Ingress Controller sufficient for my needs?** - Sometimes having a capability like a/b testing or traffic splitting at the ingress is sufficient to support the required scenario. Don't add complexity to your

environment with no upside.

- **Can my workloads and environment tolerate the additional overheads?** - All the additional components required to support the service mesh require additional resources like cpu and memory. In addition, all the proxies and their associated policy checks add latency to your traffic. If you have workloads that are very sensitive to latency or cannot provide the additional resources to cover the service mesh components, then re-consider.
- **Is this adding additional complexity unnecessarily?** - If the reason for installing a service mesh is to gain a capability that is not necessarily critical to the business or operational teams, then consider whether the additional complexity of installation, maintenance, and configuration is worth it.
- **Can this be adopted in an incremental approach?** - Some of the service meshes that provide a lot of capabilities can be adopted in a more incremental approach. Install just the components you need to ensure your success. Once you are more confident and additional capabilities are required, then explore those. Resist the urge to install *everything* from the start.

If, after careful consideration, you decide that you need a service mesh to provide the capabilities required, then your next decision is *which service mesh?*

Consider the following areas and which of them are most aligned with your requirements. This will guide you towards the best fit for your environment and workloads. The [Next steps](#) section will take you to further detailed information about specific service meshes and how they map to these areas.

- **Technical** - traffic management, policy, security, observability
- **Business** - commercial support, foundation (CNCF), OSS license, governance
- **Operational** – installation/upgrades, resource requirements, performance requirements, integrations (metrics, telemetry, dashboards, tools, SMI), mixed workloads (Linux and Windows node pools), compute (Kubernetes, virtual machines), multi-cluster
- **Security** - auth, identity, certificate management and rotation, pluggable external CA

## Next steps

The following documentation provides more information about service meshes that you can try out on Azure Kubernetes Service (AKS):

[Learn more about Istio ...](#)

[Learn more about Linkerd ...](#)

[Learn more about Consul ...](#)

You may also want to explore Service Mesh Interface (SMI), a standard interface for service meshes on Kubernetes:

- [Service Mesh Interface \(SMI\)](#)

# Open Service Mesh AKS add-on (Preview)

4/28/2021 • 61 minutes to read • [Edit Online](#)

## Overview

**Open Service Mesh (OSM)** is a lightweight, extensible, Cloud Native service mesh that allows users to uniformly manage, secure, and get out-of-the-box observability features for highly dynamic microservice environments.

OSM runs an Envoy-based control plane on Kubernetes, can be configured with [SMI APIs](#), and works by injecting an Envoy proxy as a sidecar container next to each instance of your application. The Envoy proxy contains and executes rules around access control policies, implements routing configuration, and captures metrics. The control plane continually configures proxies to ensure policies and routing rules are up to date and ensures proxies are healthy.

### IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

## Capabilities and Features

OSM provides the following set of capabilities and features to provide a cloud native service mesh for your Azure Kubernetes Service (AKS) clusters:

- Secure service to service communication by enabling mTLS
- Easily onboard applications onto the mesh by enabling automatic sidecar injection of Envoy proxy
- Easily and transparent configurations for traffic shifting on deployments
- Ability to define and execute fine grained access control policies for services
- Observability and insights into application metrics for debugging and monitoring services
- Integration with external certificate management services/solutions with a pluggable interface

## Scenarios

OSM can assist your AKS deployments with the following scenarios:

- Provide encrypted communications between service endpoints deployed in the cluster
- Traffic authorization of both HTTP/HTTPS and TCP traffic in the mesh
- Configuration of weighted traffic controls between two or more services for A/B or canary deployments
- Collection and viewing of KPIs from application traffic

## OSM Service Quotas and Limits (Preview)

OSM preview limitations for service quotas and limits can be found on the AKS [Quotas and regional limits page](#).

## Download and install the OSM client binary

In a bash-based shell on Linux or [Windows Subsystem for Linux](#), use `curl` to download the OSM release and then extract with `tar` as follows:

```
Specify the OSM version that will be leveraged throughout these instructions
OSM_VERSION=v0.8.2

curl -sL "https://github.com/openservicemesh/osm/releases/download/$OSM_VERSION/osm-$OSM_VERSION-linux-amd64.tar.gz" | tar -vxzf -
```

The `osm` client binary runs on your client machine and allows you to manage OSM in your AKS cluster. Use the following commands to install the OSM `osm` client binary in a bash-based shell on Linux or [Windows Subsystem for Linux](#). These commands copy the `osm` client binary to the standard user program location in your `PATH`.

```
sudo mv ./linux-amd64/osm /usr/local/bin/osm
sudo chmod +x /usr/local/bin/osm
```

You can verify the `osm` client library has been correctly added to your path and its version number with the following command.

```
osm version
```

## Download and install the OSM client binary

In a bash-based shell, use `curl` to download the OSM release and then extract with `tar` as follows:

```
Specify the OSM version that will be leveraged throughout these instructions
OSM_VERSION=v0.8.2

curl -sL "https://github.com/openservicemesh/osm/releases/download/$OSM_VERSION/osm-$OSM_VERSION-darwin-amd64.tar.gz" | tar -vxzf -
```

The `osm` client binary runs on your client machine and allows you to manage OSM in your AKS cluster. Use the following commands to install the OSM `osm` client binary in a bash-based shell. These commands copy the `osm` client binary to the standard user program location in your `PATH`.

```
sudo mv ./darwin-amd64/osm /usr/local/bin/osm
sudo chmod +x /usr/local/bin/osm
```

You can verify the `osm` client library has been correctly added to your path and its version number with the following command.

```
osm version
```

## Download and install the OSM client binary

In a PowerShell-based shell on Windows, use `Invoke-WebRequest` to download the Istio release and then extract

with `Expand-Archive` as follows:

```
Specify the OSM version that will be leveraged throughout these instructions
$OSM_VERSION="v0.8.2"

[Net.ServicePointManager]::SecurityProtocol = "tls12"
$ProgressPreference = 'SilentlyContinue'; Invoke-WebRequest -URI
"https://github.com/openservicemesh/osm/releases/download/$OSM_VERSION/osm-$OSM_VERSION-windows-amd64.zip" -
OutFile "osm-$OSM_VERSION.zip"
Expand-Archive -Path "osm-$OSM_VERSION.zip" -DestinationPath .
```

The `osm` client binary runs on your client machine and allows you to manage the OSM controller in your AKS cluster. Use the following commands to install the OSM `osm` client binary in a PowerShell-based shell on Windows. These commands copy the `osm` client binary to an OSM folder and then make it available both immediately (in current shell) and permanently (across shell restarts) via your `PATH`. You don't need elevated (Admin) privileges to run these commands and you don't need to restart your shell.

```
Copy osm.exe to C:\OSM
New-Item -ItemType Directory -Force -Path "C:\OSM"
Move-Item -Path .\windows-amd64\osm.exe -Destination "C:\OSM\"

Add C:\OSM to PATH.
Make the new PATH permanently available for the current User
$USER_PATH = [environment]::GetEnvironmentVariable("PATH", "User") + ";C:\OSM\
[environment]::SetEnvironmentVariable("PATH", $USER_PATH, "User")
Make the new PATH immediately available in the current shell
$env:PATH += ";C:\OSM\"
```

### WARNING

Do not attempt to install OSM from the binary using `osm install`. This will result in a installation of OSM that is not integrated as an add-on for AKS.

## Register the `AKS-OpenServiceMesh` preview feature

To create an AKS cluster that can use the Open Service Mesh add-on, you must enable the `AKS-OpenServiceMesh` feature flag on your subscription.

Register the `AKS-OpenServiceMesh` feature flag by using the `az feature register` command, as shown in the following example:

```
az feature register --namespace "Microsoft.ContainerService" --name "AKS-OpenServiceMesh"
```

It takes a few minutes for the status to show *Registered*. Verify the registration status by using the `az feature list` command:

```
az feature list -o table --query "[?contains(name, 'Microsoft.ContainerService/AKS-OpenServiceMesh')].{Name:name,State:properties.state}"
```

When ready, refresh the registration of the `Microsoft.ContainerService` resource provider by using the `az provider register` command:

```
az provider register --namespace Microsoft.ContainerService
```

## IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

## Install Open Service Mesh (OSM) Azure Kubernetes Service (AKS) add-on for a new AKS cluster

For a new AKS cluster deployment scenario, you will start with a brand new deployment of an AKS cluster enabling the OSM add-on at the cluster create operation.

### Create a resource group

In Azure, you allocate related resources to a resource group. Create a resource group by using [az group create](#). The following example creates a resource group named *myOsmAksGroup* in the *eastus2* location (region):

```
az group create --name <myosmaksgroup> --location <eastus2>
```

### Deploy an AKS cluster with the OSM add-on enabled

You'll now deploy a new AKS cluster with the OSM add-on enabled.

#### NOTE

Please be aware the following AKS deployment command utilizes OS ephemeral disks. You can find more information here about [Ephemeral OS disks for AKS](#)

```
az aks create -n osm-addon-cluster -g <myosmaksgroup> --kubernetes-version 1.19.6 --node-osdisk-type Ephemeral --node-osdisk-size 30 --network-plugin azure --enable-managed-identity -a open-service-mesh
```

### Get AKS Cluster Access Credentials

Get access credentials for the new managed Kubernetes cluster.

```
az aks get-credentials -n <myosmakscluster> -g <myosmaksgroup>
```

## Enable Open Service Mesh (OSM) Azure Kubernetes Service (AKS) add-on for an existing AKS cluster

For an existing AKS cluster scenario, you will enable the OSM add-on to an existing AKS cluster that has already been deployed.

### Enable the OSM add-on to existing AKS cluster

To enable the AKS OSM add-on, you will need to run the `az aks enable-addons --addons` command passing the parameter `open-service-mesh`

```
az aks enable-addons --addons open-service-mesh -g <resource group name> -n <AKS cluster name>
```

You should see output similar to the output shown below to confirm the AKS OSM add-on has been installed.

```
{- Finished ..
 "aadProfile": null,
 "addonProfiles": {
 "KubeDashboard": {
 "config": null,
 "enabled": false,
 "identity": null
 },
 "openServiceMesh": {
 "config": {},
 "enabled": true,
 "identity": {
 ...
 }
 }
 }
}
```

## Validate the AKS OSM add-on installation

There are several commands to run to check all of the components of the AKS OSM add-on are enabled and running:

First we can query the add-on profiles of the cluster to check the enabled state of the add-ons installed. The following command should return "true".

```
az aks list -g <resource group name> -o json | jq -r '.[].addonProfiles.openServiceMesh.enabled'
```

The following `kubectl` commands will report the status of the osm-controller.

```
kubectl get deployments -n kube-system --selector app=osm-controller
kubectl get pods -n kube-system --selector app=osm-controller
kubectl get services -n kube-system --selector app=osm-controller
```

## Accessing the AKS OSM add-on

Currently you can access and configure the OSM controller configuration via the configmap. To view the OSM controller configuration settings, query the osm-config configmap via `kubectl` to view its configuration settings.

```
kubectl get configmap -n kube-system osm-config -o json | jq '.data'
```

Output of the OSM configmap should look like the following:

```
{
 "egress": "true",
 "enable_debug_server": "true",
 "enable_privileged_init_container": "false",
 "envoy_log_level": "error",
 "outbound_ip_range_exclusion_list": "169.254.169.254/32,168.63.129.16/32,<YOUR_API_SERVER_PUBLIC_IP>/32",
 "permissive_traffic_policy_mode": "true",
 "prometheus_scraping": "false",
 "service_cert_validity_duration": "24h",
 "use_https_ingress": "false"
}
```

Notice the `permissive_traffic_policy_mode` is configured to `true`. Permissive traffic policy mode in OSM is a

mode where the [SMI](#) traffic policy enforcement is bypassed. In this mode, OSM automatically discovers services that are a part of the service mesh and programs traffic policy rules on each Envoy proxy sidecar to be able to communicate with these services.

#### WARNING

Before proceeding please verify that your permissive traffic policy mode is set to true, if not please change it to `true` using the command below

```
kubectl patch ConfigMap -n kube-system osm-config --type merge --patch '{"data": {"permissive_traffic_policy_mode":"true"}}'
```

## Deploy a new application to be managed by the Open Service Mesh (OSM) Azure Kubernetes Service (AKS) add-on

#### Before you begin

The steps detailed in this walkthrough assume that you've created an AKS cluster (Kubernetes [1.19+](#) and above, with Kubernetes RBAC enabled), have established a `kubectl` connection with the cluster (If you need help with any of these items, then see the [AKS quickstart](#), and have installed the AKS OSM add-on.

You must have the following resources installed:

- The Azure CLI, version 2.20.0 or later
- The `aks-preview` extension version 0.5.5 or later
- OSM version v0.8.0 or later
- `apt-get install jq`

#### Create namespaces for the application

In this walkthrough, we will be using the OSM bookstore application that has the following Kubernetes services:

- bookbuyer
- bookthief
- bookstore
- bookwarehouse

Create namespaces for each of these application components.

```
for i in bookstore bookbuyer bookthief bookwarehouse; do kubectl create ns $i; done
```

You should see the following output:

```
namespace/bookstore created
namespace/bookbuyer created
namespace/bookthief created
namespace/bookwarehouse created
```

#### Onboard the namespaces to be managed by OSM

When you add the namespaces to the OSM mesh, this will allow the OSM controller to automatically inject the Envoy sidecar proxy containers with your application. Run the following command to onboard the OSM bookstore application namespaces.

```
osm namespace add bookstore bookbuyer bookthief bookwarehouse
```

You should see the following output:

```
Namespace [bookstore] successfully added to mesh [osm]
Namespace [bookbuyer] successfully added to mesh [osm]
Namespace [bookthief] successfully added to mesh [osm]
Namespace [bookwarehouse] successfully added to mesh [osm]
```

## Deploy the Bookstore application to the AKS cluster

```
kubectl apply -f https://raw.githubusercontent.com/openservicemesh/osm/release-v0.8/docs/example/manifests/apps/bookbuyer.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/openservicemesh/osm/release-v0.8/docs/example/manifests/apps/bookthief.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/openservicemesh/osm/release-v0.8/docs/example/manifests/apps/bookstore.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/openservicemesh/osm/release-v0.8/docs/example/manifests/apps/bookwarehouse.yaml
```

All of the deployment outputs are summarized below.

```
serviceaccount/bookbuyer created
service/bookbuyer created
deployment.apps/bookbuyer created

serviceaccount/bookthief created
service/bookthief created
deployment.apps/bookthief created

service/bookstore created
serviceaccount/bookstore created
deployment.apps/bookstore created

serviceaccount/bookwarehouse created
service/bookwarehouse created
deployment.apps/bookwarehouse created
```

### Checkpoint: What got installed?

The example Bookstore application is a multi-tiered app that consists of four services, being the bookbuyer, bookthief, bookstore, and bookwarehouse. Both the bookbuyer and bookthief service communicate to the bookstore service to retrieve books from the bookstore service. The bookstore service retrieves books out of the bookwarehouse service to supply the bookbuyer and bookthief. This is a simple multi-tiered application that works well in showing how a service mesh can be used to protect and authorize communications between the applications services. As we continue through the walkthrough, we will be enabling and disabling Service Mesh Interface (SMI) policies to both allow and disallow the services to communicate via OSM. Below is an architecture diagram of what got installed for the bookstore application.



### Verify the Bookstore application running inside the AKS cluster

As of now we have deployed the bookstore multi-container application, but it is only accessible from within the AKS cluster. Later tutorials will assist you in exposing the application outside the cluster via an ingress controller. For now we will be utilizing port forwarding to access the bookbuyer application inside the AKS cluster to verify it is buying books from the bookstore service.

To verify that the application is running inside the cluster, we will use a port forward to view both the bookbuyer and bookthief components UI.

First let's get the bookbuyer pod's name

```
kubectl get pod -n bookbuyer
```

You should see output similar to the following. Your bookbuyer pod will have a unique name appended.

NAME	READY	STATUS	RESTARTS	AGE
bookbuyer-7676c7fcfb-mtnrz	2/2	Running	0	7m8s

Once we have the pod's name, we can now use the port-forward command to set up a tunnel from our local system to the application inside the AKS cluster. Run the following command to set up the port forward for the local system port 8080. Again use your specified bookbuyer pod name.

#### NOTE

For all port forwarding commands it is best to use an additional terminal so that you can continue to work through this walkthrough and not disconnect the tunnel. It is also best that you establish the port forward tunnel outside of the Azure Cloud Shell.

```
kubectl port-forward bookbuyer-7676c7fcfb-mtnrz -n bookbuyer 8080:14001
```

You should see output similar to this.

```
Forwarding from 127.0.0.1:8080 -> 14001
Forwarding from [::1]:8080 -> 14001
```

While the port forwarding session is in place, navigate to the following url from a browser

<http://localhost:8080>. You should now be able to see the bookbuyer application UI in the browser similar to the image below.

- Total books bought: 499
  - from bookstore V1: 499
  - from bookstore V2: 0

You will also notice that the total books bought number continues to increment to the bookstore v1 service. The bookstore v2 service has not been deployed yet. We will deploy the bookstore v2 service when we demonstrate the SMI traffic split policies.

You can also check the same for the bookthief service.

```
kubectl get pod -n bookthief
```

You should see output similar to the following. Your bookthief pod will have a unique name appended.

NAME	READY	STATUS	RESTARTS	AGE
bookthief-59549fb69c-cr8v1	2/2	Running	0	15m54s

Port forward to bookthief pod.

```
kubectl port-forward bookthief-59549fb69c-cr8v1 -n bookthief 8080:14001
```

Navigate to the following url from a browser <http://localhost:8080>. You should see the bookthief is currently stealing books from the bookstore service! Later on we will implement a traffic policy to stop the bookthief.

- Total books stolen: **14201**
  - from bookstore V1: **14201**
  - from bookstore V2: **0**

#### Disable OSM Permissive Traffic Mode for the mesh

As mentioned earlier when viewing the OSM cluster configuration, the OSM configuration defaults to enabling permissive traffic mode policy. In this mode traffic policy enforcement is bypassed and OSM automatically discovers services that are a part of the service mesh and programs traffic policy rules on each Envoy proxy sidecar to be able to communicate with these services.

We will now disable the permissive traffic mode policy and OSM will need explicit [SMI](#) policies deployed to the cluster to allow communications in the mesh from each service. To disable permissive traffic mode, run the following command to update the configmap property changing the value from `true` to `false`.

```
kubectl patch ConfigMap -n kube-system osm-config --type merge --patch '{"data": {"permissive_traffic_policy_mode":"false"}}'
```

You should see output similar to the following. Your bookthief pod will have a unique name appended.

```
configmap/osm-config patched
```

To verify permissive traffic mode has been disabled, port forward back into either the bookbuyer or bookthief pod to view their UI in the browser and see if the books bought or books stolen is no longer incrementing. Ensure to refresh the browser. If the incrementing has stopped, the policy was applied correctly. You have successfully stopped the bookthief from stealing books, but neither the bookbuyer can purchase from the bookstore nor the bookstore can retrieve books from the bookwarehouse. Next we will implement [SMI](#) policies to allow only the services in the mesh you'd like to communicate to do so.

#### Apply Service Mesh Interface (SMI) traffic access policies

Now that we have disabled all communications in the mesh, let's allow our bookbuyer service to communicate to our bookstore service for purchasing books, and allow our bookstore service to communicate to our bookwarehouse service to retrieving books to sell.

Deploy the following [SMI](#) policies.

```
kubectl apply -f - <<EOF

apiVersion: access.smi-spec.io/v1alpha3
kind: TrafficTarget
metadata:
 name: bookbuyer-access-bookstore
 namespace: bookstore
```

```

spec:
 destination:
 kind: ServiceAccount
 name: bookstore
 namespace: bookstore
 rules:
 - kind: HTTPRouteGroup
 name: bookstore-service-routes
 matches:
 - buy-a-book
 - books-bought
 sources:
 - kind: ServiceAccount
 name: bookbuyer
 namespace: bookbuyer

apiVersion: specs.smi-spec.io/v1alpha4
kind: HTTPRouteGroup
metadata:
 name: bookstore-service-routes
 namespace: bookstore
spec:
 matches:
 - name: books-bought
 pathRegex: /books-bought
 methods:
 - GET
 headers:
 - "user-agent": ".*-http-client/*.*"
 - "client-app": "bookbuyer"
 - name: buy-a-book
 pathRegex: ".*a-book.*new"
 methods:
 - GET
 - name: update-books-bought
 pathRegex: /update-books-bought
 methods:
 - POST

kind: TrafficTarget
apiVersion: access.smi-spec.io/v1alpha3
metadata:
 name: bookstore-access-bookwarehouse
 namespace: bookwarehouse
spec:
 destination:
 kind: ServiceAccount
 name: bookwarehouse
 namespace: bookwarehouse
 rules:
 - kind: HTTPRouteGroup
 name: bookwarehouse-service-routes
 matches:
 - restock-books
 sources:
 - kind: ServiceAccount
 name: bookstore
 namespace: bookstore
 - kind: ServiceAccount
 name: bookstore-v2
 namespace: bookstore

apiVersion: specs.smi-spec.io/v1alpha4
kind: HTTPRouteGroup
metadata:
 name: bookwarehouse-service-routes
 namespace: bookwarehouse
spec:
 matches:

```

```
- name: restock-books
 methods:
 - POST
 headers:
 - host: bookwarehouse.bookwarehouse
EOF
```

You should see output similar to the following.

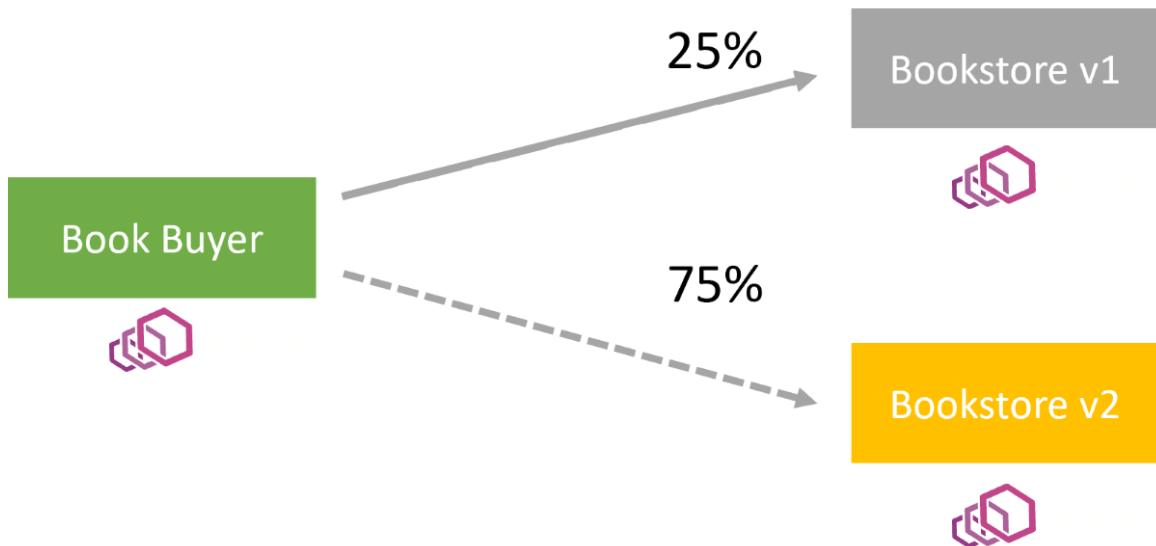
```
traffictarget.access.smi-spec.io/bookbuyer-access-bookstore-v1 created
httproutroutegroup.specs.smi-spec.io/bookstore-service-routes created
traffictarget.access.smi-spec.io/bookstore-access-bookwarehouse created
httproutroutegroup.specs.smi-spec.io/bookwarehouse-service-routes created
```

You can now set up a port forwarding session on either the bookbuyer or bookstore pods and see that both the books bought and books sold metrics are back incrementing. You can also do the same for the bookthief pod to verify it is still no longer able to steal books.

### Apply Service Mesh Interface (SMI) traffic split policies

For our final demonstration, we will create an [SMI](#) traffic split policy to configure the weight of communications from one service to multiple services as a backend. The traffic split functionality allows you to progressively move connections to one service over to another by weighting the traffic on a scale of 0 to 100.

The below graphic is a diagram of the [SMI](#) Traffic Split policy to be deployed. We will deploy an additional Bookstore version 2 and then split the incoming traffic from the bookbuyer, weighting 25% of the traffic to the bookstore v1 service and 75% to the bookstore v2 service.



Deploy the bookstore v2 service.

```
kubectl apply -f - <<EOF

apiVersion: v1
kind: Service
metadata:
 name: bookstore-v2
 namespace: bookstore
 labels:
 app: bookstore-v2
spec:
 ports:
 - port: 14001
```

```

- port: 14001
 name: bookstore-port
 selector:
 app: bookstore-v2

Deploy bookstore-v2 Service Account
apiVersion: v1
kind: ServiceAccount
metadata:
 name: bookstore-v2
 namespace: bookstore

Deploy bookstore-v2 Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
 name: bookstore-v2
 namespace: bookstore
spec:
 replicas: 1
 selector:
 matchLabels:
 app: bookstore-v2
 template:
 metadata:
 labels:
 app: bookstore-v2
 spec:
 serviceAccountName: bookstore-v2
 containers:
 - name: bookstore
 image: openservicemesh/bookstore:v0.8.0
 imagePullPolicy: Always
 ports:
 - containerPort: 14001
 name: web
 command: ["/bookstore"]
 args: ["--path", "./", "--port", "14001"]
 env:
 - name: BOOKWAREHOUSE_NAMESPACE
 value: bookwarehouse
 - name: IDENTITY
 value: bookstore-v2

kind: TrafficTarget
apiVersion: access.smi-spec.io/v1alpha3
metadata:
 name: bookbuyer-access-bookstore-v2
 namespace: bookstore
spec:
 destination:
 kind: ServiceAccount
 name: bookstore-v2
 namespace: bookstore
 rules:
 - kind: HTTPRouteGroup
 name: bookstore-service-routes
 matches:
 - buy-a-book
 - books-bought
 sources:
 - kind: ServiceAccount
 name: bookbuyer
 namespace: bookbuyer
EOF

```

You should see the following output.

```
service/bookstore-v2 configured
serviceaccount/bookstore-v2 created
deployment.apps/bookstore-v2 created
traffictarget.access.smi-spec.io/bookstore-v2 created
```

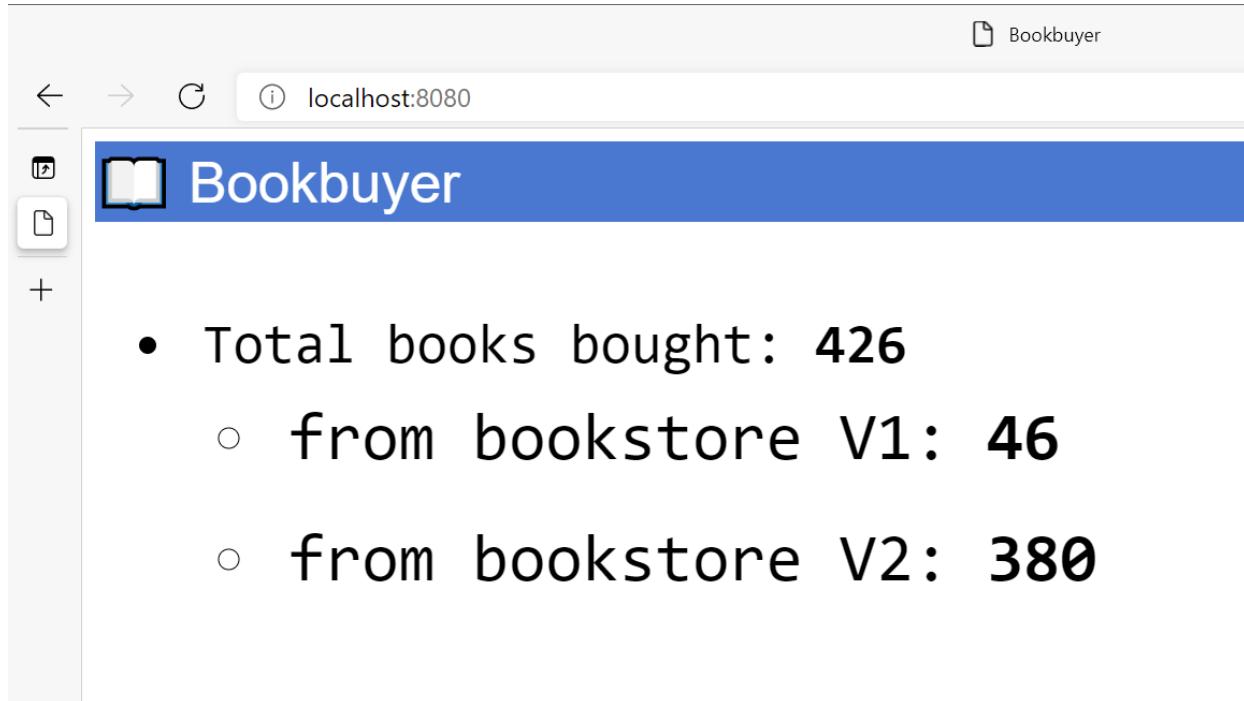
Now deploy the traffic split policy to split the bookbuyer traffic between the two bookstore v1 and v2 service.

```
kubectl apply -f - <<EOF
apiVersion: split.smi-spec.io/v1alpha2
kind: TrafficSplit
metadata:
 name: bookstore-split
 namespace: bookstore
spec:
 service: bookstore.bookstore
 backends:
 - service: bookstore
 weight: 25
 - service: bookstore-v2
 weight: 75
EOF
```

You should see the following output.

```
trafficsplit.split.smi-spec.io/bookstore-split created
```

Set up a port forward tunnel to the bookbuyer pod and you should now see books being purchased from the bookstore v2 service. If you continue to watch the increment of purchases you should notice a faster increment of purchases happening through the bookstore v2 service.



Manage existing deployed applications to be managed by the Open Service Mesh (OSM) Azure Kubernetes Service (AKS) add-on

#### Before you begin

The steps detailed in this walkthrough assume that you have previously enabled the OSM AKS add-on for your

AKS cluster. If not, review the section [Enable Open Service Mesh \(OSM\) Azure Kubernetes Service \(AKS\) add-on for an existing AKS cluster](#) before proceeding. Also, your AKS cluster needs to be version Kubernetes [1.19+](#) and above, have Kubernetes RBAC enabled, and have established a `kubectl` connection with the cluster (If you need help with any of these items, then see the [AKS quickstart](#), and have installed the AKS OSM add-on).

You must have the following resources installed:

- The Azure CLI, version 2.20.0 or later
- The `aks-preview` extension version 0.5.5 or later
- OSM version v0.8.0 or later
- `apt-get install jq`

### Verify the Open Service Mesh (OSM) Permissive Traffic Mode Policy

The OSM Permissive Traffic Policy mode is a mode where the [SMI](#) traffic policy enforcement is bypassed. In this mode, OSM automatically discovers services that are a part of the service mesh and programs traffic policy rules on each Envoy proxy sidecar to be able to communicate with these services.

To verify the current permissive traffic mode of OSM for your cluster, run the following command:

```
kubectl get configmap -n kube-system osm-config -o json | jq '.data'
```

Output of the OSM configmap should look like the following:

```
{
 "egress": "true",
 "enable_debug_server": "true",
 "envoy_log_level": "error",
 "permissive_traffic_policy_mode": "true",
 "prometheus_scraping": "false",
 "service_cert_validity_duration": "24h",
 "use_https_ingress": "false"
}
```

If the `permissive_traffic_policy_mode` is configured to `true`, you can safely onboard your namespaces without any disruption to your service-to-service communications. If the `permissive_traffic_policy_mode` is configured to `false`, You will need to ensure you have the correct [SMI](#) traffic access policy manifests deployed as well as ensuring you have a service account representing each service deployed in the namespace. Please follow the guidance for [Onboard existing deployed applications with Open Service Mesh \(OSM\) Permissive Traffic Policy configured as False](#)

### Onboard existing deployed applications with Open Service Mesh (OSM) Permissive Traffic Policy configured as True

The first thing we'll do is add the deployed application namespace(s) to OSM to manage.

```
osm namespace add bookstore
```

You should see the following output:

```
Namespace [bookstore] successfully added to mesh [osm]
```

Next we will take a look at the current pod deployment in the namespace. Run the following command to view the pods in the designated namespace.

```
kubectl get pod -n bookbuyer
```

You should see the following similar output:

NAME	READY	STATUS	RESTARTS	AGE
bookbuyer-78666dcff8-wh6wl	1/1	Running	0	43s

Notice the **READY** column showing **1/1**, meaning that the application pod has only one container. Next we will need to restart your application deployments so that OSM can inject the Envoy sidecar proxy container with your application pod. Let's get a list of deployments in the namespace.

```
kubectl get deployment -n bookbuyer
```

You should see the following output:

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
bookbuyer	1/1	1	1	23h

Now we will restart the deployment to inject the Envoy sidecar proxy container with your application pod. Run the following command.

```
kubectl rollout restart deployment bookbuyer -n bookbuyer
```

You should see the following output:

```
deployment.apps/bookbuyer restarted
```

If we take a look at the pods in the namespace again:

```
kubectl get pod -n bookbuyer
```

You will now notice that the **READY** column is now showing **2/2** containers being ready for your pod. The second container is the Envoy sidecar proxy.

NAME	READY	STATUS	RESTARTS	AGE
bookbuyer-84446dd5bd-j4t1r	2/2	Running	0	3m30s

We can further inspect the pod to view the Envoy proxy by running the `describe` command to view the configuration.

```
kubectl describe pod bookbuyer-84446dd5bd-j4t1r -n bookbuyer
```

```

Containers:
bookbuyer:
 Container ID: containerd://b7503b866f915711002292ea53970bd994e788e33fb718f1c4f8f12cd4a88198
 Image: openservicemesh/bookbuyer:v0.8.0
 Image ID:
docker.io/openservicemesh/bookbuyer@sha256:813874bd2dc9c5a259b9657995348cf0822b905e29c4e86f21fdefa0ef21dce
 Port: <none>
 Host Port: <none>
 Command:
 /bookbuyer
 State: Running
 Started: Tue, 23 Mar 2021 10:52:53 -0400
 Ready: True
 Restart Count: 0
 Environment:
 BOOKSTORE_NAMESPACE: bookstore
 BOOKSTORE_SVC: bookstore
 Mounts:
 /var/run/secrets/kubernetes.io/serviceaccount from bookbuyer-token-zft2r (ro)
envoy:
 Container ID: containerd://f5f1cb5db8d5304e23cc984eb08146ea162a3e82d4262c4472c28d5579c25e10
 Image: envoyproxy/envoy-alpine:v1.17.1
 Image ID:
alpine@sha256:511e76b9b73fccd98af2fbfb75c34833343d1999469229fdfb191abd2bbe3dfb
 Ports: 15000/TCP, 15003/TCP, 15010/TCP
 Host Ports: 0/TCP, 0/TCP, 0/TCP

```

Verify your application is still functional after the Envoy sidecar proxy injection.

### Onboard existing deployed applications with Open Service Mesh (OSM) Permissive Traffic Policy configured as False

When the OSM configuration for the permissive traffic policy is set to `false`, OSM will require explicit [SMI](#) traffic access policies deployed for the service-to-service communication to happen within your cluster.

Currently, OSM also uses Kubernetes service accounts as part of authorizing service-to-service communications as well. To ensure your existing deployed applications will communicate when managed by the OSM mesh, we will need to verify the existence of a service account to utilize, update the application deployment with the service account information, apply the [SMI](#) traffic access policies.

#### Verify Kubernetes Service Accounts

Verify if you have a kubernetes service account in the namespace your application is deployed to.

```
kubectl get serviceaccounts -n bookbuyer
```

In the following there is a service account named `bookbuyer` in the bookbuyer namespace.

NAME	SECRETS	AGE
bookbuyer	1	25h
default	1	25h

If you do not have a service account listed other than the default account, you will need to create one for your application. Use the following command as an example to create a service account in the application's deployed namespace.

```
kubectl create serviceaccount myserviceaccount -n bookbuyer
```

```
serviceaccount/myserviceaccount created
```

### **View your application's current deployment specification**

If you had to create a service account from the earlier section, chances are your application deployment is not configured with a specific `serviceAccountName` in the deployment spec. We can view your application's deployment spec with the following commands:

```
kubectl get deployment -n bookbuyer
```

A list of deployments will be listed in the output.

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
bookbuyer	1/1	1	1	25h

We will now describe the deployment as a check to see if there is a service account listed in the Pod Template section.

```
kubectl describe deployment bookbuyer -n bookbuyer
```

In this particular deployment you can see that there is a service account associated with the deployment listed under the Pod Template section. This deployment is using the service account bookbuyer. If you do not see the **Service Account:** property, your deployment is not configured to use a service account.

```
Pod Template:
 Labels: app=bookbuyer
 version=v1
 Annotations: kubectl.kubernetes.io/restartedAt: 2021-03-23T10:52:49-04:00
 Service Account: bookbuyer
 Containers:
 bookbuyer:
 Image: openservicemesh/bookbuyer:v0.8.0
```

There are several techniques to update your deployment to add a kubernetes service account. Review the Kubernetes documentation on [Updating a Deployment](#) inline, or [Configure Service Accounts for Pods](#). Once you have updated your deployment spec with the service account, redeploy (kubectl apply -f your-deployment.yaml) your deployment to the cluster.

### **Deploy the necessary Service Mesh Interface (SMI) Policies**

The last step to allowing authorized traffic to flow in the mesh is to deploy the necessary **SMI** traffic access policies for your application. The amount of configuration you can achieve with **SMI** traffic access policies is beyond the scope of this walkthrough, but we will detail some of the common components of the specification and show how to configure both a simple **TrafficTarget** and **HTTPRouteGroup** policy to enable service-to-service communication for your application.

The [SMI Traffic Access Control](#) specification allows users to define the access control policy for their applications. We will focus on the **TrafficTarget** and **HTTPRouteGroup** api resources.

The **TrafficTarget** resource consists of three main configuration settings destination, rules, and sources. An example **TrafficTarget** is shown below.

```

apiVersion: access.smi-spec.io/v1alpha3
kind: TrafficTarget
metadata:
 name: bookbuyer-access-bookstore-v1
 namespace: bookstore
spec:
 destination:
 kind: ServiceAccount
 name: bookstore
 namespace: bookstore
 rules:
 - kind: HTTPRouteGroup
 name: bookstore-service-routes
 matches:
 - buy-a-book
 - books-bought
 sources:
 - kind: ServiceAccount
 name: bookbuyer
 namespace: bookbuyer

```

In the above `TrafficTarget` spec, the `destination` denotes the service account that is configured for the destination source service. Remember the service account that was added to the deployment earlier will be used to authorize access to the deployment it is attached to. The `rules` section , in this particular example, defines the type of HTTP traffic that is allowed over the connection. You can configure fine grain regex patterns for the HTTP headers to be specific on what traffic is allowed via HTTP. The `sources` section is the service originating communications. This spec reads bookbuyer needs to communicate to the bookstore.

The `HTTPRouteGroup` resource consists of one or an array of matches of HTTP header information and is a requirement for the `TrafficTarget` spec. In the example below, you can see that the `HTTPRouteGroup` is authorizing three HTTP actions, two GET and one POST.

```

apiVersion: specs.smi-spec.io/v1alpha4
kind: HTTPRouteGroup
metadata:
 name: bookstore-service-routes
 namespace: bookstore
spec:
 matches:
 - name: books-bought
 pathRegex: /books-bought
 methods:
 - GET
 headers:
 - "user-agent": ".*-http-client/*.*"
 - "client-app": "bookbuyer"
 - name: buy-a-book
 pathRegex: ".*a-book.*new"
 methods:
 - GET
 - name: update-books-bought
 pathRegex: /update-books-bought
 methods:
 - POST

```

If you are not familiar with the type of HTTP traffic your front-end application makes to other tiers of the application, since the `TrafficTarget` spec requires a rule, you can create the equivalent of an allow all rule using the below spec for `HTTPRouteGroup`.

```

apiVersion: specs.smi-spec.io/v1alpha4
kind: HTTPRouteGroup
metadata:
 name: allow-all
 namespace: yournamespace
spec:
 matches:
 - name: allow-all
 pathRegex: '.*'
 methods: ["GET", "PUT", "POST", "DELETE", "PATCH"]

```

Once you configure your TrafficTarget and HTTPRouteGroup spec, you can put them together as one YAML and deploy. Below is the bookstore example configuration.

```

kubectl apply -f - <<EOF

apiVersion: access.smi-spec.io/v1alpha3
kind: TrafficTarget
metadata:
 name: bookbuyer-access-bookstore-v1
 namespace: bookstore
spec:
 destination:
 kind: ServiceAccount
 name: bookstore
 namespace: bookstore
 rules:
 - kind: HTTPRouteGroup
 name: bookstore-service-routes
 matches:
 - buy-a-book
 - books-bought
 sources:
 - kind: ServiceAccount
 name: bookbuyer
 namespace: bookbuyer

apiVersion: specs.smi-spec.io/v1alpha4
kind: HTTPRouteGroup
metadata:
 name: bookstore-service-routes
 namespace: bookstore
spec:
 matches:
 - name: books-bought
 pathRegex: /books-bought
 methods:
 - GET
 headers:
 - "user-agent": ".*-http-client/*.*"
 - "client-app": "bookbuyer"
 - name: buy-a-book
 pathRegex: .*a-book.*new"
 methods:
 - GET
 - name: update-books-bought
 pathRegex: /update-books-bought
 methods:
 - POST
EOF

```

Visit the [SMI](#) site for more detailed information on the specification.

**Manage the application's namespace with OSM**

Next we will configure OSM to manage the namespace and restart the deployments to get the Envoy sidecar proxy injected with the application.

Run the following command to configure the `azure-vote` namespace to be managed by OSM.

```
osm namespace add azure-vote
```

```
Namespace [azure-vote] successfully added to mesh [osm]
```

Next restart both the `azure-vote-front` and `azure-vote-back` deployments with the following commands.

```
kubectl rollout restart deployment azure-vote-front -n azure-vote
kubectl rollout restart deployment azure-vote-back -n azure-vote
```

```
deployment.apps/azure-vote-front restarted
deployment.apps/azure-vote-back restarted
```

If we view the pods for the `azure-vote` namespace, we will see the **READY** stage of both the `azure-vote-front` and `azure-vote-back` as 2/2, meaning the Envoy sidecar proxy has been injected alongside the application.

## Tutorial: Deploy an application managed by Open Service Mesh (OSM) with NGINX ingress

Open Service Mesh (OSM) is a lightweight, extensible, Cloud Native service mesh that allows users to uniformly manage, secure, and get out-of-the-box observability features for highly dynamic microservice environments.

In this tutorial, you will:

- View the current OSM cluster configuration
- Create the namespace(s) for OSM to manage deployed applications in the namespace(s)
- Onboard the namespaces to be managed by OSM
- Deploy the sample application
- Verify the application running inside the AKS cluster
- Create a NGINX ingress controller used for the application
- Expose a service via the Azure Application Gateway ingress to the internet

### Before you begin

The steps detailed in this article assume that you've created an AKS cluster (Kubernetes `1.19+` and above, with Kubernetes RBAC enabled), have established a `kubectl` connection with the cluster (If you need help with any of these items, then see the [AKS quickstart](#), and have installed the AKS OSM add-on).

You must have the following resources installed:

- The Azure CLI, version 2.20.0 or later
- The `aks-preview` extension version 0.5.5 or later
- OSM version v0.8.0 or later
- `apt-get install jq`

### View and verify the current OSM cluster configuration

Once the OSM add-on for AKS has been enabled on the AKS cluster, you can view the current configuration parameters in the `osm-config` Kubernetes ConfigMap. Run the following command to view the ConfigMap

properties:

```
kubectl get configmap -n kube-system osm-config -o json | jq '.data'
```

Output shows the current OSM configuration for the cluster.

```
{
 "egress": "true",
 "enable_debug_server": "true",
 "enable_privileged_init_container": "false",
 "envoy_log_level": "error",
 "outbound_ip_range_exclusion_list": "169.254.169.254,168.63.129.16,20.193.57.43",
 "permissive_traffic_policy_mode": "false",
 "prometheus_scraping": "false",
 "service_cert_validity_duration": "24h",
 "use_https_ingress": "false"
}
```

Notice the **permissive\_traffic\_policy\_mode** is configured to **true**. Permissive traffic policy mode in OSM is a mode where the [SMI](#) traffic policy enforcement is bypassed. In this mode, OSM automatically discovers services that are a part of the service mesh and programs traffic policy rules on each Envoy proxy sidecar to be able to communicate with these services.

### Create namespaces for the application

In this tutorial we will be using the OSM bookstore application that has the following application components:

- bookbuyer
- bookthief
- bookstore
- bookwarehouse

Create namespaces for each of these application components.

```
for i in bookstore bookbuyer bookthief bookwarehouse; do kubectl create ns $i; done
```

You should see the following output:

```
namespace/bookstore created
namespace/bookbuyer created
namespace/bookthief created
namespace/bookwarehouse created
```

### Onboard the namespaces to be managed by OSM

Adding the namespaces to the OSM mesh will allow the OSM controller to automatically inject the Envoy sidecar proxy containers with your application. Run the following command to onboard the OSM bookstore application namespaces.

```
osm namespace add bookstore bookbuyer bookthief bookwarehouse
```

You should see the following output:

```
Namespace [bookstore] successfully added to mesh [osm]
Namespace [bookbuyer] successfully added to mesh [osm]
Namespace [bookthief] successfully added to mesh [osm]
Namespace [bookwarehouse] successfully added to mesh [osm]
```

## Deploy the Bookstore application to the AKS cluster

```
kubectl apply -f https://raw.githubusercontent.com/openservicemesh/osm/release-v0.8/docs/example/manifests/apps/bookbuyer.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/openservicemesh/osm/release-v0.8/docs/example/manifests/apps/bookthief.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/openservicemesh/osm/release-v0.8/docs/example/manifests/apps/bookstore.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/openservicemesh/osm/release-v0.8/docs/example/manifests/apps/bookwarehouse.yaml
```

All of the deployment outputs are summarized below.

```
serviceaccount/bookbuyer created
service/bookbuyer created
deployment.apps/bookbuyer created

serviceaccount/bookthief created
service/bookthief created
deployment.apps/bookthief created

service/bookstore created
serviceaccount/bookstore created
deployment.apps/bookstore created

serviceaccount/bookwarehouse created
service/bookwarehouse created
deployment.apps/bookwarehouse created
```

## Update the Bookbuyer Service

Update the bookbuyer service to the correct inbound port configuration with the following service manifest.

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
 name: bookbuyer
 namespace: bookbuyer
 labels:
 app: bookbuyer
spec:
 ports:
 - port: 14001
 name: inbound-port
 selector:
 app: bookbuyer
EOF
```

## Verify the Bookstore application running inside the AKS cluster

As of now we have deployed the bookstore mult-container application, but it is only accessible from within the AKS cluster. Later we will add the Azure Application Gateway ingress controller to expose the application outside the AKS cluster. To verify that the application is running inside the cluster, we will use a port forward to view the bookbuyer component UI.

First let's get the bookbuyer pod's name

```
kubectl get pod -n bookbuyer
```

You should see output similar to the following. Your bookbuyer pod will have a unique name appended.

NAME	READY	STATUS	RESTARTS	AGE
bookbuyer-7676c7fcfb-mtnrz	2/2	Running	0	7m8s

Once we have the pod's name, we can now use the port-forward command to set up a tunnel from our local system to the application inside the AKS cluster. Run the following command to set up the port forward for the local system port 8080. Again use your specified bookbuyer pod name.

```
kubectl port-forward bookbuyer-7676c7fcfb-mtnrz -n bookbuyer 8080:14001
```

You should see output similar to this.

```
Forwarding from 127.0.0.1:8080 -> 14001
Forwarding from [::1]:8080 -> 14001
```

While the port forwarding session is in place, navigate to the following url from a browser

<http://localhost:8080>. You should now be able to see the bookbuyer application UI in the browser similar to the image below.



## Create an NGINX ingress controller in Azure Kubernetes Service (AKS)

An ingress controller is a piece of software that provides reverse proxy, configurable traffic routing, and TLS termination for Kubernetes services. Kubernetes ingress resources are used to configure the ingress rules and routes for individual Kubernetes services. Using an ingress controller and ingress rules, a single IP address can be used to route traffic to multiple services in a Kubernetes cluster.

We will utilize the ingress controller to expose the application managed by OSM to the internet. To create the

ingress controller, use Helm to install nginx-ingress. For added redundancy, two replicas of the NGINX ingress controllers are deployed with the `--set controller.replicaCount` parameter. To fully benefit from running replicas of the ingress controller, make sure there's more than one node in your AKS cluster.

The ingress controller also needs to be scheduled on a Linux node. Windows Server nodes shouldn't run the ingress controller. A node selector is specified using the `--set nodeSelector` parameter to tell the Kubernetes scheduler to run the NGINX ingress controller on a Linux-based node.

**TIP**

The following example creates a Kubernetes namespace for the ingress resources named *ingress-basic*. Specify a namespace for your own environment as needed.

```
Create a namespace for your ingress resources
kubectl create namespace ingress-basic

Add the ingress-nginx repository
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

Update the helm repo(s)
helm repo update

Use Helm to deploy an NGINX ingress controller in the ingress-basic namespace
helm install nginx-ingress ingress-nginx/nginx-inginx \
--namespace ingress-basic \
--set controller.replicaCount=1 \
--set controller.nodeSelector."beta\.kubernetes\.io/os"=linux \
--set defaultBackend.nodeSelector."beta\.kubernetes\.io/os"=linux \
--set controller.admissionWebhooks.patch.nodeSelector."beta\.kubernetes\.io/os"=linux
```

When the Kubernetes load balancer service is created for the NGINX ingress controller, a dynamic public IP address is assigned, as shown in the following example output:

```
$ kubectl --namespace ingress-basic get services -o wide -w nginx-ingress-ingress-nginx-controller

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S)
AGE SELECTOR
nginx-ingress-ingress-nginx-controller LoadBalancer 10.0.74.133 EXTERNAL_IP
80:32486/TCP,443:30953/TCP 44s app.kubernetes.io/component=controller,app.kubernetes.io/instance=nginx-
ingress,app.kubernetes.io/name=ingress-nginx
```

No ingress rules have been created yet, so the NGINX ingress controller's default 404 page is displayed if you browse to the internal IP address. Ingress rules are configured in the following steps.

### Expose the bookbuyer service to the internet

```

kubectl apply -f - <<EOF

apiVersion: extensions/v1beta1
kind: Ingress
metadata:
 name: bookbuyer-ingress
 namespace: bookbuyer
 annotations:
 kubernetes.io/ingress.class: nginx

spec:

rules:
 - host: bookbuyer.contoso.com
 http:
 paths:
 - path: /
 backend:
 serviceName: bookbuyer
 servicePort: 14001

 backend:
 serviceName: bookbuyer
 servicePort: 14001
EOF

```

You should see the following output:

```

Warning: extensions/v1beta1 Ingress is deprecated in v1.14+, unavailable in v1.22+; use networking.k8s.io/v1
Ingress
ingress.extensions/bookbuyer-ingress created

```

## View the NGINX logs

```

POD=$(kubectl get pods -n ingress-basic | grep 'nginx-ingress' | awk '{print $1}')

kubectl logs $POD -n ingress-basic -f

```

Output shows the NGINX ingress controller status when ingress rule has been applied successfully:

```

I0321 <date> 6 event.go:282] Event(v1.ObjectReference{Kind:"Pod", Namespace:"ingress-basic",
Name:"nginx-ingress-ingress-nginx-controller-54cf6c8bf4-jdvrw", UID:"3ebbe5e5-50ef-481d-954d-4b82a499ebe1",
APIVersion:"v1", ResourceVersion:"3272", FieldPath:""}): type: 'Normal' reason: 'RELOAD' NGINX reload
triggered due to a change in configuration
I0321 <date> 6 event.go:282] Event(v1.ObjectReference{Kind:"Ingress", Namespace:"bookbuyer",
Name:"bookbuyer-ingress", UID:"e1018efc-8116-493c-9999-294b4566819e",
APIVersion:"networking.k8s.io/v1beta1", ResourceVersion:"5460", FieldPath:""}): type: 'Normal' reason:
'Sync' Scheduled for sync
I0321 <date> 6 controller.go:146] "Configuration changes detected, backend reload required"
I0321 <date> 6 controller.go:163] "Backend successfully reloaded"
I0321 <date> 6 event.go:282] Event(v1.ObjectReference{Kind:"Pod", Namespace:"ingress-basic",
Name:"nginx-ingress-ingress-nginx-controller-54cf6c8bf4-jdvrw", UID:"3ebbe5e5-50ef-481d-954d-4b82a499ebe1",
APIVersion:"v1", ResourceVersion:"3272", FieldPath:""}): type: 'Normal' reason: 'RELOAD' NGINX reload
triggered due to a change in configuration

```

## View the NGINX services and bookbuyer service externally

```

kubectl get services -n ingress-basic

```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
AGE				
nginx-ingress-ingress-nginx-controller 80:31742/TCP,443:32683/TCP 4m15s	LoadBalancer	10.0.100.23	20.193.1.74	
nginx-ingress-ingress-nginx-controller-admission 4m15s	ClusterIP	10.0.163.98	<none>	443/TCP

Since the host name in the ingress manifest is a psuedo name used for testing, the DNS name will not be available on the internet. We can alternatively use the curl program and pass the hostname header to the NGINX public IP address and receive a 200 code successfully connecting us to the bookbuyer service.

```
curl -H 'Host: bookbuyer.contoso.com' http://EXTERNAL-IP/
```

You should see the following output:

```

<!doctype html>
<html itemscope="" itemtype="http://schema.org/WebPage" lang="en">
 <head>
 <meta content="Bookbuyer" name="description">
 <meta content="text/html; charset=UTF-8" http-equiv="Content-Type">
 <title>Bookbuyer</title>
 <style>
 #navbar {
 width: 100%;
 height: 50px;
 display: table;
 border-spacing: 0;
 white-space: nowrap;
 line-height: normal;
 background-color: #0078D4;
 background-position: left top;
 background-repeat-x: repeat;
 background-image: none;
 color: white;
 font: 2.2em "Fira Sans", sans-serif;
 }
 #main {
 padding: 10pt 10pt 10pt 10pt;
 font: 1.8em "Fira Sans", sans-serif;
 }
 li {
 padding: 10pt 10pt 10pt 10pt;
 font: 1.2em "Consolas", sans-serif;
 }
 </style>
 <script>
 setTimeout(function(){window.location.reload(1);}, 1500);
 </script>
 </head>
 <body bgcolor="#fff">
 <div id="navbar">
 📖 Bookbuyer
 </div>
 <div id="main">

 Total books bought: 1833

 from bookstore V1: 277
 from bookstore V2: 1556

 </div>

 Current Time: Fri, 26 Mar 2021 15:02:53 UTC
 </body>
</html>

```

## Tutorial: Deploy an application managed by Open Service Mesh (OSM) using Azure Application Gateway ingress AKS add-on

Open Service Mesh (OSM) is a lightweight, extensible, Cloud Native service mesh that allows users to uniformly manage, secure, and get out-of-the-box observability features for highly dynamic microservice environments.

In this tutorial, you will:

- View the current OSM cluster configuration
- Create the namespace(s) for OSM to manage deployed applications in the namespace(s)
- Onboard the namespaces to be managed by OSM
- Deploy the sample application
- Verify the application running inside the AKS cluster
- Create an Azure Application Gateway to be used as the ingress controller for the application
- Expose a service via the Azure Application Gateway ingress to the internet

## Before you begin

The steps detailed in this article assume that you've created an AKS cluster (Kubernetes 1.19+ and above, with Kubernetes RBAC enabled), have established a `kubectl` connection with the cluster (If you need help with any of these items, then see the [AKS quickstart](#), have installed the AKS OSM add-on, and will be creating a new Azure Application Gateway for ingress.

You must have the following resources installed:

- The Azure CLI, version 2.20.0 or later
- The `aks-preview` extension version 0.5.5 or later
- AKS cluster version 1.19+ using Azure CNI networking (Attached to an Azure Vnet)
- OSM version v0.8.0 or later
- `apt-get install jq`

## View and verify the current OSM cluster configuration

Once the OSM add-on for AKS has been enabled on the AKS cluster, you can view the current configuration parameters in the `osm-config` Kubernetes ConfigMap. Run the following command to view the ConfigMap properties:

```
kubectl get configmap -n kube-system osm-config -o json | jq '.data'
```

Output shows the current OSM configuration for the cluster.

```
{
 "egress": "true",
 "enable_debug_server": "true",
 "enable_privileged_init_container": "false",
 "envoy_log_level": "error",
 "outbound_ip_range_exclusion_list": "169.254.169.254,168.63.129.16,20.193.57.43",
 "permissive_traffic_policy_mode": "false",
 "prometheus_scraping": "false",
 "service_cert_validity_duration": "24h",
 "use_https_ingress": "false"
}
```

Notice the `permissive_traffic_policy_mode` is configured to `true`. Permissive traffic policy mode in OSM is a mode where the [SMI](#) traffic policy enforcement is bypassed. In this mode, OSM automatically discovers services that are a part of the service mesh and programs traffic policy rules on each Envoy proxy sidecar to be able to communicate with these services.

## Create namespaces for the application

In this tutorial we will be using the OSM bookstore application that has the following application components:

- bookbuyer
- bookthief
- bookstore

- bookwarehouse

Create namespaces for each of these application components.

```
for i in bookstore bookbuyer bookthief bookwarehouse; do kubectl create ns $i; done
```

You should see the following output:

```
namespace/bookstore created
namespace/bookbuyer created
namespace/bookthief created
namespace/bookwarehouse created
```

### Onboard the namespaces to be managed by OSM

When you add the namespaces to the OSM mesh, this will allow the OSM controller to automatically inject the Envoy sidecar proxy containers with your application. Run the following command to onboard the OSM bookstore application namespaces.

```
osm namespace add bookstore bookbuyer bookthief bookwarehouse
```

You should see the following output:

```
Namespace [bookstore] successfully added to mesh [osm]
Namespace [bookbuyer] successfully added to mesh [osm]
Namespace [bookthief] successfully added to mesh [osm]
Namespace [bookwarehouse] successfully added to mesh [osm]
```

### Deploy the Bookstore application to the AKS cluster

```
kubectl apply -f https://raw.githubusercontent.com/openservicemesh/osm/release-v0.8/docs/example/manifests/apps/bookbuyer.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/openservicemesh/osm/release-v0.8/docs/example/manifests/apps/bookthief.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/openservicemesh/osm/release-v0.8/docs/example/manifests/apps/bookstore.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/openservicemesh/osm/release-v0.8/docs/example/manifests/apps/bookwarehouse.yaml
```

All of the deployment outputs are summarized below.

```
serviceaccount/bookbuyer created
service/bookbuyer created
deployment.apps/bookbuyer created

serviceaccount/bookthief created
service/bookthief created
deployment.apps/bookthief created

service/bookstore created
serviceaccount/bookstore created
deployment.apps/bookstore created

serviceaccount/bookwarehouse created
service/bookwarehouse created
deployment.apps/bookwarehouse created
```

## Update the Bookbuyer Service

Update the bookbuyer service to the correct inbound port configuration with the following service manifest.

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
 name: bookbuyer
 namespace: bookbuyer
 labels:
 app: bookbuyer
spec:
 ports:
 - port: 14001
 name: inbound-port
 selector:
 app: bookbuyer
EOF
```

## Verify the Bookstore application running inside the AKS cluster

As of now we have deployed the bookstore multi-container application, but it is only accessible from within the AKS cluster. Later we will add the Azure Application Gateway ingress controller to expose the application outside the AKS cluster. To verify that the application is running inside the cluster, we will use a port forward to view the bookbuyer component UI.

First let's get the bookbuyer pod's name

```
kubectl get pod -n bookbuyer
```

You should see output similar to the following. Your bookbuyer pod will have a unique name appended.

NAME	READY	STATUS	RESTARTS	AGE
bookbuyer-7676c7fcfb-mtnrz	2/2	Running	0	7m8s

Once we have the pod's name, we can now use the port-forward command to set up a tunnel from our local system to the application inside the AKS cluster. Run the following command to set up the port forward for the local system port 8080. Again use your specific bookbuyer pod name.

```
kubectl port-forward bookbuyer-7676c7fcfb-mtnrz -n bookbuyer 8080:14001
```

You should see output similar to this.

```
Forwarding from 127.0.0.1:8080 -> 14001
Forwarding from [::1]:8080 -> 14001
```

While the port forwarding session is in place, navigate to the following url from a browser

<http://localhost:8080>. You should now be able to see the bookbuyer application UI in the browser similar to the image below.



## Create an Azure Application Gateway to expose the bookbuyer application outside the AKS cluster

### NOTE

The following directions will create a new instance of the Azure Application Gateway to be used for ingress. If you have an existing Azure Application Gateway you wish to use, skip to the section for enabling the Application Gateway Ingress Controller add-on.

## Deploy a new Application Gateway

### NOTE

We are referencing existing documentation for enabling the Application Gateway Ingress Controller add-on for an existing AKS cluster. Some modifications have been made to suit the OSM materials. More detailed documentation on the subject can be found [here](#).

You'll now deploy a new Application Gateway, to simulate having an existing Application Gateway that you want to use to load balance traffic to your AKS cluster, *myCluster*. The name of the Application Gateway will be *myApplicationGateway*, but you will need to first create a public IP resource, named *myPublicIp*, and a new virtual network called *myVnet* with address space 11.0.0.0/8, and a subnet with address space 11.1.0.0/16 called *mySubnet*, and deploy your Application Gateway in *mySubnet* using *myPublicIp*.

When using an AKS cluster and Application Gateway in separate virtual networks, the address spaces of the two virtual networks must not overlap. The default address space that an AKS cluster deploys in is 10.0.0.0/8, so we set the Application Gateway virtual network address prefix to 11.0.0.0/8.

```
az group create --name myResourceGroup --location eastus2
az network public-ip create -n myPublicIp -g MyResourceGroup --allocation-method Static --sku Standard
az network vnet create -n myVnet -g myResourceGroup --address-prefix 11.0.0.0/8 --subnet-name mySubnet --
subnet-prefix 11.1.0.0/16
az network application-gateway create -n myApplicationGateway -l eastus2 -g myResourceGroup --sku
Standard_v2 --public-ip-address myPublicIp --vnet-name myVnet --subnet mySubnet
```

#### NOTE

Application Gateway Ingress Controller (AGIC) add-on **only** supports Application Gateway v2 SKUs (Standard and WAF), and **not** the Application Gateway v1 SKUs.

#### Enable the AGIC add-on for an existing AKS cluster through Azure CLI

If you'd like to continue using Azure CLI, you can continue to enable the AGIC add-on in the AKS cluster you created, *myCluster*, and specify the AGIC add-on to use the existing Application Gateway you created, *myApplicationGateway*.

```
appgwId=$(az network application-gateway show -n myApplicationGateway -g myResourceGroup -o tsv --query
"id")
az aks enable-addons -n myCluster -g myResourceGroup -a ingress-appgw --appgw-id $appgwId
```

You can verify the Azure Application Gateway AKS add-on has been enabled by the following command.

```
az aks list -g osm-aks-rg -o json | jq -r .[].addonProfiles.ingressApplicationGateway.enabled
```

This command should show the output as `true`.

#### Peer the two virtual networks together

Since we deployed the AKS cluster in its own virtual network and the Application Gateway in another virtual network, you'll need to peer the two virtual networks together in order for traffic to flow from the Application Gateway to the pods in the cluster. Peering the two virtual networks requires running the Azure CLI command two separate times, to ensure that the connection is bi-directional. The first command will create a peering connection from the Application Gateway virtual network to the AKS virtual network; the second command will create a peering connection in the other direction.

```
nodeResourceGroup=$(az aks show -n myCluster -g myResourceGroup -o tsv --query "nodeResourceGroup")
aksVnetName=$(az network vnet list -g $nodeResourceGroup -o tsv --query "[0].name")

aksVnetId=$(az network vnet show -n $aksVnetName -g $nodeResourceGroup -o tsv --query "id")
az network vnet peering create -n AppGwtoAKSVnetPeering -g myResourceGroup --vnet-name myVnet --remote-vnet
$aksVnetId --allow-vnet-access

appGWVnetId=$(az network vnet show -n myVnet -g myResourceGroup -o tsv --query "id")
az network vnet peering create -n AKStoAppGWVnetPeering -g $nodeResourceGroup --vnet-name $aksVnetName --
remote-vnet $appGWVnetId --allow-vnet-access
```

#### Expose the bookbuyer service to the internet

Apply the following ingress manifest to the AKS cluster to expose the bookbuyer service to the internet via the Azure Application Gateway.

```
kubectl apply -f - <<EOF

apiVersion: extensions/v1beta1
kind: Ingress
metadata:
 name: bookbuyer-ingress
 namespace: bookbuyer
 annotations:
 kubernetes.io/ingress.class: azure/application-gateway

spec:

rules:
 - host: bookbuyer.contoso.com
 http:
 paths:
 - path: /
 backend:
 serviceName: bookbuyer
 servicePort: 14001

 backend:
 serviceName: bookbuyer
 servicePort: 14001
EOF
```

You should see the following output

```
Warning: extensions/v1beta1 Ingress is deprecated in v1.14+, unavailable in v1.22+; use networking.k8s.io/v1
Ingress
ingress.extensions/bookbuyer-ingress created
```

Since the host name in the ingress manifest is a pseudo name used for testing, the DNS name will not be available on the internet. We can alternatively use the curl program and pass the hostname header to the Azure Application Gateway public IP address and receive a 200 code successfully connecting us to the bookbuyer service.

```
appGWPIP=$(az network public-ip show -g MyResourceGroup -n myPublicIp -o tsv --query "ipAddress")
curl -H 'Host: bookbuyer.contoso.com' http://$appGWPIP/
```

You should see the following output

```

<!doctype html>
<html itemscope="" itemtype="http://schema.org/WebPage" lang="en">
 <head>
 <meta content="Bookbuyer" name="description">
 <meta content="text/html; charset=UTF-8" http-equiv="Content-Type">
 <title>Bookbuyer</title>
 <style>
 #navbar {
 width: 100%;
 height: 50px;
 display: table;
 border-spacing: 0;
 white-space: nowrap;
 line-height: normal;
 background-color: #0078D4;
 background-position: left top;
 background-repeat-x: repeat;
 background-image: none;
 color: white;
 font: 2.2em "Fira Sans", sans-serif;
 }
 #main {
 padding: 10pt 10pt 10pt 10pt;
 font: 1.8em "Fira Sans", sans-serif;
 }
 li {
 padding: 10pt 10pt 10pt 10pt;
 font: 1.2em "Consolas", sans-serif;
 }
 </style>
 <script>
 setTimeout(function(){window.location.reload(1);}, 1500);
 </script>
 </head>
 <body bgcolor="#ffff">
 <div id="navbar">
 📖 Bookbuyer
 </div>
 <div id="main">

 Total books bought: 5969

 from bookstore V1: 277
 from bookstore V2: 5692

 </div>

 Current Time: Fri, 26 Mar 2021 16:34:30 UTC
 </body>
</html>

```

## Troubleshooting

- [AGIC Troubleshooting Documentation](#)
- Additional troubleshooting tools are available on AGIC's GitHub repo

# Open Service Mesh (OSM) Monitoring and Observability using Azure Monitor and Applications Insights

Both Azure Monitor and Azure Application Insights helps you maximize the availability and performance of your applications and services by delivering a comprehensive solution for collecting, analyzing, and acting on telemetry from your cloud and on-premises environments.

The OSM AKS add-on will have deep integrations into both of these Azure services, and provide a seamless Azure experience for viewing and responding to critical KPIs provided by OSM metrics. For more information on how to enable and configure these services for the OSM AKS add-on, visit the [Azure Monitor for OSM](#) page for more information.

## Tutorial: Manually deploy Prometheus, Grafana, and Jaeger to view Open Service Mesh (OSM) metrics for observability

### WARNING

The installation of Prometheus, Grafana and Jaeger are provided as general guidance to show how these tools can be utilized to view OSM metric data. The installation guidance is not to be utilized for a production setup. Please refer to each tool's documentation on how best to suit thier installations to your needs. Most notable will be the lack of persistent storage, meaning that all data is lost once a Prometheus Grafana, and/or Jaeger pod(s) are terminated.

Open Service Mesh (OSM) generates detailed metrics related to all traffic within the mesh. These metrics provide insights into the behavior of applications in the mesh helping users to troubleshoot, maintain, and analyze their applications.

As of today OSM collects metrics directly from the sidecar proxies (Envoy). OSM provides rich metrics for incoming and outgoing traffic for all services in the mesh. With these metrics, the user can get information about the overall volume of traffic, errors within traffic and the response time for requests.

OSM uses Prometheus to gather and store consistent traffic metrics and statistics for all applications running in the mesh. Prometheus is an open-source monitoring and alerting toolkit, which is commonly used on (but not limited to) Kubernetes and Service Mesh environments.

Each application that is part of the mesh runs in a Pod that contains an Envoy sidecar that exposes metrics (proxy metrics) in the Prometheus format. Furthermore, every Pod that is a part of the mesh has Prometheus annotations, which makes it possible for the Prometheus server to scrape the application dynamically. This mechanism automatically enables scraping of metrics whenever a new namespace/pod/service is added to the mesh.

OSM metrics can be viewed with Grafana, which is an open-source visualization and analytics software. It allows you to query, visualize, alert on, and explore your metrics.

In this tutorial, you will:

- Create and deploy a Prometheus instance
- Configure OSM to allow Prometheus scraping
- Update the Prometheus Configmap
- Create and deploy a Grafana instance
- Configure Grafana with the Prometheus datasource
- Import OSM dashboard for Grafana
- Create and deploy a Jaeger instance
- Configure Jaeger tracing for OSM

### Deploy and configure a Prometheus instance for OSM

We will use Helm to deploy the Prometheus instance. Run the following commands to install Prometheus via Helm:

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
helm install stable prometheus-community/prometheus
```

You should see similar output below if the installation was successful. Make note of the Prometheus server port and cluster DNS name. This information will be used later for to configure Prometheus as a data source for Grafana.

```
NAME: stable
LAST DEPLOYED: Fri Mar 26 13:34:51 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The Prometheus server can be accessed via port 80 on the following DNS name from within your cluster:
stable-prometheus-server.default.svc.cluster.local
```

Get the Prometheus server URL by running these commands in the same shell:

```
export POD_NAME=$(kubectl get pods --namespace default -l "app=prometheus,component=server" -o jsonpath=".items[0].metadata.name")
kubectl --namespace default port-forward $POD_NAME 9090
```

The Prometheus alertmanager can be accessed via port 80 on the following DNS name from within your cluster:  
stable-prometheus-alertmanager.default.svc.cluster.local

Get the Alertmanager URL by running these commands in the same shell:

```
export POD_NAME=$(kubectl get pods --namespace default -l "app=prometheus,component=alertmanager" -o jsonpath=".items[0].metadata.name")
kubectl --namespace default port-forward $POD_NAME 9093
#####
WARNING: Pod Security Policy has been moved to a global property.
use .Values.podSecurityPolicy.enabled with pod-based
annotations
(e.g. .Values.nodeExporter.podSecurityPolicy.annotations)
#####
```

The Prometheus PushGateway can be accessed via port 9091 on the following DNS name from within your cluster:  
stable-prometheus-pushgateway.default.svc.cluster.local

Get the PushGateway URL by running these commands in the same shell:

```
export POD_NAME=$(kubectl get pods --namespace default -l "app=prometheus,component=pushgateway" -o jsonpath=".items[0].metadata.name")
kubectl --namespace default port-forward $POD_NAME 9091
```

For more information on running Prometheus, visit:  
<https://prometheus.io/>

### Configure OSM to allow Prometheus scraping

To ensure that the OSM components are configured for Prometheus scrapes, we'll want to check the **prometheus\_scraping** configuration located in the osm-config config file. View the configuration with the following command:

```
kubectl get configmap -n kube-system osm-config -o json | jq '.data.prometheus_scraping'
```

The output of the previous command should return `true` if OSM is configured for Prometheus scraping. If the

returned value is `false`, we will need to update the configuration to be `true`. Run the following command to turn **on** OSM Prometheus scraping:

```
kubectl patch ConfigMap -n kube-system osm-config --type merge --patch '{"data": {"prometheus_scraping":"true"}}'
```

You should see the following output.

```
configmap/osm-config patched
```

#### Update the Prometheus Configmap

The default installation of Prometheus will contain two Kubernetes configmaps. You can view the list of Prometheus configmaps with the following command.

```
kubectl get configmap | grep prometheus
```

```
stable-prometheus-alertmanager 1 4h34m
stable-prometheus-server 5 4h34m
```

We will need to replace the `prometheus.yml` configuration located in the **stable-prometheus-server** configmap with the following OSM configuration. There are several file editing techniques to accomplish this task. A simple and safe way is to export the configmap, create a copy of it for backup, then edit it with an editor such as Visual Studio code.

#### NOTE

If you do not have Visual Studio Code installed you can go download and install it [here](#).

Let's first export out the **stable-prometheus-server** configmap and then make a copy for backup.

```
kubectl get configmap stable-prometheus-server -o yaml > cm-stable-prometheus-server.yaml
cp cm-stable-prometheus-server.yaml cm-stable-prometheus-server.yaml.copy
```

Next let's open the file using Visual Studio Code to edit.

```
code cm-stable-prometheus-server.yaml
```

Once you have the configmap opened in the Visual Studio Code editor, replace the `prometheus.yml` file with the OSM configuration below and save the file.

#### WARNING

It is extremely important that you ensure you keep the indentation structure of the yaml file. Any changes to the yaml file structure could result in the configmap not being able to be re-applied.

```
prometheus.yml: |
 global:
 scrape_interval: 10s
 scrape_timeout: 10s
 evaluation_interval: 1m
```

```

scrape_configs:
 - job_name: 'kubernetes-apiservers'
 kubernetes_sd_configs:
 - role: endpoints
 scheme: https
 tls_config:
 ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
 # TODO need to remove this when the CA and SAN match
 insecure_skip_verify: true
 bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
 metric_relabel_configs:
 - source_labels: [__name__]
 regex: '(apiserver_watch_events_total|apiserver_admission_webhook_rejection_count)'
 action: keep
 relabel_configs:
 - source_labels: [__meta_kubernetes_namespace, __meta_kubernetes_service_name,
 __meta_kubernetes_endpoint_port_name]
 action: keep
 regex: default;kubernetes;https

 - job_name: 'kubernetes-nodes'
 scheme: https
 tls_config:
 ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
 bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
 kubernetes_sd_configs:
 - role: node
 relabel_configs:
 - action: labelmap
 regex: __meta_kubernetes_node_label_(.+)
 - target_label: __address__
 replacement: kubernetes.default.svc:443
 - source_labels: [__meta_kubernetes_node_name]
 regex: (.+)
 target_label: __metrics_path__
 replacement: /api/v1/nodes/${1}/proxy/metrics

 - job_name: 'kubernetes-pods'
 kubernetes_sd_configs:
 - role: pod
 metric_relabel_configs:
 - source_labels: [__name__]
 regex:
 '(envoy_server_live|envoy_cluster_upstream_rq_xx|envoy_cluster_upstream_cx_active|envoy_cluster_upstream_cx_tx_bytes_total|envoy_cluster_upstream_cx_rx_bytes_total|envoy_cluster_upstream_cx_destroy_remote_with_active_rq|envoy_cluster_upstream_cx_connect_timeout|envoy_cluster_upstream_cx_destroy_local_with_active_rq|envoy_cluster_upstream_rq_pending_failure_eject|envoy_cluster_upstream_rq_pending_overflow|envoy_cluster_upstream_rq_timeout|envoy_cluster_upstream_rq_rx_reset|^osm.*)'
 action: keep
 relabel_configs:
 - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
 action: keep
 regex: true
 - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
 action: replace
 target_label: __metrics_path__
 regex: (.+)
 - source_labels: [__address__, __meta_kubernetes_pod_annotation_prometheus_io_port]
 action: replace
 regex: ([^:]+)(?::\d+)?;(\d+)
 replacement: $1:$2
 target_label: __address__
 - source_labels: [__meta_kubernetes_namespace]
 action: replace
 target_label: source_namespace
 - source_labels: [__meta_kubernetes_pod_name]
 action: replace
 target_label: source_pod_name
 regex: '(__meta_kubernetes_pod_label_*)'

```

```

 - regex: __meta_kubernetes_pod_label_osm_envoy_uid|__meta_kubernetes_pod_label_template_hash|__meta_kubernetes_
 pod_label_version'
 action: drop
 # for non-ReplicaSets (DaemonSet, StatefulSet)
 # __meta_kubernetes_pod_controller_kind=DaemonSet
 # __meta_kubernetes_pod_controller_name=foo
 # =>
 # workload_kind=DaemonSet
 # workload_name=foo
 - source_labels: [__meta_kubernetes_pod_controller_kind]
 action: replace
 target_label: source_workload_kind
 - source_labels: [__meta_kubernetes_pod_controller_name]
 action: replace
 target_label: source_workload_name
 # for ReplicaSets
 # __meta_kubernetes_pod_controller_kind=ReplicaSet
 # __meta_kubernetes_pod_controller_name=foo-bar-123
 # =>
 # workload_kind=Deployment
 # workload_name=foo-bar
 # deployment=foo
 - source_labels: [__meta_kubernetes_pod_controller_kind]
 action: replace
 regex: ^ReplicaSet$
 target_label: source_workload_kind
 replacement: Deployment
 - source_labels:
 - __meta_kubernetes_pod_controller_kind
 - __meta_kubernetes_pod_controller_name
 action: replace
 regex: ^ReplicaSet;(.*)-[^-]+$
 target_label: source_workload_name

 - job_name: 'smi-metrics'
 kubernetes_sd_configs:
 - role: pod
 relabel_configs:
 - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
 action: keep
 regex: true
 - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
 action: replace
 target_label: __metrics_path__
 regex: (.+)
 - source_labels: [__address__, __meta_kubernetes_pod_annotation_prometheus_io_port]
 action: replace
 regex: ([^:])(?:\d+)?;(\d+)
 replacement: $1:$2
 target_label: __address__
 metric_relabel_configs:
 - source_labels: [__name__]
 regex: 'envoy_.*osm_request_(total|duration_ms_(bucket|count|sum))'
 action: keep
 - source_labels: [__name__]
 action: replace
 regex:
envoy_response_code_(\d{3})_source_namespace_.*_source_kind_.*_source_name_.*_source_pod_.*_destination_name
space_.*_destination_kind_.*_destination_name_.*_destination_pod_.*_osm_request_total
 target_label: response_code
 - source_labels: [__name__]
 action: replace
 regex:
envoy_response_code_\d{3}_source_namespace_.*_source_kind_.*_source_name_.*_source_pod_.*_destination_name
space_.*_destination_kind_.*_destination_name_.*_destination_pod_.*_osm_request_total

```

```

 target_label: source_namespace
- source_labels: [__name__]
 action: replace
 regex:
envoy_response_code_\d{3}_source_namespace_.*_source_kind_(.*)_source_name_.*_source_pod_.*_destination_name
space_.*_destination_kind_.*_destination_name_.*_destination_pod_.*_osm_request_total
 target_label: source_kind
- source_labels: [__name__]
 action: replace
 regex:
envoy_response_code_\d{3}_source_namespace_.*_source_kind_.*_source_name_(.*)_source_pod_.*_destination_name
space_.*_destination_kind_.*_destination_name_.*_destination_pod_.*_osm_request_total
 target_label: source_name
- source_labels: [__name__]
 action: replace
 regex:
envoy_response_code_\d{3}_source_namespace_.*_source_kind_.*_source_name_.*_source_pod_(.*)_destination_name
space_.*_destination_kind_.*_destination_name_.*_destination_pod_.*_osm_request_total
 target_label: source_pod
- source_labels: [__name__]
 action: replace
 regex:
envoy_response_code_\d{3}_source_namespace_.*_source_kind_.*_source_name_.*_source_pod_.*_destination_namesp
ace_(.*)_destination_kind_.*_destination_name_.*_destination_pod_.*_osm_request_total
 target_label: destination_namespace
- source_labels: [__name__]
 action: replace
 regex:
envoy_response_code_\d{3}_source_namespace_.*_source_kind_.*_source_name_.*_source_pod_.*_destination_namesp
ace_.*_destination_kind_(.*)_destination_name_.*_destination_pod_.*_osm_request_total
 target_label: destination_kind
- source_labels: [__name__]
 action: replace
 regex:
envoy_response_code_\d{3}_source_namespace_.*_source_kind_.*_source_name_.*_source_pod_.*_destination_namesp
ace_.*_destination_kind_.*_destination_name_(.*)_destination_pod_.*_osm_request_total
 target_label: destination_name
- source_labels: [__name__]
 action: replace
 regex:
envoy_response_code_\d{3}_source_namespace_.*_source_kind_.*_source_name_.*_source_pod_.*_destination_namesp
ace_.*_destination_kind_.*_destination_name_.*_destination_pod_(.*)_osm_request_total
 target_label: destination_pod
- source_labels: [__name__]
 action: replace
 regex: .*(osm_request_total)
 target_label: __name__

- source_labels: [__name__]
 action: replace
 regex:
envoy_source_namespace_(.*)_source_kind_.*_source_name_.*_source_pod_.*_destination_namespace_.*_destinat
ion_kind_.*_destination_name_.*_destination_pod_.*_osm_request_duration_ms_(bucket|sum|count)
 target_label: source_namespace
- source_labels: [__name__]
 action: replace
 regex:
envoy_source_namespace_.*_source_kind_(.*)_source_name_.*_source_pod_.*_destination_namespace_.*_destinat
ion_kind_.*_destination_name_.*_destination_pod_.*_osm_request_duration_ms_(bucket|sum|count)
 target_label: source_kind
- source_labels: [__name__]
 action: replace
 regex:
envoy_source_namespace_.*_source_kind_.*_source_name_(.*)_source_pod_.*_destination_namespace_.*_destinat
ion_kind_.*_destination_name_.*_destination_pod_.*_osm_request_duration_ms_(bucket|sum|count)
 target_label: source_name
- source_labels: [__name__]
 action: replace
 regex:

```

```

envoy_source_namespace_.*_source_kind_.*_source_name_.*_source_pod_(.*)_destination_namespace_.*_destination
kind.*_destination_name_.*_destination_pod_.*_osm_request_duration_ms_(bucket|sum|count)
 target_label: source_pod
 - source_labels: [__name__]
 action: replace
 regex:
envoy_source_namespace_.*_source_kind_.*_source_name_.*_source_pod_.*_destination_namespace_(.*)_destination
kind.*_destination_name_.*_destination_pod_.*_osm_request_duration_ms_(bucket|sum|count)
 target_label: destination_namespace
 - source_labels: [__name__]
 action: replace
 regex:
envoy_source_namespace_.*_source_kind_.*_source_name_.*_source_pod_.*_destination_namespace_.*_destination_k
ind_(.*)_destination_name_.*_destination_pod_.*_osm_request_duration_ms_(bucket|sum|count)
 target_label: destination_kind
 - source_labels: [__name__]
 action: replace
 regex:
envoy_source_namespace_.*_source_kind_.*_source_name_.*_source_pod_.*_destination_namespace_.*_destination_k
ind_.*_destination_name_(.*)_destination_pod_.*_osm_request_duration_ms_(bucket|sum|count)
 target_label: destination_name
 - source_labels: [__name__]
 action: replace
 regex:
envoy_source_namespace_.*_source_kind_.*_source_name_.*_source_pod_.*_destination_namespace_.*_destination_k
ind_.*_destination_name_.*_destination_pod_(.*)_osm_request_duration_ms_(bucket|sum|count)
 target_label: destination_pod
 - source_labels: [__name__]
 action: replace
 regex: .*(osm_request_duration_ms_(bucket|sum|count))
 target_label: __name__

 - job_name: 'kubernetes-cadvisor'
 scheme: https
 tls_config:
 ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
 bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
 kubernetes_sd_configs:
 - role: node
 metric_relabel_configs:
 - source_labels: [__name__]
 regex: '(container_cpu_usage_seconds_total|container_memory_rss)'
 action: keep
 relabel_configs:
 - action: labelmap
 regex: __meta_kubernetes_node_label_(.+)
 - target_label: __address__
 replacement: kubernetes.default.svc:443
 - source_labels: [__meta_kubernetes_node_name]
 regex: (.+)
 - target_label: __metrics_path__
 replacement: /api/v1/nodes/${1}/proxy/metrics/cadvisor

```

Apply the updated configmap yaml file with the following command.

```
kubectl apply -f cm-stable-prometheus-server.yml
```

```
configmap/stable-prometheus-server configured
```

#### NOTE

You may receive a message about a missing kubernetes annotation needed. This can be ignored for now.

## Verify Prometheus is configured to scrape the OSM mesh and API endpoints

To verify that Prometheus is correctly configured to scrape the OSM mesh and API endpoints, we will port forward to the Prometheus pod and view the target configuration. Run the following commands.

```
PROM POD NAME=$(kubectl get pods -l "app=prometheus,component=server" -o jsonpath=".items[0].metadata.name")
kubectl --namespace <promNamespace> port-forward $PROM POD NAME 9090
```

Open a browser up to <http://localhost:9090/targets>

If you scroll down you should see all the SMI metric endpoints state being UP as well as other OSM metrics defined as pictured below.

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
<a href="http://10.240.0.12:9091/metrics">http://10.240.0.12:9091/metrics</a>	UP	instance="10.240.0.12:9091", job="kubernetes-pods", source_namespace="kube-system", source_pod_name="osm-injector-9c48b8d46-wxrvn", source_service="osm-injector", source_workload_kind="Deployment", source_workload_name="osm-injector"	251.000ms	0.988ms	
<a href="http://10.240.0.36:9091/metrics">http://10.240.0.36:9091/metrics</a>	UP	instance="10.240.0.36:9091", job="smi-metrics", source_namespace="osm-mesh", source_pod_name="osm-mesh-0", source_service="osm-mesh", source_workload_kind="OSM", source_workload_name="osm-mesh"	7.28s	1.424ms	
<a href="http://10.240.0.56:8123/metrics">http://10.240.0.56:8123/metrics</a>	UP	instance="10.240.0.56:8123", job="smi-metrics", source_namespace="osm-mesh", source_pod_name="osm-mesh-0", source_service="osm-mesh", source_workload_kind="OSM", source_workload_name="osm-mesh"	4.474s	1.094ms	

## Deploy and configure a Grafana Instance for OSM

We will use Helm to deploy the Grafana instance. Run the following commands to install Grafana via Helm:

```
helm repo add grafana https://grafana.github.io/helm-charts
helm repo update
helm install osm-grafana grafana/grafana
```

Next we'll retrieve the default Grafana password to log into the Grafana site.

```
kubectl get secret --namespace default osm-grafana -o jsonpath="{.data.admin-password}" | base64 --decode ;
echo
```

Make note of the Grafana password.

Next we will retrieve the Grafana pod to port forward to the Grafana dashboard to login.

```
GRAF POD NAME=$(kubectl get pods -l "app.kubernetes.io/name=grafana" -o jsonpath=".items[0].metadata.name")
kubectl port-forward $GRAF POD NAME 3000
```

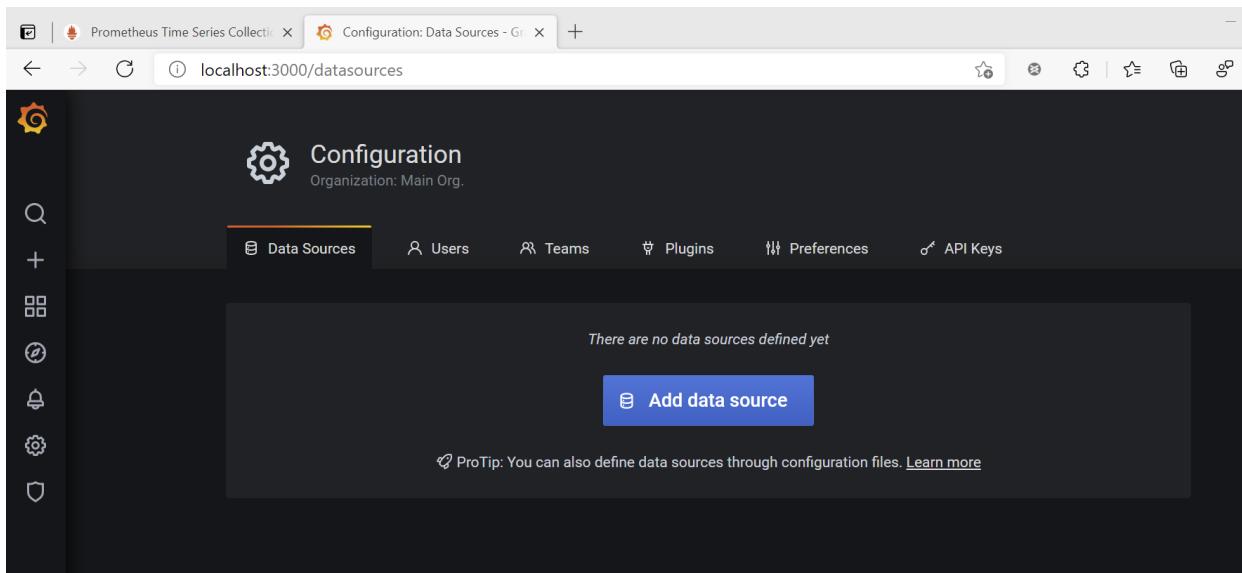
Open a browser up to <http://localhost:3000>

At the login screen pictured below, enter **admin** as the username and use the Grafana password captured earlier.



#### Configure the Grafana Prometheus data source

Once you have successfully logged into Grafana, the next step is to add Prometheus as data sources for Grafana. To do so, navigate on the configuration icon on the left menu and select Data Sources as shown below.



Click the **Add data source** button and select Prometheus under time series databases.



On the **Configure your Prometheus data source** below page, enter the Kubernetes cluster FQDN for the Prometheus service for the HTTP URL setting. The default FQDN should be

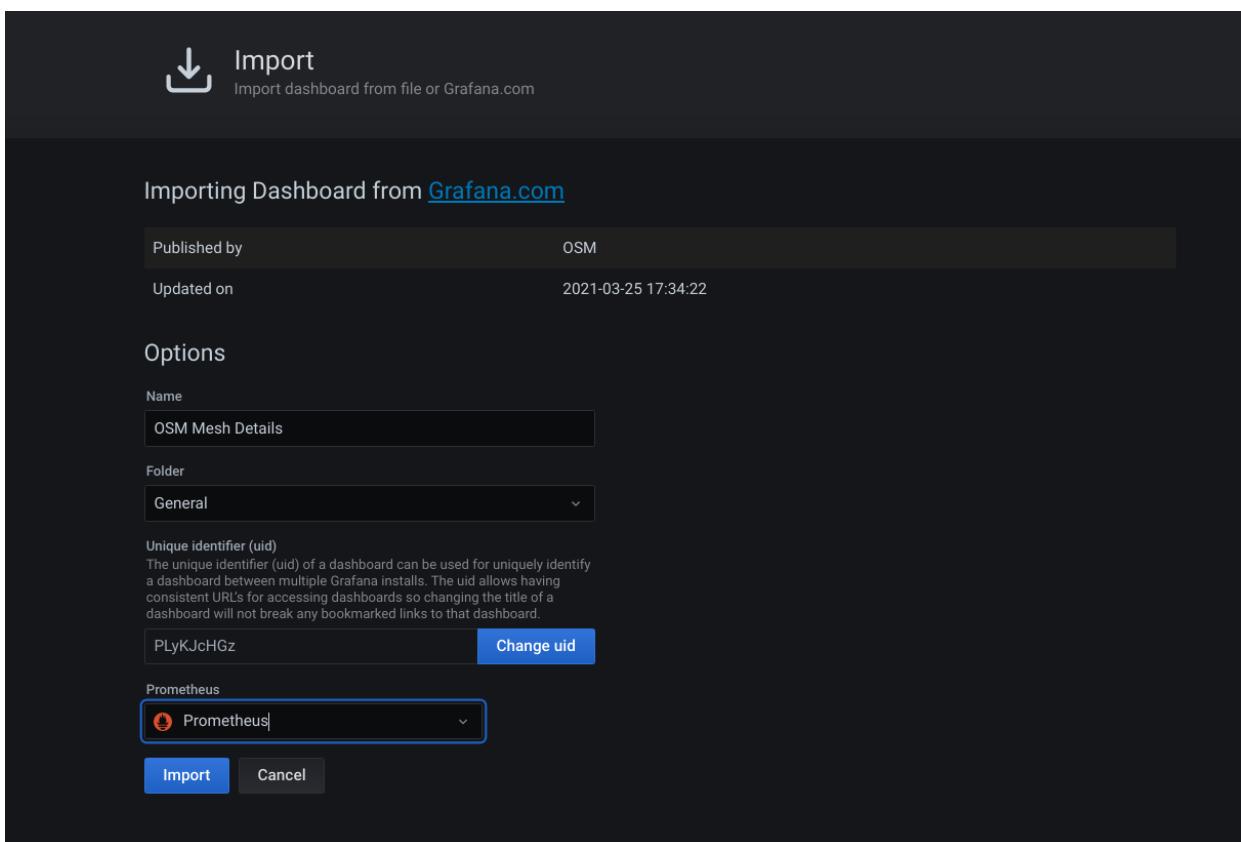
`stable-prometheus-server.default.svc.cluster.local`. Once you have entered that Prometheus service endpoint, scroll to the bottom of the page and select **Save & Test**. You should receive a green checkbox indicating the data source is working.

#### Importing OSM Dashboards

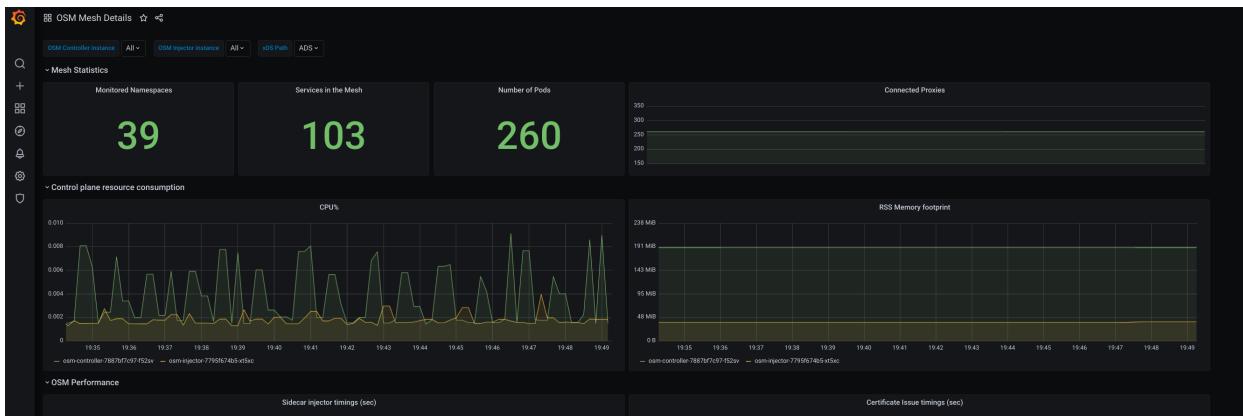
OSM Dashboards are available both through:

- [Our repository](#), and are importable as json blobs through the web admin portal
- or [online at Grafana.com](#)

To import a dashboard, look for the `+` sign on the left menu and select `import`. You can directly import dashboard by their ID on [Grafana.com](#). For example, our `OSM Mesh Details` dashboard uses ID `14145`, you can use the ID directly on the form and select `import`:



As soon as you select import, it will bring you automatically to your imported dashboard.



## Deploy and configure a Jaeger Operator on Kubernetes for OSM

Jaeger is an open-source tracing system used for monitoring and troubleshooting distributed systems. It can be deployed with OSM as a new instance or you may bring your own instance. The following instructions deploy a new instance of Jaeger to the `jaeger` namespace on the AKS cluster.

### Deploy Jaeger to the AKS cluster

Apply the following manifest to install Jaeger:

```

kubectl apply -f - <<EOF

apiVersion: apps/v1
kind: Deployment
metadata:
 name: jaeger
 namespace: jaeger
 labels:
 app: jaeger
spec:
 replicas: 1
 selector:
 matchLabels:
 app: jaeger
 template:
 metadata:
 labels:
 app: jaeger
 spec:
 containers:
 - name: jaeger
 image: jaegertracing/all-in-one
 args:
 - --collector.zipkin.host-port=9411
 imagePullPolicy: IfNotPresent
 ports:
 - containerPort: 9411
 resources:
 limits:
 cpu: 500m
 memory: 512M
 requests:
 cpu: 100m
 memory: 256M

kind: Service
apiVersion: v1
metadata:
 name: jaeger
 namespace: jaeger
 labels:
 app: jaeger
spec:
 selector:
 app: jaeger
 ports:
 - protocol: TCP
 # Service port and target port are the same
 port: 9411
 type: ClusterIP
EOF

```

```

deployment.apps/jaeger created
service/jaeger created

```

#### Enable Tracing for the OSM add-on

Next we will need to enable tracing for the OSM add-on.

#### NOTE

As of now the tracing properties are not visible in the osm-config configmap at this time. This will be made visible in a new release of the OSM AKS add-on.

Run the following command to enable tracing for the OSM add-on:

```
kubectl patch configmap osm-config -n kube-system -p '{"data":{"tracing_enable":"true", "tracing_address":"jaeger.jaeger.svc.cluster.local", "tracing_port":"9411", "tracing_endpoint":"/api/v2/spans"}}' --type=merge
```

```
configmap/osm-config patched
```

### View the Jaeger UI with port forwarding

Jaeger's UI is running on port 16686. To view the web UI, you can use kubectl port-forward:

```
JAEGER_POD=$(kubectl get pods -n jaeger --no-headers --selector app=jaeger | awk 'NR==1{print $1}')
kubectl port-forward -n jaeger $JAEGER_POD 16686:16686
http://localhost:16686/
```

In the browser, you should see a Service dropdown, which allows you to select from the various applications deployed by the bookstore demo. Select a service to view all spans from it. For example, if you select bookbuyer with a Lookback of one hour, you can see its interactions with bookstore-v1 and bookstore-v2 sorted by time.



Select any item to view it in further detail. Select multiple items to compare traces. For example, you can compare the bookbuyer's interactions with bookstore and bookstore-v2 at a particular moment in time.

You can also select the System Architecture tab to view a graph of how the various applications have been interacting/communicating. This provides an idea of how traffic is flowing between the applications.



## Open Service Mesh (OSM) AKS add-on Troubleshooting Guides

When you deploy the OSM AKS add-on, you might occasionally experience a problem. The following guides will assist you on how to troubleshoot errors and resolve common problems.

### Verifying and Troubleshooting OSM components

#### Check OSM Controller Deployment

```
kubectl get deployment -n kube-system --selector app=osm-controller
```

A healthy OSM Controller would look like this:

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
osm-controller	1/1	1	1	59m

#### Check the OSM Controller Pod

```
kubectl get pods -n kube-system --selector app=osm-controller
```

A healthy OSM Pod would look like this:

NAME	READY	STATUS	RESTARTS	AGE
osm-controller-b5bd66db-wglzl	0/1	Evicted	0	61m
osm-controller-b5bd66db-wv19w	1/1	Running	0	31m

Even though we had one controller evicted at some point, we have another one that is READY 1/1 and Running with 0 restarts. If the column READY is anything other than 1/1 the service mesh would be in a broken state. Column READY with 0/1 indicates the control plane container is crashing - we need to get logs. See Get OSM Controller Logs from Azure Support Center section below. Column READY with a number higher than 1 after the / would indicate that there are sidecars installed. OSM Controller would most likely not work with any sidecars attached to it.

#### **NOTE**

As of version v0.8.2 the OSM Controller is not in HA mode and will run in a deployed with replica count of 1 - single pod. The pod does have health probes and will be restarted by the kubelet if needed.

#### **Check OSM Controller Service**

```
kubectl get service -n kube-system osm-controller
```

A healthy OSM Controller service would look like this:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
osm-controller	ClusterIP	10.0.31.254	<none>	15128/TCP,9092/TCP	67m

#### **NOTE**

The CLUSTER-IP would be different. The service NAME and PORT(S) must be the same as the example above.

#### **Check OSM Controller Endpoints**

```
kubectl get endpoints -n kube-system osm-controller
```

A healthy OSM Controller endpoint(s) would look like this:

NAME	ENDPOINTS	AGE
osm-controller	10.240.1.115:9092,10.240.1.115:15128	69m

#### **Check OSM Injector Deployment**

```
kubectl get pod -n kube-system --selector app=osm-injector
```

A healthy OSM Injector deployment would look like this:

NAME	READY	STATUS	RESTARTS	AGE
osm-injector-5986c57765-vlsdk	1/1	Running	0	73m

#### **Check OSM Injector Pod**

```
kubectl get pod -n kube-system --selector app=osm-injector
```

A healthy OSM Injector pod would look like this:

NAME	READY	STATUS	RESTARTS	AGE
osm-injector-5986c57765-vlsdk	1/1	Running	0	73m

#### **Check OSM Injector Service**

```
kubectl get service -n kube-system osm-injector
```

A healthy OSM Injector service would look like this:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
osm-injector	ClusterIP	10.0.39.54	<none>	9090/TCP	75m

### Check OSM Endpoints

```
kubectl get endpoints -n kube-system osm-injector
```

A healthy OSM endpoint would look like this:

NAME	ENDPOINTS	AGE
osm-injector	10.240.1.172:9090	75m

### Check Validating and Mutating webhooks

```
kubectl get ValidatingWebhookConfiguration --selector app=osm-controller
```

A healthy OSM Validating Webhook would look like this:

NAME	WEBHOOKS	AGE
aks-osm-webhook-osm	1	81m

```
kubectl get MutatingWebhookConfiguration --selector app=osm-injector
```

A healthy OSM Mutating Webhook would look like this:

NAME	WEBHOOKS	AGE
aks-osm-webhook-osm	1	102m

### Check for the service and the CA bundle of the Validating webhook

```
kubectl get ValidatingWebhookConfiguration aks-osm-webhook-osm -o json | jq '.webhooks[0].clientConfig.service'
```

A well configured Validating Webhook Configuration would look exactly like this:

```
{
 "name": "osm-config-validator",
 "namespace": "kube-system",
 "path": "/validate-webhook",
 "port": 9093
}
```

### Check for the service and the CA bundle of the Mutating webhook

```
kubectl get MutatingWebhookConfiguration aks-osm-webhook-osm -o json | jq '.webhooks[0].clientConfig.service'
```

A well configured Mutating Webhook Configuration would look exactly like this:

```
{
 "name": "osm-injector",
 "namespace": "kube-system",
 "path": "/mutate-pod-creation",
 "port": 9090
}
```

#### Check whether OSM Controller has given the Validating (or Mutating) Webhook a CA Bundle

##### NOTE

As of v0.8.2 It is important to know that AKS RP installs the Validating Webhook, AKS Reconciler ensures it exists, but OSM Controller is the one that fills the CA Bundle.

```
kubectl get ValidatingWebhookConfiguration aks-osm-webhook-osm -o json | jq -r
'.webhooks[0].clientConfig.caBundle' | wc -c
```

```
kubectl get MutatingWebhookConfiguration aks-osm-webhook-osm -o json | jq -r
'.webhooks[0].clientConfig.caBundle' | wc -c
```

1845

This number indicates the number of bytes, or the size of the CA Bundle. If this is empty, 0, or some number under 1000 it would indicate that the CA Bundle is not correctly provisioned. Without a correct CA Bundle, the Validating Webhook would be erroring out and prohibiting the user from making changes to the osm-config ConfigMap in the kube-system namespace.

A sample error when the CA Bundle is incorrect:

- An attempt to change the osm-config ConfigMap:

```
kubectl patch ConfigMap osm-config -n kube-system --type merge --patch '{"data":
{"config_resync_interval":"2m"}}'
```

- Error:

```
Error from server (InternalError): Internal error occurred: failed calling webhook "osm-config-webhook.k8s.io": Post https://osm-config-validator.kube-system.svc:9093/validate-webhook?timeout=30s: x509: certificate signed by unknown authority
```

Work around for when the **Validating** Webhook Configuration has a bad certificate:

- Option 1 - Restart OSM Controller - this will restart the OSM Controller. On start, it will overwrite the CA Bundle of both the Mutating and Validating webhooks.

```
kubectl rollout restart deployment -n kube-system osm-controller
```

- Option 2 - Option 2. Delete the Validating Webhook - removing the Validating Webhook makes mutations of the `osm-config` ConfigMap no longer validated. Any patch will go through. The AKS Reconciler will at some point ensure the Validating Webhook exists and will recreate it. The OSM Controller may have to be restarted to quickly rewrite the CA Bundle.

```
kubectl delete ValidatingWebhookConfiguration aks-osm-webhook-osm
```

- Option 3 - Delete and Patch: The following command will delete the validating webhook, allowing us to add any values, and will immediately try to apply a patch. Most likely the AKS Reconciler will not have enough time to reconcile and restore the Validating Webhook giving us the opportunity to apply a change as a last resort:

```
kubectl delete ValidatingWebhookConfiguration aks-osm-webhook-osm; kubectl patch ConfigMap osm-config -n kube-system --type merge --patch '{"data":{"config_resync_interval":"15s"}}'
```

Check the `osm-config` ConfigMap

#### NOTE

The OSM Controller does not require for the `osm-config` ConfigMap to be present in the kube-system namespace. The controller has reasonable default values for the config and can operate without it.

Check for the existence:

```
kubectl get ConfigMap -n kube-system osm-config
```

Check the content of the `osm-config` ConfigMap

```
kubectl get ConfigMap -n kube-system osm-config -o json | jq '.data'
```

```
{
 "egress": "true",
 "enable_debug_server": "true",
 "enable_privileged_init_container": "false",
 "envoy_log_level": "error",
 "outbound_ip_range_exclusion_list": "169.254.169.254,168.63.129.16,20.193.20.233",
 "permissive_traffic_policy_mode": "true",
 "prometheus_scraping": "false",
 "service_cert_validity_duration": "24h",
 "use_https_ingress": "false"
}
```

`osm-config` ConfigMap values:

KEY	TYPE	ALLOWED VALUES	DEFAULT VALUE	FUNCTION
egress	bool	true, false	"false"	Enables egress in the mesh.
enable_debug_server	bool	true, false	"true"	Enables a debug endpoint on the osm-controller pod to list information regarding the mesh such as proxy connections, certificates, and SMI policies.

KEY	TYPE	ALLOWED VALUES	DEFAULT VALUE	FUNCTION
enable_privileged_init_container	bool	true, false	"false"	Enables privileged init containers for pods in mesh. When false, init containers only have NET_ADMIN.
envoy_log_level	string	trace, debug, info, warning, warn, error, critical, off	"error"	Sets the logging verbosity of Envoy proxy sidecar, only applicable to newly created pods joining the mesh. To update the log level for existing pods, restart the deployment with <code>kubectl rollout restart</code>
outbound_ip_range_exclusion_list	string	comma-separated list of IP ranges of the form a.b.c.d/x	-	Global list of IP address ranges to exclude from outbound traffic interception by the sidecar proxy.
permissive_traffic_policy_mode	bool	true, false	"false"	Setting to <code>true</code> , enables allow-all mode in the mesh i.e. no traffic policy enforcement in the mesh. If set to <code>false</code> , enables deny-all traffic policy in mesh i.e. an <code>SMI Traffic Target</code> is necessary for services to communicate.
prometheus_scraping	bool	true, false	"true"	Enables Prometheus metrics scraping on sidecar proxies.
service_cert_validity_duration	string	24h, 1h30m (any time duration)	"24h"	Sets the service certificate validity duration, represented as a sequence of decimal numbers each with optional fraction and a unit suffix.
tracing_enable	bool	true, false	"false"	Enables Jaeger tracing for the mesh.

KEY	TYPE	ALLOWED VALUES	DEFAULT VALUE	FUNCTION
tracing_address	string	jaeger.mesh-namespace.svc.cluster.local	jaeger.kube-system.svc.cluster.local	Address of the Jaeger deployment, if tracing is enabled.
tracing_endpoint	string	/api/v2/spans	/api/v2/spans	Endpoint for tracing data, if tracing enabled.
tracing_port	int	any non-zero integer value	"9411"	Port on which tracing is enabled.
use_https_ingress	bool	true, false	"false"	Enables HTTPS ingress on the mesh.
config_resync_interval	string	under 1 minute disables this	0 (disabled)	When a value above 1m (60s) is provided, OSM Controller will send all available config to each connected Envoy at the given interval

## Check Namespaces

### NOTE

The kube-system namespace will never participate in a service mesh and will never be labeled and/or annotated with the key/values below.

We use the `osm namespace add` command to join namespaces to a given service mesh. When a k8s namespace is part of the mesh (or for it to be part of the mesh) the following must be true:

View the annotations with

```
kubectl get namespace bookbuyer -o json | jq '.metadata.annotations'
```

The following annotation must be present:

```
{
 "openservicemesh.io/sidecar-injection": "enabled"
}
```

View the labels with

```
kubectl get namespace bookbuyer -o json | jq '.metadata.labels'
```

The following label must be present:

```
{
 "openservicemesh.io/monitored-by": "osm"
}
```

If a namespace is not annotated with `"openservicemesh.io/sidecar-injection": "enabled"` or not labeled with `"openservicemesh.io/monitored-by": "osm"` the OSM Injector will not add Envoy sidecars.

Note: After `osm namespace add` is called only **new** pods will be injected with an Envoy sidecar. Existing pods must be restarted with `kubectl rollout restart deployment ...`

#### Verify the SMI CRDs:

Check whether the cluster has the required CRDs:

```
kubectl get crds
```

We must have the following installed on the cluster:

- `httproulegroups.specs.smi-spec.io`
- `tcproutes.specs.smi-spec.io`
- `trafficsplits.split.smi-spec.io`
- `traffictargets.access.smi-spec.io`
- `udproutes.specs.smi-spec.io`

Get the versions of the CRDs installed with this command:

```
for x in $(kubectl get crds --no-headers | awk '{print $1}' | grep 'smi-spec.io'); do
 kubectl get crd $x -o json | jq -r '(.metadata.name, \"---\" , .spec.versions[].name, "\n")'
done
```

Expected output:

```
httproutegroups.specs.smi-spec.io
```

```

```

```
v1alpha4
v1alpha3
v1alpha2
v1alpha1
```

```
tcproutes.specs.smi-spec.io
```

```

```

```
v1alpha4
v1alpha3
v1alpha2
v1alpha1
```

```
trafficsplits.split.smi-spec.io
```

```

```

```
v1alpha2
```

```
traffictargets.access.smi-spec.io
```

```

```

```
v1alpha3
v1alpha2
v1alpha1
```

```
udproutes.specs.smi-spec.io
```

```

```

```
v1alpha4
v1alpha3
v1alpha2
v1alpha1
```

OSM Controller v0.8.2 requires the following versions:

- traffictargets.access.smi-spec.io - [v1alpha3](#)
- httproutegroups.specs.smi-spec.io - [v1alpha4](#)
- tcproutes.specs.smi-spec.io - [v1alpha4](#)
- udproutes.specs.smi-spec.io - Not supported
- trafficsplits.split.smi-spec.io - [v1alpha2](#)
- \*.metrics.smi-spec.io - [v1alpha1](#)

If CRDs are missing use the following commands to install these on the cluster:

```
kubectl apply -f https://raw.githubusercontent.com/openservicemesh/osm/v0.8.2/charts/osm/crds/access.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/openservicemesh/osm/v0.8.2/charts/osm/crds/specs.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/openservicemesh/osm/v0.8.2/charts/osm/crds/split.yaml
```

## Disable Open Service Mesh (OSM) add-on for your AKS cluster

To disable the OSM add-on, run the following command:

```
az aks disable-addons -n <AKS-cluster-name> -g <AKS-resource-group-name> -a open-service-mesh
```

# Istio

2/25/2020 • 2 minutes to read • [Edit Online](#)

## Overview

Istio is a full featured, customisable, and extensible service mesh.

## Architecture

Istio provides a data plane that is composed of [Envoy](#)-based sidecars. These intelligent proxies control all network traffic in and out of your meshed apps and workloads.

The control plane manages the configuration, policy, and telemetry via the following [components](#):

- **Mixer** - Enforces access control and usage policies. Collects telemetry from the proxies that is pushed into [Prometheus](#).
- **Pilot** - Provides service discovery and traffic management policy/configuration for the proxies.
- **Citadel** - Provides identity and security capabilities that allow for mTLS between services.
- **Galley** - Abstracts and provides configuration to components.

The following architecture diagram demonstrates how the various components within the data plane and control plane interact.



## Selection criteria

It's important to understand and consider the following areas when evaluating Istio for your workloads:

- [Design Goals](#)
- [Capabilities](#)
- [Scenarios](#)

## Design goals

The following design goals [guide](#) the Istio project:

- **Maximize Transparency** - Allow adoption with the minimum amount of work to get real value from the system.
- **Extensibility** - Must be able to grow and adapt with changing needs.
- **Portability** - Run easily in different kinds of environments - cloud, on-premises.
- **Policy Uniformity** - Consistency in policy definition across variety of resources.

## Capabilities

Istio provides the following set of capabilities:

- **Mesh** – gateways (multi-cluster), virtual machines (mesh expansion)
- **Traffic Management** – routing, splitting, timeouts, circuit breakers, retries, ingress, egress
- **Policy** – access control, rate limit, quota, custom policy adapters
- **Security** – authentication (jwt), authorisation, encryption (mTLS), external CA (HashiCorp Vault)
- **Observability** – golden metrics, mirror, tracing, custom adapters, prometheus, grafana

## Scenarios

Istio is well suited to and suggested for the following scenarios:

- Require extensibility and rich set of capabilities
- Mesh expansion to include VM based workloads
- Multi-cluster service mesh

## Next steps

The following documentation describes how you can install Istio on Azure Kubernetes Service (AKS):

### [Install Istio in Azure Kubernetes Service \(AKS\)](#)

You can also further explore Istio concepts and additional deployment models:

- [Istio Concepts](#)
- [Istio Deployment Models](#)

# Install and use Istio in Azure Kubernetes Service (AKS)

3/5/2021 • 10 minutes to read • [Edit Online](#)

Istio is an open-source service mesh that provides a key set of functionality across the microservices in a Kubernetes cluster. These features include traffic management, service identity and security, policy enforcement, and observability. For more information about Istio, see the official [What is Istio?](#) documentation.

This article shows you how to install Istio. The Istio `istioctl` client binary is installed onto your client machine and the Istio components are installed into a Kubernetes cluster on AKS.

## NOTE

The following instructions reference Istio version `1.7.3`.

The Istio `1.7.x` releases have been tested by the Istio team against Kubernetes version `1.16+`. You can find additional Istio versions at [GitHub - Istio Releases](#), information about each of the releases at [Istio News](#) and supported Kubernetes versions at [Istio General FAQ](#).

In this article, you learn how to:

- Download and install the Istio `istioctl` client binary
- Install Istio on AKS
- Validate the Istio installation
- Access the add-ons
- Uninstall Istio from AKS

## Before you begin

The steps detailed in this article assume that you've created an AKS cluster (Kubernetes `1.16` and above, with Kubernetes RBAC enabled) and have established a `kubectl` connection with the cluster. If you need help with any of these items, then see the [AKS quickstart](#).

Make sure that you have read the [Istio Performance and Scalability](#) documentation to understand the additional resource requirements for running Istio in your AKS cluster. The core and memory requirements will vary based on your specific workload. Choose an appropriate number of nodes and VM size to cater for your setup.

This article separates the Istio installation guidance into several discrete steps. The end result is the same in structure as the official Istio installation [guidance](#).

## Download and install the Istio `istioctl` client binary

In a bash-based shell on Linux or [Windows Subsystem for Linux](#), use `curl` to download the Istio release and then extract with `tar` as follows:

```
Specify the Istio version that will be leveraged throughout these instructions
ISTIO_VERSION=1.7.3

curl -sL "https://github.com/istio/istio/releases/download/$ISTIO_VERSION/istioctl-$ISTIO_VERSION-linux-amd64.tar.gz" | tar xz
```

The `istioctl` client binary runs on your client machine and allows you to install and manage Istio in your AKS cluster. Use the following commands to install the Istio `istioctl` client binary in a bash-based shell on Linux or [Windows Subsystem for Linux](#). These commands copy the `istioctl` client binary to the standard user program location in your `PATH`.

```
sudo mv ./istioctl /usr/local/bin/istioctl
sudo chmod +x /usr/local/bin/istioctl
```

If you'd like command-line completion for the Istio `istioctl` client binary, then set it up as follows:

```
Generate the bash completion file and source it in your current shell
mkdir -p ~/completions && istioctl collateral --bash -o ~/completions
source ~/completions/istioctl.bash

Source the bash completion file in your .bashrc so that the command-line completions
are permanently available in your shell
echo "source ~/completions/istioctl.bash" >> ~/.bashrc
```

## Download and install the Istio istioctl client binary

In a bash-based shell on macOS, use `curl` to download the Istio release and then extract with `tar` as follows:

```
Specify the Istio version that will be leveraged throughout these instructions
ISTIO_VERSION=1.7.3

curl -sL "https://github.com/istio/istio/releases/download/$ISTIO_VERSION/istioctl-$ISTIO_VERSION-
osx.tar.gz" | tar xz
```

The `istioctl` client binary runs on your client machine and allows you to install and manage Istio in your AKS cluster. Use the following commands to install the Istio `istioctl` client binary in a bash-based shell on macOS. These commands copy the `istioctl` client binary to the standard user program location in your `PATH`.

```
sudo mv ./istioctl /usr/local/bin/istioctl
sudo chmod +x /usr/local/bin/istioctl
```

If you'd like command-line completion for the Istio `istioctl` client binary, then set it up as follows:

```
Generate the bash completion file and source it in your current shell
mkdir -p ~/completions && istioctl collateral --bash -o ~/completions
source ~/completions/istioctl.bash

Source the bash completion file in your .bashrc so that the command-line completions
are permanently available in your shell
echo "source ~/completions/istioctl.bash" >> ~/.bashrc
```

## Download and install the Istio istioctl client binary

In a PowerShell-based shell on Windows, use `Invoke-WebRequest` to download the Istio release and then extract with `Expand-Archive` as follows:

```
Specify the Istio version that will be leveraged throughout these instructions
$ISTIO_VERSION="1.7.3"

[Net.ServicePointManager]::SecurityProtocol = "tls12"
$ProgressPreference = 'SilentlyContinue'; Invoke-WebRequest -URI
"https://github.com/istio/istio/releases/download/$ISTIO_VERSION/istioctl-$ISTIO_VERSION-win.zip" -OutFile
"istioctl-$ISTIO_VERSION.zip"
Expand-Archive -Path "istioctl-$ISTIO_VERSION.zip" -DestinationPath .
```

The `istioctl` client binary runs on your client machine and allows you to install and manage Istio in your AKS cluster. Use the following commands to install the Istio `istioctl` client binary in a PowerShell-based shell on Windows. These commands copy the `istioctl` client binary to an Istio folder and then make it available both immediately (in current shell) and permanently (across shell restarts) via your `PATH`. You don't need elevated (Admin) privileges to run these commands and you don't need to restart your shell.

```
Copy istioctl.exe to C:\Istio
New-Item -ItemType Directory -Force -Path "C:\Istio"
Move-Item -Path .\istioctl.exe -Destination "C:\Istio\"

Add C:\Istio to PATH.
Make the new PATH permanently available for the current User
$USER_PATH = [environment]::GetEnvironmentVariable("PATH", "User") + ";C:\Istio\
[environment]::SetEnvironmentVariable("PATH", $USER_PATH, "User")
Make the new PATH immediately available in the current shell
$env:PATH += ";C:\Istio\"
```

## Install the Istio Operator on AKS

Istio provides an [Operator](#) to manage installation and updates to the Istio components within your AKS cluster. We'll install the Istio Operator using the `istioctl` client binary.

```
istioctl operator init
```

You should see something like the following output to confirm that the Istio Operator has been installed.

```
Using operator Deployment image: docker.io/istio/operator:1.7.3
✓ Istio operator installed
✓ Installation complete
```

The Istio Operator is installed into the `istio-operator` namespace. Query the namespace.

```
kubectl get all -n istio-operator
```

You should see the following components deployed.

NAME	READY	STATUS	RESTARTS	AGE	
pod/istio-operator-6d7958b7bf-wxgdc	1/1	Running	0	2m43s	
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/istio-operator	ClusterIP	10.0.8.57	<none>	8383/TCP	2m43s
NAME	READY	UP-TO-DATE	AVAILABLE	AGE	
deployment.apps/istio-operator	1/1	1	1	2m43s	
NAME	DESIRED	CURRENT	READY	AGE	
replicaset.apps/istio-operator-6d7958b7bf	1	1	1	2m43s	

You can learn more about the Operator pattern and how it can help automate complex tasks via [kubernetes.io](#).

## Install Istio components

Now that we've successfully installed the Istio Operator in our AKS cluster, it's time to install the Istio components.

We will leverage the `default` Istio Configuration Profile to build the [Istio Operator Spec](#).

You can run the following `istioctl` command to view the configuration for the `default` Istio Configuration Profile.

```
istioctl profile dump default
```

### NOTE

Istio currently must be scheduled to run on Linux nodes. If you have Windows Server nodes in your cluster, you must ensure that the Istio pods are only scheduled to run on Linux nodes. We'll use [node selectors](#) to make sure pods are scheduled to the correct nodes.

### Caution

The [Istio CNI](#) Istio features are currently in [Alpha](#), so thought should be given before enabling these.

Create a file called `istio.aks.yaml` with the following content. This file will hold the [Istio Operator Spec](#) for configuring Istio.

```

apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
metadata:
 namespace: istio-system
 name: istio-control-plane
spec:
 # Use the default profile as the base
 # More details at: https://istio.io/docs/setup/additional-setup/config-profiles/
 profile: default
 # Enable the addons that we will want to use
 addonComponents:
 grafana:
 enabled: true
 prometheus:
 enabled: true
 tracing:
 enabled: true
 kiali:
 enabled: true
 values:
 global:
 # Ensure that the Istio pods are only scheduled to run on Linux nodes
 defaultNodeSelector:
 beta.kubernetes.io/os: linux
 kiali:
 dashboard:
 auth:
 strategy: anonymous

```

Create the `istio-system` namespace and deploy the Istio Operator Spec to that namespace. The Istio Operator will be watching for the Istio Operator Spec and will use it to install and configure Istio in your AKS cluster.

```

kubectl create ns istio-system

kubectl apply -f istio.aks.yaml

```

At this point, you've deployed Istio to your AKS cluster. To ensure that we have a successful deployment of Istio, let's move on to the next section to [Validate the Istio installation](#).

## Validate the Istio installation

Query the `istio-system` namespace, where the Istio and add-on components were installed by the Istio Operator:

```

kubectl get all -n istio-system

```

You should see the following components:

- `istio*` - the Istio components
- `jaeger-*`, `tracing`, and `zipkin` - tracing addon
- `prometheus` - metrics addon
- `grafana` - analytics and monitoring dashboard addon
- `kiali` - service mesh dashboard addon

NAME	READY	STATUS	RESTARTS	AGE
pod/grafana-7cf9794c74-mpfbp	1/1	Running	0	5m53s
pod/istio-ingressgateway-86b5dbdcb9-ndrp5	1/1	Running	0	5m57s
pod/istio-tracing-c98f4b8fc-zqklg	1/1	Running	0	82s
pod/istiod-6965c56995-4ph9h	1/1	Running	0	6m15s
pod/kiali-7b44985d68-p87zh	1/1	Running	0	81s
pod/prometheus-6868989549-5ghzz	1/1	Running	0	81s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
service/grafana	ClusterIP	10.0.226.39	<none>	3000/TCP
service/istio-ingressgateway	LoadBalancer	10.0.143.56	20.53.72.254 15021:32166/TCP, 80:31684/TCP, 443:31302/TCP, 15443:30863/TCP	5m57s
service/istiod	ClusterIP	10.0.211.228	<none>	15010/TCP, 15012/TCP, 443/TCP, 15014/TCP, 853/TCP 6m16s
service/jaeger-agent	ClusterIP	None	<none>	5775/UDP, 6831/UDP, 6832/UDP 82s
service/jaeger-collector	ClusterIP	10.0.7.62	<none>	14267/TCP, 14268/TCP, 14250/TCP 82s
service/jaeger-collector-headless	ClusterIP	None	<none>	14250/TCP 82s
service/jaeger-query	ClusterIP	10.0.52.172	<none>	16686/TCP 82s
service/kiali	ClusterIP	10.0.71.179	<none>	20001/TCP 82s
service/prometheus	ClusterIP	10.0.171.151	<none>	9090/TCP 82s
service/tracing	ClusterIP	10.0.195.137	<none>	80/TCP 82s
service/zipkin	ClusterIP	10.0.136.111	<none>	9411/TCP 82s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/grafana	1/1	1	1	5m54s
deployment.apps/istio-ingressgateway	1/1	1	1	5m58s
deployment.apps/istio-tracing	1/1	1	1	83s
deployment.apps/istiod	1/1	1	1	6m16s
deployment.apps/kiali	1/1	1	1	83s
deployment.apps/prometheus	1/1	1	1	82s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/grafana-7cf9794c74	1	1	1	5m54s
replicaset.apps/istio-ingressgateway-86b5dbdcb9	1	1	1	5m58s
replicaset.apps/istio-tracing-c98f4b8fc	1	1	1	83s
replicaset.apps/istiod-6965c56995	1	1	1	6m16s
replicaset.apps/kiali-7b44985d68	1	1	1	82s
replicaset.apps/prometheus-6868989549	1	1	1	82s

NAME	REFERENCE	TARGETS
MINPODS MAXPODS REPLICAS AGE		
horizontalpodautoscaler.autoscaling/istio-ingressgateway 5 1 5m57s	Deployment/istio-ingressgateway	7%/80% 1
horizontalpodautoscaler.autoscaling/istiod 5 1 6m16s	Deployment/istiod	1%/80% 1

You can also gain additional insight into the installation by watching the logs for the Istio Operator.

```
kubectl logs -n istio-operator -l name=istio-operator -f
```

If the `istio-ingressgateway` shows an external ip of `<pending>`, wait a few minutes until an IP address has been assigned by Azure networking.

All of the pods should show a status of `Running`. If your pods don't have these statuses, wait a minute or two

until they do. If any pods report an issue, use the `kubectl describe pod` command to review their output and status.

## Accessing the add-ons

A number of add-ons were installed by the Istio Operator that provide additional functionality. The web applications for the add-ons are **not** exposed publicly via an external ip address.

To access the add-on user interfaces, use the `istioctl dashboard` command. This command uses `kubectl port-forward` and a random port to create a secure connection between your client machine and the relevant pod in your AKS cluster. It will then automatically open the add-on web application in your default browser.

### Grafana

The analytics and monitoring dashboards for Istio are provided by [Grafana](#). Remember to use the credentials you created via the Grafana secret earlier when prompted. Open the Grafana dashboard securely as follows:

```
istioctl dashboard grafana
```

### Prometheus

Metrics for Istio are provided by [Prometheus](#). Open the Prometheus dashboard securely as follows:

```
istioctl dashboard prometheus
```

### Jaeger

Tracing within Istio is provided by [Jaeger](#). Open the Jaeger dashboard securely as follows:

```
istioctl dashboard jaeger
```

### Kiali

A service mesh observability dashboard is provided by [Kiali](#). Remember to use the credentials you created via the Kiali secret earlier when prompted. Open the Kiali dashboard securely as follows:

```
istioctl dashboard kiali
```

### Envoy

A simple interface to the [Envoy](#) proxies is available. It provides configuration information and metrics for an Envoy proxy running in a specified pod. Open the Envoy interface securely as follows:

```
istioctl dashboard envoy <pod-name>.<namespace>
```

## Uninstall Istio from AKS

### WARNING

Deleting Istio from a running system may result in traffic related issues between your services. Ensure that you have made provisions for your system to still operate correctly without Istio before proceeding.

### Remove Istio

To remove Istio from your AKS cluster, delete the `IstioOperator` resource named `istio-control-plane` that we

added earlier. The Istio Operator will recognize that the Istio Operator Spec has been removed, and then delete all the associated Istio components.

```
kubectl delete istiooperator istio-control-plane -n istio-system
```

You can run the following to check when all the Istio components have been deleted.

```
kubectl get all -n istio-system
```

## Remove Istio Operator

Once Istio has been successfully uninstalled, you can also remove the Istio Operator.

```
istioctl operator remove
```

And then finally, remove the `istio-` namespaces.

```
kubectl delete ns istio-system
kubectl delete ns istio-operator
```

## Next steps

To explore more installation and configuration options for Istio, see the following official Istio guidance:

- [Istio - installation guides](#)

You can also follow additional scenarios using:

- [Istio Bookinfo Application example](#)

# Use intelligent routing and canary releases with Istio in Azure Kubernetes Service (AKS)

3/5/2021 • 15 minutes to read • [Edit Online](#)

Istio is an open-source service mesh that provides a key set of functionality across the microservices in a Kubernetes cluster. These features include traffic management, service identity and security, policy enforcement, and observability. For more information about Istio, see the official [What is Istio?](#) documentation.

This article shows you how to use the traffic management functionality of Istio. A sample AKS voting app is used to explore intelligent routing and canary releases.

In this article, you learn how to:

- Deploy the application
- Update the application
- Roll out a canary release of the application
- Finalize the rollout

## Before you begin

### NOTE

This scenario has been tested against Istio version [1.3.2](#).

The steps detailed in this article assume you've created an AKS cluster (Kubernetes [1.13](#) and above, with Kubernetes RBAC enabled) and have established a `kubectl` connection with the cluster. You'll also need Istio installed in your cluster.

If you need help with any of these items, then see the [AKS quickstart](#) and [Install Istio in AKS](#) guidance.

## About this application scenario

The sample AKS voting app provides two voting options (**Cats** or **Dogs**) to users. There is a storage component that persists the number of votes for each option. Additionally, there is an analytics component that provides details around the votes cast for each option.

In this application scenario, you start by deploying version [1.0](#) of the voting app and version [1.0](#) of the analytics component. The analytics component provides simple counts for the number of votes. The voting app and analytics component interact with version [1.0](#) of the storage component, which is backed by Redis.

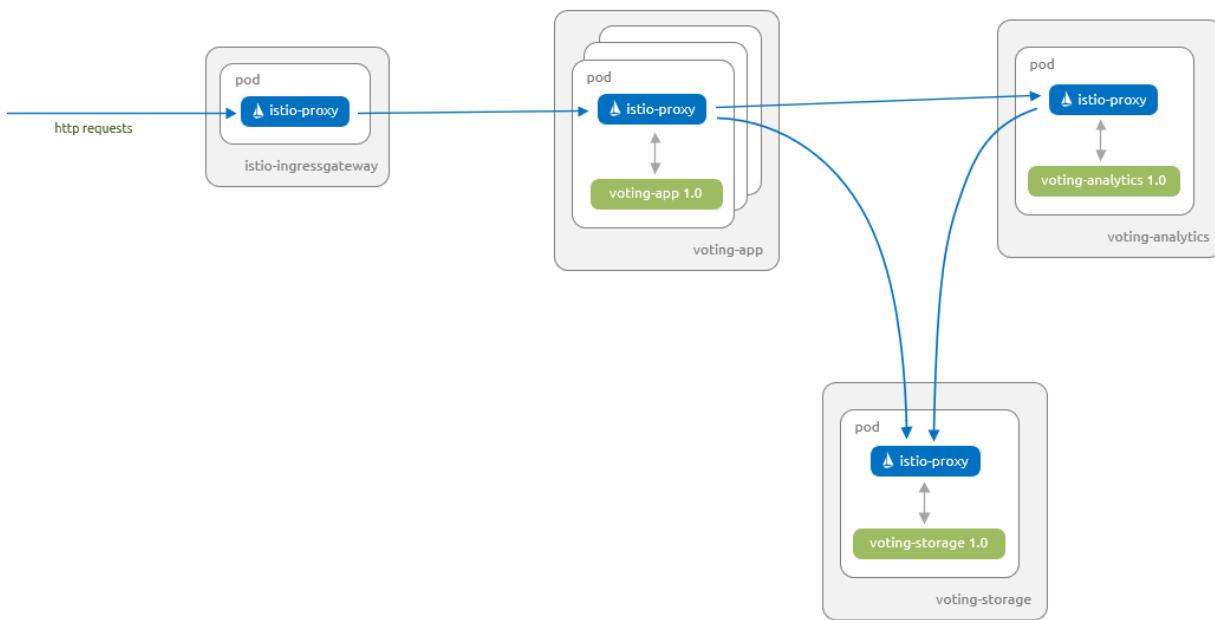
You upgrade the analytics component to version [1.1](#), which provides counts, and now totals and percentages.

A subset of users test version [2.0](#) of the app via a canary release. This new version uses a storage component that is backed by a MySQL database.

Once you're confident that version [2.0](#) works as expected on your subset of users, you roll out version [2.0](#) to all your users.

## Deploy the application

Let's start by deploying the application into your Azure Kubernetes Service (AKS) cluster. The following diagram shows what runs by the end of this section - version 1.0 of all components with inbound requests serviced via the Istio ingress gateway:



The artifacts you need to follow along with this article are available in the [Azure-Samples/aks-voting-app](https://github.com/Azure-Samples/aks-voting-app) GitHub repo. You can either download the artifacts or clone the repo as follows:

```
git clone https://github.com/Azure-Samples/aks-voting-app.git
```

Change to the following folder in the downloaded / cloned repo and run all subsequent steps from this folder:

```
cd aks-voting-app/scenarios/intelligent-routing-with-istio
```

First, create a namespace in your AKS cluster for the sample AKS voting app named `voting` as follows:

```
kubectl create namespace voting
```

Label the namespace with `istio-injection=enabled`. This label instructs Istio to automatically inject the Istio proxies as sidecars into all of your pods in this namespace.

```
kubectl label namespace voting istio-injection=enabled
```

Now let's create the components for the AKS Voting app. Create these components in the `voting` namespace created in a previous step.

```
kubectl apply -f kubernetes/step-1-create-voting-app.yaml --namespace voting
```

The following example output shows the resources being created:

```
deployment.apps/voting-storage-1-0 created
service/voting-storage created
deployment.apps/voting-analytics-1-0 created
service/voting-analytics created
deployment.apps/voting-app-1-0 created
service/voting-app created
```

#### NOTE

Istio has some specific requirements around pods and services. For more information, see the [Istio Requirements for Pods and Services documentation](#).

To see the pods that have been created, use the `kubectl get pods` command as follows:

```
kubectl get pods -n voting --show-labels
```

The following example output shows there are three instances of the `voting-app` pod and a single instance of both the `voting-analytics` and `voting-storage` pods. Each of the pods has two containers. One of these containers is the component, and the other is the `istio-proxy`:

NAME	READY	STATUS	RESTARTS	AGE	LABELS
voting-analytics-1-0-57c7fccb44-ng7dl template-hash=57c7fccb44,version=1.0	2/2	Running	0	39s	app=voting-analytics,pod-
voting-app-1-0-956756fd-d5w7z hash=956756fd,version=1.0	2/2	Running	0	39s	app=voting-app,pod-template-
voting-app-1-0-956756fd-f6h69 hash=956756fd,version=1.0	2/2	Running	0	39s	app=voting-app,pod-template-
voting-app-1-0-956756fd-wsxvt hash=956756fd,version=1.0	2/2	Running	0	39s	app=voting-app,pod-template-
voting-storage-1-0-5d8fcc89c4-2jhms template-hash=5d8fcc89c4,version=1.0	2/2	Running	0	39s	app=voting-storage,pod-

To see information about the pod, we'll use the `kubectl describe pod` command with label selectors to select the `voting-analytics` pod. We'll filter the output to show the details of the two containers present in the pod:

```
kubectl describe pod -l "app=voting-analytics, version=1.0" -n voting | egrep "istio-proxy:|voting-analytics:" -A2
```

The `istio-proxy` container has automatically been injected by Istio to manage the network traffic to and from your components, as shown in the following example output:

```
voting-analytics:
 Container ID: docker://35efa1f31d95ca737ff2e2229ab8fe7d9f2f8a39ac11366008f31287be4cea4d
 Image: mcr.microsoft.com/aks/samples/voting/analytics:1.0
 --
 istio-proxy:
 Container ID: docker://1fa4eb43e8d4f375058c23cc062084f91c0863015e58eb377276b20c809d43c6
 Image: docker.io/istio/proxyv2:1.3.2
```

```
kubectl describe pod -l "app=voting-analytics, version=1.0" -n voting | egrep "istio-proxy:|voting-analytics:" -A2
```

The `istio-proxy` container has automatically been injected by Istio to manage the network traffic to and from

your components, as shown in the following example output:

```
voting-analytics:
 Container ID: docker://35efa1f31d95ca737ff2e2229ab8fe7d9f2f8a39ac11366008f31287be4cea4d
 Image: mcr.microsoft.com/aks/samples/voting/analytics:1.0
--
istio-proxy:
 Container ID: docker://1fa4eb43e8d4f375058c23cc062084f91c0863015e58eb377276b20c809d43c6
 Image: docker.io/istio/proxyv2:1.3.2
```

```
kubectl describe pod -l "app=voting-analytics, version=1.0" -n voting | Select-String -Pattern "istio-proxy:|voting-analytics:" -Context 0,2
```

The `istio-proxy` container has automatically been injected by Istio to manage the network traffic to and from your components, as shown in the following example output:

```
> voting-analytics:
 Container ID: docker://35efa1f31d95ca737ff2e2229ab8fe7d9f2f8a39ac11366008f31287be4cea4d
 Image: mcr.microsoft.com/aks/samples/voting/analytics:1.0
> istio-proxy:
 Container ID: docker://1fa4eb43e8d4f375058c23cc062084f91c0863015e58eb377276b20c809d43c6
 Image: docker.io/istio/proxyv2:1.3.2
```

You can't connect to the voting app until you create the Istio [Gateway](#) and [Virtual Service](#). These Istio resources route traffic from the default Istio ingress gateway to our application.

#### NOTE

A **Gateway** is a component at the edge of the service mesh that receives inbound or outbound HTTP and TCP traffic.

A **Virtual Service** defines a set of routing rules for one or more destination services.

Use the `kubectl apply` command to deploy the Gateway and Virtual Service yaml. Remember to specify the namespace that these resources are deployed into.

```
kubectl apply -f istio/step-1-create-voting-app-gateway.yaml --namespace voting
```

The following example output shows the new Gateway and Virtual Service being created:

```
virtualservice.networking.istio.io/voting-app created
gateway.networking.istio.io/voting-app-gateway created
```

Obtain the IP address of the Istio Ingress Gateway using the following command:

```
kubectl get service istio-ingressgateway --namespace istio-system -o
jsonpath='{.status.loadBalancer.ingress[0].ip}'
```

The following example output shows the IP address of the Ingress Gateway:

```
20.188.211.19
```

Open up a browser and paste in the IP address. The sample AKS voting app is displayed.



The information at the bottom of the screen shows that the app uses version `1.0` of `voting-app` and version `1.0` of `voting-storage` (Redis).

## Update the application

Let's deploy a new version of the analytics component. This new version `1.1` displays totals and percentages in addition to the count for each category.

The following diagram shows what will be running at the end of this section - only version `1.1` of our `voting-analytics` component has traffic routed from the `voting-app` component. Even though version `1.0` of our `voting-analytics` component continues to run and is referenced by the `voting-analytics` service, the Istio proxies disallow traffic to and from it.



Let's deploy version `1.1` of the `voting-analytics` component. Create this component in the `voting` namespace:

```
kubectl apply -f kubernetes/step-2-update-voting-analytics-to-1.1.yaml --namespace voting
```

The following example output shows the resources being created:

```
deployment.apps/voting-analytics-1-1 created
```

Open the sample AKS voting app in a browser again, using the IP address of the Istio Ingress Gateway obtained in the previous step.

Your browser alternates between the two views shown below. Since you are using a Kubernetes [Service](#) for the `voting-analytics` component with only a single label selector (`app: voting-analytics`), Kubernetes uses the default behavior of round-robin between the pods that match that selector. In this case, it is both version `1.0` and `1.1` of your `voting-analytics` pods.



You can visualize the switching between the two versions of the `voting-analytics` component as follows.  
Remember to use the IP address of your own Istio Ingress Gateway.

```
INGRESS_IP=20.188.211.19
for i in {1..5}; do curl -si $INGRESS_IP | grep results; done
```

```
INGRESS_IP=20.188.211.19
for i in {1..5}; do curl -si $INGRESS_IP | grep results; done
```

```
$INGRESS_IP="20.188.211.19"
(1..5) |% { (Invoke-WebRequest -Uri $INGRESS_IP).Content.Split("`n") | Select-String -Pattern "results" }
```

The following example output shows the relevant part of the returned web site as the site switches between versions:

```
<div id="results"> Cats: 2 | Dogs: 4 </div>
<div id="results"> Cats: 2 | Dogs: 4 </div>
<div id="results"> Cats: 2/6 (33%) | Dogs: 4/6 (67%) </div>
<div id="results"> Cats: 2 | Dogs: 4 </div>
<div id="results"> Cats: 2/6 (33%) | Dogs: 4/6 (67%) </div>
```

### Lock down traffic to version 1.1 of the application

Now let's lock down traffic to only version `1.1` of the `voting-analytics` component and to version `1.0` of the `voting-storage` component. You then define routing rules for all of the other components.

- A **Virtual Service** defines a set of routing rules for one or more destination services.
- A **Destination Rule** defines traffic policies and version specific policies.
- A **Policy** defines what authentication methods can be accepted on workload(s).

Use the `kubectl apply` command to replace the Virtual Service definition on your `voting-app` and add **Destination Rules** and **Virtual Services** for the other components. You will add a **Policy** to the `voting` namespace to ensure that all communication between services is secured using mutual TLS and client certificates.

- The Policy has `peers.mtls.mode` set to `STRICT` to ensure that mutual TLS is enforced between your services within the `voting` namespace.
- We also set the `trafficPolicy.tls.mode` to `ISTIO_MUTUAL` in all our Destination Rules. Istio provides services with strong identities and secures communications between services using mutual TLS and client certificates that Istio transparently manages.

```
kubectl apply -f istio/step-2-update-and-add-routing-for-all-components.yaml --namespace voting
```

The following example output shows the new Policy, Destination Rules, and Virtual Services being updated/created:

```
virtualservice.networking.istio.io/voting-app configured
policy.authentication.istio.io/default created
destinationrule.networking.istio.io/voting-app created
destinationrule.networking.istio.io/voting-analytics created
virtualservice.networking.istio.io/voting-analytics created
destinationrule.networking.istio.io/voting-storage created
virtualservice.networking.istio.io/voting-storage created
```

If you open the AKS Voting app in a browser again, only the new version `1.1` of the `voting-analytics` component is used by the `voting-app` component.



You can visualize that you are now only routed to version `1.1` of your `voting-analytics` component as follows. Remember to use the IP address of your own Istio Ingress Gateway:

```
INGRESS_IP=20.188.211.19
for i in {1..5}; do curl -si $INGRESS_IP | grep results; done
```

```
INGRESS_IP=20.188.211.19
for i in {1..5}; do curl -si $INGRESS_IP | grep results; done
```

```
$INGRESS_IP="20.188.211.19"
(1..5) |% { (Invoke-WebRequest -Uri $INGRESS_IP).Content.Split("`n") | Select-String -Pattern "results" }
```

The following example output shows the relevant part of the returned web site:

```
<div id="results"> Cats: 2/6 (33%) | Dogs: 4/6 (67%) </div>
<div id="results"> Cats: 2/6 (33%) | Dogs: 4/6 (67%) </div>
<div id="results"> Cats: 2/6 (33%) | Dogs: 4/6 (67%) </div>
<div id="results"> Cats: 2/6 (33%) | Dogs: 4/6 (67%) </div>
<div id="results"> Cats: 2/6 (33%) | Dogs: 4/6 (67%) </div>
```

Let's now confirm that Istio is using mutual TLS to secure communications between each of our services. For this we will use the `authn tls-check` command on the `istioctl` client binary, which takes the following form.

```
istioctl authn tls-check <pod-name[.namespace]> [<service>]
```

This set of commands provide information about the access to the specified services, from all pods that are in a namespace and match a set of labels:

```

mTLS configuration between each of the istio ingress pods and the voting-app service
kubectl get pod -n istio-system -l app=istio-ingressgateway | grep Running | cut -d ' ' -f1 | xargs -n1 -I{} istioctl authn tls-check {}.istio-system voting-app.voting.svc.cluster.local

mTLS configuration between each of the voting-app pods and the voting-analytics service
kubectl get pod -n voting -l app=voting-app | grep Running | cut -d ' ' -f1 | xargs -n1 -I{} istioctl authn tls-check {}.voting voting-analytics.voting.svc.cluster.local

mTLS configuration between each of the voting-app pods and the voting-storage service
kubectl get pod -n voting -l app=voting-app | grep Running | cut -d ' ' -f1 | xargs -n1 -I{} istioctl authn tls-check {}.voting voting-storage.voting.svc.cluster.local

mTLS configuration between each of the voting-analytics version 1.1 pods and the voting-storage service
kubectl get pod -n voting -l app=voting-analytics,version=1.1 | grep Running | cut -d ' ' -f1 | xargs -n1 -I{} istioctl authn tls-check {}.voting voting-storage.voting.svc.cluster.local

```

```

mTLS configuration between each of the istio ingress pods and the voting-app service
kubectl get pod -n istio-system -l app=istio-ingressgateway | grep Running | cut -d ' ' -f1 | xargs -n1 -I{} istioctl authn tls-check {}.istio-system voting-app.voting.svc.cluster.local

mTLS configuration between each of the voting-app pods and the voting-analytics service
kubectl get pod -n voting -l app=voting-app | grep Running | cut -d ' ' -f1 | xargs -n1 -I{} istioctl authn tls-check {}.voting voting-analytics.voting.svc.cluster.local

mTLS configuration between each of the voting-app pods and the voting-storage service
kubectl get pod -n voting -l app=voting-app | grep Running | cut -d ' ' -f1 | xargs -n1 -I{} istioctl authn tls-check {}.voting voting-storage.voting.svc.cluster.local

mTLS configuration between each of the voting-analytics version 1.1 pods and the voting-storage service
kubectl get pod -n voting -l app=voting-analytics,version=1.1 | grep Running | cut -d ' ' -f1 | xargs -n1 -I{} istioctl authn tls-check {}.voting voting-storage.voting.svc.cluster.local

```

```

mTLS configuration between each of the istio ingress pods and the voting-app service
(kubectl get pod -n istio-system -l app=istio-ingressgateway | Select-String -Pattern "Running").Line |% { $_.Split()[0] |% { istioctl authn tls-check $($_.Split()[0]) voting-app.voting.svc.cluster.local } }

mTLS configuration between each of the voting-app pods and the voting-analytics service
(kubectl get pod -n voting -l app=voting-app | Select-String -Pattern "Running").Line |% { $_.Split()[0] |% { istioctl authn tls-check $($_.Split()[0]) voting-analytics.voting.svc.cluster.local } }

mTLS configuration between each of the voting-app pods and the voting-storage service
(kubectl get pod -n voting -l app=voting-app | Select-String -Pattern "Running").Line |% { $_.Split()[0] |% { istioctl authn tls-check $($_.Split()[0]) voting-storage.voting.svc.cluster.local } }

mTLS configuration between each of the voting-analytics version 1.1 pods and the voting-storage service
(kubectl get pod -n voting -l app=voting-analytics,version=1.1 | Select-String -Pattern "Running").Line |% { $_.Split()[0] |% { istioctl authn tls-check $($_.Split()[0]) voting-storage.voting.svc.cluster.local } }

```

This following example output shows that mutual TLS is enforced for each of our queries above. The output also shows the Policy and Destination Rules that enforces the mutual TLS:

```

mTLS configuration between istio ingress pods and the voting-app service
HOST:PORT STATUS SERVER CLIENT AUTHN POLICY DESTINATION
RULE
voting-app.voting.svc.cluster.local:8080 OK mTLS mTLS default/voting voting-
app/voting

mTLS configuration between each of the voting-app pods and the voting-analytics service
HOST:PORT STATUS SERVER CLIENT AUTHN POLICY
DESTINATION RULE
voting-analytics.voting.svc.cluster.local:8080 OK mTLS mTLS default/voting
voting-analytics/voting
HOST:PORT STATUS SERVER CLIENT AUTHN POLICY
DESTINATION RULE
voting-analytics.voting.svc.cluster.local:8080 OK mTLS mTLS default/voting
voting-analytics/voting
HOST:PORT STATUS SERVER CLIENT AUTHN POLICY
DESTINATION RULE
voting-analytics.voting.svc.cluster.local:8080 OK mTLS mTLS default/voting
voting-analytics/voting

mTLS configuration between each of the voting-app pods and the voting-storage service
HOST:PORT STATUS SERVER CLIENT AUTHN POLICY
DESTINATION RULE
voting-storage.voting.svc.cluster.local:6379 OK mTLS mTLS default/voting voting-
storage/voting
HOST:PORT STATUS SERVER CLIENT AUTHN POLICY
DESTINATION RULE
voting-storage.voting.svc.cluster.local:6379 OK mTLS mTLS default/voting voting-
storage/voting
HOST:PORT STATUS SERVER CLIENT AUTHN POLICY
DESTINATION RULE
voting-storage.voting.svc.cluster.local:6379 OK mTLS mTLS default/voting voting-
storage/voting

mTLS configuration between each of the voting-analytics version 1.1 pods and the voting-storage service
HOST:PORT STATUS SERVER CLIENT AUTHN POLICY
DESTINATION RULE
voting-storage.voting.svc.cluster.local:6379 OK mTLS mTLS default/voting voting-
storage/voting

```

## Roll out a canary release of the application

Now let's deploy a new version 2.0 of the voting-app, voting-analytics, and voting-storage components. The new voting-storage component uses MySQL instead of Redis, and the voting-app and voting-analytics components are updated to allow them to use this new voting-storage component.

The voting-app component now supports feature flag functionality. This feature flag allows you to test the canary release capability of Istio for a subset of users.

The following diagram shows what you will have running at the end of this section.

- Version 1.0 of the voting-app component, version 1.1 of the voting-analytics component and version 1.0 of the voting-storage component are able to communicate with each other.
- Version 2.0 of the voting-app component, version 2.0 of the voting-analytics component and version 2.0 of the voting-storage component are able to communicate with each other.
- Version 2.0 of the voting-app component are only accessible to users that have a specific feature flag set. This change is managed using a feature flag via a cookie.



First, update the Istio Destination Rules and Virtual Services to cater for these new components. These updates ensure that you don't route traffic incorrectly to the new components and users don't get unexpected access:

```
kubectl apply -f istio/step-3-add-routing-for-2.0-components.yaml --namespace voting
```

The following example output shows the Destination Rules and Virtual Services being updated:

```
destinationrule.networking.istio.io/voting-app configured
virtualservice.networking.istio.io/voting-app configured
destinationrule.networking.istio.io/voting-analytics configured
virtualservice.networking.istio.io/voting-analytics configured
destinationrule.networking.istio.io/voting-storage configured
virtualservice.networking.istio.io/voting-storage configured
```

Next, let's add the Kubernetes objects for the new version `2.0` components. You also update the `voting-storage` service to include the `3306` port for MySQL:

```
kubectl apply -f kubernetes/step-3-update-voting-app-with-new-storage.yaml --namespace voting
```

The following example output shows the Kubernetes objects are successfully updated or created:

```
service/voting-storage configured
secret/voting-storage-secret created
deployment.apps/voting-storage-2-0 created
persistentvolumeclaim/mysql-pv-claim created
deployment.apps/voting-analytics-2-0 created
deployment.apps/voting-app-2-0 created
```

Wait until all the version `2.0` pods are running. Use the `kubectl get pods` command with the `-w` watch switch

to watch for changes on all pods in the `voting` namespace:

```
kubectl get pods --namespace voting -w
```

You should now be able to switch between the version `1.0` and version `2.0` (canary) of the voting application. The feature flag toggle at the bottom of the screen sets a cookie. This cookie is used by the `voting-app` Virtual Service to route users to the new version `2.0`.





The vote counts are different between the versions of the app. This difference highlights that you are using two different storage backends.

## Finalize the rollout

Once you've successfully tested the canary release, update the `voting-app` Virtual Service to route all traffic to version `2.0` of the `voting-app` component. All users then see version `2.0` of the application, regardless of whether the feature flag is set or not:



Update all the Destination Rules to remove the versions of the components you no longer want active. Then, update all the Virtual Services to stop referencing those versions.

Since there's no longer any traffic to any of the older versions of the components, you can now safely delete all the deployments for those components.



You have now successfully rolled out a new version of the AKS Voting App.

## Clean up

You can remove the AKS voting app we used in this scenario from your AKS cluster by deleting the `voting` namespace as follows:

```
kubectl delete namespace voting
```

The following example output shows that all the components of the AKS voting app have been removed from your AKS cluster.

```
namespace "voting" deleted
```

## Next steps

You can explore additional scenarios using the [Istio Bookinfo Application example](#).

# Linkerd

2/25/2020 • 2 minutes to read • [Edit Online](#)

## Overview

Linkerd is an easy to use and lightweight service mesh.

## Architecture

Linkerd provides a data plane that is composed of ultralight Linkerd specialised proxy sidecars. These intelligent proxies control all network traffic in and out of your meshed apps and workloads. The proxies also expose metrics via Prometheus metrics endpoints.

The control plane manages the configuration and aggregated telemetry via the following components:

- **Controller** - Provides api that drives the Linkerd CLI and Dashboard. Provides configuration for proxies.
- **Tap** - Establish real-time watches on requests and responses.
- **Identity** - Provides identity and security capabilities that allow for mTLS between services.
- **Web** - Provides the Linkerd dashboard.

The following architecture diagram demonstrates how the various components within the data plane and control plane interact.



## Selection criteria

It's important to understand and consider the following areas when evaluating Linkerd for your workloads:

- [Design Principles](#)
- [Capabilities](#)
- [Scenarios](#)

## Design principles

The following design principles [guide](#) the Linkerd project:

- **Keep it Simple** - Must be easy to use and understand.
- **Minimize Resource Requirements** - Impose minimal performance and resource cost.
- **Just Work** - Don't break existing applications and don't require complex configuration.

## Capabilities

Linkerd provides the following set of capabilities:

- **Mesh** – built in debugging option
- **Traffic Management** – splitting, timeouts, retries, ingress
- **Security** – encryption (mTLS), certificates autorotated every 24 hours
- **Observability** – golden metrics, tap, tracing, service profiles and per route metrics, web dashboard with topology graphs, prometheus, grafana

## Scenarios

Linkerd is well suited to and suggested for the following scenarios:

- Simple to use with just the essential set of capability requirements
- Low latency, low overhead, with focus on observability and simple traffic management

## Next steps

The following documentation describes how you can install Linkerd on Azure Kubernetes Service (AKS):

[Install Linkerd in Azure Kubernetes Service \(AKS\)](#)

You can also further explore Linkerd features and architecture:

- [Linkerd Features](#)
- [Linkerd Architecture](#)

# Install Linkerd in Azure Kubernetes Service (AKS)

4/27/2021 • 8 minutes to read • [Edit Online](#)

Linkerd is an open-source service mesh and [CNCF incubating project](#). Linkerd is an ultralight service mesh that provides features that include traffic management, service identity and security, reliability, and observability. For more information about Linkerd, see the official [Linkerd FAQ](#) and [Linkerd Architecture](#) documentation.

This article shows you how to install Linkerd. The Linkerd `linkerd` client binary is installed onto your client machine and the Linkerd components are installed into a Kubernetes cluster on AKS.

## NOTE

These instructions reference Linkerd version `stable-2.6.0`.

The Linkerd `stable-2.6.x` can be run against Kubernetes versions `1.13+`. You can find additional stable and edge Linkerd versions at [GitHub - Linkerd Releases](#).

In this article, you learn how to:

- Download and install the Linkerd `linkerd` client binary
- Install Linkerd on AKS
- Validate the Linkerd installation
- Access the Dashboard
- Uninstall Linkerd from AKS

## Before you begin

The steps detailed in this article assume that you've created an AKS cluster (Kubernetes `1.13` and above, with Kubernetes RBAC enabled) and have established a `kubectl` connection with the cluster. If you need help with any of these items, then see the [AKS quickstart](#).

All Linkerd pods must be scheduled to run on Linux nodes - this setup is the default in the installation method detailed below and requires no additional configuration.

This article separates the Linkerd installation guidance into several discrete steps. The result is the same in structure as the official Linkerd getting started [guidance](#).

## Download and install the Linkerd `linkerd` client binary

In a bash-based shell on Linux or [Windows Subsystem for Linux](#), use `curl` to download the Linkerd release as follows:

```
Specify the Linkerd version that will be leveraged throughout these instructions
LINKERD_VERSION=stable-2.6.0

curl -sLO "https://github.com/linkerd/linkerd2/releases/download/$LINKERD_VERSION/linkerd2-cli-$LINKERD_VERSION-linux"
```

The `linkerd` client binary runs on your client machine and allows you to interact with the Linkerd service mesh. Use the following commands to install the Linkerd `linkerd` client binary in a bash-based shell on Linux or [Windows Subsystem for Linux](#). These commands copy the `linkerd` client binary to the standard user program

location in your `PATH`.

```
sudo cp ./linkerd2-cli-$LINKERD_VERSION-linux /usr/local/bin/linkerd
sudo chmod +x /usr/local/bin/linkerd
```

If you'd like command-line completion for the Linkerd `linkerd` client binary, then set it up as follows:

```
Generate the bash completion file and source it in your current shell
mkdir -p ~/completions && linkerd completion bash > ~/completions/linkerd.bash
source ~/completions/linkerd.bash

Source the bash completion file in your .bashrc so that the command-line completions
are permanently available in your shell
echo "source ~/completions/linkerd.bash" >> ~/.bashrc
```

## Download and install the Linkerd linkerd client binary

In a bash-based shell on MacOS, use `curl` to download the Linkerd release as follows:

```
Specify the Linkerd version that will be leveraged throughout these instructions
LINKERD_VERSION=stable-2.6.0

curl -sLO "https://github.com/linkerd/linkerd2/releases/download/$LINKERD_VERSION/linkerd2-cli-
$LINKERD_VERSION-darwin"
```

The `linkerd` client binary runs on your client machine and allows you to interact with the Linkerd service mesh. Use the following commands to install the Linkerd `linkerd` client binary in a bash-based shell on MacOS. These commands copy the `linkerd` client binary to the standard user program location in your `PATH`.

```
sudo cp ./linkerd2-cli-$LINKERD_VERSION-darwin /usr/local/bin/linkerd
sudo chmod +x /usr/local/bin/linkerd
```

If you'd like command-line completion for the Linkerd `linkerd` client binary, then set it up as follows:

```
Generate the bash completion file and source it in your current shell
mkdir -p ~/completions && linkerd completion bash > ~/completions/linkerd.bash
source ~/completions/linkerd.bash

Source the bash completion file in your .bashrc so that the command-line completions
are permanently available in your shell
echo "source ~/completions/linkerd.bash" >> ~/.bashrc
```

## Download and install the Linkerd linkerd client binary

In a PowerShell-based shell on Windows, use `Invoke-WebRequest` to download the Linkerd release as follows:

```
Specify the Linkerd version that will be leveraged throughout these instructions
$LINKERD_VERSION="stable-2.6.0"

Enforce TLS 1.2
[Net.ServicePointManager]::SecurityProtocol = "tls12"
$ProgressPreference = 'SilentlyContinue'; Invoke-WebRequest -URI
"https://github.com/linkerd/linkerd2/releases/download/$LINKERD_VERSION/linkerd2-cli-$LINKERD_VERSION-
windows.exe" -OutFile "linkerd2-cli-$LINKERD_VERSION-windows.exe"
```

The `linkerd` client binary runs on your client machine and allows you to interact with the Linkerd service mesh. Use the following commands to install the Linkerd `linkerd` client binary in a PowerShell-based shell on Windows. These commands copy the `linkerd` client binary to a Linkerd folder and then make it available both immediately (in current shell) and permanently (across shell restarts) via your `PATH`. You don't need elevated (Admin) privileges to run these commands and you don't need to restart your shell.

```
Copy linkerd.exe to C:\Linkerd
New-Item -ItemType Directory -Force -Path "C:\Linkerd"
Copy-Item -Path ".\linkerd2-cli-$LINKERD_VERSION-windows.exe" -Destination "C:\Linkerd\linkerd.exe"

Add C:\Linkerd to PATH.
Make the new PATH permanently available for the current User
$USER_PATH = [environment]::GetEnvironmentVariable("PATH", "User") + ";C:\Linkerd\
[environment]::SetEnvironmentVariable("PATH", $USER_PATH, "User")
Make the new PATH immediately available in the current shell
$env:PATH += ";C:\Linkerd\"
```

## Install Linkerd on AKS

Before we install Linkerd, we'll run pre-installation checks to determine if the control plane can be installed on our AKS cluster:

```
linkerd check --pre
```

You should see something like the following to indicate that your AKS cluster is a valid installation target for Linkerd:

```
kubernetes-api

✓ can initialize the client
✓ can query the Kubernetes API

kubernetes-version

✓ is running the minimum Kubernetes API version
✓ is running the minimum kubectl version

pre-kubernetes-setup

✓ control plane namespace does not already exist
✓ can create Namespaces
✓ can create ClusterRoles
✓ can create ClusterRoleBindings
✓ can create CustomResourceDefinitions
✓ can create PodSecurityPolicies
✓ can create ServiceAccounts
✓ can create Services
✓ can create Deployments
✓ can create CronJobs
✓ can create ConfigMaps
✓ no clock skew detected

pre-kubernetes-capability

✓ has NET_ADMIN capability
✓ has NET_RAW capability

pre-linkerd-global-resources

✓ no ClusterRoles exist
✓ no ClusterRoleBindings exist
✓ no CustomResourceDefinitions exist
✓ no MutatingWebhookConfigurations exist
✓ no ValidatingWebhookConfigurations exist
✓ no PodSecurityPolicies exist

linkerd-version

✓ can determine the latest version
✓ cli is up-to-date

Status check results are ✓
```

Now it's time to install the Linkerd components. Use the `linkerd` and `kubectl` binaries to install the Linkerd components into your AKS cluster. A `linkerd` namespace will be automatically created, and the components will be installed into this namespace.

```
linkerd install | kubectl apply -f -
```

Linkerd deploys a number of objects. You'll see the list from the output of your `linkerd install` command above. The deployment of the Linkerd components should take around 1 minute to complete, depending on your cluster environment.

At this point, you've deployed Linkerd to your AKS cluster. To ensure we have a successful deployment of Linkerd, let's move on to the next section to [Validate the Linkerd installation](#).

## Validate the Linkerd installation

Confirm that the resources have been successfully created. Use the `kubectl get svc` and `kubectl get pod`

commands to query the `linkerd` namespace, where the Linkerd components were installed by the `linkerd install` command:

```
kubectl get svc --namespace linkerd --output wide
kubectl get pod --namespace linkerd --output wide
```

The following example output shows the services and pods (scheduled on Linux nodes) that should now be running:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR	
linkerd-controller-api	ClusterIP	10.0.110.67	<none>	8085/TCP	66s		
linkerd.io/control-plane-component=controller							
linkerd-destination	ClusterIP	10.0.224.29	<none>	8086/TCP	66s		
linkerd.io/control-plane-component=controller							
linkerd-dst	ClusterIP	10.0.225.148	<none>	8086/TCP	66s		
linkerd.io/control-plane-component=destination							
linkerd-grafana	ClusterIP	10.0.61.124	<none>	3000/TCP	65s		
linkerd.io/control-plane-component=grafana							
linkerd-identity	ClusterIP	10.0.6.104	<none>	8080/TCP	67s		
linkerd.io/control-plane-component=identity							
linkerd-prometheus	ClusterIP	10.0.27.168	<none>	9090/TCP	65s		
linkerd.io/control-plane-component=prometheus							
linkerd-proxy-injector	ClusterIP	10.0.100.133	<none>	443/TCP	64s		
linkerd.io/control-plane-component=proxy-injector							
linkerd-sp-validator	ClusterIP	10.0.221.5	<none>	443/TCP	64s		
linkerd.io/control-plane-component=sp-validator							
linkerd-tap	ClusterIP	10.0.18.14	<none>	8088/TCP,443/TCP	64s		
linkerd.io/control-plane-component=tap							
linkerd-web	ClusterIP	10.0.37.108	<none>	8084/TCP,9994/TCP	66s		
linkerd.io/control-plane-component=web							
NAME		READY	STATUS	RESTARTS	AGE	IP	
NOMINATED NODE	READINESS GATES					NODE	
linkerd-controller-66ddc9f94f-cm9kt	16165125-vmss000001	3/3	Running	0	66s	10.240.0.50	aks-linux-
<none>	<none>						
linkerd-destination-c94bc454-qpkng	16165125-vmss000002	2/2	Running	0	66s	10.240.0.78	aks-linux-
<none>	<none>						
linkerd-grafana-6868fdcb66-4cmq2	16165125-vmss000002	2/2	Running	0	65s	10.240.0.69	aks-linux-
<none>	<none>						
linkerd-identity-74d8df4b85-tqq8f	16165125-vmss000001	2/2	Running	0	66s	10.240.0.48	aks-linux-
<none>	<none>						
linkerd-prometheus-699587cf8-k8ghg	16165125-vmss000001	2/2	Running	0	65s	10.240.0.41	aks-linux-
<none>	<none>						
linkerd-proxy-injector-6556447f64-n29wr	16165125-vmss000000	2/2	Running	0	64s	10.240.0.32	aks-linux-
<none>	<none>						
linkerd-sp-validator-56745cd567-v4x7h	16165125-vmss000000	2/2	Running	0	64s	10.240.0.6	aks-linux-
<none>	<none>						
linkerd-tap-5cd9fc566-ct988	16165125-vmss000000	2/2	Running	0	64s	10.240.0.15	aks-linux-
<none>	<none>						
linkerd-web-774c79b6d5-dhhwf	16165125-vmss000002	2/2	Running	0	65s	10.240.0.70	aks-linux-
<none>	<none>						

Linkerd provides a command via the `linkerd` client binary to validate that the Linkerd control plane was successfully installed and configured.

```
linkerd check
```

You should see something like the following to indicate that your installation was successful:

```

kubernetes-api

✓ can initialize the client
✓ can query the Kubernetes API

kubernetes-version

✓ is running the minimum Kubernetes API version
✓ is running the minimum kubectl version

linkerd-config

✓ control plane Namespace exists
✓ control plane ClusterRoles exist
✓ control plane ClusterRoleBindings exist
✓ control plane ServiceAccounts exist
✓ control plane CustomResourceDefinitions exist
✓ control plane MutatingWebhookConfigurations exist
✓ control plane ValidatingWebhookConfigurations exist
✓ control plane PodSecurityPolicies exist

linkerd-existence

✓ 'linkerd-config' config map exists
✓ heartbeat ServiceAccount exist
✓ control plane replica sets are ready
✓ no unschedulable pods
✓ controller pod is running
✓ can initialize the client
✓ can query the control plane API

linkerd-api

✓ control plane pods are ready
✓ control plane self-check
✓ [kubernetes] control plane can talk to Kubernetes
✓ [prometheus] control plane can talk to Prometheus
✓ no invalid service profiles

linkerd-version

✓ can determine the latest version
✓ cli is up-to-date

control-plane-version

✓ control plane is up-to-date
✓ control plane and cli versions match

Status check results are ✓

```

## Access the dashboard

Linkerd comes with a dashboard that provides insight into the service mesh and workloads. To access the dashboard, use the `linkerd dashboard` command. This command leverages [kubectl port-forward](#) to create a secure connection between your client machine and the relevant pods in your AKS cluster. It will then automatically open the dashboard in your default browser.

```
linkerd dashboard
```

The command will also create a port-forward and return a link for the Grafana dashboards.

```
Linkerd dashboard available at:
http://127.0.0.1:50750
Grafana dashboard available at:
http://127.0.0.1:50750/grafana
Opening Linkerd dashboard in the default browser
```

## Uninstall Linkerd from AKS

### WARNING

Deleting Linkerd from a running system may result in traffic related issues between your services. Ensure that you have made provisions for your system to still operate correctly without Linkerd before proceeding.

First you'll need to remove the data plane proxies. Remove any Automatic Proxy Injection [annotations](#) from workload namespaces and roll out your workload deployments. Your workloads should no longer have any associated data plane components.

Finally, remove the control plane as follows:

```
linkerd install --ignore-cluster | kubectl delete -f -
```

## Next steps

To explore more installation and configuration options for Linkerd, see the following official Linkerd guidance:

- [Linkerd - Helm installation](#)
- [Linkerd - Multi-stage installation to cater for role privileges](#)

You can also follow additional scenarios using:

- [Linkerd emojivoto demo](#)
- [Linkerd books demo](#)

# Consul

5/21/2020 • 2 minutes to read • [Edit Online](#)

## Overview

Consul is a multi data centre aware service networking solution to connect and secure services across runtime platforms. Connect is the component that provides service mesh capabilities.

## Architecture

Consul provides a data plane that is composed of Envoy-based sidecars by default. Consul has a pluggable proxy architecture. These intelligent proxies control all network traffic in and out of your meshed apps and workloads.

The control plane manages the configuration, and policy via the following components:

- **Server** - A Consul Agent running in Server mode that maintains Consul cluster state.
- **Client** - A Consul Agent running in lightweight Client Mode. Each compute node must have a Client agent running. This client brokers configuration and policy between the workloads and the Consul configuration.

The following architecture diagram demonstrates how the various components within the data plane and control plane interact.



## Selection criteria

It's important to understand and consider the following areas when evaluating Consul for your workloads:

- [Consul Principles](#)
- [Capabilities](#)
- [Scenarios](#)

## Consul principles

The following principles [guide](#) the Consul project:

- **API-Driven** - Codify all configuration and policy.
- **Run and Connect Anywhere** - Connect workloads across runtime platforms (Kubernetes, VMs, Serverless).
- **Extend and Integrate** - Securely connect workloads across infrastructure.

## Capabilities

Consul provides the following set of capabilities:

- **Mesh** – gateway (multi data centre), virtual machines (out of cluster nodes), service sync, built in debugging option
- **Proxies** – Envoy, built-in proxy, pluggable, l4 proxy available for Windows workloads
- **Traffic Management** – routing, splitting, resolution
- **Policy** – intentions, ACLs
- **Security** – authorisation, authentication, encryption, SPIFFE-based identities, external CA (Vault), certificate management, and rotation
- **Observability** – metrics, ui dashboard, prometheus, grafana

## Scenarios

Consul is well suited to and suggested for the following scenarios:

- Extending existing Consul connected workloads
- Compliance requirements around certificate management
- Multi cluster service mesh
- VM-based workloads to be included in the service mesh

## Next steps

The following documentation describes how you can install Consul on Azure Kubernetes Service (AKS):

### [Install Consul in Azure Kubernetes Service \(AKS\)](#)

You can also further explore Consul features and architecture:

- [Consul Getting Started Tutorials](#)
- [Consul Features](#)
- [Consul Architecture](#)
- [Consul - How Connect Works](#)

# Install and use Consul in Azure Kubernetes Service (AKS)

3/5/2021 • 5 minutes to read • [Edit Online](#)

Consul is an open-source service mesh that provides a key set of functionality across the microservices in a Kubernetes cluster. These features include service discovery, health checking, service segmentation, and observability. For more information about Consul, see the official [What is Consul?](#) documentation.

This article shows you how to install Consul. The Consul components are installed into a Kubernetes cluster on AKS.

## NOTE

These instructions reference Consul version 1.6.0, and use at least Helm version 2.14.2.

The Consul 1.6.x releases can be run against Kubernetes versions 1.13+. You can find additional Consul versions at [GitHub - Consul Releases](#) and information about each of the releases at [Consul- Release Notes](#).

In this article, you learn how to:

- Install the Consul components on AKS
- Validate the Consul installation
- Uninstall Consul from AKS

## Before you begin

The steps detailed in this article assume that you've created an AKS cluster (Kubernetes 1.13 and above, with Kubernetes RBAC enabled) and have established a kubectl connection with the cluster. If you need help with any of these items, then see the [AKS quickstart](#). Ensure that your cluster has at least 3 nodes in the Linux node pool.

You'll need [Helm](#) to follow these instructions and install Consul. It's recommended that you have the latest stable version correctly installed and configured in your cluster. If you need help with installing Helm, then see the [AKS Helm installation guidance](#). All Consul pods must also be scheduled to run on Linux nodes.

This article separates the Consul installation guidance into several discrete steps. The end result is the same in structure as the official Consul installation [guidance](#).

### Install the Consul components on AKS

We'll start by downloading version v0.10.0 of the Consul Helm chart. This version of the chart includes Consul version 1.6.0.

In a bash-based shell on Linux, [Windows Subsystem for Linux](#) or MacOS, use curl to download the Consul Helm chart release as follows:

```
Specify the Consul Helm chart version that will be leveraged throughout these instructions
CONSUL_HELM_VERSION=0.10.0
```

```
curl -sL "https://github.com/hashicorp/consul-helm/archive/v$CONSUL_HELM_VERSION.tar.gz" | tar xz
mv consul-helm-$CONSUL_HELM_VERSION consul-helm
```

In a bash-based shell on Linux, Windows Subsystem for Linux or MacOS, use `curl` to download the Consul Helm chart release as follows:

```
Specify the Consul Helm chart version that will be leveraged throughout these instructions
CONSUL_HELM_VERSION=0.10.0

curl -sL "https://github.com/hashicorp/consul-helm/archive/v$CONSUL_HELM_VERSION.tar.gz" | tar xz
mv consul-helm-$CONSUL_HELM_VERSION consul-helm
```

In a PowerShell-based shell on Windows, use `Invoke-WebRequest` to download the Consul Helm chart release and then extract with `Expand-Archive` as follows:

```
Specify the Consul Helm chart version that will be leveraged throughout these instructions
$CONSUL_HELM_VERSION="0.10.0"

Enforce TLS 1.2
[Net.ServicePointManager]::SecurityProtocol = "tls12"
$ProgressPreference = 'SilentlyContinue'; Invoke-WebRequest -URI "https://github.com/hashicorp/consul-helm/archive/v$CONSUL_HELM_VERSION.zip" -OutFile "consul-helm-$CONSUL_HELM_VERSION.zip"
Expand-Archive -Path "consul-helm-$CONSUL_HELM_VERSION.zip" -DestinationPath .
Move-Item -Path consul-helm-$CONSUL_HELM_VERSION -Destination consul-helm
```

Use Helm and the downloaded `consul-helm` chart to install the Consul components into the `consul` namespace in your AKS cluster.

## NOTE

### Installation options

We are using the following options as part of our installation:

- `connectInject.enabled=true` - enable proxies to be injected into pods
- `client.enabled=true` - enable Consul clients to run on every node
- `client.grpc=true` - enable gRPC listener for connectInject
- `syncCatalog.enabled=true` - sync Kubernetes and Consul services

### Node selectors

Consul currently must be scheduled to run on Linux nodes. If you have Windows Server nodes in your cluster, you must ensure that the Consul pods are only scheduled to run on Linux nodes. We'll use node selectors to make sure pods are scheduled to the correct nodes.

```
helm install -f consul-helm/values.yaml --name consul --namespace consul ./consul-helm \
--set connectInject.enabled=true --set connectInject.nodeSelector="beta.kubernetes.io/os: linux" \
--set client.enabled=true --set client.grpc=true --set client.nodeSelector="beta.kubernetes.io/os: linux" \
\
--set server.nodeSelector="beta.kubernetes.io/os: linux" \
--set syncCatalog.enabled=true --set syncCatalog.nodeSelector="beta.kubernetes.io/os: linux"
```

```
helm install -f consul-helm/values.yaml --name consul --namespace consul ./consul-helm \
--set connectInject.enabled=true --set connectInject.nodeSelector="beta.kubernetes.io/os: linux" \
--set client.enabled=true --set client.grpc=true --set client.nodeSelector="beta.kubernetes.io/os: linux" \
\
--set server.nodeSelector="beta.kubernetes.io/os: linux" \
--set syncCatalog.enabled=true --set syncCatalog.nodeSelector="beta.kubernetes.io/os: linux"
```

```
helm install -f consul-helm/values.yaml --name consul --namespace consul ./consul-helm \
--set connectInject.enabled=true --set connectInject.nodeSelector="beta.kubernetes.io/os: linux" \
--set client.enabled=true --set client.grpc=true --set client.nodeSelector="beta.kubernetes.io/os: linux" \
--set server.nodeSelector="beta.kubernetes.io/os: linux" \
--set syncCatalog.enabled=true --set syncCatalog.nodeSelector="beta.kubernetes.io/os: linux"
```

The `consul` Helm chart deploys a number of objects. You can see the list from the output of your `helm install` command above. The deployment of the Consul components can take around 3 minutes to complete, depending on your cluster environment.

At this point, you've deployed Consul to your AKS cluster. To ensure that we have a successful deployment of Consul, let's move on to the next section to validate the Consul installation.

## Validate the Consul installation

Confirm that the resources have been successfully created. Use the `kubectl get svc` and `kubectl get pod` commands to query the `consul` namespace, where the Consul components were installed by the `helm install` command:

```
kubectl get svc --namespace consul --output wide
kubectl get pod --namespace consul --output wide
```

The following example output shows the services and pods (scheduled on Linux nodes) that should now be running:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)			
AGE	SELECTOR	ExternalName	<none>	consul.service.consul	<none>		
38s	<none>						
3m26s	consul-consul-connect-injector-svc	ClusterIP	10.0.98.102	<none>		443/TCP	
3m26s	app=consul,component=connect-injector,release=consul						
3m26s	consul-consul-dns	ClusterIP	10.0.46.194	<none>		53/TCP,53/UDP	
3m26s	app=consul,hasDNS=true,release=consul						
3m26s	consul-consul-server	ClusterIP	None	<none>			
8500/TCP,8301/TCP,8301/UDP,8302/TCP,8302/UDP,8300/TCP,8600/TCP,8600/UDP	app=consul,component=server,release=consul			3m26s			
3m26s	consul-consul-ui	ClusterIP	10.0.50.188	<none>		80/TCP	
3m26s	app=consul,component=server,release=consul						
NAME	NODE	NOMINATED NODE	READY	STATUS	RESTARTS	AGE	IP
NAME	NODE	NOMINATED NODE	READINESS	GATES			
consul-consul-connect-injector-webhook-deployment-99f74fdbcr5zj	10.240.0.68	aks-linux-92468653-vmss000002	1/1	<none>	Running	0	3m9s
consul-consul-jbksc	10.240.0.44	aks-linux-92468653-vmss000001	1/1	<none>	Running	0	3m9s
consul-consul-jkwtq	10.240.0.70	aks-linux-92468653-vmss000002	1/1	<none>	Running	0	3m9s
consul-consul-server-0	10.240.0.91	aks-linux-92468653-vmss000002	1/1	<none>	Running	0	3m9s
consul-consul-server-1	10.240.0.38	aks-linux-92468653-vmss000001	1/1	<none>	Running	0	3m9s
consul-consul-server-2	10.240.0.10	aks-linux-92468653-vmss000000	1/1	<none>	Running	0	3m9s
consul-consul-sync-catalog-d846b79c-8ssr8	10.240.0.94	aks-linux-92468653-vmss000002	1/1	<none>	Running	2	3m9s
consul-consul-tz2t5	10.240.0.12	aks-linux-92468653-vmss000000	1/1	<none>	Running	0	3m9s

All of the pods should show a status of `Running`. If your pods don't have these statuses, wait a minute or two

until they do. If any pods report an issue, use the `kubectl describe pod` command to review their output and status.

## Accessing the Consul UI

The Consul UI was installed in our setup above and provides UI based configuration for Consul. The UI for Consul is not exposed publicly via an external ip address. To access the Consul user interface, use the `kubectl port-forward` command. This command creates a secure connection between your client machine and the relevant pod in your AKS cluster.

```
kubectl port-forward -n consul svc/consul-consul-ui 8080:80
```

You can now open a browser and point it to `http://localhost:8080/ui` to open the Consul UI. You should see the following when you open the UI:

The screenshot shows the Consul UI's Services page. At the top, there are navigation tabs: dc1, Services (which is selected), Nodes, Key/Value, ACL, and Intentions. Below the tabs, a search bar contains the placeholder text "service:name tag:name status:critical search-term". The main table lists six services:

Service	Type	Health Checks ⓘ	Tags
consul		3	
consul-consul-connect-injector-svc		0	k8s
consul-consul-dns		0	k8s
consul-consul-server		0	k8s
consul-consul-ui		0	k8s
kubernetes		0	k8s

## Uninstall Consul from AKS

### WARNING

Deleting Consul from a running system may result in traffic related issues between your services. Ensure that you have made provisions for your system to still operate correctly without Consul before proceeding.

### Remove Consul components and namespace

To remove Consul from your AKS cluster, use the following commands. The `helm delete` commands will remove the `consul` chart, and the `kubectl delete namespace` command will remove the `consul` namespace.

```
helm delete --purge consul
kubectl delete namespace consul
```

## Next steps

To explore more installation and configuration options for Consul, see the following official Consul articles:

- [Consul - Helm installation guide](#)
- [Consul - Helm installation options](#)

You can also follow additional scenarios using:

- [Consul Example Application](#)
- [Consul Kubernetes Reference Architecture](#)
- [Consul Mesh Gateways](#)

# Deploy ASP.NET Core apps to Azure Kubernetes Service with Azure DevOps Starter

3/5/2021 • 7 minutes to read • [Edit Online](#)

Azure DevOps Starter presents a simplified experience where you can bring your existing code and Git repo or choose a sample application to create a continuous integration (CI) and continuous delivery (CD) pipeline to Azure.

DevOps Starter also:

- Automatically creates Azure resources, such as Azure Kubernetes Service (AKS).
- Creates and configures a release pipeline in Azure DevOps that sets up a build and release pipeline for CI/CD.
- Creates an Azure Application Insights resource for monitoring.
- Enables [Azure Monitor for containers](#) to monitor performance for the container workloads on the AKS cluster

In this tutorial, you will:

- Use DevOps Starter to deploy an ASP.NET Core app to AKS
- Configure Azure DevOps and an Azure subscription
- Examine the AKS cluster
- Examine the CI pipeline
- Examine the CD pipeline
- Commit changes to Git and automatically deploy them to Azure
- Clean up resources

## Prerequisites

- An Azure subscription. You can get one free through [Visual Studio Dev Essentials](#).

## Use DevOps Starter to deploy an ASP.NET Core app to AKS

DevOps Starter creates a CI/CD pipeline in Azure Pipelines. You can create a new Azure DevOps organization or use an existing organization. DevOps Starter also creates Azure resources, such as an AKS cluster, in the Azure subscription of your choice.

1. Sign in to the [Azure portal](#).
2. In the search box, type **DevOps Starter**, and then select. Click on **Add** to create a new one.



3. Select .NET, and then select **Next**.
4. Under **Choose an application framework**, select **ASP.NET Core** and then select **Next**.
5. Select **Kubernetes Service**, and then select **Next**.

## Configure Azure DevOps and an Azure subscription

1. Create a new Azure DevOps organization, or select an existing organization.
2. Enter a name for your Azure DevOps project.
3. Select your Azure subscription.
4. To view additional Azure configuration settings and to identify the number of nodes for the AKS cluster, select **Change**. This pane displays various options for configuring the type and location of Azure services.
5. Exit the Azure configuration area, and then select **Done**. After a few minutes, the process is completed. A sample ASP.NET Core app is set up in a Git repo in your Azure DevOps organization, an AKS cluster is created, a CI/CD pipeline is executed, and your app is deployed to Azure.

After all this is completed, the Azure DevOps Starter dashboard is displayed in the Azure portal. You can also go to the DevOps Starter dashboard directly from **All resources** in the Azure portal.

This dashboard provides visibility into your Azure DevOps code repository, your CI/CD pipeline, and your AKS cluster. You can configure additional CI/CD options in your Azure DevOps pipeline. At the right, select **Browse** to view your running app.

## Examine the AKS cluster

DevOps Starter automatically configures an AKS cluster, which you can explore and customize. To familiarize yourself with the AKS cluster, do the following:

1. Go to the DevOps Starter dashboard.
2. At the right, select the AKS service. A pane opens for the AKS cluster. From this view you can perform various actions, such as monitoring container health, searching logs, and opening the Kubernetes dashboard.
3. At the right, select **View Kubernetes dashboard**. Optionally, follow the steps to open the Kubernetes dashboard.

## Examine the CI pipeline

DevOps Starter automatically configures a CI/CD pipeline in your Azure DevOps organization. You can explore and customize the pipeline. To familiarize yourself with it, do the following:

1. Go to the DevOps Starter dashboard.
2. At the top of the DevOps Starter dashboard, select **Build Pipelines**. A browser tab displays the build pipeline for your new project.
3. Point to the **Status** field, and then select the ellipsis (...). A menu displays several options, such as queueing a new build, pausing a build, and editing the build pipeline.
4. Select **Edit**.
5. In this pane, you can examine the various tasks for your build pipeline. The build performs various tasks, such as fetching sources from the Git repo, restoring dependencies, and publishing outputs used for deployments.
6. At the top of the build pipeline, select the build pipeline name.
7. Change the name of your build pipeline to something more descriptive, select **Save & queue**, and then select **Save**.
8. Under your build pipeline name, select **History**. This pane displays an audit trail of your recent changes for the build. Azure DevOps keeps track of any changes made to the build pipeline, and it allows you to compare versions.
9. Select **Triggers**. DevOps Starter automatically creates a CI trigger, and every commit to the repo starts a new build. Optionally, you can choose to include or exclude branches from the CI process.
10. Select **Retention**. Depending on your scenario, you can specify policies to keep or remove a certain number of builds.

## Examine the CD release pipeline

DevOps Starter automatically creates and configures the necessary steps to deploy from your Azure DevOps organization to your Azure subscription. These steps include configuring an Azure service connection to authenticate Azure DevOps to your Azure subscription. The automation also creates a release pipeline, which provides the CD to Azure. To learn more about the release pipeline, do the following:

1. Select **Build and Release**, and then select **Releases**. DevOps Starter creates a release pipeline to manage deployments to Azure.
2. Select the ellipsis (...) next to your release pipeline, and then select **Edit**. The release pipeline contains a *pipeline*, which defines the release process.
3. Under **Artifacts**, select **Drop**. The build pipeline you examined in the previous steps produces the output that's used for the artifact.
4. At the right of the **Drop** icon, select **Continuous deployment trigger**. This release pipeline has an enabled CD trigger, which executes a deployment every time a new build artifact is available. Optionally, you can disable the trigger so that your deployments require manual execution.
5. At the right, select **View releases** to display a history of releases.
6. Select the ellipsis (...) next to a release, and then select **Open**. You can explore several menus, such as a release summary, associated work items, and tests.
7. Select **Commits**. This view shows code commits that are associated with this deployment. Compare

releases to view the commit differences between deployments.

8. Select **Logs**. The logs contain useful information about the deployment process. You can view them both during and after deployments.

## Commit changes to Azure Repos and automatically deploy them to Azure

### NOTE

The following procedure tests the CI/CD pipeline by making a simple text change.

You're now ready to collaborate with a team on your app by using a CI/CD process that automatically deploys your latest work to your website. Each change to the Git repo starts a build in Azure DevOps, and a CD pipeline executes a deployment to Azure. Follow the procedure in this section, or use another technique to commit changes to your repo. For example, you can clone the Git repo in your favorite tool or IDE, and then push changes to this repo.

1. In the Azure DevOps menu, select **Code > Files**, and then go to your repo.
2. Go to the *Views\Home* directory, select the ellipsis (...) next to the *Index.cshtml* file, and then select **Edit**.
3. Make a change to the file, such as adding some text within one of the div tags.
4. At the top right, select **Commit**, and then select **Commit** again to push your change. After a few moments, a build starts in Azure DevOps and a release executes to deploy the changes. Monitor the build status on the DevOps Starter dashboard or in the browser with your Azure DevOps organization.
5. After the release is completed, refresh your app to verify your changes.

## Clean up resources

If you are testing, you can avoid accruing billing charges by cleaning up your resources. When they are no longer needed, you can delete the AKS cluster and related resources that you created in this tutorial. To do so, use the **Delete** functionality on the DevOps Starter dashboard.

### IMPORTANT

The following procedure permanently deletes resources. The **Delete** functionality destroys the data that's created by the project in DevOps Starter in both Azure and Azure DevOps, and you will be unable to retrieve it. Use this procedure only after you've carefully read the prompts.

1. In the Azure portal, go to the DevOps Starter dashboard.
2. At the top right, select **Delete**.
3. At the prompt, select **Yes** to *permanently delete* the resources.

## Next steps

You can optionally modify these build and release pipelines to meet the needs of your team. You can also use this CI/CD pattern as a template for your other pipelines. In this tutorial, you learned how to:

- Use DevOps Starter to deploy an ASP.NET Core app to AKS
- Configure Azure DevOps and an Azure subscription
- Examine the AKS cluster

- Examine the CI pipeline
- Examine the CD pipeline
- Commit changes to Git and automatically deploy them to Azure
- Clean up resources

To learn more about using the Kubernetes dashboard, see:

[Use the Kubernetes dashboard](#)

# Deployment Center for Azure Kubernetes

3/5/2021 • 4 minutes to read • [Edit Online](#)

Deployment Center in Azure DevOps simplifies setting up a robust Azure DevOps pipeline for your application. By default, Deployment Center configures an Azure DevOps pipeline to deploy your application updates to the Kubernetes cluster. You can extend the default configured Azure DevOps pipeline and also add richer capabilities: the ability to gain approval before deploying, provision additional Azure resources, run scripts, upgrade your application, and even run more validation tests.

In this tutorial, you will:

- Configure an Azure DevOps pipeline to deploy your application updates to the Kubernetes cluster.
- Examine the continuous integration (CI) pipeline.
- Examine the continuous delivery (CD) pipeline.
- Clean up the resources.

## Prerequisites

- An Azure subscription. You can get one free through [Visual Studio Dev Essentials](#).
- An Azure Kubernetes Service (AKS) cluster.

## Create an AKS cluster

1. Sign in to your [Azure portal](#).
2. Select the [Cloud Shell](#) option on the right side of the menu bar in the Azure portal.
3. To create the AKS cluster, run the following commands:

```
Create a resource group in the South India location:

az group create --name azoaks --location southindia

Create a cluster named azookubectl with one node.

az aks create --resource-group azoaks --name azookubectl --node-count 1 --enable-addons monitoring -
-generate-ssh-keys
```

## Deploy application updates to a Kubernetes cluster

1. Go to the resource group that you created in the previous section.
2. Select the AKS cluster, and then select **Deployment Center (preview)** on the left blade. Select **Get started**.

The screenshot shows the 'Deployment center' blade in the Azure portal. On the left, there's a sidebar with various options like Overview, Activity log, Access control (IAM), Tags, Settings, Upgrade, Scale, Dev Spaces, Deployment center (preview) (which has a red arrow pointing to it), Properties, Locks, Export template, Monitoring, and Insights. The main content area is titled 'Deployment Center' and contains a detailed description of what DevOps simplifies, mentioning Kubernetes, DevOps pipelines, approvals, and additional validation tests.

3. Choose the location of the code and select **Next**. Then, select one of the currently supported repositories: [Azure Repos](#) or [GitHub](#).

Azure Repos is a set of version control tools that help you manage your code. Whether your software project is large or small, using version control as early as possible is a good idea.

- **Azure Repos:** Choose a repository from your existing project and organization.

The screenshot shows the 'Select the code location' step of a deployment wizard. At the top, there's a progress bar with four steps: 1. Source (highlighted in blue), 2. Repository, 3. Application, and 4. Resources. Below the progress bar, the title 'Select the code location' is centered. There are two options displayed: 'Azure Repos' (selected, indicated by a checked checkbox) and 'GitHub'. Both options have their respective logos and brief descriptions.

1 Source      2 Repository      3 Application      4 Resources

Select the code location

 <b>Azure Repos</b> <input checked="" type="checkbox"/>	 <b>GitHub</b>
Unlimited free private repos	Home to the world's largest community of developers

Previous **Next**

- **GitHub:** Authorize and select the repository for your GitHub account.



4. Deployment Center analyzes the repository and detects your Dockerfile. If you want to update the Dockerfile, you can edit the identified port number.

Detected the following Dockerfile

```
src/MyHealth.Web/Dockerfile
1 FROM microsoft/aspnetcore:1.0
2 ARG source
3 WORKDIR /app
4 EXPOSE 80
5 COPY ${source:-obj}/Docker/publish .
6 ENTRYPOINT ["dotnet",
7 "myhealth.web.dll"]
```

We have detected the following values from your Dockerfile.  
Please update if required.

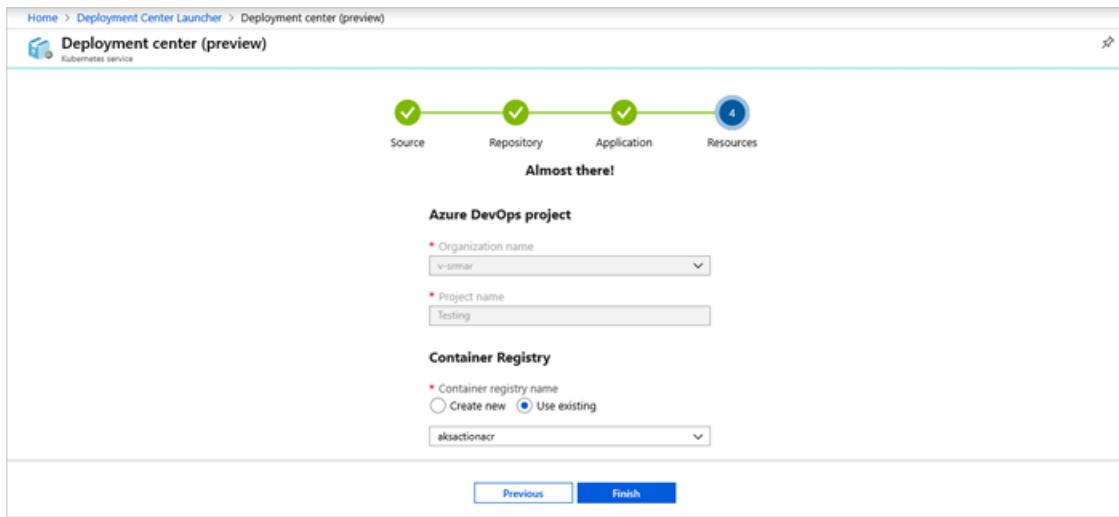
Port

If the repository doesn't contain the Dockerfile, the system displays a message to commit one.

Could not find a Dockerfile in the repository  
Please commit the Dockerfile to the repository to proceed further.

5. Select an existing container registry or create one, and then select **Finish**. The pipeline is created automatically and queues a build in [Azure Pipelines](#).

Azure Pipelines is a cloud service that you can use to automatically build and test your code project and make it available to other users. Azure Pipelines combines continuous integration and continuous delivery to constantly and consistently test and build your code and ship it to any target.



6. Select the link to see the ongoing pipeline.
7. You'll see the successful logs after deployment is complete.

## Examine the CI pipeline

Deployment Center automatically configures your Azure DevOps organization's CI/CD pipeline. The pipeline can be explored and customized.

1. Go to the Deployment Center dashboard.
2. Select the build number from the list of successful logs to view the build pipeline for your project.
3. Select the ellipsis (...) in the upper-right corner. A menu shows several options, such as queuing a new build, retaining a build, and editing the build pipeline. Select **Edit pipeline**.
4. You can examine the different tasks for your build pipeline in this pane. The build performs various tasks, such as collecting sources from the Git repository, creating an image, pushing an image to the container registry, and publishing outputs that are used for deployments.
5. Select the name of the build pipeline at the top of the pipeline.
6. Change your build pipeline name to something more descriptive, select **Save & queue**, and then select **Save**.
7. Under your build pipeline, select **History**. This pane shows an audit trail of your recent build changes. Azure DevOps monitors any changes made to the build pipeline and allows you to compare versions.
8. Select **Triggers**. You can include or exclude branches from the CI process.

9. Select **Retention**. You can specify policies to keep or remove a number of builds, depending on your scenario.

## Examine the CD pipeline

Deployment Center automatically creates and configures the relationship between your Azure DevOps organization and your Azure subscription. The steps involved include setting up an Azure service connection to authenticate your Azure subscription with Azure DevOps. The automated process also creates a release pipeline, which provides continuous delivery to Azure.

1. Select **Pipelines**, and then select **Releases**.
2. To edit the release pipeline, select **Edit**.
3. Select **Drop** from the **Artifacts** list. In the previous steps, the construction pipeline you examined produces the output used for the artifact.
4. Select the **Continuous deployment** trigger on the right of the **Drop** option. This release pipeline has an enabled CD trigger that runs a deployment whenever a new build artifact is available. You can also disable the trigger to require manual execution for your deployments.
5. To examine all the tasks for your pipeline, select **Tasks**. The release sets the tiller environment, configures the `imagePullSecrets` parameter, installs Helm tools, and deploys the Helm charts to the Kubernetes cluster.
6. To view the release history, select **View releases**.
7. To see the summary, select **Release**. Select any of the stages to explore multiple menus, such as a release summary, associated work items, and tests.
8. Select **Commits**. This view shows code commits related to this deployment. Compare releases to see the commit differences between deployments.
9. Select **Logs**. The logs contain useful deployment information, which you can view during and after deployments.

## Clean up resources

You can delete the related resources that you created when you don't need them anymore. Use the delete functionality on the DevOps Projects dashboard.

## Next steps

You can modify these build and release pipelines to meet the needs of your team. Or, you can use this CI/CD model as a template for your other pipelines.

# GitHub Actions for deploying to Kubernetes service

5/11/2021 • 5 minutes to read • [Edit Online](#)

[GitHub Actions](#) gives you the flexibility to build an automated software development lifecycle workflow. You can use multiple Kubernetes actions to deploy to containers from Azure Container Registry to Azure Kubernetes Service with GitHub Actions.

## Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- A GitHub account. If you don't have one, sign up for [free](#).
- A working Kubernetes cluster
  - [Tutorial: Prepare an application for Azure Kubernetes Service](#)

## Workflow file overview

A workflow is defined by a YAML (.yml) file in the `/.github/workflows/` path in your repository. This definition contains the various steps and parameters that make up the workflow.

For a workflow targeting AKS, the file has three sections:

SECTION	TASKS
Authentication	Login to a private container registry (ACR)
Build	Build & push the container image
Deploy	<ol style="list-style-type: none"><li>1. Set the target AKS cluster</li><li>2. Create a generic/docker-registry secret in Kubernetes cluster</li><li>3. Deploy to the Kubernetes cluster</li></ol>

## Create a service principal

You can create a [service principal](#) by using the `az ad sp create-for-rbac` command in the [Azure CLI](#). You can run this command using [Azure Cloud Shell](#) in the Azure portal or by selecting the Try it button.

```
az ad sp create-for-rbac --name "myApp" --role contributor --scopes
/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP> --sdk-auth
```

In the above command, replace the placeholders with your subscription ID, and resource group. The output is the role assignment credentials that provide access to your resource. The command should output a JSON object similar to this.

```
{
 "clientId": "<GUID>",
 "clientSecret": "<GUID>",
 "subscriptionId": "<GUID>",
 "tenantId": "<GUID>",
 (...)
}
```

Copy this JSON object, which you can use to authenticate from GitHub.

## Configure the GitHub secrets

Follow the steps to configure the secrets:

1. In [GitHub](#), browse to your repository, select **Settings > Secrets > Add a new secret**.



2. Paste the contents of the above `az cli` command as the value of secret variable. For example,

`AZURE_CREDENTIALS`.

3. Similarly, define the following additional secrets for the container registry credentials and set them in Docker login action.

- `REGISTRY_USERNAME`
- `REGISTRY_PASSWORD`

4. You will see the secrets as shown below once defined.



# Build a container image and deploy to Azure Kubernetes Service cluster

The build and push of the container images is done using `Azure/docker-login@v1` action.

```
env:
 REGISTRY_NAME: {registry-name}
 CLUSTER_NAME: {cluster-name}
 CLUSTER_RESOURCE_GROUP: {resource-group-name}
 NAMESPACE: {namespace-name}
 APP_NAME: {app-name}

jobs:
 build:
 runs-on: ubuntu-latest
 steps:
 - uses: actions/checkout@main

 # Connect to Azure Container Registry (ACR)
 - uses: azure/docker-login@v1
 with:
 login-server: ${{ env.REGISTRY_NAME }}.azurecr.io
 username: ${{ secrets.REGISTRY_USERNAME }}
 password: ${{ secrets.REGISTRY_PASSWORD }}

 # Container build and push to a Azure Container Registry (ACR)
 - run: |
 docker build . -t ${{ env.REGISTRY_NAME }}.azurecr.io/${{ env.APP_NAME }}:${{ github.sha }}
 docker push ${{ env.REGISTRY_NAME }}.azurecr.io/${{ env.APP_NAME }}:${{ github.sha }}

```

## Deploy to Azure Kubernetes Service cluster

To deploy a container image to AKS, you will need to use the `Azure/k8s-deploy@v1` action. This action has five parameters:

PARAMETER	EXPLANATION
<code>namespace</code>	(Optional) Choose the target Kubernetes namespace. If the namespace is not provided, the commands will run in the default namespace
<code>manifests</code>	(Required) Path to the manifest files, that will be used for deployment
<code>images</code>	(Optional) Fully qualified resource URL of the image(s) to be used for substitutions on the manifest files
<code>imagepullsecrets</code>	(Optional) Name of a docker-registry secret that has already been set up within the cluster. Each of these secret names is added under <code>imagePullSecrets</code> field for the workloads found in the input manifest files
<code>kubectl-version</code>	(Optional) Installs a specific version of kubectl binary

### NOTE

The manifest files should be created manually by you. Currently there are no tools that will generate such files in an automated way, for more information see [this sample repository with example manifest files](#).

Before you can deploy to AKS, you'll need to set target Kubernetes namespace and create an image pull secret.

See [Pull images from an Azure container registry to a Kubernetes cluster](#), to learn more about how pulling images works.

```
Create namespace if doesn't exist
- run: |
 kubectl create namespace ${{ env.NAMESPACE }} --dry-run -o json | kubectl apply -f -

Create image pull secret for ACR
- uses: azure/k8s-create-secret@v1
 with:
 container-registry-url: ${{ env.REGISTRY_NAME }}.azurecr.io
 container-registry-username: ${{ secrets.REGISTRY_USERNAME }}
 container-registry-password: ${{ secrets.REGISTRY_PASSWORD }}
 secret-name: ${{ env.SECRET }}
 namespace: ${{ env.NAMESPACE }}
 arguments: --force true
```

Complete your deployment with the `k8s-deploy` action. Replace the environment variables with values for your application.

```

on: [push]

Environment variables available to all jobs and steps in this workflow
env:
 REGISTRY_NAME: {registry-name}
 CLUSTER_NAME: {cluster-name}
 CLUSTER_RESOURCE_GROUP: {resource-group-name}
 NAMESPACE: {namespace-name}
 SECRET: {secret-name}
 APP_NAME: {app-name}

jobs:
 build:
 runs-on: ubuntu-latest
 steps:
 - uses: actions/checkout@main

 # Connect to Azure Container Registry (ACR)
 - uses: azure/docker-login@v1
 with:
 login-server: ${{ env.REGISTRY_NAME }}.azurecr.io
 username: ${{ secrets.REGISTRY_USERNAME }}
 password: ${{ secrets.REGISTRY_PASSWORD }}

 # Container build and push to a Azure Container Registry (ACR)
 - run: |
 docker build . -t ${{ env.REGISTRY_NAME }}.azurecr.io/${{ env.APP_NAME }}:${{ github.sha }}
 docker push ${{ env.REGISTRY_NAME }}.azurecr.io/${{ env.APP_NAME }}:${{ github.sha }}

 # Set the target Azure Kubernetes Service (AKS) cluster.
 - uses: azure/aks-set-context@v1
 with:
 creds: '${{ secrets.AZURE_CREDENTIALS }}'
 cluster-name: ${{ env.CLUSTER_NAME }}
 resource-group: ${{ env.CLUSTER_RESOURCE_GROUP }}

 # Create namespace if doesn't exist
 - run: |
 kubectl create namespace ${{ env.NAMESPACE }} --dry-run -o json | kubectl apply -f -

 # Create image pull secret for ACR
 - uses: azure/k8s-create-secret@v1
 with:
 container-registry-url: ${{ env.REGISTRY_NAME }}.azurecr.io
 container-registry-username: ${{ secrets.REGISTRY_USERNAME }}
 container-registry-password: ${{ secrets.REGISTRY_PASSWORD }}
 secret-name: ${{ env.SECRET }}
 namespace: ${{ env.NAMESPACE }}
 force: true

 # Deploy app to AKS
 - uses: azure/k8s-deploy@v1
 with:
 manifests: |
 manifests/deployment.yml
 manifests/service.yml
 images: |
 ${{ env.REGISTRY_NAME }}.azurecr.io/${{ env.APP_NAME }}:${{ github.sha }}
 imagepullsecrets: |
 ${{ env.SECRET }}
 namespace: ${{ env.NAMESPACE }}

```

## Clean up resources

When your Kubernetes cluster, container registry, and repository are no longer needed, clean up the resources you deployed by deleting the resource group and your GitHub repository.

## Next steps

[Learn about Azure Kubernetes Service](#)

[Learn how to create multiple pipelines on GitHub Actions with AKS](#)

### More Kubernetes GitHub Actions

- [Kubectl tool installer](#) ( `azure/setup-kubectl` ): Installs a specific version of kubectl on the runner.
- [Kubernetes set context](#) ( `azure/k8s-set-context` ): Set the target Kubernetes cluster context which will be used by other actions or run any kubectl commands.
- [AKS set context](#) ( `azure/aks-set-context` ): Set the target Azure Kubernetes Service cluster context.
- [Kubernetes create secret](#) ( `azure/k8s-create-secret` ): Create a generic secret or docker-registry secret in the Kubernetes cluster.
- [Kubernetes deploy](#) ( `azure/k8s-deploy` ): Bake and deploy manifests to Kubernetes clusters.
- [Setup Helm](#) ( `azure/setup-helm` ): Install a specific version of Helm binary on the runner.
- [Kubernetes bake](#) ( `azure/k8s-bake` ): Bake manifest file to be used for deployments using helm2, kustomize or kompose.
- [Kubernetes lint](#) ( `azure/k8s-lint` ): Validate/lint your manifest files.

# AKS troubleshooting

5/4/2021 • 21 minutes to read • [Edit Online](#)

When you create or manage Azure Kubernetes Service (AKS) clusters, you might occasionally come across problems. This article details some common problems and troubleshooting steps.

## In general, where do I find information about debugging Kubernetes problems?

Try the [official guide to troubleshooting Kubernetes clusters](#). There's also a [troubleshooting guide](#), published by a Microsoft engineer for troubleshooting pods, nodes, clusters, and other features.

### I'm getting a `quota exceeded` error during creation or upgrade. What should I do?

[Request more cores.](#)

### I'm getting an `insufficientSubnetSize` error while deploying an AKS cluster with advanced networking. What should I do?

This error indicates a subnet in use for a cluster no longer has available IPs within its CIDR for successful resource assignment. For Kubenet clusters, the requirement is sufficient IP space for each node in the cluster. For Azure CNI clusters, the requirement is sufficient IP space for each node and pod in the cluster. Read more about the [design of Azure CNI to assign IPs to pods](#).

These errors are also surfaced in [AKS Diagnostics](#), which proactively surfaces issues such as an insufficient subnet size.

The following three (3) cases cause an insufficient subnet size error:

#### 1. AKS Scale or AKS Node pool scale

- If using Kubenet, when the `number of free IPs in the subnet` is **less than** the `number of new nodes requested`.
- If using Azure CNI, when the `number of free IPs in the subnet` is **less than** the `number of nodes requested times (*) the node pool's --max-pod value`.

#### 2. AKS Upgrade or AKS Node pool upgrade

- If using Kubenet, when the `number of free IPs in the subnet` is **less than** the `number of buffer nodes needed to upgrade`.
- If using Azure CNI, when the `number of free IPs in the subnet` is **less than** the `number of buffer nodes needed to upgrade times (*) the node pool's --max-pod value`.

By default AKS clusters set a max surge (upgrade buffer) value of one (1), but this upgrade behavior can be customized by setting the [max surge value of a node pool, which will increase the number of available IPs needed to complete an upgrade.

#### 3. AKS create or AKS Node pool add

- If using Kubenet, when the `number of free IPs in the subnet` is **less than** the `number of nodes requested for the node pool`.

- b. If using Azure CNI, when the `number of free IPs in the subnet` is less than the `number of nodes requested times (*) the node pool's --max-pod value`.

The following mitigation can be taken by creating new subnets. The permission to create a new subnet is required for mitigation due to the inability to update an existing subnet's CIDR range.

1. Rebuild a new subnet with a larger CIDR range sufficient for operation goals:
  - a. Create a new subnet with a new desired non-overlapping range.
  - b. Create a new node pool on the new subnet.
  - c. Drain pods from the old node pool residing in the old subnet to be replaced.
  - d. Delete the old subnet and old node pool.

## My pod is stuck in CrashLoopBackOff mode. What should I do?

There might be various reasons for the pod being stuck in that mode. You might look into:

- The pod itself, by using `kubectl describe pod <pod-name>`.
- The logs, by using `kubectl logs <pod-name>`.

For more information on how to troubleshoot pod problems, see [Debug applications](#).

## I'm receiving `TCP timeouts` when using `kubectl` or other third-party tools connecting to the API server

AKS has HA control planes that scale vertically according to the number of cores to ensure its Service Level Objectives (SLOs) and Service Level Agreements (SLAs). If you're experiencing connections timing out, check the below:

- **Are all your API commands timing out consistently or only a few?** If it's only a few, your `tunnelfront` pod or `aks-link` pod, responsible for node -> control plane communication, might not be in a running state. Make sure the nodes hosting this pod aren't over-utilized or under stress. Consider moving them to their own `system` node pool.
- **Have you opened all required ports, FQDNs, and IPs noted on the AKS restrict egress traffic docs?** Otherwise several commands calls can fail.
- **Is your current IP covered by API IP Authorized Ranges?** If you're using this feature and your IP is not included in the ranges your calls will be blocked.
- **Do you have a client or application leaking calls to the API server?** Make sure to use watches instead of frequent get calls and that your third-party applications aren't leaking such calls. For example, a bug in the Istio mixer causes a new API Server watch connection to be created every time a secret is read internally. Because this behavior happens at a regular interval, watch connections quickly accumulate, and eventually cause the API Server to become overloaded no matter the scaling pattern.  
<https://github.com/istio/istio/issues/19481>
- **Do you have many releases in your helm deployments?** This scenario can cause both tiller to use too much memory on the nodes, as well as a large amount of `configmaps`, which can cause unnecessary spikes on the API server. Consider configuring `--history-max` at `helm init` and leverage the new Helm 3. More details on the following issues:
  - <https://github.com/helm/helm/issues/4821>
  - <https://github.com/helm/helm/issues/3500>
  - <https://github.com/helm/helm/issues/4543>
- **Is internal traffic between nodes being blocked?**

I'm receiving `TCP timeouts`, such as

```
dial tcp <Node_IP>:10250: i/o timeout
```

These timeouts may be related to internal traffic between nodes being blocked. Verify that this traffic is not being blocked, such as by [network security groups](#) on the subnet for your cluster's nodes.

I'm trying to enable Kubernetes role-based access control (Kubernetes RBAC) on an existing cluster. How can I do that?

Enabling Kubernetes role-based access control (Kubernetes RBAC) on existing clusters isn't supported at this time, it must be set when creating new clusters. Kubernetes RBAC is enabled by default when using CLI, Portal, or an API version later than `2020-03-01`.

I can't get logs by using `kubectl logs` or I can't connect to the API server. I'm getting "Error from server: error dialing backend: dial `tcp...`". What should I do?

Ensure ports 22, 9000 and 1194 are open to connect to the API server. Check whether the `tunnelfront` or `aks-link` pod is running in the `kube-system` namespace using the `kubectl get pods --namespace kube-system` command. If it isn't, force deletion of the pod and it will restart.

I'm getting `"tls: client offered only unsupported versions"` from my client when connecting to AKS API. What should I do?

The minimum supported TLS version in AKS is TLS 1.2.

I'm trying to upgrade or scale and am getting a `"Changing property 'imageReference' is not allowed"` error. How do I fix this problem?

You might be getting this error because you've modified the tags in the agent nodes inside the AKS cluster. Modify or delete tags and other properties of resources in the `MC_*` resource group can lead to unexpected results. Altering the resources under the `MC_*` group in the AKS cluster breaks the service-level objective (SLO).

I'm receiving errors that my cluster is in failed state and upgrading or scaling will not work until it is fixed

*This troubleshooting assistance is directed from <https://aka.ms/aks-cluster-failed>*

This error occurs when clusters enter a failed state for multiple reasons. Follow the steps below to resolve your cluster failed state before retrying the previously failed operation:

1. Until the cluster is out of `failed` state, `upgrade` and `scale` operations won't succeed. Common root issues and resolutions include:
  - Scaling with **insufficient compute (CRP)** quota. To resolve, first scale your cluster back to a stable goal state within quota. Then follow these [steps to request a compute quota increase](#) before trying to scale up again beyond initial quota limits.
  - Scaling a cluster with advanced networking and **insufficient subnet (networking) resources**. To resolve, first scale your cluster back to a stable goal state within quota. Then follow [these steps to request a resource quota increase](#) before trying to scale up again beyond initial quota limits.

- Once the underlying cause for upgrade failure is resolved, your cluster should be in a succeeded state. Once a succeeded state is verified, retry the original operation.

## I'm receiving errors when trying to upgrade or scale that state my cluster is being upgraded or has failed upgrade

*This troubleshooting assistance is directed from <https://aka.ms/aks-pending-upgrade>*

You can't have a cluster or node pool simultaneously upgrade and scale. Instead, each operation type must complete on the target resource before the next request on that same resource. As a result, operations are limited when active upgrade or scale operations are occurring or attempted.

To help diagnose the issue run `az aks show -g myResourceGroup -n myAKSCluster -o table` to retrieve detailed status on your cluster. Based on the result:

- If cluster is actively upgrading, wait until the operation finishes. If it succeeded, retry the previously failed operation again.
- If cluster has failed upgrade, follow steps outlined in previous section.

## Can I move my cluster to a different subscription or my subscription with my cluster to a new tenant?

If you've moved your AKS cluster to a different subscription or the cluster's subscription to a new tenant, the cluster won't function because of missing cluster identity permissions. **AKS doesn't support moving clusters across subscriptions or tenants** because of this constraint.

## I'm receiving errors trying to use features that require virtual machine scale sets

*This troubleshooting assistance is directed from [aka.ms/aks-vmss-enablement](https://aka.ms/aks-vmss-enablement)*

You may receive errors that indicate your AKS cluster isn't on a virtual machine scale set, such as the following example:

AgentPool `<agentpoolname>` has set auto scaling as enabled but isn't on Virtual Machine Scale Sets

Features such as the cluster autoscaler or multiple node pools require virtual machine scale sets as the `vm-set-type`.

Follow the *Before you begin* steps in the appropriate doc to correctly create an AKS cluster:

- [Use the cluster autoscaler](#)
- [Create and use multiple node pools](#)

## What naming restrictions are enforced for AKS resources and parameters?

*This troubleshooting assistance is directed from [aka.ms/aks-naming-rules](https://aka.ms/aks-naming-rules)*

Naming restrictions are implemented by both the Azure platform and AKS. If a resource name or parameter breaks one of these restrictions, an error is returned that asks you provide a different input. The following common naming guidelines apply:

- Cluster names must be 1-63 characters. The only allowed characters are letters, numbers, dashes, and underscore. The first and last character must be a letter or a number.

- The AKS Node/MC\_ resource group name combines resource group name and resource name. The autogenerated syntax of `MC_resourceGroupName_resourceName_AzureRegion` must be no greater than 80 chars. If needed, reduce the length of your resource group name or AKS cluster name. You may also [customize your node resource group name](#)
- The *dnsPrefix* must start and end with alphanumeric values and must be between 1-54 characters. Valid characters include alphanumeric values and hyphens (-). The *dnsPrefix* can't include special characters such as a period (.).
- AKS Node Pool names must be all lowercase and be 1-11 characters for linux node pools and 1-6 characters for windows node pools. The name must start with a letter and the only allowed characters are letters and numbers.
- The *admin-username*, which sets the administrator username for Linux nodes, must start with a letter, may only contain letters, numbers, hyphens, and underscores, and has a maximum length of 64 characters.

I'm receiving errors when trying to create, update, scale, delete or upgrade cluster, that operation is not allowed as another operation is in progress.

*This troubleshooting assistance is directed from aka.ms/aks-pending-operation*

Cluster operations are limited when a previous operation is still in progress. To retrieve a detailed status of your cluster, use the `az aks show -g myResourceGroup -n myAKScluster -o table` command. Use your own resource group and AKS cluster name as needed.

Based on the output of the cluster status:

- If the cluster is in any provisioning state other than *Succeeded* or *Failed*, wait until the operation (*Upgrading / Updating / Creating / Scaling / Deleting / Migrating*) finishes. When the previous operation has completed, retry your latest cluster operation.
- If the cluster has a failed upgrade, follow the steps outlined [I'm receiving errors that my cluster is in failed state and upgrading or scaling will not work until it is fixed](#).

Received an error saying my service principal wasn't found or is invalid when I try to create a new cluster.

When creating an AKS cluster, it requires a service principal or managed identity to create resources on your behalf. AKS can automatically create a new service principal at cluster creation time or receive an existing one. When using an automatically created one, Azure Active Directory needs to propagate it to every region so the creation succeeds. When the propagation takes too long, the cluster will fail validation to create as it can't find an available service principal to do so.

Use the following workarounds for this issue:

- Use an existing service principal, which has already propagated across regions and exists to pass into AKS at cluster create time.
- If using automation scripts, add time delays between service principal creation and AKS cluster creation.
- If using Azure portal, return to the cluster settings during create and retry the validation page after a few minutes.

I'm getting `"AADSTS7000215: Invalid client secret is provided."` when using AKS API. What should I do?

This issue is due to the expiration of service principal credentials. [Update the credentials for an AKS cluster](#).

# I can't access my cluster API from my automation/dev machine/tooling when using API server authorized IP ranges. How do I fix this problem?

To resolve this issue, ensure `--api-server-authorized-ip-ranges` includes the IP(s) or IP range(s) of automation/dev/tooling systems being used. Refer section 'How to find my IP' in [Secure access to the API server using authorized IP address ranges](#).

# I'm unable to view resources in Kubernetes resource viewer in Azure portal for my cluster configured with API server authorized IP ranges. How do I fix this problem?

The [Kubernetes resource viewer](#) requires `--api-server-authorized-ip-ranges` to include access for the local client computer or IP address range (from which the portal is being browsed). Refer section 'How to find my IP' in [Secure access to the API server using authorized IP address ranges](#).

# I'm receiving errors after restricting egress traffic

When restricting egress traffic from an AKS cluster, there are [required and optional recommended](#) outbound ports / network rules and FQDN / application rules for AKS. If your settings are in conflict with any of these rules, certain `kubectl` commands won't work correctly. You may also see errors when creating an AKS cluster.

Verify that your settings aren't conflicting with any of the required or optional recommended outbound ports / network rules and FQDN / application rules.

# I'm receiving "429 - Too Many Requests" errors

When a kubernetes cluster on Azure (AKS or no) does a frequent scale up/down or uses the cluster autoscaler (CA), those operations can result in a large number of HTTP calls that in turn exceed the assigned subscription quota leading to failure. The errors will look like

```
Service returned an error. Status=429 Code=\"OperationNotAllowed\" Message=\"The server rejected the request because too many requests have been received for this subscription.\" Details=[{"code": "TooManyRequests", "message": "\\\\\"operationGroup\\\\\" : \\\\\"HighCostGetVMScaleSet30Min\\\\\", \\\\\"startTime\\\\\" : \\\\\"2020-09-20T07:13:55.2177346+00\\\\\", \\\\\"endTime\\\\\" : \\\\\"2020-09-20T07:28:55.2177346+00\\\\\", \\\\\"allowedRequestCount\\\\\" : 1800, \\\\\"measuredRequestCount\\\\\" : 2208}]", "target": "\\\\\"HighCostGetVMScaleSet30Min\\\\\"}] InnerError={\"internalErrorCode\": \"TooManyRequestsReceived\\\"}}}
```

These throttling errors are described in detail [here](#) and [here](#)

The recommendation from AKS Engineering Team is to ensure you are running version at least 1.18.x, which contains many improvements. More details can be found on these improvements [here](#) and [here](#).

Given these throttling errors are measured at the subscription level, they might still happen if:

- There are 3rd party applications making GET requests (for example, monitoring applications, and so on). The recommendation is to reduce the frequency of these calls.
- There are numerous AKS clusters / node pools using virtual machine scale sets. Try to split your number of clusters into different subscriptions, in particular if you expect them to be very active (for example, an active cluster autoscaler) or have multiple clients (for example, rancher, terraform, and so on).

# My cluster's provisioning status changed from Ready to Failed with or

## without me performing an operation. What should I do?

If your cluster's provisioning status changes from *Ready* to *Failed* with or without you performing any operations, but the applications on your cluster are continuing to run, this issue may be resolved automatically by the service and your applications should not be affected.

If your cluster's provisioning status remains as *Failed* or the applications on your cluster stop working, [submit a support request](#).

## My watch is stale or Azure AD Pod Identity NMI is returning status 500

If you're using Azure Firewall like on this [example](#), you may encounter this issue as the long lived TCP connections via firewall using Application Rules currently have a bug (to be resolved in Q1CY21) that causes the Go `keepalives` to be terminated on the firewall. Until this issue is resolved, you can mitigate by adding a Network rule (instead of application rule) to the AKS API server IP.

## Azure Storage and AKS Troubleshooting

### Failure when setting uid and `GID` in mountOptions for Azure Disk

Azure Disk uses the ext4,xfs filesystem by default and mountOptions such as `uid=x,gid=x` can't be set at mount time. For example if you tried to set mountOptions `uid=999,gid=999`, would see an error like:

```
Warning FailedMount 63s kubelet, aks-nodepool1-29460110-0
MountVolume.MountDevice failed for volume "pvc-d783d0e4-85a1-11e9-8a90-369885447933" : azureDisk -
mountDevice:FormatAndMount failed with mount failed: exit status 32
Mounting command: systemd-run
Mounting arguments: --description=Kubernetes transient mount for
/var/lib/kubelet/plugins/kubernetes.io/azure-disk/mounts/m436970985 --scope -- mount -t xfs -o
dir_mode=0777,file_mode=0777,uid=1000,gid=1000,defaults /dev/disk/azure/scsi1/lun2
/var/lib/kubelet/plugins/kubernetes.io/azure-disk/mounts/m436970985
Output: Running scope as unit run-rb21966413ab449b3a242ae9b0fbc9398.scope.
mount: wrong fs type, bad option, bad superblock on /dev/sde,
 missing codepage or helper program, or other error
```

You can mitigate the issue by doing one the options:

- [Configure the security context for a pod](#) by setting uid in `runAsUser` and gid in `fsGroup`. For example, the following setting will set pod run as root, make it accessible to any file:

```
apiVersion: v1
kind: Pod
metadata:
 name: security-context-demo
spec:
 securityContext:
 runAsUser: 0
 fsGroup: 0
```

### NOTE

Since gid and uid are mounted as root or 0 by default. If gid or uid are set as non-root, for example 1000, Kubernetes will use `chown` to change all directories and files under that disk. This operation can be time consuming and may make mounting the disk very slow.

- Use `chown` in initContainers to set `GID` and `UID`. For example:

```
initContainers:
- name: volume-mount
 image: mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11
 command: ["sh", "-c", "chown -R 100:100 /data"]
 volumeMounts:
- name: <your data volume>
 mountPath: /data
```

### Azure Disk detach failure leading to potential race condition issue and invalid data disk list

When an Azure Disk fails to detach, it will retry up to six times to detach the disk using exponential back off. It will also hold a node-level lock on the data disk list for about 3 minutes. If the disk list is updated manually during that time, it will cause the disk list held by the node-level lock to be obsolete and cause instability on the node.

This issue has been fixed in the following versions of Kubernetes:

KUBERNETES VERSION	FIXED VERSION
1.12	1.12.9 or later
1.13	1.13.6 or later
1.14	1.14.2 or later
1.15 and later	N/A

If you're using a version of Kubernetes that doesn't have the fix for this issue and your node has an obsolete disk list, you can mitigate by detaching all non-existing disks from the VM as a bulk operation. **Individually detaching non-existing disks may fail.**

### Large number of Azure Disks causes slow attach/detach

When the numbers of Azure Disk attach/detach operations targeting a single node VM is larger than 10, or larger than 3 when targeting single virtual machine scale set pool they may be slower than expected as they are done sequentially. This issue is a known limitation and there are no workarounds at this time. [User voice item to support parallel attach/detach beyond number.](#)

### Azure Disk detach failure leading to potential node VM in failed state

In some edge cases, an Azure Disk detach may partially fail and leave the node VM in a failed state.

This issue has been fixed in the following versions of Kubernetes:

KUBERNETES VERSION	FIXED VERSION
1.12	1.12.10 or later
1.13	1.13.8 or later
1.14	1.14.4 or later
1.15 and later	N/A

If you're using a version of Kubernetes that doesn't have the fix for this issue and your node is in a failed state, you can mitigate by manually updating the VM status using one of the below:

- For an availability set-based cluster:

```
az vm update -n <VM_NAME> -g <RESOURCE_GROUP_NAME>
```

- For a VMSS-based cluster:

```
az vmss update-instances -g <RESOURCE_GROUP_NAME> --name <VMSS_NAME> --instance-id <ID>
```

## Azure Files and AKS Troubleshooting

### What are the recommended stable versions of Kubernetes for Azure files?

KUBERNETES VERSION	RECOMMENDED VERSION
1.12	1.12.6 or later
1.13	1.13.4 or later
1.14	1.14.0 or later

### What are the default mountOptions when using Azure Files?

Recommended settings:

KUBERNETES VERSION	FILEMODE AND DIRMODE VALUE
1.12.0 - 1.12.1	0755
1.12.2 and later	0777

Mount options can be specified on the storage class object. The following example sets 0777:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
 name: azurefile
provisioner: kubernetes.io/azure-file
mountOptions:
 - dir_mode=0777
 - file_mode=0777
 - uid=1000
 - gid=1000
 - mfsymlinks
 - nobrl
 - cache=none
parameters:
 skuName: Standard_LRS
```

Some additional useful *mountOptions* settings:

- `mfsymlinks` will make Azure Files mount (cifs) support symbolic links
- `nobrl` will prevent sending byte range lock requests to the server. This setting is necessary for certain applications that break with cifs style mandatory byte range locks. Most cifs servers don't yet support requesting advisory byte range locks. If not using `nobrl`, applications that break with cifs style mandatory byte range locks may cause error messages similar to:

```
Error: SQLITE_BUSY: database is locked
```

## Error "could not change permissions" when using Azure Files

When running PostgreSQL on the Azure Files plugin, you may see an error similar to:

```
initdb: could not change permissions of directory "/var/lib/postgresql/data": Operation not permitted
fixing permissions on existing directory /var/lib/postgresql/data
```

This error is caused by the Azure Files plugin using the cifs/SMB protocol. When using the cifs/SMB protocol, the file and directory permissions couldn't be changed after mounting.

To resolve this issue, use `subPath` together with the Azure Disk plugin.

### NOTE

For ext3/4 disk type, there is a lost+found directory after the disk is formatted.

## Azure Files has high latency compared to Azure Disk when handling many small files

In some case, such as handling many small files, you may experience high latency when using Azure Files when compared to Azure Disk.

## Error when enabling "Allow access allow access from selected network" setting on storage account

If you enable *allow access from selected network* on a storage account that's used for dynamic provisioning in AKS, you'll get an error when AKS creates a file share:

```
persistentvolume-controller (combined from similar events): Failed to provision volume with StorageClass
"azurefile": failed to create share kubernetes-dynamic-pvc-xxx in account xxx: failed to create file share,
err: storage: service returned error: StatusCode=403, ErrorCode=AuthorizationFailure, ErrorMessage=This
request is not authorized to perform this operation.
```

This error is because of the Kubernetes *persistentvolume-controller* not being on the network chosen when setting *allow access from selected network*.

You can mitigate the issue by using [static provisioning with Azure Files](#).

## Azure Files fails to remount in Windows pod

If a Windows pod with an Azure Files mount is deleted and then scheduled to be recreated on the same node, the mount will fail. This failure is because of the `New-SmbGlobalMapping` command failing since the Azure Files mount is already mounted on the node.

For example, you may see an error similar to:

```
E0118 08:15:52.041014 2112 nestedpendingoperations.go:267] Operation for "\"kubernetes.io/azure-
file/42c0ea39-1af9-11e9-8941-000d3af95268-pvc-d7e1b5f9-1af3-11e9-8941-000d3af95268\" ("42c0ea39-1af9-11e9-
8941-000d3af95268\")" failed. No retries permitted until 2019-01-18 08:15:53.0410149 +0000 GMT
m=+732.446642701 (durationBeforeRetry 1s). Error: "MountVolume.SetUp failed for volume \"pvc-d7e1b5f9-1af3-
11e9-8941-000d3af95268\" (UniqueName: \"kubernetes.io/azure-file/42c0ea39-1af9-11e9-8941-000d3af95268-pvc-
d7e1b5f9-1af3-11e9-8941-000d3af95268\") pod \"deployment-azurefile-697f98d559-6zrlf\" (UID: \"42c0ea39-1af9-
11e9-8941-000d3af95268\") : azureMount: SmbGlobalMapping failed: exit status 1, only SMB mount is supported
now, output: \"New-SmbGlobalMapping : Generic failure \\r\\nAt line:1 char:190\\r\\n+ ... ser, $PWord;New-
SmbGlobalMapping -RemotePath $Env:smbremotepath -Cred ...\\r\\n+
~~~~~\\r\\n      + CategoryInfo          : NotSpecified:  
(MSFT_SmbGlobalMapping:ROOT/Microsoft/...mbGlobalMapping) [New-SmbGlobalMa \\r\\n      pping],  
CimException\\r\\n      + FullyQualifiedErrorId : HRESULT 0x80041001,New-SmbGlobalMapping\\r\\n      \\r\\n\""
```

This issue has been fixed in the following versions of Kubernetes:

KUBERNETES VERSION	FIXED VERSION
1.12	1.12.6 or later
1.13	1.13.4 or later
1.14 and later	N/A

### Azure Files mount fails because of storage account key changed

If your storage account key has changed, you may see Azure Files mount failures.

You can mitigate by manually updating the `azurerestorageaccountkey` field manually in an Azure file secret with your base64-encoded storage account key.

To encode your storage account key in base64, you can use `base64`. For example:

```
echo X+ALAAUgMhWHL7QmQ87E1kSF1qLKfgC03Guy7/xk9MyIg2w4Jzque60CVw2r/dm6v6E0DWHTnJUEJGVQoPaBc== | base64
```

To update your Azure secret file, use `kubectl edit secret`. For example:

```
kubectl edit secret azure-storage-account-{storage-account-name}-secret
```

After a few minutes, the agent node will retry the Azure File mount with the updated storage key.

### Cluster autoscaler fails to scale with error failed to fix node group sizes

If your cluster autoscaler isn't scaling up/down and you see an error like the below on the [cluster autoscaler logs](#).

```
E1114 09:58:55.367731 1 static_autoscaler.go:239] Failed to fix node group sizes: failed to decrease aks-default-35246781-vmss: attempt to delete existing nodes
```

This error is because of an upstream cluster autoscaler race condition. In such a case, cluster autoscaler ends with a different value than the one that is actually in the cluster. To get out of this state, disable and re-enable the [cluster autoscaler](#).

### Slow disk attachment, `GetAzureDiskLun` takes 10 to 15 minutes and you receive an error

On Kubernetes versions **older than 1.15.0**, you may receive an error such as `Error WaitForAttach Cannot find Lun for disk`. The workaround for this issue is to wait approximately 15 minutes and retry.

### Why do upgrades to Kubernetes 1.16 fail when using node labels with a kubernetes.io prefix

As of Kubernetes 1.16 [only a defined subset of labels with the kubernetes.io prefix](#) can be applied by the kubelet to nodes. AKS cannot remove active labels on your behalf without consent, as it may cause downtime to impacted workloads.

As a result, to mitigate this issue you can:

1. Upgrade your cluster control plane to 1.16 or higher
2. Add a new nodepool on 1.16 or higher without the unsupported kubernetes.io labels
3. Delete the older node pool

AKS is investigating the capability to mutate active labels on a node pool to improve this mitigation.

# Checking for Kubernetes best practices in your cluster

3/5/2021 • 2 minutes to read • [Edit Online](#)

There are several best practices that you should follow on your Kubernetes deployments to ensure the best performance and resilience for your applications. You can use the kube-advisor tool to look for deployments that aren't following those suggestions.

## About kube-advisor

The [kube-advisor tool](#) is a single container designed to be run on your cluster. It queries the Kubernetes API server for information about your deployments and returns a set of suggested improvements.

The kube-advisor tool can report on resource request and limits missing in PodSpecs for Windows applications as well as Linux applications, but the kube-advisor tool itself must be scheduled on a Linux pod. You can schedule a pod to run on a node pool with a specific OS using a [node selector](#) in the pod's configuration.

### NOTE

The kube-advisor tool is supported by Microsoft on a best-effort basis. Issues and suggestions should be filed on GitHub.

## Running kube-advisor

To run the tool on a cluster that is configured for [Kubernetes role-based access control \(Kubernetes RBAC\)](#), using the following commands. The first command creates a Kubernetes service account. The second command runs the tool in a pod using that service account and configures the pod for deletion after it exits.

```
kubectl apply -f https://raw.githubusercontent.com/Azure/kube-advisor/master/sa.yaml  
kubectl run --rm -i -t kubeadvisor --image=mcr.microsoft.com/aks/kubeadvisor --restart=Never --overrides='{"apiVersion": "v1", "spec": { "serviceAccountName": "kube-advisor" }}' --namespace default
```

If you aren't using Kubernetes RBAC, you can run the command as follows:

```
kubectl run --rm -i -t kubeadvisor --image=mcr.microsoft.com/aks/kubeadvisor --restart=Never
```

Within a few seconds, you should see a table describing potential improvements to your deployments.

Kubernetes Best Practice Checks					
Namespace	Pod Name	Pod CPU/Memory	Container	Issue	
postgresdemo	postgresdemo-worker-0	1m / 9440Ki	postgresdemo-worker	CPU Resource Limits Missing	
	postgresdemo-worker-1	1m / 9152Ki	postgresdemo-worker	Memory Resource Limits Missing	
				CPU Resource Limits Missing	

## Checks performed

The tool validates several Kubernetes best practices, each with their own suggested remediation.

### Resource requests and limits

Kubernetes supports defining [resource requests and limits on pod specifications](#). The request defines the minimum CPU and memory required to run the container. The limit defines the maximum CPU and memory that should be allowed.

By default, no requests or limits are set on pod specifications. This can lead to nodes being overscheduled and containers being starved. The kube-advisor tool highlights pods without requests and limits set.

## Cleaning up

If your cluster has Kubernetes RBAC enabled, you can clean up the `ClusterRoleBinding` after you've run the tool using the following command:

```
kubectl delete -f https://raw.githubusercontent.com/Azure/kube-advisor/master/sa.yaml
```

If you are running the tool against a cluster that is not Kubernetes RBAC-enabled, no cleanup is required.

## Next steps

- [Troubleshoot issues with Azure Kubernetes Service](#)

# Connect with SSH to Azure Kubernetes Service (AKS) cluster nodes for maintenance or troubleshooting

4/21/2021 • 7 minutes to read • [Edit Online](#)

Throughout the lifecycle of your Azure Kubernetes Service (AKS) cluster, you may need to access an AKS node. This access could be for maintenance, log collection, or other troubleshooting operations. You can access AKS nodes using SSH, including Windows Server nodes. You can also [connect to Windows Server nodes using remote desktop protocol \(RDP\) connections](#). For security purposes, the AKS nodes aren't exposed to the internet. To SSH to the AKS nodes, you use the private IP address.

This article shows you how to create an SSH connection with an AKS node using their private IP addresses.

## Before you begin

This article assumes that you have an existing AKS cluster. If you need an AKS cluster, see the AKS quickstart [using the Azure CLI](#) or [using the Azure portal](#).

By default, SSH keys are obtained, or generated, then added to nodes when you create an AKS cluster. This article shows you how to specify different SSH keys than the SSH keys used when you created your AKS cluster. The article also shows you how to determine the private IP address of your node and connect to it using SSH. If you don't need to specify a different SSH key, then you may skip the step for adding the SSH public key to the node.

This article also assumes you have an SSH key. You can create an SSH key using [macOS or Linux](#) or [Windows](#). If you use PuTTY Gen to create the key pair, save the key pair in an OpenSSH format rather than the default PuTTY private key format (.ppk file).

You also need the Azure CLI version 2.0.64 or later installed and configured. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

## Configure virtual machine scale set-based AKS clusters for SSH access

To configure your virtual machine scale set-based for SSH access, find the name of your cluster's virtual machine scale set and add your SSH public key to that scale set.

Use the `az aks show` command to get the resource group name of your AKS cluster, then the `az vmss list` command to get the name of your scale set.

```
CLUSTER_RESOURCE_GROUP=$(az aks show --resource-group myResourceGroup --name myAKSCluster --query nodeResourceGroup -o tsv)
SCALE_SET_NAME=$(az vmss list --resource-group $CLUSTER_RESOURCE_GROUP --query '[0].name' -o tsv)
```

The above example assigns the name of the cluster resource group for the *myAKSCluster* in *myResourceGroup* to *CLUSTER\_RESOURCE\_GROUP*. The example then uses *CLUSTER\_RESOURCE\_GROUP* to list the scale set name and assign it to *SCALE\_SET\_NAME*.

## IMPORTANT

At this time, you should only update your SSH keys for your virtual machine scale set-based AKS clusters using the Azure CLI.

For Linux nodes, SSH keys can currently only be added using the Azure CLI. If you want to connect to a Windows Server node using SSH, use the SSH keys provided when you created the AKS cluster and skip the next set of commands for adding your SSH public key. You will still need the IP address of the node you wish to troubleshoot, which is shown in the final command of this section. Alternatively, you can [connect to Windows Server nodes using remote desktop protocol \(RDP\) connections](#) instead of using SSH.

To add your SSH keys to the nodes in a virtual machine scale set, use the [az vmss extension set](#) and [az vmss update-instances](#) commands.

```
az vmss extension set \
--resource-group $CLUSTER_RESOURCE_GROUP \
--vmss-name $SCALE_SET_NAME \
--name VMAccessForLinux \
--publisher Microsoft.OSTCExtensions \
--version 1.4 \
--protected-settings "{\"username\":\"azureuser\", \"ssh_key\":$(cat ~/.ssh/id_rsa.pub)}"
```

```
az vmss update-instances --instance-ids '*' \
--resource-group $CLUSTER_RESOURCE_GROUP \
--name $SCALE_SET_NAME
```

The above example uses the *CLUSTER\_RESOURCE\_GROUP* and *SCALE\_SET\_NAME* variables from the previous commands. The above example also uses *~/.ssh/id\_rsa.pub* as the location for your SSH public key.

## NOTE

By default, the username for the AKS nodes is *azureuser*.

After you add your SSH public key to the scale set, you can SSH into a node virtual machine in that scale set using its IP address. View the private IP addresses of the AKS cluster nodes using the [kubectl get command](#).

```
kubectl get nodes -o wide
```

The follow example output shows the internal IP addresses of all the nodes in the cluster, including a Windows Server node.

```
$ kubectl get nodes -o wide
```

NAME	KERNEL-VERSION	CONTAINER-RUNTIME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE
aks-nodepool1-42485177-vmss000000	16.04.6 LTS	4.15.0-1040-azure	Ready	agent	18h	v1.12.7	10.240.0.4	<none>	Ubuntu
aksnpnwin000000	Server Datacenter		Ready	agent	13h	v1.12.7	10.240.0.67	<none>	Windows

Record the internal IP address of the node you wish to troubleshoot.

To access your node using SSH, follow the steps in [Create the SSH connection](#).

## Configure virtual machine availability set-based AKS clusters for SSH

## access

To configure your virtual machine availability set-based AKS cluster for SSH access, find the name of your cluster's Linux node, and add your SSH public key to that node.

Use the [az aks show](#) command to get the resource group name of your AKS cluster, then the [az vm list](#) command to list the virtual machine name of your cluster's Linux node.

```
CLUSTER_RESOURCE_GROUP=$(az aks show --resource-group myResourceGroup --name myAKSCluster --query nodeResourceGroup -o tsv)
az vm list --resource-group $CLUSTER_RESOURCE_GROUP -o table
```

The above example assigns the name of the cluster resource group for the *myAKSCluster* in *myResourceGroup* to *CLUSTER\_RESOURCE\_GROUP*. The example then uses *CLUSTER\_RESOURCE\_GROUP* to list the virtual machine name. The example output shows the name of the virtual machine:

Name	ResourceGroup	Location
aks-nodepool1-79590246-0	MC_myResourceGroupAKS_myAKSClusterRBAC_eastus	eastus

To add your SSH keys to the node, use the [az vm user update](#) command.

```
az vm user update \
--resource-group $CLUSTER_RESOURCE_GROUP \
--name aks-nodepool1-79590246-0 \
--username azureuser \
--ssh-key-value ~/.ssh/id_rsa.pub
```

The above example uses the *CLUSTER\_RESOURCE\_GROUP* variable and the node virtual machine name from previous commands. The above example also uses *~/.ssh/id\_rsa.pub* as the location for your SSH public key. You could also use the contents of your SSH public key instead of specifying a path.

### NOTE

By default, the username for the AKS nodes is *azureuser*.

After you add your SSH public key to the node virtual machine, you can SSH into that virtual machine using its IP address. View the private IP address of an AKS cluster node using the [az vm list-ip-addresses](#) command.

```
az vm list-ip-addresses --resource-group $CLUSTER_RESOURCE_GROUP -o table
```

The above example uses the *CLUSTER\_RESOURCE\_GROUP* variable set in the previous commands. The following example output shows the private IP addresses of the AKS nodes:

VirtualMachine	PrivateIPAddresses
aks-nodepool1-79590246-0	10.240.0.4

## Create the SSH connection

To create an SSH connection to an AKS node, you run a helper pod in your AKS cluster. This helper pod provides you with SSH access into the cluster and then additional SSH node access. To create and use this helper pod,

complete the following steps:

1. Run a `debian` container image and attach a terminal session to it. This container can be used to create an SSH session with any node in the AKS cluster:

```
kubectl run -it --rm aks-ssh --image=mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11
```

**TIP**

If you use Windows Server nodes, add a node selector to the command to schedule the Debian container on a Linux node:

```
kubectl run -it --rm aks-ssh --image=mcr.microsoft.com/aks/fundamental/base-ubuntu:v0.0.11 --overrides='{"apiVersion": "v1", "spec": {"nodeSelector": {"beta.kubernetes.io/os": "linux"}}}'
```

2. Once the terminal session is connected to the container, install an SSH client using `apt-get`:

```
apt-get update && apt-get install openssh-client -y
```

3. Open a new terminal window, not connected to your container, copy your private SSH key into the helper pod. This private key is used to create the SSH into the AKS node.

If needed, change `~/ssh/id_rsa` to location of your private SSH key:

```
kubectl cp ~/.ssh/id_rsa $(kubectl get pod -l run=aks-ssh -o jsonpath='{.items[0].metadata.name}'):/id_rsa
```

4. Return to the terminal session to your container, update the permissions on the copied `id_rsa` private SSH key so that it is user read-only:

```
chmod 0400 id_rsa
```

5. Create an SSH connection to your AKS node. Again, the default username for AKS nodes is `azureuser`. Accept the prompt to continue with the connection as the SSH key is first trusted. You are then provided with the bash prompt of your AKS node:

```
$ ssh -i id_rsa azureuser@10.240.0.4
ECDSA key fingerprint is SHA256:A6rnRkfpG21TaZ8XmQCCgdi9G/MYIMc+gFAuY9RUY70.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.240.0.4' (ECDSA) to the list of known hosts.

Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.15.0-1018-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

Get cloud support with Ubuntu Advantage Cloud Guest:
https://www.ubuntu.com/business/services/cloud

[...]
azureuser@aks-nodepool1-79590246-0:~$
```

## Remove SSH access

When done, `exit` the SSH session and then `exit` the interactive container session. When this container session closes, the pod used for SSH access from the AKS cluster is deleted.

## Next steps

If you need additional troubleshooting data, you can [view the kubelet logs](#) or [view the Kubernetes master node logs](#).

# Linux Performance Troubleshooting

11/2/2020 • 11 minutes to read • [Edit Online](#)

Resource exhaustion on Linux machines is a common issue and can manifest through a wide variety of symptoms. This document provides a high-level overview of the tools available to help diagnose such issues.

Many of these tools accept an interval on which to produce rolling output. This output format typically makes spotting patterns much easier. Where accepted, the example invocation will include `[interval]`.

Many of these tools have an extensive history and wide set of configuration options. This page provides only a simple subset of invocations to highlight common problems. The canonical source of information is always the reference documentation for each particular tool. That documentation will be much more thorough than what is provided here.

## Guidance

Be systematic in your approach to investigating performance issues. Two common approaches are USE (utilization, saturation, errors) and RED (rate, errors, duration). RED is typically used in the context of services for request-based monitoring. USE is typically used for monitoring resources: for each resource in a machine, monitor utilization, saturation, and errors. The four main kinds of resources on any machine are cpu, memory, disk, and network. High utilization, saturation, or error rates for any of these resources indicates a possible problem with the system. When a problem exists, investigate the root cause: why is disk IO latency high? Are the disks or virtual machine SKU throttled? What processes are writing to the devices, and to what files?

Some examples of common issues and indicators to diagnose them:

- IOPS throttling: use iostat to measure per-device IOPS. Ensure no individual disk is above its limit, and the sum for all disks is less than the limit for the virtual machine.
- Bandwidth throttling: use iostat as for IOPS, but measuring read/write throughput. Ensure both per-device and aggregate throughput are below the bandwidth limits.
- SNAT exhaustion: this can manifest as high active (outbound) connections in SAR.
- Packet loss: this can be measured by proxy via TCP retransmit count relative to sent/received count. Both `sar` and `netstat` can show this information.

## General

These tools are general purpose and cover basic system information. They are a good starting point for further investigation.

### `uptime`

```
$ uptime  
19:32:33 up 17 days, 12:36, 0 users, load average: 0.21, 0.77, 0.69
```

`uptime` provides system uptime and 1, 5, and 15-minute load averages. These load averages roughly correspond to threads doing work or waiting for uninterruptible work to complete. In absolute these numbers can be difficult to interpret, but measured over time they can tell us useful information:

- 1-minute average > 5-minute average means load is increasing.
- 1-minute average < 5-minute average means load is decreasing.

uptime can also illuminate why information is not available: the issue may have resolved on its own or by a restart before the user could access the machine.

Load averages higher than the number of CPU threads available may indicate a performance issue with a given workload.

## dmesg

```
$ dmesg | tail  
$ dmesg --level=err | tail
```

dmesg dumps the kernel buffer. Events like OOMKill add an entry to the kernel buffer. Finding an OOMKill or other resource exhaustion messages in dmesg logs is a strong indicator of a problem.

## top

```
$ top  
Tasks: 249 total, 1 running, 158 sleeping, 0 stopped, 0 zombie  
%Cpu(s): 2.2 us, 1.3 sy, 0.0 ni, 95.4 id, 1.0 wa, 0.0 hi, 0.2 si, 0.0 st  
KiB Mem : 65949064 total, 43415136 free, 2349328 used, 20184600 buff/cache  
KiB Swap: 0 total, 0 free, 0 used. 62739060 avail Mem  
  
 PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND  
116004 root      20   0 144400  41124  27028 S 11.8  0.1 248:45.45 coredns  
 4503 root      20   0 1677980 167964  89464 S  5.9  0.3 1326:25 kubelet  
    1 root      20   0 120212   6404   4044 S  0.0  0.0  48:20.38 systemd  
    ...
```

`top` provides a broad overview of current system state. The headers provide some useful aggregate information:

- state of tasks: running, sleeping, stopped.
- CPU utilization, in this case mostly showing idle time.
- total, free, and used system memory.

`top` may miss short-lived processes; alternatives like `htop` and `atop` provide similar interfaces while fixing some of these shortcomings.

## CPU

These tools provide CPU utilization information. This is especially useful with rolling output, where patterns become easy to spot.

## mpstat

```
$ mpstat -P ALL [interval]  
Linux 4.15.0-1064-azure (aks-main-10212767-vmss000001) 02/10/20          _x86_64_          (8 CPU)  
  
19:49:03    CPU    %usr    %nice    %sys %iowait    %irq    %soft    %steal    %guest    %gnice    %idle  
19:49:04    all    1.01    0.00    0.63    2.14    0.00    0.13    0.00    0.00    0.00    0.00    96.11  
19:49:04      0    1.01    0.00    1.01   17.17    0.00    0.00    0.00    0.00    0.00    0.00    80.81  
19:49:04      1    1.98    0.00    0.99    0.00    0.00    0.00    0.00    0.00    0.00    0.00    97.03  
19:49:04      2    1.01    0.00    0.00    0.00    0.00    1.01    0.00    0.00    0.00    0.00    97.98  
19:49:04      3    0.00    0.00    0.99    0.00    0.00    0.99    0.00    0.00    0.00    0.00    98.02  
19:49:04      4    1.98    0.00    1.98    0.00    0.00    0.00    0.00    0.00    0.00    0.00    96.04  
19:49:04      5    1.00    0.00    1.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    98.00  
19:49:04      6    1.00    0.00    1.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    98.00  
19:49:04      7    1.98    0.00    0.99    0.00    0.00    0.00    0.00    0.00    0.00    0.00    97.03
```

`mpstat` prints similar CPU information to `top`, but broken down by CPU thread. Seeing all cores at once can be useful for detecting highly imbalanced CPU usage, for example when a single threaded application uses one core at 100% utilization. This problem may be more difficult to spot when aggregated over all CPUs in the system.

## vmstat

```
$ vmstat [interval]
procs -----memory----- --swap-- -----io---- -system-- -----cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa st
2 0      0 43300372 545716 19691456    0   0    3   50   3   3   2   1 95   1   0
```

`vmstat` provides similar information `mpstat` and `top`, enumerating number of processes waiting on CPU (r column), memory statistics, and percent of CPU time spent in each work state.

## Memory

Memory is a very important, and thankfully easy, resource to track. Some tools can report both CPU and memory, like `vmstat`. But tools like `free` may still be useful for quick debugging.

## free

```
$ free -m
              total        used         free       shared  buff/cache   available
Mem:       64403        2338       42485           1       19579        61223
Swap:          0          0          0
```

`free` presents basic information about total memory as well as used and free memory. `vmstat` may be more useful even for basic memory analysis due to its ability to provide rolling output.

## Disk

These tools measure disk IOPS, wait queues, and total throughput.

## iostat

```
$ iostat -xy [interval] [count]
$ iostat -xy 1 1
Linux 4.15.0-1064-azure (aks-main-10212767-vmss000001) 02/10/20      _x86_64_      (8 CPU)

avg-cpu: %user  %nice %system %iowait  %steal  %idle
          3.42    0.00   2.92    1.90    0.00   91.76

Device:    rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s  avgrq-sz avgqu-sz  await r_await w_await
svctm %util
loop0      0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
0.00  0.00
sdb        0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
0.00  0.00
sda        0.00   56.00    0.00   65.00    0.00   504.00   15.51    0.01    3.02    0.00   3.02
0.12  0.80
scd0      0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
0.00  0.00
```

`iostat` provides deep insights into disk utilization. This invocation passes `-x` for extended statistics, `-y` to skip the initial output printing system averages since boot, and `1 1` to specify we want 1-second interval, ending after one block of output.

`iostat` exposes many useful statistics:

- `r/s` and `w/s` are reads per second and writes per second. The sum of these values is IOPS.
- `rkB/s` and `wkB/s` are kilobytes read/written per second. The sum of these values is throughput.
- `await` is the average iowait time in milliseconds for queued requests.
- `avgqu-sz` is the average queue size over the provided interval.

On an Azure VM:

- the sum of `r/s` and `w/s` for an individual block device may not exceed that disk's SKU limits.
- the sum of `rkB/s` and `wkB/s` for an individual block device may not exceed that disk's SKU limits
- the sum of `r/s` and `w/s` for all block devices may not exceed the limits for the VM SKU.
- the sum of `rkB/s` and `wkB/s` for all block devices may not exceed the limits for the VM SKU.

Note that the OS disk counts as a managed disk of the smallest SKU corresponding to its capacity. For example, a 1024GB OS Disk corresponds to a P30 disk. Ephemeral OS disks and temporary disks do not have individual disk limits; they are only limited by the full VM limits.

Non-zero values of `await` or `avgqu-sz` are also good indicators of IO contention.

## Network

These tools measure network statistics like throughput, transmission failures, and utilization. Deeper analysis can expose fine-grained TCP statistics about congestion and dropped packets.

`sar`

```
$ sar -n DEV [interval]
22:36:57      IFACE    rxpck/s    txpck/s    rxkB/s    txkB/s    rxcmp/s    txcmp/s    rxmcst/s    %ifutil
22:36:58      docker0    0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
22:36:58      azv604be75d832   1.00     9.00      0.06      1.04      0.00      0.00      0.00      0.00
22:36:58      azure0     68.00    79.00     27.79     52.79      0.00      0.00      0.00      0.00
22:36:58      azv4a8e7704a5b   202.00   207.00     37.51     21.86      0.00      0.00      0.00      0.00
22:36:58      azve83c28f6d1c   21.00     30.00     24.12      4.11      0.00      0.00      0.00      0.00
22:36:58      eth0       314.00   321.00     70.87    163.28      0.00      0.00      0.00      0.00
22:36:58      azva3128390bff   12.00     20.00      1.14      2.29      0.00      0.00      0.00      0.00
22:36:58      azvf46c95dde3   10.00     18.00     31.47      1.36      0.00      0.00      0.00      0.00
22:36:58      enP1s1      74.00    374.00     29.36    166.94      0.00      0.00      0.00      0.00
22:36:58      lo          0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
22:36:58      azvdbf16b0b2fc   9.00     19.00      3.36      1.18      0.00      0.00      0.00      0.00
```

`sar` is a powerful tool for a wide range of analysis. While this example uses its ability to measure network stats, it is equally powerful for measuring CPU and memory consumption. This example invokes `sar` with `-n` flag to specify the `DEV` (network device) keyword, displaying network throughput by device.

- The sum of `rxKb/s` and `txKb/s` is total throughput for a given device. When this value exceeds the limit for the provisioned Azure NIC, workloads on the machine will experience increased network latency.
- `%ifutil` measures utilization for a given device. As this value approaches 100%, workloads will experience increased network latency.

```
$ sar -n TCP,ETCP [interval]
Linux 4.15.0-1064-azure (aks-main-10212767-vmss000001) 02/10/20      _x86_64_      (8 CPU)

22:50:08    active/s passive/s    iseg/s    oseg/s
22:50:09        2.00       0.00     19.00     24.00

22:50:08    atmptf/s estres/s retrans/s isegerr/s   orsts/s
22:50:09        0.00       0.00       0.00       0.00       0.00

Average:    active/s passive/s    iseg/s    oseg/s
Average:        2.00       0.00     19.00     24.00

Average:    atmptf/s estres/s retrans/s isegerr/s   orsts/s
Average:        0.00       0.00       0.00       0.00       0.00
```

This invocation of `sar` uses the `TCP,ETCP` keywords to examine TCP connections. The third column of the last row, "retrans", is the number of TCP retransmits per second. High values for this field indicate an unreliable network connection. In The first and third rows, "active" means a connection originated from the local device, while "remote" indicates an incoming connection. A common issue on Azure is SNAT port exhaustion, which `sar` can help detect. SNAT port exhaustion would manifest as high "active" values, since the problem is due to a high rate of outbound, locally-initiated TCP connections.

As `sar` takes an interval, it prints rolling output and then prints final rows of output containing the average results from the invocation.

## netstat

```
$ netstat -s
Ip:
    71046295 total packets received
    78 forwarded
    0 incoming packets discarded
    71046066 incoming packets delivered
    83774622 requests sent out
    40 outgoing packets dropped

Icmp:
    103 ICMP messages received
    0 input ICMP message failed.
    ICMP input histogram:
        destination unreachable: 103
    412802 ICMP messages sent
    0 ICMP messages failed
    ICMP output histogram:
        destination unreachable: 412802

IcmpMsg:
    InType3: 103
    OutType3: 412802

Tcp:
    11487089 active connections openings
    592 passive connection openings
    1137 failed connection attempts
    404 connection resets received
    17 connections established
    70880911 segments received
    95242567 segments send out
    176658 segments retransmited
    3 bad segments received.
    163295 resets sent

Udp:
    164968 packets received
    84 packets to unknown port received.
    0 packet receive errors
    165082 packets sent

UdpLite:
```

```

TcpExt:
  5 resets received for embryonic SYN_RECV sockets
  1670559 TCP sockets finished time wait in fast timer
  95 packets rejects in established connections because of timestamp
  756870 delayed acks sent
  2236 delayed acks further delayed because of locked socket
  Quick ack mode was activated 479 times
  11983969 packet headers predicted
  25061447 acknowledgments not containing data payload received
  5596263 predicted acknowledgments
  19 times recovered from packet loss by selective acknowledgements
  Detected reordering 114 times using SACK
  Detected reordering 4 times using time stamp
  5 congestion windows fully recovered without slow start
  1 congestion windows partially recovered using Hoe heuristic
  5 congestion windows recovered without slow start by DSACK
  111 congestion windows recovered without slow start after partial ack
  73 fast retransmits
  26 retransmits in slow start
  311 other TCP timeouts
  TCPLossProbes: 198845
  TCPLossProbeRecovery: 147
  480 DSACKs sent for old packets
  175310 DSACKs received
  316 connections reset due to unexpected data
  272 connections reset due to early user close
  5 connections aborted due to timeout
  TCPDSACKIgnoredNoUndo: 8498
  TCPSpuriousRTOs: 1
  TCPSackShifted: 3
  TCPSackMerged: 9
  TCPSackShiftFallback: 177
  IPReversePathFilter: 4
  TCPRcvCoalesce: 1501457
  TCPOFQQueue: 9898
  TCPChallengeACK: 342
  TCPSYNChallenge: 3
  TCPSpuriousRtxHostQueues: 17
  TCPAutoCorking: 2315642
  TCPFromZeroWindowAdv: 483
  TCPToZeroWindowAdv: 483
  TCPWantZeroWindowAdv: 115
  TCPSynRetrans: 885
  TCPOrigDataSent: 51140171
  TCPHystartTrainDetect: 349
  TCPHystartTrainCwnd: 7045
  TCPHystartDelayDetect: 26
  TCPHystartDelayCwnd: 862
  TCPACKSkippedPAWS: 3
  TCPACKSkippedSeq: 4
  TCPKeepAlive: 62517

IpExt:
  InOctets: 36416951539
  OutOctets: 41520580596
  InNoECTPkts: 86631440
  InECT0Pkts: 14

```

`netstat` can introspect a wide variety of network stats, here invoked with summary output. There are many useful fields here depending on the issue. One useful field in the TCP section is "failed connection attempts". This may be an indication of SNAT port exhaustion or other issues making outbound connections. A high rate of retransmitted segments (also under the TCP section) may indicate issues with packet delivery.

# Check for Resource Health events impacting your AKS cluster (Preview)

11/2/2020 • 2 minutes to read • [Edit Online](#)

When running your container workloads on AKS, you want to ensure you can troubleshoot and fix problems as soon as they arise to minimize the impact on the availability of your workloads. [Azure Resource Health](#) gives you visibility into various health events that may cause your AKS cluster to be unavailable.

## IMPORTANT

AKS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. AKS previews are partially covered by customer support on a best-effort basis. As such, these features aren't meant for production use. AKS preview features aren't available in Azure Government or Azure China 21Vianet clouds. For more information, see the following support articles:

- [AKS support policies](#)
- [Azure support FAQ](#)

## Open Resource Health

### To access Resource Health for your AKS cluster:

- Navigate to your AKS cluster in the [Azure portal](#).
- Select **Resource Health** in the left navigation.

### To access Resource Health for all clusters on your subscription:

- Search for **Service Health** in the [Azure portal](#) and navigate to it.
- Select **Resource health** in the left navigation.
- Select your subscription and set the resource type to Azure Kubernetes Service (AKS).

The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with 'Microsoft Azure', a search bar, and various icons. Below the navigation bar, the main title is 'myAKSCluster | Resource health'. On the left, there's a sidebar with links like 'Dev Spaces', 'Deployment center (preview)', 'Policies (preview)', 'Properties', 'Locks', 'Export template', 'Monitoring' (which is expanded to show 'Insights', 'Alerts', 'Metrics', 'Diagnostic settings', 'Advisor recommendations', 'Logs', and 'Workbooks'), 'Support + troubleshooting' (which is expanded to show 'Resource health' and 'New support request'), and 'Home'. The main content area has a heading 'Resource health watches your resource and tells you if it's running as expected. [Learn more](#)'. It shows a 'Degraded' status with a message: 'Your service principal has expired or is invalid. Please check if you are using the correct secret or if the key has expired.' There's a link to 'Report incorrect health status'. Below this, a section titled 'What actions can you take?' lists '1. Renew the service principal credentials using the [Service principals with Azure Kubernetes Service documentation](#)'. The 'Health history' section shows a table of events over the last 4 weeks:

Date	Description
08/20/2020	1 health event(s)
08/19/2020	Available
08/18/2020	Available
08/17/2020	Available
08/16/2020	Available
08/15/2020	Available

## Check the health status

Azure Resource Health helps you diagnose and get support for service problems that affect your Azure resources. Resource Health reports on the current and past health of your resources and helps you determine if the problem is caused by a user-initiated action or a platform event.

Resource Health receives signals for your managed cluster to determine the cluster's health state. It examines the health state of your AKS cluster and reports actions required for each health signal. These signals range from auto-resolving issues, planned updates, unplanned health events, and unavailability caused by user-initiated actions. These signals are classified using the Azure Resource Health's health status: *Available*, *Unavailable*, *Unknown*, and *Degraded*.

- **Available:** When there are no known issues affecting your cluster's health, Resource Health reports your cluster as *Available*.
- **Unavailable:** When there is a platform or non-platform event affecting your cluster's health, Resource Health reports your cluster as *Unavailable*.
- **Unknown:** When there is a temporary connection loss to your cluster's health metrics, Resource Health reports your cluster as *Unknown*.
- **Degraded:** When there is a health issue requiring your action, Resource Health reports your cluster as *Degraded*.

For additional details on what each health status indicates, visit [Resource Health overview](#).

### View historical data

You can also view the past 30 days of historical Resource Health information in the **Health history** section.

## Next steps

Run checks on your cluster to further troubleshoot cluster issues by using [AKS Diagnostics](#).

# Azure Policy built-in definitions for Azure Kubernetes Service

5/14/2021 • 18 minutes to read • [Edit Online](#)

This page is an index of [Azure Policy built-in policy definitions](#) for Azure Kubernetes Service. For additional Azure Policy built-ins for other services, see [Azure Policy built-in definitions](#).

The name of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Version** column to view the source on the [Azure Policy GitHub repo](#).

## Initiatives

NAME	DESCRIPTION	POLICIES	VERSION
<a href="#">Kubernetes cluster pod security baseline standards for Linux-based workloads</a>	This initiative includes the policies for the Kubernetes cluster pod security baseline standards. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For instructions on using this policy, visit <a href="https://aka.ms/kubepolicydoc">https://aka.ms/kubepolicydoc</a> .	5	1.1.1
<a href="#">Kubernetes cluster pod security restricted standards for Linux-based workloads</a>	This initiative includes the policies for the Kubernetes cluster pod security restricted standards. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For instructions on using this policy, visit <a href="https://aka.ms/kubepolicydoc">https://aka.ms/kubepolicydoc</a> .	8	2.1.1

## Policy definitions

### Microsoft.ContainerService

NAME (AZURE PORTAL)	DESCRIPTION	EFFECT(S)	VERSION (GITHUB)

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Authorized IP ranges should be defined on Kubernetes Services</a>	Restrict access to the Kubernetes Service Management API by granting API access only to IP addresses in specific ranges. It is recommended to limit access to authorized IP ranges to ensure that only applications from allowed networks can access the cluster.	Audit, Disabled	2.0.1
<a href="#">Azure Kubernetes Service Private Clusters should be enabled</a>	Enable the private cluster feature for your Azure Kubernetes Service cluster to ensure network traffic between your API server and your node pools remains on the private network only. This is a common requirement in many regulatory and industry compliance standards.	Audit, Deny, Disabled	1.0.0
<a href="#">Azure Policy Add-on for Kubernetes service (AKS) should be installed and enabled on your clusters</a>	Azure Policy Add-on for Kubernetes service (AKS) extends Gatekeeper v3, an admission controller webhook for Open Policy Agent (OPA), to apply at-scale enforcements and safeguards on your clusters in a centralized, consistent manner.	Audit, Disabled	1.0.2
<a href="#">Both operating systems and data disks in Azure Kubernetes Service clusters should be encrypted by customer-managed keys</a>	Encrypting OS and data disks using customer-managed keys provides more control and greater flexibility in key management. This is a common requirement in many regulatory and industry compliance standards.	Audit, Deny, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
Configure Kubernetes clusters with specified GitOps configuration using HTTPS secrets	Deploy a 'sourceControlConfiguration' to Kubernetes clusters to assure that the clusters get their source of truth for workloads and configurations from the defined git repo. This definition requires HTTPS user and key secrets stored in Key Vault. For instructions, visit <a href="https://aka.ms/K8sGitOpsPolicy">https://aka.ms/K8sGitOpsPolicy</a> .	deployIfNotExists, auditIfNotExists, disabled	1.0.0
Configure Kubernetes clusters with specified GitOps configuration using no secrets	Deploy a 'sourceControlConfiguration' to Kubernetes clusters to assure that the clusters get their source of truth for workloads and configurations from the defined git repo. This definition requires no secrets. For instructions, visit <a href="https://aka.ms/K8sGitOpsPolicy">https://aka.ms/K8sGitOpsPolicy</a> .	deployIfNotExists, auditIfNotExists, disabled	1.0.0
Configure Kubernetes clusters with specified GitOps configuration using SSH secrets	Deploy a 'sourceControlConfiguration' to Kubernetes clusters to assure that the clusters get their source of truth for workloads and configurations from the defined git repo. This definition requires a SSH private key secret in Key Vault. For instructions, visit <a href="https://aka.ms/K8sGitOpsPolicy">https://aka.ms/K8sGitOpsPolicy</a> .	deployIfNotExists, auditIfNotExists, disabled	1.0.0
Deploy - Configure diagnostic settings for Azure Kubernetes Service to Log Analytics workspace	Deploys the diagnostic settings for Azure Kubernetes Service to stream resource logs to a Log Analytics workspace.	DeployIfNotExists, Disabled	1.0.0
Deploy Azure Policy Add-on to Azure Kubernetes Service clusters	Use Azure Policy Add-on to manage and report on the compliance state of your Azure Kubernetes Service (AKS) clusters. For more information, see <a href="https://aka.ms/akspolicydoc">https://aka.ms/akspolicydoc</a> .	deployIfNotExists	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Kubernetes cluster containers CPU and memory resource limits should not exceed the specified limits</a>	Enforce container CPU and memory resource limits to prevent resource exhaustion attacks in a Kubernetes cluster. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	6.0.0
<a href="#">Kubernetes cluster containers should not share host process ID or host IPC namespace</a>	Block pod containers from sharing the host process ID namespace and host IPC namespace in a Kubernetes cluster. This recommendation is part of CIS 5.2.2 and CIS 5.2.3 which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	3.0.0
<a href="#">Kubernetes cluster containers should only listen on allowed ports</a>	Restrict containers to listen only on allowed ports to secure access to the Kubernetes cluster. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	6.1.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Kubernetes cluster containers should only use allowed AppArmor profiles</a>	<p>Containers should only use allowed AppArmor profiles in a Kubernetes cluster. This recommendation is part of Pod Security Policies which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicydoc">https://aka.ms/kubepolicydoc</a>.</p>	audit, deny, disabled	3.0.0
<a href="#">Kubernetes cluster containers should only use allowed capabilities</a>	<p>Restrict the capabilities to reduce the attack surface of containers in a Kubernetes cluster. This recommendation is part of CIS 5.2.8 and CIS 5.2.9 which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicydoc">https://aka.ms/kubepolicydoc</a>.</p>	audit, deny, disabled	3.0.0
<a href="#">Kubernetes cluster containers should only use allowed images</a>	<p>Use images from trusted registries to reduce the Kubernetes cluster's exposure risk to unknown vulnerabilities, security issues and malicious images. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicydoc">https://aka.ms/kubepolicydoc</a>.</p>	audit, deny, disabled	6.1.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Kubernetes cluster containers should only use allowed seccomp profiles</a>	<p>Pod containers can only use allowed seccomp profiles in a Kubernetes cluster. This recommendation is part of Pod Security Policies which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicydoc">https://aka.ms/kubepolicydoc</a>.</p>	audit, deny, disabled	3.0.0
<a href="#">Kubernetes cluster containers should run with a read only root file system</a>	<p>Run containers with a read only root file system to protect from changes at run-time with malicious binaries being added to PATH in a Kubernetes cluster. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicydoc">https://aka.ms/kubepolicydoc</a>.</p>	audit, deny, disabled	3.0.0
<a href="#">Kubernetes cluster pod FlexVolume volumes should only use allowed drivers</a>	<p>Pod FlexVolume volumes should only use allowed drivers in a Kubernetes cluster. This recommendation is part of Pod Security Policies which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicydoc">https://aka.ms/kubepolicydoc</a>.</p>	audit, deny, disabled	3.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Kubernetes cluster pod hostPath volumes should only use allowed host paths</a>	<p>Limit pod HostPath volume mounts to the allowed host paths in a Kubernetes Cluster. This recommendation is part of Pod Security Policies which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a>.</p>	audit, deny, disabled	3.0.0
<a href="#">Kubernetes cluster pods and containers should only run with approved user and group IDs</a>	<p>Control the user, primary group, supplemental group and file system group IDs that pods and containers can use to run in a Kubernetes Cluster. This recommendation is part of Pod Security Policies which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a>.</p>	audit, deny, disabled	3.0.0
<a href="#">Kubernetes cluster pods should only use allowed volume types</a>	<p>Pods can only use allowed volume types in a Kubernetes cluster. This recommendation is part of Pod Security Policies which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a>.</p>	audit, deny, disabled	3.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Kubernetes cluster pods should only use approved host network and port range</a>	Restrict pod access to the host network and the allowable host port range in a Kubernetes cluster. This recommendation is part of CIS 5.2.4 which is intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	3.0.0
<a href="#">Kubernetes cluster pods should use specified labels</a>	Use specified labels to identify the pods in a Kubernetes cluster. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	6.0.0
<a href="#">Kubernetes cluster services should listen only on allowed ports</a>	Restrict services to listen only on allowed ports to secure access to the Kubernetes cluster. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	6.1.0
<a href="#">Kubernetes cluster services should only use allowed external IPs</a>	Use allowed external IPs to avoid the potential attack (CVE-2020-8554) in a Kubernetes cluster. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	3.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Kubernetes cluster should not allow privileged containers</a>	Do not allow privileged containers creation in a Kubernetes cluster. This recommendation is part of CIS 5.2.1 which is intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	7.0.0
<a href="#">Kubernetes clusters should be accessible only over HTTPS</a>	Use of HTTPS ensures authentication and protects data in transit from network layer eavesdropping attacks. This capability is currently generally available for Kubernetes Service (AKS), and in preview for AKS Engine and Azure Arc enabled Kubernetes. For more info, visit <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a>	audit, deny, disabled	6.0.0
<a href="#">Kubernetes clusters should disable automounting API credentials</a>	Disable automounting API credentials to prevent a potentially compromised Pod resource to run API commands against Kubernetes clusters. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	2.0.0-preview
<a href="#">Kubernetes clusters should not allow container privilege escalation</a>	Do not allow containers to run with privilege escalation to root in a Kubernetes cluster. This recommendation is part of CIS 5.2.5 which is intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	3.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Kubernetes clusters should not grant CAP_SYS_ADMIN security capabilities</a>	To reduce the attack surface of your containers, restrict CAP_SYS_ADMIN Linux capabilities. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	2.0.0-preview
<a href="#">Kubernetes clusters should not use specific security capabilities</a>	Prevent specific security capabilities in Kubernetes clusters to prevent ungranted privileges on the Pod resource. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	2.0.0-preview
<a href="#">Kubernetes clusters should not use the default namespace</a>	Prevent usage of the default namespace in Kubernetes clusters to protect against unauthorized access for ConfigMap, Pod, Secret, Service, and ServiceAccount resource types. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	2.0.0-preview
<a href="#">Kubernetes clusters should use internal load balancers</a>	Use internal load balancers to make a Kubernetes service accessible only to applications running in the same virtual network as the Kubernetes cluster. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	6.0.0
<a href="#">Kubernetes Services should be upgraded to a non-vulnerable Kubernetes version</a>	Upgrade your Kubernetes service cluster to a later Kubernetes version to protect against known vulnerabilities in your current Kubernetes version. Vulnerability CVE-2019-9946 has been patched in Kubernetes versions 1.11.9+, 1.12.7+, 1.13.5+, and 1.14.0+	Audit, Disabled	1.0.2

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Role-Based Access Control (RBAC) should be used on Kubernetes Services</a>	To provide granular filtering on the actions that users can perform, use Role-Based Access Control (RBAC) to manage permissions in Kubernetes Service Clusters and configure relevant authorization policies.	Audit, Disabled	1.0.2
<a href="#">Temp disks and cache for agent node pools in Azure Kubernetes Service clusters should be encrypted at host</a>	To enhance data security, the data stored on the virtual machine (VM) host of your Azure Kubernetes Service nodes VMs should be encrypted at rest. This is a common requirement in many regulatory and industry compliance standards.	Audit, Deny, Disabled	1.0.0

## AKS Engine

NAME (AZURE PORTAL)	DESCRIPTION	EFFECT(S)	VERSION (GITHUB)
<a href="#">Kubernetes cluster containers CPU and memory resource limits should not exceed the specified limits</a>	Enforce container CPU and memory resource limits to prevent resource exhaustion attacks in a Kubernetes cluster. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	6.0.0
<a href="#">Kubernetes cluster containers should not share host process ID or host IPC namespace</a>	Block pod containers from sharing the host process ID namespace and host IPC namespace in a Kubernetes cluster. This recommendation is part of CIS 5.2.2 and CIS 5.2.3 which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	3.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Kubernetes cluster containers should not use forbidden sysctl interfaces</a>	<p>Containers should not use forbidden sysctl interfaces in a Kubernetes cluster. This recommendation is part of Pod Security Policies which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a>.</p>	audit, deny, disabled	4.0.0
<a href="#">Kubernetes cluster containers should only listen on allowed ports</a>	<p>Restrict containers to listen only on allowed ports to secure access to the Kubernetes cluster. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a>.</p>	audit, deny, disabled	6.1.0
<a href="#">Kubernetes cluster containers should only use allowed AppArmor profiles</a>	<p>Containers should only use allowed AppArmor profiles in a Kubernetes cluster. This recommendation is part of Pod Security Policies which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a>.</p>	audit, deny, disabled	3.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Kubernetes cluster containers should only use allowed capabilities</a>	Restrict the capabilities to reduce the attack surface of containers in a Kubernetes cluster. This recommendation is part of CIS 5.2.8 and CIS 5.2.9 which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	3.0.0
<a href="#">Kubernetes cluster containers should only use allowed images</a>	Use images from trusted registries to reduce the Kubernetes cluster's exposure risk to unknown vulnerabilities, security issues and malicious images. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	6.1.0
<a href="#">Kubernetes cluster containers should only use allowed ProcMountType</a>	Pod containers can only use allowed ProcMountTypes in a Kubernetes cluster. This recommendation is part of Pod Security Policies which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	4.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Kubernetes cluster containers should only use allowed seccomp profiles</a>	<p>Pod containers can only use allowed seccomp profiles in a Kubernetes cluster. This recommendation is part of Pod Security Policies which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicydoc">https://aka.ms/kubepolicydoc</a>.</p>	audit, deny, disabled	3.0.0
<a href="#">Kubernetes cluster containers should run with a read only root file system</a>	<p>Run containers with a read only root file system to protect from changes at run-time with malicious binaries being added to PATH in a Kubernetes cluster. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicydoc">https://aka.ms/kubepolicydoc</a>.</p>	audit, deny, disabled	3.0.0
<a href="#">Kubernetes cluster pod FlexVolume volumes should only use allowed drivers</a>	<p>Pod FlexVolume volumes should only use allowed drivers in a Kubernetes cluster. This recommendation is part of Pod Security Policies which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicydoc">https://aka.ms/kubepolicydoc</a>.</p>	audit, deny, disabled	3.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Kubernetes cluster pod hostPath volumes should only use allowed host paths</a>	<p>Limit pod HostPath volume mounts to the allowed host paths in a Kubernetes Cluster. This recommendation is part of Pod Security Policies which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a>.</p>	audit, deny, disabled	3.0.0
<a href="#">Kubernetes cluster pods and containers should only run with approved user and group IDs</a>	<p>Control the user, primary group, supplemental group and file system group IDs that pods and containers can use to run in a Kubernetes Cluster. This recommendation is part of Pod Security Policies which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a>.</p>	audit, deny, disabled	3.0.0
<a href="#">Kubernetes cluster pods and containers should only use allowed SELinux options</a>	<p>Pods and containers should only use allowed SELinux options in a Kubernetes cluster. This recommendation is part of Pod Security Policies which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a>.</p>	audit, deny, disabled	4.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Kubernetes cluster pods should only use allowed volume types</a>	<p>Pods can only use allowed volume types in a Kubernetes cluster. This recommendation is part of Pod Security Policies which are intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicydoc">https://aka.ms/kubepolicydoc</a>.</p>	audit, deny, disabled	3.0.0
<a href="#">Kubernetes cluster pods should only use approved host network and port range</a>	<p>Restrict pod access to the host network and the allowable host port range in a Kubernetes cluster. This recommendation is part of CIS 5.2.4 which is intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicydoc">https://aka.ms/kubepolicydoc</a>.</p>	audit, deny, disabled	3.0.0
<a href="#">Kubernetes cluster pods should use specified labels</a>	<p>Use specified labels to identify the pods in a Kubernetes cluster. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicydoc">https://aka.ms/kubepolicydoc</a>.</p>	audit, deny, disabled	6.0.0
<a href="#">Kubernetes cluster services should listen only on allowed ports</a>	<p>Restrict services to listen only on allowed ports to secure access to the Kubernetes cluster. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicydoc">https://aka.ms/kubepolicydoc</a>.</p>	audit, deny, disabled	6.1.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Kubernetes cluster services should only use allowed external IPs</a>	Use allowed external IPs to avoid the potential attack (CVE-2020-8554) in a Kubernetes cluster. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	3.0.0
<a href="#">Kubernetes cluster should not allow privileged containers</a>	Do not allow privileged containers creation in a Kubernetes cluster. This recommendation is part of CIS 5.2.1 which is intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	7.0.0
<a href="#">Kubernetes clusters should be accessible only over HTTPS</a>	Use of HTTPS ensures authentication and protects data in transit from network layer eavesdropping attacks. This capability is currently generally available for Kubernetes Service (AKS), and in preview for AKS Engine and Azure Arc enabled Kubernetes. For more info, visit <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a>	audit, deny, disabled	6.0.0
<a href="#">Kubernetes clusters should disable automounting API credentials</a>	Disable automounting API credentials to prevent a potentially compromised Pod resource to run API commands against Kubernetes clusters. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	2.0.0-preview

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Kubernetes clusters should not allow container privilege escalation</a>	Do not allow containers to run with privilege escalation to root in a Kubernetes cluster. This recommendation is part of CIS 5.2.5 which is intended to improve the security of your Kubernetes environments. This policy is generally available for Kubernetes Service (AKS), and preview for AKS Engine and Azure Arc enabled Kubernetes. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	3.0.0
<a href="#">Kubernetes clusters should not grant CAP_SYS_ADMIN security capabilities</a>	To reduce the attack surface of your containers, restrict CAP_SYS_ADMIN Linux capabilities. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	2.0.0-preview
<a href="#">Kubernetes clusters should not use specific security capabilities</a>	Prevent specific security capabilities in Kubernetes clusters to prevent ungranted privileges on the Pod resource. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	2.0.0-preview
<a href="#">Kubernetes clusters should not use the default namespace</a>	Prevent usage of the default namespace in Kubernetes clusters to protect against unauthorized access for ConfigMap, Pod, Secret, Service, and ServiceAccount resource types. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	2.0.0-preview
<a href="#">Kubernetes clusters should use internal load balancers</a>	Use internal load balancers to make a Kubernetes service accessible only to applications running in the same virtual network as the Kubernetes cluster. For more information, see <a href="https://aka.ms/kubepolicyd_oc">https://aka.ms/kubepolicyd_oc</a> .	audit, deny, disabled	6.0.0

## Next steps

- See the built-ins on the [Azure Policy GitHub repo](#).
- Review the [Azure Policy definition structure](#).
- Review [Understanding policy effects](#).

# Support policies for Azure Kubernetes Service

5/10/2021 • 9 minutes to read • [Edit Online](#)

This article provides details about technical support policies and limitations for Azure Kubernetes Service (AKS). The article also details agent node management, managed control plane components, third-party open-source components, and security or patch management.

## Service updates and releases

- For release information, see [AKS release notes](#).
- For information on features in preview, see [AKS preview features and related projects](#).

## Managed features in AKS

Base infrastructure as a service (IaaS) cloud components, such as compute or networking components, allow you access to low-level controls and customization options. By contrast, AKS provides a turnkey Kubernetes deployment that gives you the common set of configurations and capabilities you need for your cluster. As an AKS user, you have limited customization and deployment options. In exchange, you don't need to worry about or manage Kubernetes clusters directly.

With AKS, you get a fully managed *control plane*. The control plane contains all of the components and services you need to operate and provide Kubernetes clusters to end users. All Kubernetes components are maintained and operated by Microsoft.

Microsoft manages and monitors the following components through the control pane:

- Kubelet or Kubernetes API servers
- Etcd or a compatible key-value store, providing Quality of Service (QoS), scalability, and runtime
- DNS services (for example, kube-dns or CoreDNS)
- Kubernetes proxy or networking
- Any additional addon or system component running in the kube-system namespace

AKS isn't a Platform-as-a-Service (PaaS) solution. Some components, such as agent nodes, have *shared responsibility*, where users must help maintain the AKS cluster. User input is required, for example, to apply an agent node operating system (OS) security patch.

The services are *managed* in the sense that Microsoft and the AKS team deploys, operates, and is responsible for service availability and functionality. Customers can't alter these managed components. Microsoft limits customization to ensure a consistent and scalable user experience. For a fully customizable solution, see [AKS Engine](#).

## Shared responsibility

When a cluster is created, you define the Kubernetes agent nodes that AKS creates. Your workloads are executed on these nodes.

Because your agent nodes execute private code and store sensitive data, Microsoft Support can access them only in a very limited way. Microsoft Support can't sign in to, execute commands in, or view logs for these nodes without your express permission or assistance.

Any modification done directly to the agent nodes using any of the IaaS APIs renders the cluster unsupportable.

Any modification done to the agent nodes must be done using Kubernetes-native mechanisms such as [Daemon Sets](#).

Similarly, while you may add any metadata to the cluster and nodes, such as tags and labels, changing any of the system created metadata will render the cluster unsupported.

## AKS support coverage

Microsoft provides technical support for the following examples:

- Connectivity to all Kubernetes components that the Kubernetes service provides and supports, such as the API server.
- Management, uptime, QoS, and operations of Kubernetes control plane services (Kubernetes control plane, API server, etcd, and coreDNS, for example).
- Etcd data store. Support includes automated, transparent backups of all etcd data every 30 minutes for disaster planning and cluster state restoration. These backups aren't directly available to you or any users. They ensure data reliability and consistency. Etcd. on-demand rollback or restore is not supported as a feature.
- Any integration points in the Azure cloud provider driver for Kubernetes. These include integrations into other Azure services such as load balancers, persistent volumes, or networking (Kubernetes and Azure CNI).
- Questions or issues about customization of control plane components such as the Kubernetes API server, etcd, and coreDNS.
- Issues about networking, such as Azure CNI, kubenet, or other network access and functionality issues. Issues could include DNS resolution, packet loss, routing, and so on. Microsoft supports various networking scenarios:
  - Kubenet and Azure CNI using managed VNETs or with custom (bring your own) subnets.
  - Connectivity to other Azure services and applications
  - Ingress controllers and ingress or load balancer configurations
  - Network performance and latency
  - [Network policies](#)

### NOTE

Any cluster actions taken by Microsoft/AKS are made with user consent under a built-in Kubernetes role `aks-service` and built-in role binding `aks-service-rolebinding`. This role enables AKS to troubleshoot and diagnose cluster issues, but can't modify permissions nor create roles or role bindings, or other high privilege actions. Role access is only enabled under active support tickets with just-in-time (JIT) access.

Microsoft doesn't provide technical support for the following examples:

- Questions about how to use Kubernetes. For example, Microsoft Support doesn't provide advice on how to create custom ingress controllers, use application workloads, or apply third-party or open-source software packages or tools.

### NOTE

Microsoft Support can advise on AKS cluster functionality, customization, and tuning (for example, Kubernetes operations issues and procedures).

- Third-party open-source projects that aren't provided as part of the Kubernetes control plane or deployed with AKS clusters. These projects might include Istio, Helm, Envoy, or others.

**NOTE**

Microsoft can provide best-effort support for third-party open-source projects such as Helm. Where the third-party open-source tool integrates with the Kubernetes Azure cloud provider or other AKS-specific bugs, Microsoft supports examples and applications from Microsoft documentation.

- Third-party closed-source software. This software can include security scanning tools and networking devices or software.
- Network customizations other than the ones listed in the [AKS documentation](#).

## AKS support coverage for agent nodes

### Microsoft responsibilities for AKS agent nodes

Microsoft and users share responsibility for Kubernetes agent nodes where:

- The base OS image has required additions (such as monitoring and networking agents).
- The agent nodes receive OS patches automatically.
- Issues with the Kubernetes control plane components that run on the agent nodes are automatically remediated. These components include the below:
  - `Kube-proxy`
  - Networking tunnels that provide communication paths to the Kubernetes master components
  - `Kubelet`
  - `Moby` or `ContainerD`

**NOTE**

If an agent node is not operational, AKS might restart individual components or the entire agent node. These restart operations are automated and provide auto-remediation for common issues. If you want to know more about the auto-remediation mechanisms, see [Node Auto-Repair](#)

### Customer responsibilities for AKS agent nodes

Microsoft provides patches and new images for your image nodes weekly, but doesn't automatically patch them by default. To keep your agent node OS and runtime components patched, you should keep a regular [node image upgrade](#) schedule or automate it.

Similarly, AKS regularly releases new kubernetes patches and minor versions. These updates can contain security or functionality improvements to Kubernetes. You're responsible to keep your clusters' kubernetes version updated and according to the [AKS Kubernetes Support Version Policy](#).

### User customization of agent nodes

**NOTE**

AKS agent nodes appear in the Azure portal as regular Azure IaaS resources. But these virtual machines are deployed into a custom Azure resource group (usually prefixed with MC\_\*) . You cannot change the base OS image or do any direct customizations to these nodes using the IaaS APIs or resources. Any custom changes that are not done via the AKS API will not persist through an upgrade, scale, update or reboot. Avoid performing changes to the agent nodes unless Microsoft Support directs you to make changes.

AKS manages the lifecycle and operations of agent nodes on your behalf - modifying the IaaS resources associated with the agent nodes is **not supported**. An example of an unsupported operation is customizing a node pool virtual machine scale set by manually changing configurations through the virtual machine scale set

portal or API.

For workload-specific configurations or packages, AKS recommends using [Kubernetes daemon sets](#).

Using Kubernetes privileged [daemon sets](#) and init containers enables you to tune/modify or install 3rd party software on cluster agent nodes. Examples of such customizations include adding custom security scanning software or updating sysctl settings.

While this path is recommended if the above requirements apply, AKS engineering and support cannot assist in troubleshooting or diagnosing modifications that render the node unavailable due to a custom deployed [daemon set](#).

### Security issues and patching

If a security flaw is found in one or more of the managed components of AKS, the AKS team will patch all affected clusters to mitigate the issue. Alternatively, the team will give users upgrade guidance.

For agent nodes affected by a security flaw, Microsoft will notify you with details on the impact and the steps to fix or mitigate the security issue (normally a node image upgrade or a cluster patch upgrade).

### Node maintenance and access

Although you can sign in to and change agent nodes, doing this operation is discouraged because changes can make a cluster unsupportable.

## Network ports, access, and NSGs

You may only customize the NSGs on custom subnets. You may not customize NSGs on managed subnets or at the NIC level of the agent nodes. AKS has egress requirements to specific endpoints, to control egress and ensure the necessary connectivity, see [limit egress traffic](#).

## Stopped or de-allocated clusters

As stated earlier, manually de-allocating all cluster nodes via the IaaS APIs/CLI/portal renders the cluster out of support. The only supported way to stop/de-allocate all nodes is to [stop the AKS cluster](#), which preserves the cluster state for up to 12 months.

Clusters that are stopped for more than 12 months will no longer preserve state.

Clusters that are de-allocated outside of the AKS APIs have no state preservation guarantees. The control planes for clusters in this state will be archived after 30 days, and deleted after 12 months.

AKS reserves the right to archive control planes that have been configured out of support guidelines for extended periods equal to and beyond 30 days. AKS maintains backups of cluster etcd metadata and can readily reallocate the cluster. This reallocation can be initiated by any PUT operation bringing the cluster back into support, such as an upgrade or scale to active agent nodes.

If your subscription is suspended or deleted, your cluster's control plane and state will be deleted after 90 days.

## Unsupported alpha and beta Kubernetes features

AKS only supports stable and beta features within the upstream Kubernetes project. Unless otherwise documented, AKS doesn't support any alpha feature that is available in the upstream Kubernetes project.

## Preview features or feature flags

For features and functionality that requires extended testing and user feedback, Microsoft releases new preview features or features behind a feature flag. Consider these features as prerelease or beta features.

Preview features or feature-flag features aren't meant for production. Ongoing changes in APIs and behavior, bug fixes, and other changes can result in unstable clusters and downtime.

Features in public preview are fall under 'best effort' support as these features are in preview and not meant for production and are supported by the AKS technical support teams during business hours only. For more information, see:

- [Azure Support FAQ](#)

## Upstream bugs and issues

Given the speed of development in the upstream Kubernetes project, bugs invariably arise. Some of these bugs can't be patched or worked around within the AKS system. Instead, bug fixes require larger patches to upstream projects (such as Kubernetes, node or agent operating systems, and kernel). For components that Microsoft owns (such as the Azure cloud provider), AKS and Azure personnel are committed to fixing issues upstream in the community.

When a technical support issue is root-caused by one or more upstream bugs, AKS support and engineering teams will:

- Identify and link the upstream bugs with any supporting details to help explain why this issue affects your cluster or workload. Customers receive links to the required repositories so they can watch the issues and see when a new release will provide fixes.
- Provide potential workarounds or mitigation. If the issue can be mitigated, a [known issue](#) will be filed in the AKS repository. The known-issue filing explains:
  - The issue, including links to upstream bugs.
  - The workaround and details about an upgrade or another persistence of the solution.
  - Rough timelines for the issue's inclusion, based on the upstream release cadence.

# Frequently asked questions about Azure Kubernetes Service (AKS)

5/17/2021 • 14 minutes to read • [Edit Online](#)

This article addresses frequent questions about Azure Kubernetes Service (AKS).

## Which Azure regions currently provide AKS?

For a complete list of available regions, see [AKS regions and availability](#).

## Can I spread an AKS cluster across regions?

No. AKS clusters are regional resources and can't span regions. See [best practices for business continuity and disaster recovery](#) for guidance on how to create an architecture that includes multiple regions.

## Can I spread an AKS cluster across availability zones?

Yes. You can deploy an AKS cluster across one or more [availability zones](#) in [regions that support them](#).

## Can I limit who has access to the Kubernetes API server?

Yes. There are two options for limiting access to the API server:

- Use [API Server Authorized IP Ranges](#) if you want to maintain a public endpoint for the API server but restrict access to a set of trusted IP ranges.
- Use a [private cluster](#) if you want to limit the API server to *only* be accessible from within your virtual network.

## Can I have different VM sizes in a single cluster?

Yes, you can use different virtual machine sizes in your AKS cluster by creating [multiple node pools](#).

## Are security updates applied to AKS agent nodes?

Azure automatically applies security patches to the Linux nodes in your cluster on a nightly schedule. However, you're responsible for ensuring that those Linux nodes are rebooted as required. You have several options for rebooting nodes:

- Manually, through the Azure portal or the Azure CLI.
- By upgrading your AKS cluster. The cluster upgrades [cordon and drain nodes](#) automatically and then bring a new node online with the latest Ubuntu image and a new patch version or a minor Kubernetes version. For more information, see [Upgrade an AKS cluster](#).
- By using [node image upgrade](#).

### Windows Server nodes

For Windows Server nodes, Windows Update does not automatically run and apply the latest updates. On a regular schedule around the Windows Update release cycle and your own validation process, you should perform an upgrade on the cluster and the Windows Server node pool(s) in your AKS cluster. This upgrade process creates nodes that run the latest Windows Server image and patches, then removes the older nodes. For more information on this process, see [Upgrade a node pool in AKS](#).

## Why are two resource groups created with AKS?

AKS builds upon a number of Azure infrastructure resources, including virtual machine scale sets, virtual networks, and managed disks. This enables you to leverage many of the core capabilities of the Azure platform within the managed Kubernetes environment provided by AKS. For example, most Azure virtual machine types can be used directly with AKS and Azure Reservations can be used to receive discounts on those resources automatically.

To enable this architecture, each AKS deployment spans two resource groups:

1. You create the first resource group. This group contains only the Kubernetes service resource. The AKS resource provider automatically creates the second resource group during deployment. An example of the second resource group is *MC\_myResourceGroup\_myAKSCluster\_eastus*. For information on how to specify the name of this second resource group, see the next section.
2. The second resource group, known as the *node resource group*, contains all of the infrastructure resources associated with the cluster. These resources include the Kubernetes node VMs, virtual networking, and storage. By default, the node resource group has a name like *MC\_myResourceGroup\_myAKSCluster\_eastus*. AKS automatically deletes the node resource whenever the cluster is deleted, so it should only be used for resources that share the cluster's lifecycle.

## Can I provide my own name for the AKS node resource group?

Yes. By default, AKS will name the node resource group *MC\_resourcegroupname\_clusternamespace\_location*, but you can also provide your own name.

To specify your own resource group name, install the [aks-preview](#) Azure CLI extension version 0.3.2 or later. When you create an AKS cluster by using the `az aks create` command, use the `--node-resource-group` parameter and specify a name for the resource group. If you [use an Azure Resource Manager template](#) to deploy an AKS cluster, you can define the resource group name by using the `nodeResourceGroup` property.

- The secondary resource group is automatically created by the Azure resource provider in your own subscription.
- You can specify a custom resource group name only when you're creating the cluster.

As you work with the node resource group, keep in mind that you can't:

- Specify an existing resource group for the node resource group.
- Specify a different subscription for the node resource group.
- Change the node resource group name after the cluster has been created.
- Specify names for the managed resources within the node resource group.
- Modify or delete Azure-created tags of managed resources within the node resource group. (See additional information in the next section.)

## Can I modify tags and other properties of the AKS resources in the node resource group?

If you modify or delete Azure-created tags and other resource properties in the node resource group, you could get unexpected results such as scaling and upgrading errors. AKS allows you to create and modify custom tags created by end users, and you can add those tags when [creating a node pool](#). You might want to create or modify custom tags, for example, to assign a business unit or cost center. This can also be achieved by creating Azure Policies with a scope on the managed resource group.

However, modifying any **Azure-created tags** on resources under the node resource group in the AKS cluster is

an unsupported action, which breaks the service-level objective (SLO). For more information, see [Does AKS offer a service-level agreement?](#)

## What Kubernetes admission controllers does AKS support? Can admission controllers be added or removed?

AKS supports the following [admission controllers](#):

- *NamespaceLifecycle*
- *LimitRanger*
- *ServiceAccount*
- *DefaultStorageClass*
- *DefaultTolerationSeconds*
- *MutatingAdmissionWebhook*
- *ValidatingAdmissionWebhook*
- *ResourceQuota*
- *PodNodeSelector*
- *PodTolerationRestriction*
- *ExtendedResourceToleration*

Currently, you can't modify the list of admission controllers in AKS.

## Can I use admission controller webhooks on AKS?

Yes, you may use admission controller webhooks on AKS. It's recommended you exclude internal AKS namespaces, which are marked with the **control-plane** label. For example, by adding the below to the webhook configuration:

```
namespaceSelector:  
  matchExpressions:  
  - key: control-plane  
    operator: DoesNotExist
```

AKS firewalls the API server egress so your admission controller webhooks need to be accessible from within the cluster.

## Can admission controller webhooks impact kube-system and internal AKS namespaces?

To protect the stability of the system and prevent custom admission controllers from impacting internal services in the kube-system, namespace AKS has an **Admissions Enforcer**, which automatically excludes kube-system and AKS internal namespaces. This service ensures the custom admission controllers don't affect the services running in kube-system.

If you have a critical use case for having something deployed on kube-system (not recommended) which you require to be covered by your custom admission webhook, you may add the below label or annotation so that Admissions Enforcer ignores it.

Label: `"admissions.enforcer/disabled": "true"` or Annotation: `"admissions.enforcer/disabled": true`

## Is Azure Key Vault integrated with AKS?

AKS isn't currently natively integrated with Azure Key Vault. However, the [Azure Key Vault provider for CSI](#)

[Secrets Store](#) enables direct integration from Kubernetes pods to Key Vault secrets.

## Can I run Windows Server containers on AKS?

Yes, Windows Server containers are available on AKS. To run Windows Server containers in AKS, you create a node pool that runs Windows Server as the guest OS. Windows Server containers can use only Windows Server 2019. To get started, see [Create an AKS cluster with a Windows Server node pool](#).

Windows Server support for node pool includes some limitations that are part of the upstream Windows Server in Kubernetes project. For more information on these limitations, see [Windows Server containers in AKS limitations](#).

## Does AKS offer a service-level agreement?

AKS provides SLA guarantees as an optional add-on feature with [Uptime SLA](#).

The Free SKU offered by default doesn't have a associated Service Level *Agreement*, but has a Service Level *Objective* of 99.5%. It could happen that transient connectivity issues are observed in case of upgrades, unhealthy underlay nodes, platform maintenance, application overwhelming the API Server with requests, etc. If your workload doesn't tolerate API Server restarts, then we suggest using Uptime SLA.

## Can I apply Azure reservation discounts to my AKS agent nodes?

AKS agent nodes are billed as standard Azure virtual machines, so if you've purchased [Azure reservations](#) for the VM size that you're using in AKS, those discounts are automatically applied.

## Can I move/migrate my cluster between Azure tenants?

Moving your AKS cluster between tenants is currently unsupported.

## Can I move/migrate my cluster between subscriptions?

Movement of clusters between subscriptions is currently unsupported.

## Can I move my AKS clusters from the current Azure subscription to another?

Moving your AKS cluster and its associated resources between Azure subscriptions isn't supported.

## Can I move my AKS cluster or AKS infrastructure resources to other resource groups or rename them?

Moving or renaming your AKS cluster and its associated resources isn't supported.

## Why is my cluster delete taking so long?

Most clusters are deleted upon user request; in some cases, especially where customers are bringing their own Resource Group, or doing cross-RG tasks deletion can take additional time or fail. If you have an issue with deletes, double-check that you do not have locks on the RG, that any resources outside of the RG are disassociated from the RG, and so on.

## If I have pod / deployments in state 'NodeLost' or 'Unknown' can I still upgrade my cluster?

You can, but AKS doesn't recommend this. Upgrades should be performed when the state of the cluster is known and healthy.

## If I have a cluster with one or more nodes in an Unhealthy state or shut down, can I perform an upgrade?

No, delete/remove any nodes in a failed state or otherwise removed from the cluster prior to upgrading.

## I ran a cluster delete, but see the error

[Errno 11001] getaddrinfo failed

Most commonly, this is caused by users having one or more Network Security Groups (NSGs) still in use and associated with the cluster. Remove them and attempt the delete again.

## I ran an upgrade, but now my pods are in crash loops, and readiness probes fail?

Confirm your service principal hasn't expired. See: [AKS service principal](#) and [AKS update credentials](#).

## My cluster was working, but suddenly can't provision LoadBalancers, mount PVCs, etc.?

Confirm your service principal hasn't expired. See: [AKS service principal](#) and [AKS update credentials](#).

## Can I scale my AKS cluster to zero?

You can completely [stop a running AKS cluster](#), saving on the respective compute costs. Additionally, you may also choose to [scale or autoscale all or specific user node pools](#) to 0, maintaining only the necessary cluster configuration. You can't directly scale [system node pools](#) to zero.

## Can I use the virtual machine scale set APIs to scale manually?

No, scale operations by using the virtual machine scale set APIs aren't supported. Use the AKS APIs ([az aks scale](#)).

## Can I use virtual machine scale sets to manually scale to zero nodes?

No, scale operations by using the virtual machine scale set APIs aren't supported. You can use the AKS API to scale to zero non-system node pools or [stop your cluster](#) instead.

## Can I stop or de-allocate all my VMs?

While AKS has resilience mechanisms to withstand such a config and recover from it, this isn't a supported configuration. [Stop your cluster](#) instead.

## Can I use custom VM extensions?

The Log Analytics agent is supported because it's an extension managed by Microsoft. Otherwise no, AKS is a managed service, and manipulation of the IaaS resources isn't supported. To install custom components, use the Kubernetes APIs and mechanisms. For example, use DaemonSets to install required components.

## Does AKS store any customer data outside of the cluster's region?

The feature to enable storing customer data in a single region is currently only available in the Southeast Asia Region (Singapore) of the Asia Pacific Geo and Brazil South (Sao Paulo State) Region of Brazil Geo. For all other regions, customer data is stored in Geo.

## Are AKS images required to run as root?

Except for the following two images, AKS images aren't required to run as root:

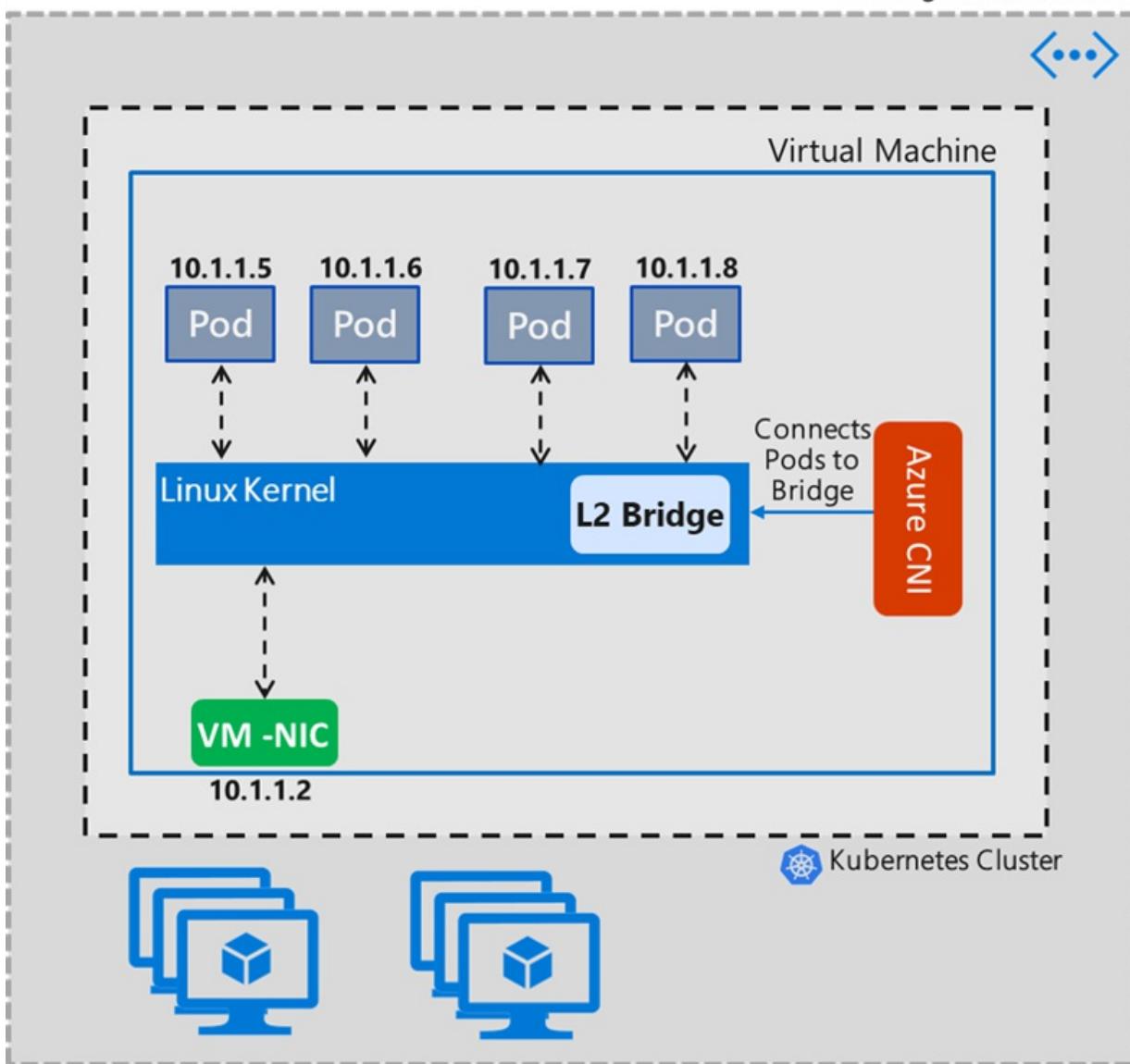
- [mcr.microsoft.com/oss/kubernetes/coredns](https://mcr.microsoft.com/oss/kubernetes/coredns)
- [mcr.microsoft.com/azuremonitor/containerinsights/ciprod](https://mcr.microsoft.com/azuremonitor/containerinsights/ciprod)

## What is Azure CNI Transparent Mode vs. Bridge Mode?

From v1.2.0 Azure CNI will have Transparent mode as default for single tenancy Linux CNI deployments. Transparent mode is replacing bridge mode. In this section, we will discuss more about the differences about both modes and what are the benefits/limitation for using Transparent mode in Azure CNI.

### Bridge mode

As the name suggests, bridge mode Azure CNI, in a "just in time" fashion, will create a L2 bridge named "azure0". All the host side pod `veth` pair interfaces will be connected to this bridge. So Pod-Pod intra VM communication and the remaining traffic goes through this bridge. The bridge in question is a layer 2 virtual device that on its own cannot receive or transmit anything unless you bind one or more real devices to it. For this reason, eth0 of the Linux VM has to be converted into a subordinate to "azure0" bridge. This creates a complex network topology within the Linux VM and as a symptom CNI had to take care of other networking functions like DNS server update and so on.



Below is an example of how the ip route setup looks like in Bridge mode. Regardless of how many pods the node has, there will only ever be two routes. The first one saying, all traffic excluding local on azure0 will go to the default gateway of the subnet through the interface with ip "src 10.240.0.4" (which is Node primary IP) and the second one saying "10.20.x.x" Pod space to kernel for kernel to decide.

```
default via 10.240.0.1 dev azure0 proto dhcp src 10.240.0.4 metric 100
10.240.0.0/12 dev azure0 proto kernel scope link src 10.240.0.4
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1 linkdown
root@k8s-agentpool1-20465682-1:/#
```

### Transparent mode

Transparent mode takes a straight forward approach to setting up Linux networking. In this mode, Azure CNI won't change any properties of eth0 interface in the Linux VM. This minimal approach of changing the Linux networking properties helps reduce complex corner case issues that clusters could face with Bridge mode. In Transparent Mode, Azure CNI will create and add host-side pod `veth` pair interfaces that will be added to the host network. Intra VM Pod-to-Pod communication is through ip routes that the CNI will add. Essentially Pod-to-Pod communication is over layer 3 and pod traffic is routed by L3 routing rules.



Below is an example ip route setup of transparent mode, each Pod's interface will get a static route attached so that traffic with dest IP as the Pod will be sent directly to the Pod's host side `veth` pair interface.

```

10.240.0.216 dev azv79d05038592 proto static
10.240.0.218 dev azv8184320e2bf proto static
10.240.0.219 dev azvc0339d223b9 proto static
10.240.0.222 dev azv722a6b28449 proto static
10.240.0.223 dev azve7f326f1507 proto static
10.240.0.224 dev azvb3bfcdd75a proto static
168.63.129.16 via 10.240.0.1 dev eth0 proto dhcp src 10.240.0.4 metric 100
169.254.169.254 via 10.240.0.1 dev eth0 proto dhcp src 10.240.0.4 metric 100
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1 linkdown

```

### Benefits of transparent mode

- Provides mitigation for `conntrack` DNS parallel race condition and avoidance of 5-sec DNS latency issues without the need to set up node local DNS (you may still use node local DNS for performance reasons).
- Eliminates the initial 5-sec DNS latency CNI bridge mode introduces today due to "just in time" bridge setup.
- One of the corner cases in bridge mode is that the Azure CNI can't keep updating the custom DNS server lists users add to either VNET or NIC. This results in the CNI picking up only the first instance of the DNS server list. Solved in Transparent mode as CNI doesn't change any eth0 properties. See more [here](#).
- Provides better handling of UDP traffic and mitigation for UDP flood storm when ARP times out. In bridge mode, when bridge doesn't know a MAC address of destination pod in intra-VM Pod-to-Pod communication,

by design, this results in storm of the packet to all ports. Solved in Transparent mode as there are no L2 devices in path. See more [here](#).

- Transparent mode performs better in Intra VM Pod-to-Pod communication in terms of throughput and latency when compared to bridge mode.

## How to avoid permission ownership setting slow issues when the volume has a lot of files?

Traditionally if your pod is running as a non-root user (which you should), you must specify a `fsGroup` inside the pod's security context so that the volume can be readable and writable by the Pod. This requirement is covered in more detail in [here](#).

But one side-effect of setting `fsGroup` is that, each time a volume is mounted, Kubernetes must recursively `chown()` and `chmod()` all the files and directories inside the volume - with a few exceptions noted below. This happens even if group ownership of the volume already matches the requested `fsGroup`, and can be pretty expensive for larger volumes with lots of small files, which causes pod startup to take a long time. This scenario has been a known problem before v1.20 and the workaround is setting the Pod run as root:

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 0
    fsGroup: 0
```

The issue has been resolved by Kubernetes v1.20, refer [Kubernetes 1.20: Granular Control of Volume Permission Changes](#) for more details.

## Can I use FIPS cryptographic libraries with deployments on AKS?

FIPS-enabled nodes are currently available in preview on Linux-based node pools. For more details, see [Add a FIPS-enabled node pool \(preview\)](#).