Deploy an Azure Kubernetes Service (AKS) cluster using Azure CLI

- 10/26/2023

In this article

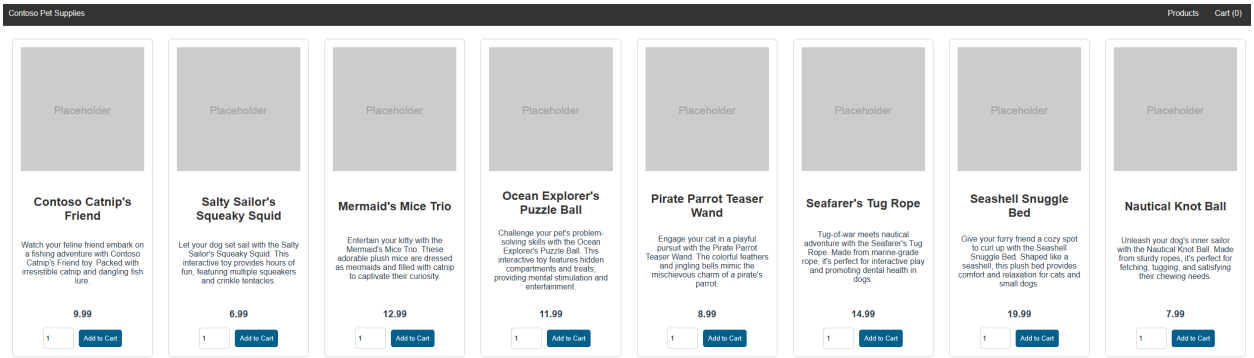
Show less

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

- Deploy an AKS cluster using the Azure CLI.
- Run a sample multi-container application with a group of microservices and web front ends simulating a retail scenario.

Note

This sample application is just for demo purposes and doesn't represent all the best practices for Kubernetes applications.



Before you begin

- This quickstart assumes a basic understanding of Kubernetes concepts. For more information, see Kubernetes core concepts for Azure Kubernetes Service (AKS).
- You need an Azure account with an active subscription. If you don't have one, create an account for free.
- To learn more about creating a Windows Server node pool, see Create an AKS cluster that supports Windows Server containers.
- This article requires Azure CLI version 2.0.64 or later. If you're using Azure Cloud Shell, the latest version is already installed.
- Make sure the identity you use to create your cluster has the appropriate minimum permissions. For more details on access and identity for AKS, see Access and identity options for Azure Kubernetes Service (AKS).
- If you have multiple Azure subscriptions, select the appropriate subscription ID in which the resources should be billed using the az account command.
- Verify you have the *Microsoft.OperationsManagement* and *Microsoft.OperationalInsights* providers registered on your subscription. These Azure resource providers are required to support Container insights. Check the registration status using the following commands:

  Azure CLICopy
  Open Cloudshell
  ```
  az provider show -n Microsoft.OperationsManagement -o table
  az provider show -n Microsoft.OperationalInsights -o table
  ```

  If they're not registered, register them using the following commands:

  Azure CLICopy
  Open Cloudshell
  ```
  az provider register --namespace Microsoft.OperationsManagement
  az provider register --namespace Microsoft.OperationalInsights
  ```

Note

If you plan to run the commands locally instead of in Azure Cloud Shell, make sure you run the commands with administrative privileges.

Note

The Azure Linux node pool is now generally available (GA). To learn about the benefits and deployment steps, see the **Introduction to the Azure Linux Container Host for AKS**.

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're prompted to specify a location. This location is the storage location of your resource group metadata and where your resources 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.

  Azure CLICopy
  Open Cloudshell
  ```
  az group create --name myResourceGroup --location eastus
  ```

  The following example output resembles successful creation of the resource group:

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

Create an AKS cluster

The following example creates a cluster named *myAKSCluster* with one node and enables a system-assigned managed identity.

- Create an AKS cluster using the az aks create command with the `--enable-addons monitoring` and `--enable-msi-auth-for-monitoring` parameters to enable Azure Monitor Container insights with managed identity authentication (preview).

  Azure CLICopy
  Open Cloudshell
  ```
  az aks create -g myResourceGroup -n myAKSCluster --enable-managed-identity --node-count 1 --enable-addons monitoring --enable-msi-auth-for-monitoring  --generate-ssh-keys
  ```

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

  Note

  When you create a new cluster, AKS automatically creates a second resource group 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.

   Azure CLICopy
   Open Cloudshell
   ```
   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.

   Azure CLICopy
   Open Cloudshell
   ```
   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.

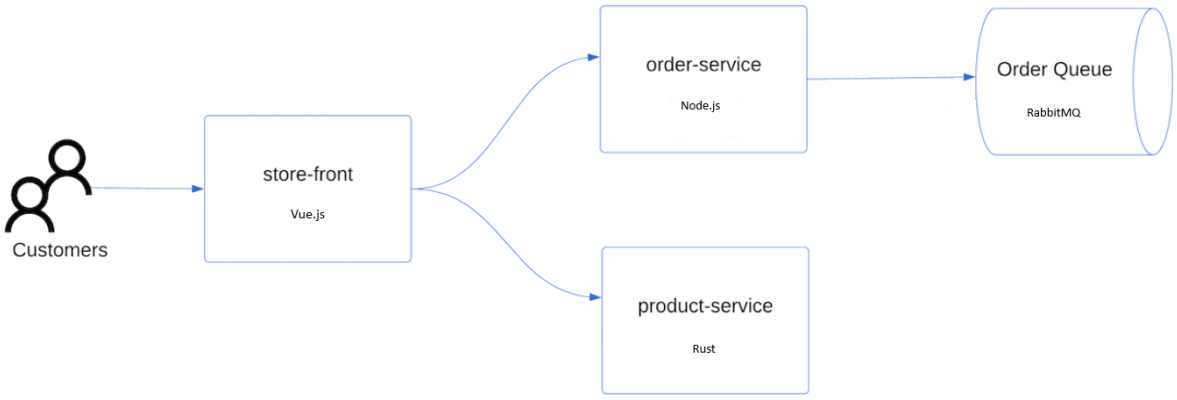
   Azure CLICopy
   Open Cloudshell
   ```
   kubectl get nodes
   ```

   The following example output shows the single node created in the previous steps. Make sure the node status is *Ready*.

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

Deploy the application

To deploy the application, you use a manifest file to create all the objects required to run the AKS Store application. A Kubernetes manifest file defines a cluster's desired state, such as which container images to run. The manifest includes the following Kubernetes deployments and services:



- **Store front**: Web application for customers to view products and place orders.
- **Product service**: Shows product information.
- **Order service**: Places orders.
- **Rabbit MQ**: Message queue for an order queue.

Note

We don't recommend running stateful containers, such as Rabbit MQ, without persistent storage for production. These are used here for simplicity, but we recommend using managed services, such as Azure CosmosDB or Azure Service Bus.

1. Create a file named `aks-store-quickstart.yaml` and copy in the following manifest:

   YAMLCopy

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rabbitmq
spec:
  replicas: 1
  selector:
    matchLabels:
      app: rabbitmq
  template:
    metadata:
      labels:
        app: rabbitmq
    spec:
      nodeSelector:
        "kubernetes.io/os": linux
      containers:
      - name: rabbitmq
        image: mcr.microsoft.com/mirror/docker/library/rabbitmq:3.10-
management-alpine
        ports:
        - containerPort: 5672
```

```
          name: rabbitmq-amqp
        - containerPort: 15672
          name: rabbitmq-http
        env:
        - name: RABBITMQ_DEFAULT_USER
          value: "username"
        - name: RABBITMQ_DEFAULT_PASS
          value: "password"
        resources:
          requests:
            cpu: 10m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        volumeMounts:
        - name: rabbitmq-enabled-plugins
          mountPath: /etc/rabbitmq/enabled_plugins
          subPath: enabled_plugins
      volumes:
      - name: rabbitmq-enabled-plugins
        configMap:
          name: rabbitmq-enabled-plugins
          items:
          - key: rabbitmq_enabled_plugins
            path: enabled_plugins
---
apiVersion: v1
data:
  rabbitmq_enabled_plugins: |
    [rabbitmq_management,rabbitmq_prometheus,rabbitmq_amqp1_0].
kind: ConfigMap
metadata:
  name: rabbitmq-enabled-plugins
---
apiVersion: v1
kind: Service
metadata:
  name: rabbitmq
spec:
  selector:
    app: rabbitmq
  ports:
    - name: rabbitmq-amqp
      port: 5672
      targetPort: 5672
    - name: rabbitmq-http
      port: 15672
      targetPort: 15672
  type: ClusterIP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: order-service
```

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: order-service
  template:
    metadata:
      labels:
        app: order-service
    spec:
      nodeSelector:
        "kubernetes.io/os": linux
      containers:
      - name: order-service
        image: ghcr.io/azure-samples/aks-store-demo/order-service:latest
        ports:
        - containerPort: 3000
        env:
        - name: ORDER_QUEUE_HOSTNAME
          value: "rabbitmq"
        - name: ORDER_QUEUE_PORT
          value: "5672"
        - name: ORDER_QUEUE_USERNAME
          value: "username"
        - name: ORDER_QUEUE_PASSWORD
          value: "password"
        - name: ORDER_QUEUE_NAME
          value: "orders"
        - name: FASTIFY_ADDRESS
          value: "0.0.0.0"
        resources:
          requests:
            cpu: 1m
            memory: 50Mi
          limits:
            cpu: 75m
            memory: 128Mi
      initContainers:
      - name: wait-for-rabbitmq
        image: busybox
        command: ['sh', '-c', 'until nc -zv rabbitmq 5672; do echo
waiting for rabbitmq; sleep 2; done;']
        resources:
          requests:
            cpu: 1m
            memory: 50Mi
          limits:
            cpu: 75m
            memory: 128Mi
---
apiVersion: v1
kind: Service
metadata:
  name: order-service
spec:
```

```
  type: ClusterIP
  ports:
  - name: http
    port: 3000
    targetPort: 3000
  selector:
    app: order-service
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product-service
  template:
    metadata:
      labels:
        app: product-service
    spec:
      nodeSelector:
        "kubernetes.io/os": linux
      containers:
      - name: product-service
        image: ghcr.io/azure-samples/aks-store-demo/product-
service:latest
        ports:
        - containerPort: 3002
        resources:
          requests:
            cpu: 1m
            memory: 1Mi
          limits:
            cpu: 1m
            memory: 7Mi
---
apiVersion: v1
kind: Service
metadata:
  name: product-service
spec:
  type: ClusterIP
  ports:
  - name: http
    port: 3002
    targetPort: 3002
  selector:
    app: product-service
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: store-front
```

```yaml
spec:
  replicas: 1
  selector:
    matchLabels:
      app: store-front
  template:
    metadata:
      labels:
        app: store-front
    spec:
      nodeSelector:
        "kubernetes.io/os": linux
      containers:
      - name: store-front
        image: ghcr.io/azure-samples/aks-store-demo/store-front:latest
        ports:
        - containerPort: 8080
          name: store-front
        env:
        - name: VUE_APP_ORDER_SERVICE_URL
          value: "http://order-service:3000/"
        - name: VUE_APP_PRODUCT_SERVICE_URL
          value: "http://product-service:3002/"
        resources:
          requests:
            cpu: 1m
            memory: 200Mi
          limits:
            cpu: 1000m
            memory: 512Mi
---
apiVersion: v1
kind: Service
metadata:
  name: store-front
spec:
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: store-front
  type: LoadBalancer
```

For a breakdown of YAML manifest files, see Deployments and YAML manifests.

2. Deploy the application using the kubectl apply command and specify the name of your YAML manifest.

   Azure CLICopy
   Open Cloudshell
   ```
   kubectl apply -f aks-store-quickstart.yaml
   ```

   The following example output shows the deployments and services:

OutputCopy

```
deployment.apps/rabbitmq created
service/rabbitmq created
deployment.apps/order-service created
service/order-service created
deployment.apps/product-service created
service/product-service created
deployment.apps/store-front created
service/store-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.

1. Check the status of the deployed pods using the kubectl get pods command. Make all pods are `Running` before proceeding.
2. Check for a public IP address for the store-front application. Monitor progress using the kubectl get service command with the `--watch` argument.

   Azure CLICopy
   Open Cloudshell

   ```
   kubectl get service store-front --watch
   ```

   The **EXTERNAL-IP** output for the `store-front` service initially shows as *pending*:

   OutputCopy

   ```
   NAME          TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)
   AGE
   store-front   LoadBalancer   10.0.100.10   <pending>     80:30025/TCP
   4h4m
   ```

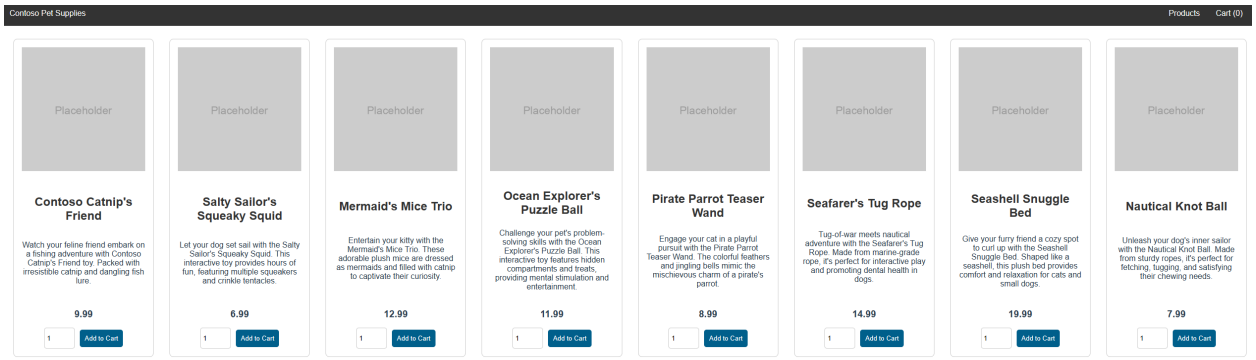
3. Once the **EXTERNAL-IP** address changes from *pending* to an actual public IP address, use `CTRL-C` to stop the `kubectl` watch process.

   The following example output shows a valid public IP address assigned to the service:

   OutputCopy

   ```
   NAME          TYPE           CLUSTER-IP    EXTERNAL-IP    PORT(S)
   AGE
   store-front   LoadBalancer   10.0.100.10   20.62.159.19   80:30025/TCP
   4h5m
   ```

4. Open a web browser to the external IP address of your service to see the Azure Store app in action.



Delete the cluster

If you don't plan on going through the following tutorials, clean up unnecessary resources to avoid Azure charges.

- Remove the resource group, container service, and all related resources using the az group delete command.

  Azure CLICopy
  Open Cloudshell
  ```
  az group delete --name myResourceGroup --yes --no-wait
  ```
  Note

  The AKS cluster was created with a system-assigned managed identity, which is the default identity option used in this quickstart. The platform manages this identity so you don't need to manually remove it.

Next steps

In this quickstart, you deployed a Kubernetes cluster and deployed a simple multi-container application to it.

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

AKS tutorial

This quickstart is for introductory purposes. For guidance on creating full solutions with AKS for production, see AKS solution guidance.

Prepare an application for Azure Kubernetes Service (AKS)

- 10/26/2023

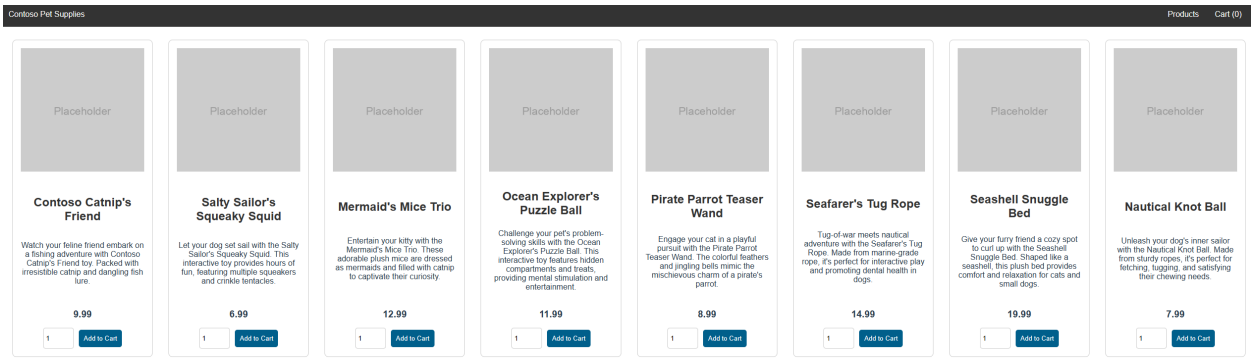
In this article

Show less

In this tutorial, part one of seven, you prepare a multi-container application to use in Kubernetes. You use existing development tools like Docker Compose to locally build and test the 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, you upload the container image to an Azure Container Registry (ACR), and then deploy it 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.

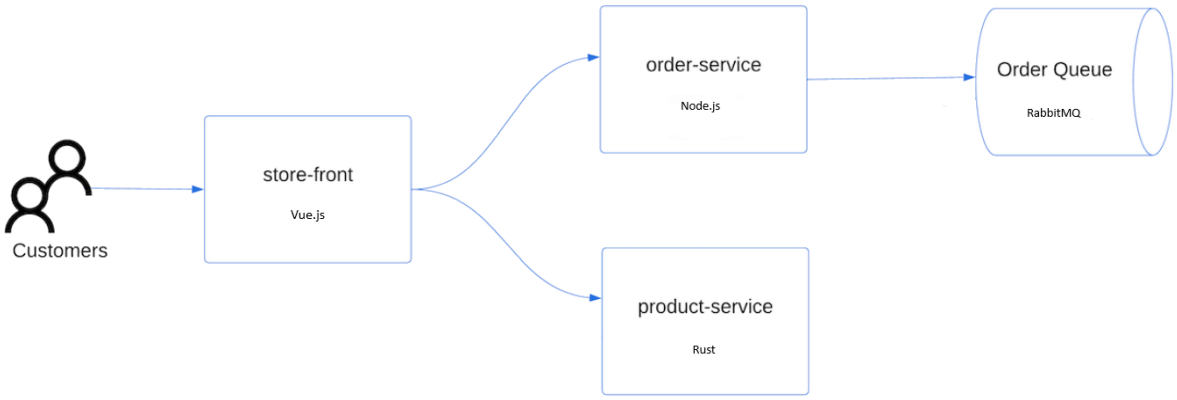
To complete this tutorial, you need a local Docker development environment running Linux containers. Docker provides packages that configure Docker on a Mac, Windows, or Linux system.

Note

Azure Cloud Shell doesn't 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 store front app including the following Kubernetes deployments and services:



- **Store front**: Web application for customers to view products and place orders.
- **Product service**: Shows product information.
- **Order service**: Places orders.
- **Rabbit MQ**: Message queue for an order queue.

1. Use git to clone the sample application to your development environment.

   ConsoleCopy
   ```
   git clone https://github.com/Azure-Samples/aks-store-demo.git
   ```

2. Change into the cloned directory.

   ConsoleCopy
   ```
   cd aks-store-demo
   ```

Review Docker Compose file

The sample application you create in this tutorial uses the *docker-compose-quickstart* YAML file in the repository you cloned in the previous step.

YAMLCopy

```yaml
version: "3.7"
services:
  rabbitmq:
    image: rabbitmq:3.11.17-management-alpine
    container_name: 'rabbitmq'
    restart: always
    environment:
      - "RABBITMQ_DEFAULT_USER=username"
      - "RABBITMQ_DEFAULT_PASS=password"
    ports:
      - 15672:15672
      - 5672:5672
    healthcheck:
      test: ["CMD", "rabbitmqctl", "status"]
      interval: 30s
      timeout: 10s
      retries: 5
    volumes:
      - ./rabbitmq_enabled_plugins:/etc/rabbitmq/enabled_plugins
    networks:
      - backend_services
  orderservice:
    build: src/order-service
    container_name: 'orderservice'
    restart: always
    ports:
      - 3000:3000
    healthcheck:
      test: ["CMD", "wget", "-O", "/dev/null", "-q",
"http://orderservice:3000/health"]
      interval: 30s
      timeout: 10s
      retries: 5
    environment:
      - ORDER_QUEUE_HOSTNAME=rabbitmq
      - ORDER_QUEUE_PORT=5672
      - ORDER_QUEUE_USERNAME=username
      - ORDER_QUEUE_PASSWORD=password
      - ORDER_QUEUE_NAME=orders
      - ORDER_QUEUE_RECONNECT_LIMIT=3
    networks:
      - backend_services
    depends_on:
      rabbitmq:
        condition: service_healthy
  productservice:
    build: src/product-service
```

```
    container_name: 'productservice'
    restart: always
    ports:
      - 3002:3002
    healthcheck:
      test: ["CMD", "wget", "-O", "/dev/null", "-q",
"http://productservice:3002/health"]
      interval: 30s
      timeout: 10s
      retries: 5
    networks:
      - backend_services
  storefront:
    build: src/store-front
    container_name: 'storefront'
    restart: always
    ports:
      - 8080:8080
    healthcheck:
      test: ["CMD", "wget", "-O", "/dev/null", "-q", "http://storefront:80/health"]
      interval: 30s
      timeout: 10s
      retries: 5
    environment:
      - VUE_APP_PRODUCT_SERVICE_URL=http://productservice:3002/
      - VUE_APP_ORDER_SERVICE_URL=http://orderservice:3000/
    networks:
      - backend_services
    depends_on:
      - productservice
      - orderservice
networks:
  backend_services:
    driver: bridge
```

Create container images and run application

You can use Docker Compose to automate building container images and the deployment of multi-container applications.

1. Create the container image, download the Redis image, and start the application using the `docker compose` command.

   ConsoleCopy
   ```
   docker compose -f docker-compose-quickstart.yml up -d
   ```

2. View the created images using the docker images command.

   ConsoleCopy
   ```
   docker images
   ```

The following condensed example output shows the created images:

```
OutputCopy
REPOSITORY                          TAG                              IMAGE ID
aks-store-demo-productservice       latest
2b66a7e91eca
aks-store-demo-orderservice         latest
54ad5de546f9
aks-store-demo-storefront           latest
d9e3ac46a225
rabbitmq                            3.11.17-management-alpine
79a570297657
...
```

3. View the running containers using the docker ps command.

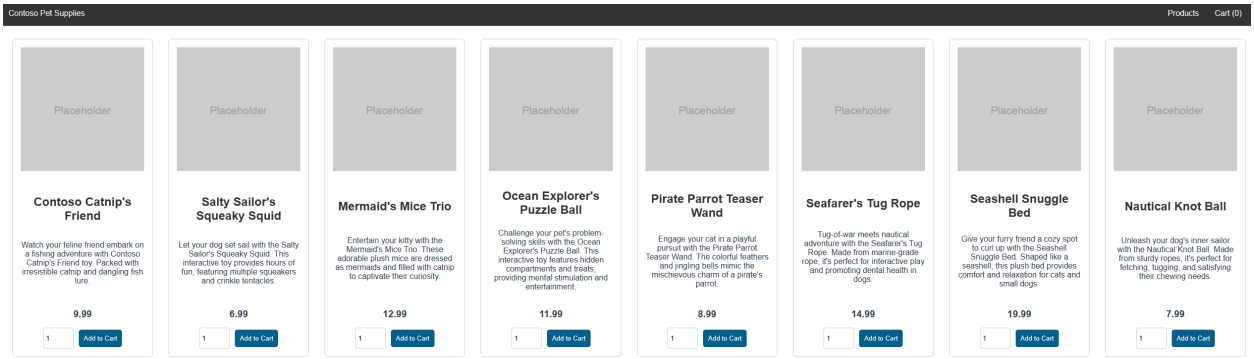
```
ConsoleCopy
docker ps
```

The following condensed example output shows four running containers:

```
OutputCopy
CONTAINER ID        IMAGE
21574cb38c1f        aks-store-demo-productservice
c30a5ed8d86a        aks-store-demo-orderservice
d10e5244f237        aks-store-demo-storefront
94e00b50b86a        rabbitmq:3.11.17-management-alpine
```

Test application locally

To see your running application, navigate to `http://localhost:8080` in a local web browser. The sample application loads, as shown in the following example:



On this page, you can view products, add them to your cart, and then place an order.

Clean up resources

Since you validated the application's functionality, you can stop and remove the running containers. ***Do not delete the container images*** - you use them in the next tutorial.

- Stop and remove the container instances and resources using the docker-compose down command.

  ConsoleCopy
  ```
  docker compose down
  ```

Next steps

In this tutorial, you created a sample application, created container images for the application, and then tested 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.

In the next tutorial, you learn how to store container images in an ACR.

Push images to Azure Container Registry

Create an Azure Container Registry (ACR) and build images

- 11/07/2023

In this article

Azure Container Registry (ACR) is a private registry for container images. A private container registry allows you to 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 ACR instance.
- Use ACR Tasks to build and push container images to ACR.
- View images in your registry.

Before you begin

In the previous tutorial, you used Docker to create a container image for a simple Azure Store Front application. If you haven't created the Azure Store Front app image, return to Tutorial 1 - Prepare an application for AKS.

- Azure CLI
- Azure PowerShell

This tutorial requires 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

Before creating an ACR instance, you need a resource group. An Azure resource group is a logical container into which you deploy and manage Azure resources.

Important

This tutorial uses *myResourceGroup* as a placeholder for the resource group name. If you want to use a different name, replace *myResourceGroup* with your own resource group name.

- Azure CLI
- Azure PowerShell

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

   Azure CLICopy
   Open Cloudshell
   ```
   az group create --name myResourceGroup --location eastus
   ```

2. Create an ACR instance using the az acr create command and provide your own unique registry name. The registry name must be unique within Azure and contain 5-50 alphanumeric characters. The rest of this tutorial uses an environment variable, `$ACRNAME`, as a placeholder for the container registry name. You can set this environment variable to your unique ACR name to use in future commands. The *Basic* SKU is a cost-optimized entry point for development purposes that provides a balance of storage and throughput.

   Azure CLICopy
   Open Cloudshell
   ```
   az acr create --resource-group myResourceGroup --name $ACRNAME --sku
   Basic
   ```

Build and push container images to registry

- Build and push the images to your ACR using the az acr build command.

  Note

  In the following example, we don't build the `rabbitmq` image. This image is available from the Docker Hub public repository and doesn't need to be built or pushed to your ACR instance.

Azure CLICopy
Open Cloudshell

```
az acr build --registry $ACRNAME --image aks-store-demo/product-
service:latest ./src/product-service/
az acr build --registry $ACRNAME --image aks-store-demo/order-
service:latest ./src/order-service/
az acr build --registry $ACRNAME --image aks-store-demo/store-
front:latest ./src/store-front/
```

List images in registry

- Azure CLI
- Azure PowerShell

- View the images in your ACR instance using the az acr repository list command.

  Azure CLICopy
  Open Cloudshell

  ```
  az acr repository list --name $ACRNAME --output table
  ```

  The following example output lists the available images in your registry:

  OutputCopy

  ```
  Result
  ----------------
  aks-store-demo/product-service
  aks-store-demo/order-service
  aks-store-demo/store-front
  ```

Next steps

In this tutorial, you created an ACR and pushed images to it to use in an AKS cluster. You learned how to:

- Create an ACR instance.
- Use ACR Tasks to build and push container images to ACR.
- View images in your registry.

In the next tutorial, you learn how to deploy a Kubernetes cluster in Azure.

Deploy Kubernetes cluster

Deploy an Azure Kubernetes Service (AKS) cluster

- 10/26/2023

In this article

Show less

Kubernetes provides a distributed platform for containerized applications. With Azure Kubernetes Service (AKS), you can quickly create a production ready Kubernetes cluster.

In this tutorial, part three of seven, you deploy a Kubernetes cluster in AKS. You learn how to:

- Deploy an AKS cluster that can authenticate to an Azure Container Registry (ACR).
- Install the Kubernetes CLI, `kubectl`.
- Configure `kubectl` to connect to your AKS cluster.

Before you begin

In previous tutorials, you created a container image and uploaded it to an ACR instance. If you haven't completed these steps and want to follow along, start with Tutorial 1 - Prepare application for AKS.

- If you're using Azure CLI, 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.
- If you're using Azure PowerShell, this tutorial requires that you're running Azure PowerShell version 5.9.0 or later. Run `Get-InstalledModule -Name Az` to

find the version. If you need to install or upgrade, see Install Azure PowerShell.

Create a Kubernetes cluster

AKS clusters can use Kubernetes role-based access control (Kubernetes RBAC), which allows you to define access to resources based on roles assigned to users. If a user is assigned multiple roles, permissions are combined. Permissions can be scoped to either a single namespace or across the whole cluster.

To learn more about AKS and Kubernetes RBAC, see Control access to cluster resources using Kubernetes RBAC and Microsoft Entra identities in AKS.

- Azure CLI
- Azure PowerShell

This tutorial requires 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 AKS cluster

AKS clusters can use Kubernetes role-based access control (Kubernetes RBAC), which allows you to define access to resources based on roles assigned to users. Permissions are combined when users are assigned multiple roles. Permissions can be scoped to either a single namespace or across the whole cluster. For more information, see Control access to cluster resources using Kubernetes RBAC and Azure Active Directory identities in AKS.

For information about AKS resource limits and region availability, see Quotas, virtual machine size restrictions, and region availability in AKS.

Note

To ensure your cluster operates reliably, you should run at least two nodes.

- Azure CLI
- Azure PowerShell

To allow an AKS cluster to interact with other Azure resources, the Azure platform automatically creates a cluster identity. In this example, the cluster identity is granted the right to pull images from the ACR instance you created in the previous tutorial. To execute the command successfully, you need to have an **Owner** or **Azure account administrator** role in your Azure subscription.

- Create an AKS cluster using the az aks create command. 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.

  Azure CLICopy
  Open Cloudshell

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

  Note

  If you already generated SSH keys, you may encounter an error similar to `linuxProfile.ssh.publicKeys.keyData is invalid`. To proceed, retry the command without the `--generate-ssh-keys` parameter.

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.

Install the Kubernetes CLI

You use the Kubernetes CLI, kubectl, to connect to your Kubernetes cluster. If you use the Azure Cloud Shell, `kubectl` is already installed. If you're running the commands locally, you can use the Azure CLI or Azure PowerShell to install `kubectl`.

- Azure CLI

- Azure PowerShell

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

      Azure CLICopy
      Open Cloudshell
      ```
      az aks install-cli
      ```

Connect to cluster using kubectl

- Azure CLI
- Azure PowerShell

1. Configure `kubectl` to connect to your Kubernetes cluster using the az aks get-credentials command. The following example gets credentials for the AKS cluster named *myAKSCluster* in *myResourceGroup*.

   Azure CLICopy
   Open Cloudshell
   ```
   az aks get-credentials --resource-group myResourceGroup --name
   myAKSCluster
   ```

2. Verify connection to your cluster using the kubectl get nodes command, which returns a list of cluster nodes.

   Azure CLICopy
   Open Cloudshell
   ```
   kubectl get nodes
   ```

   The following example output shows a list of the cluster nodes:

   OutputCopy
   ```
   NAME                                STATUS   ROLES   AGE   VERSION
   aks-nodepool1-19366578-vmss000002   Ready    agent   47h   v1.25.6
   aks-nodepool1-19366578-vmss000003   Ready    agent   47h   v1.25.6
   ```

Next steps

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

- Deploy an AKS cluster that can authenticate to an ACR.
- Install the Kubernetes CLI, `kubectl`.
- Configure `kubectl` to connect to your AKS cluster.

In the next tutorial, you learn how to deploy an application to your cluster.

Deploy an application in AKS

Deploy an application to Azure Kubernetes Service (AKS)

- 11/02/2023

In this article

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, you deploy a sample application into a Kubernetes cluster. You learn how to:

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

Tip

With AKS, you can use the following approaches for configuration management:

- **GitOps**: Enables declarations of your cluster's state to automatically apply to the cluster. To learn how to use GitOps to deploy an application with an AKS cluster, see the **prerequisites for Azure Kubernetes Service clusters** in the **GitOps with Flux v2** tutorial.
- **DevOps**: Enables you to build, test, and deploy with continuous integration (CI) and continuous delivery (CD). To see examples of how to use DevOps to deploy an application with an AKS cluster, see **Build and deploy to AKS with Azure Pipelines** or **GitHub Actions for deploying to Kubernetes**.

Before you begin

In previous tutorials, you packaged an application into a container image, uploaded the image to Azure Container Registry, and created a Kubernetes cluster. To complete this tutorial, you need the pre-created `aks-store-quickstart.yaml` Kubernetes manifest file. This file download was included with the application source code in a previous tutorial. Make sure you cloned the repo and changed directories into the cloned repo. If you haven't completed these steps and want to follow along, start with Tutorial 1 - Prepare application for AKS.

- Azure CLI
- Azure PowerShell

This tutorial requires Azure CLI version 2.34.1 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, your Azure Container Registry (ACR) instance stores the container images for the sample application. To deploy the application, you must update the image names in the Kubernetes manifest file to include your ACR login server name.

- Azure CLI
- Azure PowerShell

1. Get your login server address using the az acr list command and query for your login server.

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

2. Make sure you're in the cloned *aks-store-demo* directory, and then open the manifest file with a text editor, such as `vi`:

   Azure CLICopy
   Open Cloudshell
   ```
   vi aks-store-quickstart.yaml
   ```

3. Update the `image` property for the containers by replacing *ghcr.io/azure-samples* with your ACR login server name.

YAMLCopy

```yaml
containers:
...
- name: order-service
 image: <acrName>.azurecr.io/aks-store-demo/order-service:latest
...
- name: product-service
 image: <acrName>.azurecr.io/aks-store-demo/product-service:latest
...
- name: store-front
 image: <acrName>.azurecr.io/aks-store-demo/store-front:latest
...
```

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

Deploy the application

- Deploy the application using the kubectl apply command, which parses the manifest file and creates the defined Kubernetes objects.

  ConsoleCopy

  ```
  kubectl apply -f aks-store-quickstart.yaml
  ```

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

  OutputCopy

  ```
  deployment.apps/rabbitmq created
  service/rabbitmq created
  deployment.apps/order-service created
  service/order-service created
  deployment.apps/product-service created
  service/product-service created
  deployment.apps/store-front created
  service/store-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.

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

ConsoleCopy
```
kubectl get service store-front --watch
```

Initially, the `EXTERNAL-IP` for the *store-front* service shows as *pending*.

OutputCopy
```
store-front   LoadBalancer   10.0.34.242   <pending>     80:30676/TCP
5s
```

2. 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:

   OutputCopy
   ```
   store-front   LoadBalancer   10.0.34.242   52.179.23.131   80:30676/TCP
   67s
   ```

3. View the application in action by opening a web browser to the external IP address of your service.

If the application doesn't load, it might be an authorization problem with your image registry. To view the status of your containers, use the `kubectl get pods` command. If you can't pull the container images, see Authenticate with Azure Container Registry from Azure Kubernetes Service.

Next steps

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

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

In the next tutorial, you learn how to use PaaS services for stateful workloads in Kubernetes.

Use PaaS services for stateful workloads in AKS

Use PaaS services with an Azure Kubernetes Service (AKS) cluster

- 10/26/2023

In this article

Show less

With Kubernetes, you can use PaaS services, such as Azure Service Bus, to develop and run your applications.

In this tutorial, part five of seven, you create an Azure Service Bus namespace and queue to test your application. You learn how to:

- Create an Azure Service Bus namespace and queue.
- Update the Kubernetes manifest file to use the Azure Service Bus queue.
- Test the updated application by placing an order.

Before you begin

In previous tutorials, you packaged an application into a container image, uploaded the image to Azure Container Registry, created a Kubernetes cluster, and deployed an application. To complete this tutorial, you need the pre-created `aks-store-quickstart.yaml` Kubernetes manifest file. This file download was included with the application source code in a previous tutorial. Make sure you cloned the repo and changed directories into the cloned repo. If you haven't completed these steps and want to follow along, start with Tutorial 1 - Prepare application for AKS.

- Azure CLI
- Azure PowerShell

This tutorial requires Azure CLI version 2.34.1 or later. Run `az --version` to find the version. If you need to install or upgrade, see Install Azure CLI.

Create environment variables

- Azure CLI
- Azure PowerShell

    - Create the following environment variables to use for the commands in this tutorial:

      Azure CLICopy
      Open Cloudshell
      ```
      LOC_NAME=eastus
      RAND=$RANDOM
      RG_NAME=myResourceGroup
      AKS_NAME=myAKSCluster
      SB_NS=sb-store-demo-$RAND
      ```

Create Azure Service Bus namespace and queue

In previous tutorials, you used a RabbitMQ container to store orders submitted by the `order-service`. In this tutorial, you use an Azure Service Bus namespace to provide a scoping container for the Service Bus resources within the application. You also use an Azure Service bus queue to send and receive messages between the application components. For more information on Azure Service Bus, see Create an Azure Service Bus namespace and queue.

- Azure CLI
- Azure PowerShell

    1. Create an Azure Service Bus namespace using the az servicebus namespace create command.

       Azure CLICopy
       Open Cloudshell
       ```
       az servicebus namespace create -n $SB_NS -g $RG_NAME -l $LOC_NAME
       ```

2. Create an Azure Service Bus queue using the az servicebus queue create command.

   Azure CLICopy
   Open Cloudshell
   ```
   az servicebus queue create -n orders -g $RG_NAME --namespace-name $SB_NS
   -g $RG_NAME
   ```

3. Create an Azure Service Bus authorization rule using the az servicebus queue authorization-rule create command.

   Azure CLICopy
   Open Cloudshell
   ```
   az servicebus queue authorization-rule create \
       --name sender \
       --namespace-name $SB_NS \
       --resource-group $RG_NAME \
       --queue-name orders \
       --rights Send
   ```

4. Get the Azure Service Bus credentials for later use using the az servicebus namespace show and az servicebus queue authorization-rule keys list commands.

   Azure CLICopy
   Open Cloudshell
   ```
   az servicebus namespace show --name $SB_NS --resource-group $RG_NAME --
   query name -o tsv
   az servicebus queue authorization-rule keys list --namespace-name $SB_NS
   --resource-group $RG_NAME --queue-name orders --name sender --query
   primaryKey -o tsv
   ```

Update Kubernetes manifest file

- Azure CLI
- Azure PowerShell

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

   Azure CLICopy
   Open Cloudshell
   ```
   az aks get-credentials --resource-group myResourceGroup --name
   myAKSCluster
   ```

2. Open the `aks-store-quickstart.yaml` file in a text editor.
3. Remove the existing `rabbitmq` Deployment, ConfigMap, and Service sections and replace the existing `order-service` Deployment section with the following content:

YAMLCopy

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: order-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: order-service
  template:
    metadata:
      labels:
        app: order-service
    spec:
      nodeSelector:
        "kubernetes.io/os": linux
      containers:
      - name: order-service
        image: <REPLACE_WITH_YOUR_ACR_NAME>.azurecr.io/aks-store-
demo/order-service:latest
        ports:
        - containerPort: 3000
        env:
        - name: ORDER_QUEUE_HOSTNAME
          value: "<REPLACE_WITH_YOUR_SB_NS_HOSTNAME>" # Example: sb-
store-demo-123456.servicebus.windows.net
        - name: ORDER_QUEUE_PORT
          value: "5671"
        - name: ORDER_QUEUE_TRANSPORT
          value: "tls"
        - name: ORDER_QUEUE_USERNAME
          value: "sender"
        - name: ORDER_QUEUE_PASSWORD
          value: "<REPLACE_WITH_YOUR_SB_SENDER_PASSWORD>"
        - name: ORDER_QUEUE_NAME
          value: "orders"
        - name: FASTIFY_ADDRESS
          value: "0.0.0.0"
        resources:
          requests:
            cpu: 1m
            memory: 50Mi
          limits:
            cpu: 75m
            memory: 128Mi
```

Note

Directly adding sensitive information, such as API keys, to your Kubernetes manifest files isn't secure and may accidentally get committed to code repositories. We added it here for simplicity. For production workloads, use **Managed Identity** to authenticate with Azure Service Bus or store your secrets in **Azure Key Vault**.

4. Save and close the updated `aks-store-quickstart.yaml` file.

Deploy the updated application

- Deploy the updated application using the `kubectl apply` command.

  ConsoleCopy
  ```
  kubectl apply -f aks-store-quickstart.yaml
  ```

  The following example output shows the successfully updated resources:

  OutputCopy
  ```
  deployment.apps/order-service configured
  service/order-service unchanged
  deployment.apps/product-service unchanged
  service/product-service unchanged
  deployment.apps/store-front configured
  service/store-front unchanged
  ```

Test the application

Place a sample order

1. Get the external IP address of the `store-front` service using the `kubectl get service` command.

   ConsoleCopy
   ```
   kubectl get service store-front
   ```

2. Navigate to the external IP address of the `store-front` service in your browser.
3. Place an order by choosing a product and selecting **Add to cart**.
4. Select **Cart** to view your order, and then select **Checkout**.

View the order in the Azure Service Bus queue

1. Navigate to the Azure portal and open the Azure Service Bus namespace you created earlier.
2. Under **Entities**, select **Queues**, and then select the **orders** queue.
3. In the **orders** queue, select **Service Bus Explorer**.
4. Select **Peek from start** to view the order you submitted.

Next steps

In this tutorial, you used Azure Service Bus to update and test the sample application. You learned how to:

- Create an Azure Service Bus namespace and queue.
- Update the Kubernetes manifest file to use the Azure Service Bus queue.
- Test the updated application by placing an order.

In the next tutorial, you learn how to scale an application in AKS.

Scale applications in AKS

Scale applications in Azure Kubernetes Service (AKS)

- 10/26/2023

In this article

If you followed the previous tutorials, you have a working Kubernetes cluster and Azure Store Front app.

In this tutorial, part six of seven, you scale out the pods in the app, try pod autoscaling, and 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.

Before you begin

In previous tutorials, you packaged an application into a container image, uploaded the image to Azure Container Registry, created an AKS cluster, deployed an application, and used Azure Service Bus to redeploy an updated application. If you haven't completed these steps and want to follow along, start with Tutorial 1 - Prepare application for AKS.

- Azure CLI
- Azure PowerShell

This tutorial requires Azure CLI version 2.34.1 or later. Run `az --version` to find the version. If you need to install or upgrade, see Install Azure CLI.

Manually scale pods

1. View the pods in your cluster using the kubectl get command.

   ConsoleCopy

```
kubectl get pods
```

The following example output shows the pods running the Azure Store Front app:

OutputCopy

```
NAME                                READY     STATUS     RESTARTS   AGE
order-service-848767080-tf34m       1/1       Running    0          31m
product-service-4019737227-2q2qz    1/1       Running    0          31m
store-front-2606967446-2q2qz        1/1       Running    0          31m
```

2. Manually change the number of pods in the *store-front* deployment using the kubectl scale command.

   ConsoleCopy

   ```
   kubectl scale --replicas=5 deployment.apps/store-front
   ```

3. Verify the additional pods were created using the kubectl get pods command.

   ConsoleCopy

   ```
   kubectl get pods
   ```

   The following example output shows the additional pods running the Azure Store Front app:

   OutputCopy

   ```
                                       READY     STATUS     RESTARTS   AGE
   store-front-2606967446-2q2qzc       1/1       Running    0          15m
   store-front-3309479140-2hfh0        1/1       Running    0          3m
   store-front-3309479140-bzt05        1/1       Running    0          3m
   store-front-3309479140-fvcvm        1/1       Running    0          3m
   store-front-3309479140-hrbf2        1/1       Running    0          15m
   store-front-3309479140-qphz8        1/1       Running    0          3m
   ```

Autoscale pods

To use the horizontal pod autoscaler, all containers and pods must have defined CPU requests and limits. In the `aks-store-quickstart` deployment, the *front-end* container requests 1m CPU with a limit of 1000m CPU.

These resource requests and limits are defined for each container, as shown in the following condensed example YAML:

YAMLCopy

```yaml
...
  containers:
  - name: store-front
    image: ghcr.io/azure-samples/aks-store-demo/store-front:latest
    ports:
```

```
    - containerPort: 8080
      name: store-front
...
    resources:
      requests:
        cpu: 1m
...
      limits:
        cpu: 1000m
...
```

Autoscale pods using a manifest file

1. Create a manifest file to define the autoscaler behavior and resource limits, as shown in the following condensed example manifest file `aks-store-quickstart-hpa.yaml`:

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

2. Apply the autoscaler manifest file using the `kubectl apply` command.

   ConsoleCopy
   ```
   kubectl apply -f aks-store-quickstart-hpa.yaml
   ```

3. Check the status of the autoscaler using the `kubectl get hpa` command.

   ConsoleCopy
   ```
   kubectl get hpa
   ```

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

Note

You can enable the Kubernetes-based Event-Driven Autoscaler (KEDA) AKS add-on to your cluster to drive scaling based on the number of events needing to be processed. For more

information, see **Enable simplified application autoscaling with the Kubernetes Event-Driven Autoscaling (KEDA) add-on (Preview)**.

Manually scale AKS nodes

If you created your Kubernetes cluster using the commands in the previous tutorials, your cluster has two nodes. If you want to increase or decrease this amount, you can manually adjust the number of nodes.

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.

- Azure CLI
- Azure PowerShell

  - Scale your cluster nodes using the az aks scale command.

    Azure CLICopy
    Open Cloudshell
    ```
    az aks scale --resource-group myResourceGroup --name myAKSCluster --node-count 3
    ```

    Once the cluster successfully scales, your output will be similar to following example output:

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

You can also autoscale the nodes in your cluster. For more information, see Use the cluster autoscaler with node pools.

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.

In the next tutorial, you learn how to upgrade Kubernetes in your AKS cluster.

Upgrade Kubernetes in Azure Kubernetes Service

Upgrade an Azure Kubernetes Service (AKS) cluster

- 11/07/2023

In this article

As part of the application and cluster lifecycle, you might want to upgrade to the latest available version of Kubernetes. You can upgrade your Azure Kubernetes Service (AKS) cluster using the Azure CLI, Azure PowerShell, or the Azure portal.

In this tutorial, part seven of seven, you upgrade an AKS cluster. You learn how to:

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

Before you begin

In previous tutorials, you packaged an application into a container image and uploaded the container image to Azure Container Registry (ACR). You also created an AKS cluster and deployed an application to it. If you haven't completed these steps and want to follow along, start with Tutorial 1 - Prepare application for AKS.

If using Azure CLI, this tutorial requires Azure CLI version 2.34.1 or later. Run `az --version` to find the version. If you need to install or upgrade, see Install Azure CLI.

If using Azure PowerShell, this tutorial requires Azure PowerShell version 5.9.0 or later. Run `Get-InstalledModule -Name Az` to find the version. If you need to install or upgrade, see Install Azure PowerShell.

Get available cluster versions

- Azure CLI
- Azure PowerShell
- Azure portal

    - Before you upgrade, check which Kubernetes releases are available for your cluster using the az aks get-upgrades command.

      Azure CLICopy
      Open Cloudshell
      ```
      az aks get-upgrades --resource-group myResourceGroup --name myAKSCluster
      ```

      The following example output shows the current version as *1.26.6* and lists the available versions under `upgrades`:

      OutputCopy
      ```
      {
        "agentPoolProfiles": null,
        "controlPlaneProfile": {
          "kubernetesVersion": "1.26.6",
          ...
          "upgrades": [
            {
              "isPreview": null,
              "kubernetesVersion": "1.27.1"
            },
            {
              "isPreview": null,
              "kubernetesVersion": "1.27.3"
            }
          ]
        },
        ...
      }
      ```

Upgrade an AKS cluster

AKS nodes are carefully cordoned and drained to minimize any potential disruptions to running applications. During this process, AKS performs the following steps:

- Adds a new buffer node (or as many nodes as configured in max surge) to the cluster that runs the specified Kubernetes version.

- Cordons and drains one of the old nodes to minimize disruption to running applications. If you're using max surge, it cordons and drains as many nodes at the same time as the number of buffer nodes specified.
- When the old node is fully drained, it's reimaged to receive the new version and becomes 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 is deleted, maintaining the existing agent node count and zone balance.

Note

If no patch is specified, the cluster automatically upgrades to the specified minor version's latest GA patch. For example, setting `--kubernetes-version` to `1.21` results in the cluster upgrading to `1.21.9`.

For more information, see **Supported Kubernetes minor version upgrades in AKS**.

You can either manually upgrade your cluster or configure automatic cluster upgrades. **We recommend you configure automatic cluster upgrades to ensure your cluster is always running the latest version of Kubernetes**.

Manually upgrade cluster

- Azure CLI
- Azure PowerShell
- Azure portal

  - Upgrade your cluster using the az aks upgrade command.

    Azure CLICopy
    Open Cloudshell

    ```
    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 you can't upgrade from *1.14.x* to *1.16.x* directly. To upgrade from *1.14.x* to *1.16.x*, you must first upgrade from *1.14.x* to *1.15.x*, then perform another upgrade from *1.15.x* to *1.16.x*.

The following example output shows the result of upgrading to *1.27.3*. Notice the `kubernetesVersion` now shows *1.27.3*:

OutputCopy

```
{
  "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/m
anagedClusters/myAKSCluster",
  "kubernetesVersion": "1.27.3",
  "location": "eastus",
  "name": "myAKSCluster",
  "type": "Microsoft.ContainerService/ManagedClusters"
}
```

Configure automatic cluster upgrades

- Azure CLI
- Azure PowerShell
- Azure portal

  - Set an auto-upgrade channel on your cluster using the az aks update command with the `--auto-upgrade-channel` parameter set to `patch`.

    Azure CLICopy
    Open Cloudshell

    ```
    az aks update --resource-group myResourceGroup --name myAKSCluster --
    auto-upgrade-channel patch
    ```

For more information, see Automatically upgrade an Azure Kubernetes Service (AKS) cluster.

Upgrade AKS node images

AKS regularly provides new node images. Linux node images are updated weekly, and Windows node images are updated monthly. We recommend upgrading your node images frequently to use the latest AKS features and security updates. For more information, see Upgrade node images in Azure Kubernetes Service (AKS). To configure automatic node image upgrades, see Automatically upgrade Azure Kubernetes Service (AKS) cluster node operating system images.

View the upgrade events

Note

When you upgrade your cluster, the following Kubernetes events might occur on the nodes:

- **Surge**: Create a surge node.
- **Drain**: Evict pods from the node. Each pod has a *five minute timeout* to complete the eviction.
- **Update**: Update of a node has succeeded or failed.
- **Delete**: Delete a surge node.

- View the upgrade events in the default namespaces using the `kubectl get events` command.

  ConsoleCopy
  ```
  kubectl get events --field-selector source=upgrader
  ```

  The following example output shows some of the above events listed during an upgrade:

  OutputCopy
  ```
  ...
  default 2m1s Normal Drain node/aks-nodepool1-96663640-vmss000001 Draining
  node: [aks-nodepool1-96663640-vmss000001]
  ...
  default 9m22s Normal Surge node/aks-nodepool1-96663640-vmss000002 Created
  a surge node [aks-nodepool1-96663640-vmss000002 nodepool1] for agentpool
  %!s(MISSING)
  ...
  ```

Validate an upgrade

- Azure CLI
- Azure PowerShell
- Azure portal

    - Confirm the upgrade was successful using the az aks show command.

      Azure CLICopy
      Open Cloudshell
      ```
      az aks show --resource-group myResourceGroup --name myAKSCluster --output
      table
      ```

      The following example output shows the AKS cluster runs *KubernetesVersion 1.27.3*:

      OutputCopy
      ```
      Name          Location    ResourceGroup    KubernetesVersion
      CurrentKubernetesVersion  ProvisioningState    Fqdn
      ------------  ----------  ---------------  ------------------  ---------
      --------------  ------------------  ----------------------------------
      ----------------------------
      myAKSCluster  eastus      myResourceGroup  1.27.3              1.27.3
      Succeeded            myaksclust-myresourcegroup-19da35-
      bd54a4be.hcp.eastus.azmk8s.io
      ```

Delete the cluster

As this tutorial is the last part of the series, you might want to delete your AKS cluster to avoid incurring Azure charges.

- Azure CLI
- Azure PowerShell
- Azure portal

    - Remove the resource group, container service, and all related resources using the az group delete command.

      Azure CLICopy
      Open Cloudshell
      ```
      az group delete --name myResourceGroup --yes --no-wait
      ```

Note

When you delete the cluster, the Microsoft Entra service principal used by the AKS cluster isn't 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 doesn't require that you 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 your Kubernetes nodes.
- Validate a successful upgrade.

For more information on AKS, see the AKS overview. For guidance on how to create full solutions with AKS, see the AKS solution guidance.