**Creating an Azure Kubernetes Service Cluster**

**What is Azure Kubernetes Service?**

Kubernetes is a rapidly evolving platform that manages container-based applications and their associated networking and storage components. The focus is 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 that benefit from Kubernetes orchestrating and managing the availability of those 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.

Azure Kubernetes Service (AKS) provides a managed Kubernetes service that reduces the complexity for deployment and core management tasks, including coordinating upgrades. The AKS cluster masters are managed by the Azure platform, and you only pay for the AKS nodes that run your applications. AKS is built on top of the open-source Azure Container Service Engine (acs-engine).

**Kubernetes cluster architecture**

A Kubernetes cluster is divided into two components:

* *Cluster master* nodes provide the core Kubernetes services and orchestration of application workloads.
* *Nodes* run your application workloads.

****

**Cluster master**

When you create an AKS cluster, a cluster master is automatically created and configured. This cluster master is provided as a managed Azure resource abstracted from the user. There is no cost for the cluster master, only the nodes that are part of the AKS cluster.

The cluster master includes the following core Kubernetes components:

* **kube-apiserver** - The 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.
* **etcd** - To maintain the state of your Kubernetes cluster and configuration, the highly available *etcd* is a key value store within Kubernetes.
* **kube-scheduler** - When you create or scale applications, the Scheduler determines what nodes can run the workload and starts them.
* **kube-controller-manager** - 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 cluster master, 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 cluster master and nodes. Interaction with the cluster master occurs through Kubernetes APIs, such as kubectl or the Kubernetes dashboard.

This managed cluster master means that you do not need to configure components like a highly available etcd store, but it also means that you cannot access the cluster master directly. Upgrades to Kubernetes are orchestrated through the Azure CLI or Azure portal, which upgrades the cluster master and then the nodes. To troubleshoot possible issues, you can review the cluster master logs through Azure Log Analytics.

If you need to configure the cluster master in a particular way or need direct access to them, you can deploy your own Kubernetes cluster using aks-engine.

**Nodes and node pools**

To run your applications and supporting services, you need a Kubernetes node. An AKS cluster has one or more nodes, which is an Azure virtual machine (VM) that runs the Kubernetes node components and con­tainer runtime:

* The kubelet is the Kubernetes agent that processes the orchestration requests from the cluster master and scheduling of running the requested containers.
* Virtual networking is handled by the *kube-proxy* on each node. The proxy routes network traffic and manages IP addressing for services and pods.
* The *container runtime* is the component that allows containerized applications to run and interact with additional resources such as the virtual network and storage. In AKS, Docker is used as the container runtime.

****

The Azure VM size for your nodes defines how many CPUs, how much memory, and the size and type of storage available (such as high-performance SSD or regular HDD). If you anticipate a need for applica­tions that require large amounts of CPU and memory or high-performance storage, plan the node size accordingly. You can also scale up the number of nodes in your AKS cluster to meet demand.

In AKS, the VM image for the nodes in your cluster is currently based on Ubuntu Linux. When you create an AKS cluster or scale up the number of nodes, the Azure platform creates the requested number of VMs and configures them. There is no manual configuration for you to perform.

If you need to use a different host OS, container runtime, or include custom packages, you can deploy your own Kubernetes cluster using aks-engine. The upstream aks-engine releases features and provide configuration options before they are officially supported in AKS clusters. For example, if you wish to use Windows containers or a container runtime other than Docker, you can use aks-engine to configure and deploy a Kubernetes cluster that meets your current needs.

**Resource reservations**

You don't need to manage the core Kubernetes components on each node, such as the *kubelet*, *kube-proxy*, and *kube-dns*, but they do consume some of the available compute resources. To maintain node performance and functionality, the following compute resources are reserved on each node:

* **CPU** - 60ms
* **Memory** - 20% up to 4 GiB

These reservations mean that the amount of available CPU and memory for your applications may appear less than the node itself contains. If there are resource constraints due to the number of applications that you run, these reservations ensure CPU and memory remains available for the core Kubernetes compo­nents. The resource reservations cannot be changed.

For example:

* 1. **Standard DS2 v2** node size contains 2 vCPU and 7 GiB memory
  + 20% of 7 GiB memory = 1.4 GiB
  + A total of (7 - 1.4) = 5.6 GiB memory is available for the node
  1. **Standard E4s v3** node size contains 4 vCPU and 32 GiB memory
  + 20% of 32 GiB memory = 6.4 GiB, but AKS only reserves a maximum of 4 GiB
  + A total of (32 - 4) = 28 GiB is available for the node

**Node pools**

Nodes of the same configuration are grouped together into *node pools*. A Kubernetes cluster contains one or more node pools. 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.

When you scale or upgrade an AKS cluster, the action is performed against the default node pool. For upgrade operations, running containers are scheduled on other nodes in the node pool until all the nodes are successfully upgraded.

**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, although there are advanced scenarios where a pod may contain multiple containers. These 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 limits* to request a certain amount of CPU or memory resources. The Kubernetes Scheduler tries to schedule the pods to run on a node with available resources to meet the request. You can also specify maximum resource limits that prevent a given pod from consuming too much compute resource from the underlying node. A best practice is to include resource limits for all pods to help the Kubernetes Scheduler understand what resources are needed and permit­ted.

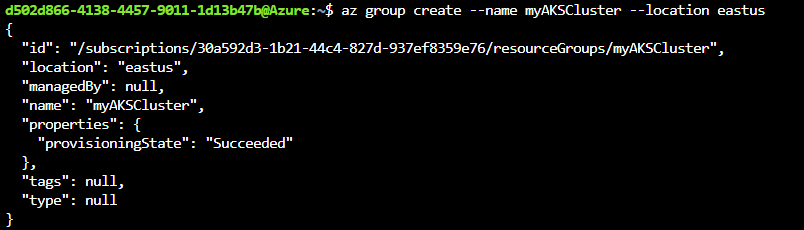
A pod is a logical resource, but the container(s) are where the application workloads run. Pods are typical­ly ephemeral, disposable resources, and individually scheduled pods miss some of the high availability and redundancy features Kubernetes provides. Instead, pods are usually deployed and managed by Kubernetes *Controllers*, such as the Deployment Controller.

**Deploy an AKS cluster using Azure CLI**

1. **Create a resource group**

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

az group create --name myAKSCluster --location eastus

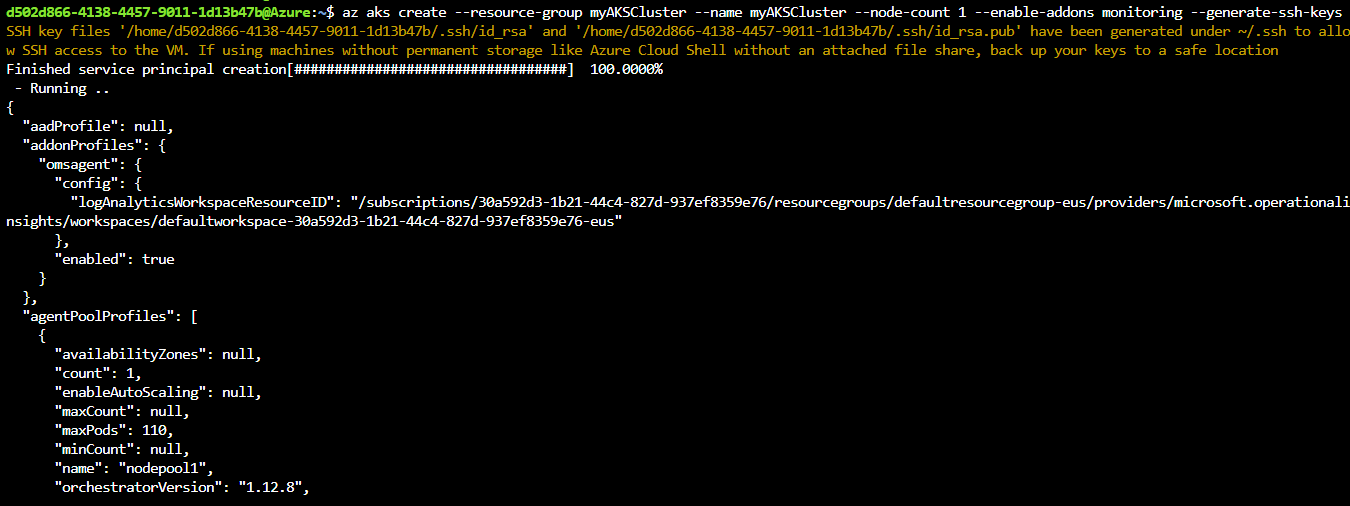


1. **Create AKS cluster**

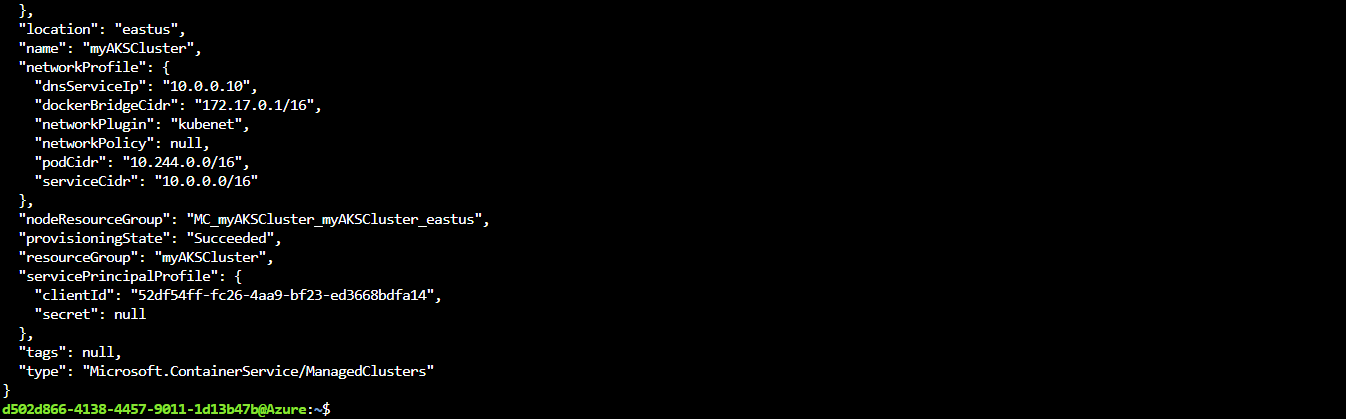
Use the az aks create command to create an AKS cluster. The following example creates a cluster named myAKSCluster with one node. Container health monitoring is also enabled using the *–enable-ad­dons monitoring* parameter.

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

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







1. **Connect to the cluster**

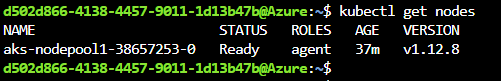
To configure kubectl to connect to your Kubernetes cluster, use the az aks get-credentials command. This step downloads credentials and configures the Kubernetes CLI to use them.

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



To verify the connection to your cluster, use the kubectl get command to return a list of the cluster nodes. It can take a few minutes for the nodes to appear.

kubectl get nodes



1. **Run the application**

Create a file named azure-vote.yaml and copy the below following YAML code in it.



**Type “i” for inserting the below scripts and then press esc button and type :wq**

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: redis

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

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:

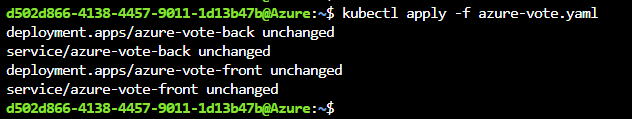
- port: 80

selector:

app: azure-vote-front

**Use the kubectl apply command to run the application.**

kubectl apply -f azure-vote.yaml



1. **Test the application**

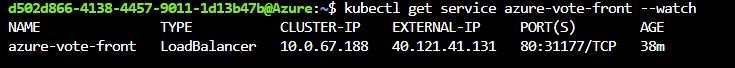
As the application is run, a Kubernetes service is created that exposes the application front end to the internet. This process can take a few minutes to complete.

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 appears as pending.

Once the EXTERNAL-IP address has changed from pending to an IP address, use CTRL-C to stop the kubectl watch process.



Now browse to the external IP address to see the Azure Vote App.

