As application development moves towards a container-based approach, the need to orchestrate and manage resources is important. Kubernetes is the leading platform that provides the ability to provide reliable scheduling of fault-tolerant application workloads. Azure Kubernetes Service (AKS) is a managed Kubernetes offering that further simplifies container-based application deployment and management.

This article introduces the core Kubernetes infrastructure components such as the *control plane*, *nodes*, and *node pools*. Workload resources such as *pods*, *deployments*, and *sets* are also introduced, along with 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. 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.

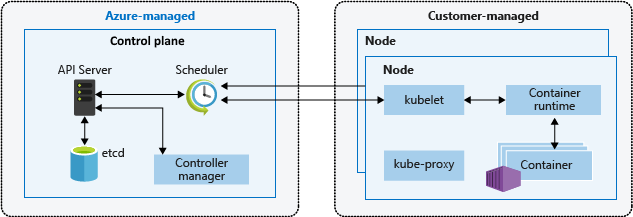
As an open platform, Kubernetes allows you to build your applications with your preferred programming language, OS, libraries, or messaging bus. Existing continuous integration and continuous delivery (CI/CD) tools can integrate with Kubernetes to schedule and deploy releases.

Azure Kubernetes Service (AKS) provides a managed Kubernetes service that reduces the complexity for deployment and core management tasks, including coordinating upgrades. The AKS control plane is 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 Kubernetes Service Engine (aks-engine).

**Kubernetes cluster architecture**

A Kubernetes cluster is divided into two components:

* *Control plane* nodes provide 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 as a managed Azure resource abstracted from the user. There's no cost for the control plane, only the nodes that are part of 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:

* *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 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.

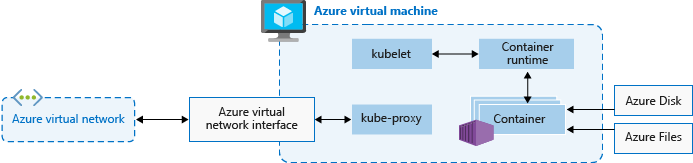
This managed control plane means you don't need to configure components like a highly available *etcd* store, but it also means you can't access the control plane directly. Upgrades to Kubernetes are orchestrated through the Azure CLI or Azure portal, which upgrades the control plane and then the nodes. To troubleshoot possible issues, you can review the control plane logs through Azure Monitor logs.

If you need to configure the control plane in a particular way or need direct access to it, 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 control plane 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, Moby 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 out 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 or Windows Server 2019. When you create an AKS cluster or scale out the number of nodes, the Azure platform creates the requested number of VMs and configures them. There's no manual configuration for you to perform. Agent nodes are billed as standard virtual machines, so any discounts you have on the VM size you're using (including Azure reservations) are automatically applied.

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 provides configuration options before they are officially supported in AKS clusters. For example, if you wish to use a container runtime other than Moby, you can use aks-engine to configure and deploy a Kubernetes cluster that meets your current needs.

**Resource reservations**

Node resources are utilized by AKS to make the node function as part of your cluster. This usage can create a discrepancy between your node's total resources and the resources allocatable when used in AKS. This information is important to note when setting requests and limits for user deployed pods.

To find a node's allocatable resources, run:

kubectlCopy

kubectl describe node [NODE\_NAME]

To maintain node performance and functionality, resources are reserved on each node by AKS. As a node grows larger in resources, the resource reservation grows due to a higher amount of user deployed pods needing management.

**Note**

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

* **CPU** - reserved CPU is dependent on node type and cluster configuration, which may cause less allocatable CPU due to running additional features
* **Memory** - memory utilized by AKS includes the sum of two values.

1. The kubelet daemon is installed on all Kubernetes agent nodes to manage container creation and termination. By default on AKS, this daemon has the following eviction rule: *memory.available<750Mi*, which means a node must always have at least 750 Mi allocatable at all times. When a host is below that threshold of available memory, the kubelet will terminate one of the running pods to free memory on the host machine and protect it. This action is triggered once available memory decreases beyond the 750Mi threshold.
2. The second value is 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

The above rules for memory and CPU allocation are used to keep agent nodes healthy, including some hosting system pods that are critical to cluster health. These allocation rules also cause the node to report less allocatable memory and CPU than it normally 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) = 0.75GB + 1GB + 0.6GB = 2.35GB / 7GB = 33.57% 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.

**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.

**Note**

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

When you scale or upgrade an AKS cluster, the action is performed against the default node pool. You can also 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.

**Node selectors**

In an AKS cluster that contains 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, such as the 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*:

YAMLCopy

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

**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 requests* 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 which resources are needed and permitted.