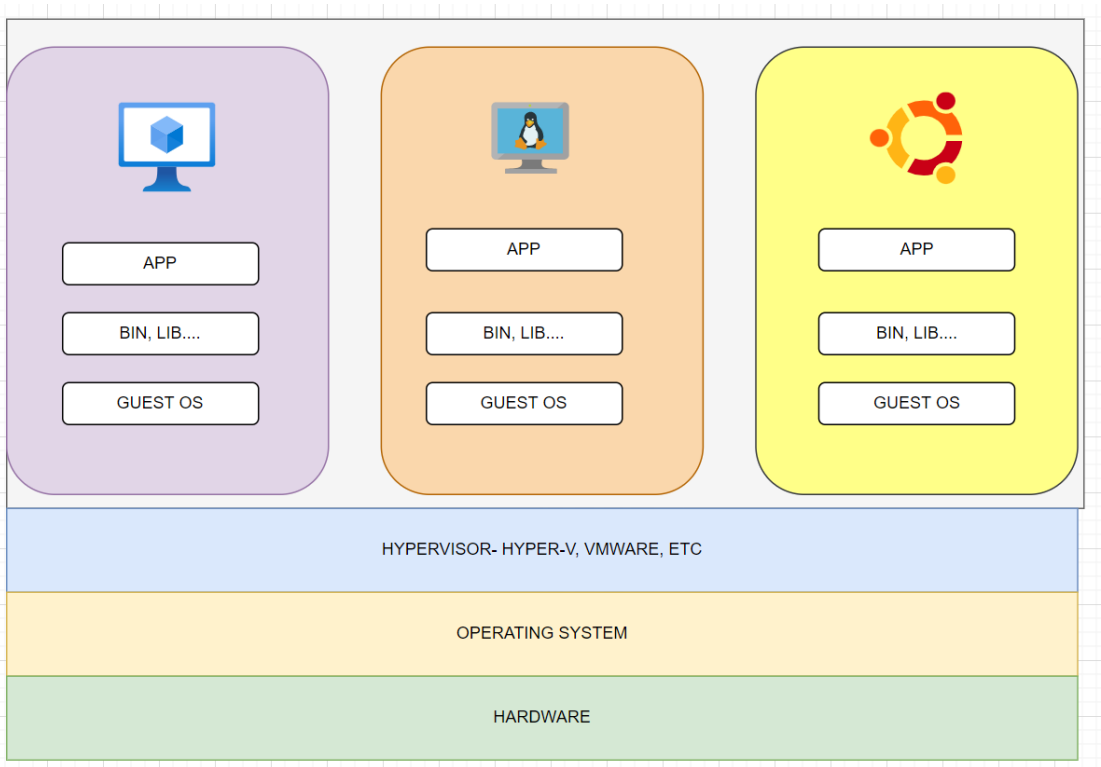
What is Azure Kubernetes Service (AKS)?

AKS is a fully managed container management service by Microsoft, it offers serverless continuous integration & continuous deployment capabilities

- 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.
- AKS provides a managed Kubernetes service that reduces the complexity of deployment and core management tasks, like upgrade coordination.

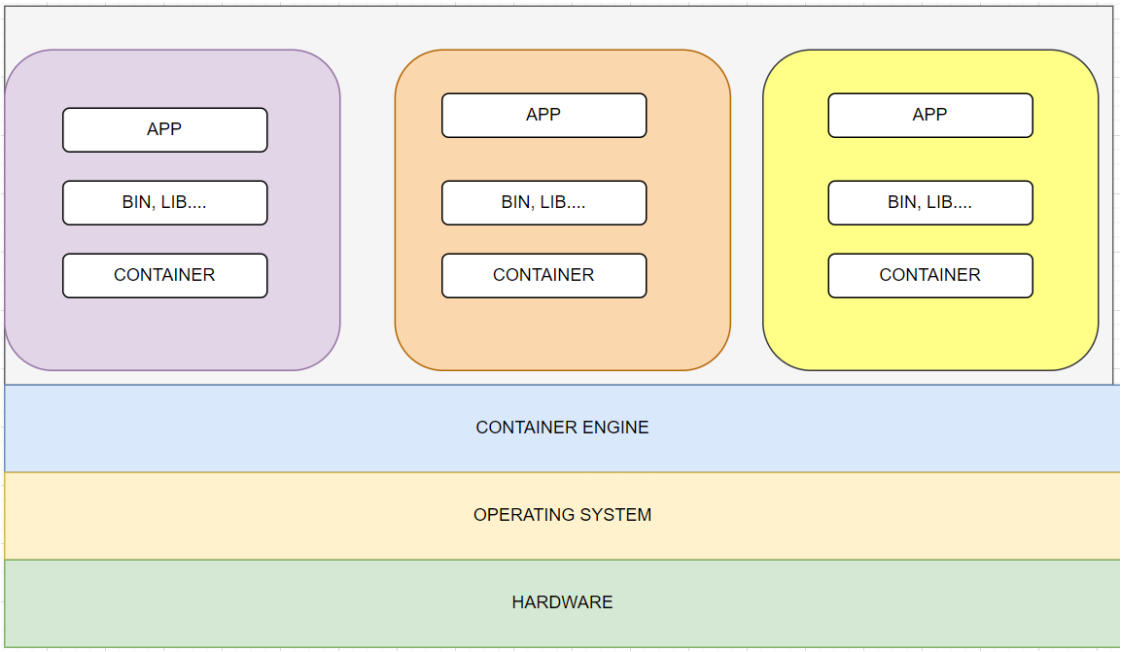
Virtual Machine vs a Container



Virtual Machine:

- It runs on top of an emulating software called the hypervisor which sit between the hardware and the virtual machine.
- The hypervisor is the key to enable virtualization. It manages the sharing of physical resources into virtual machines.
- Each virtual machine runs its own guest operating system. They are less agile and have low portability than containers

Containers



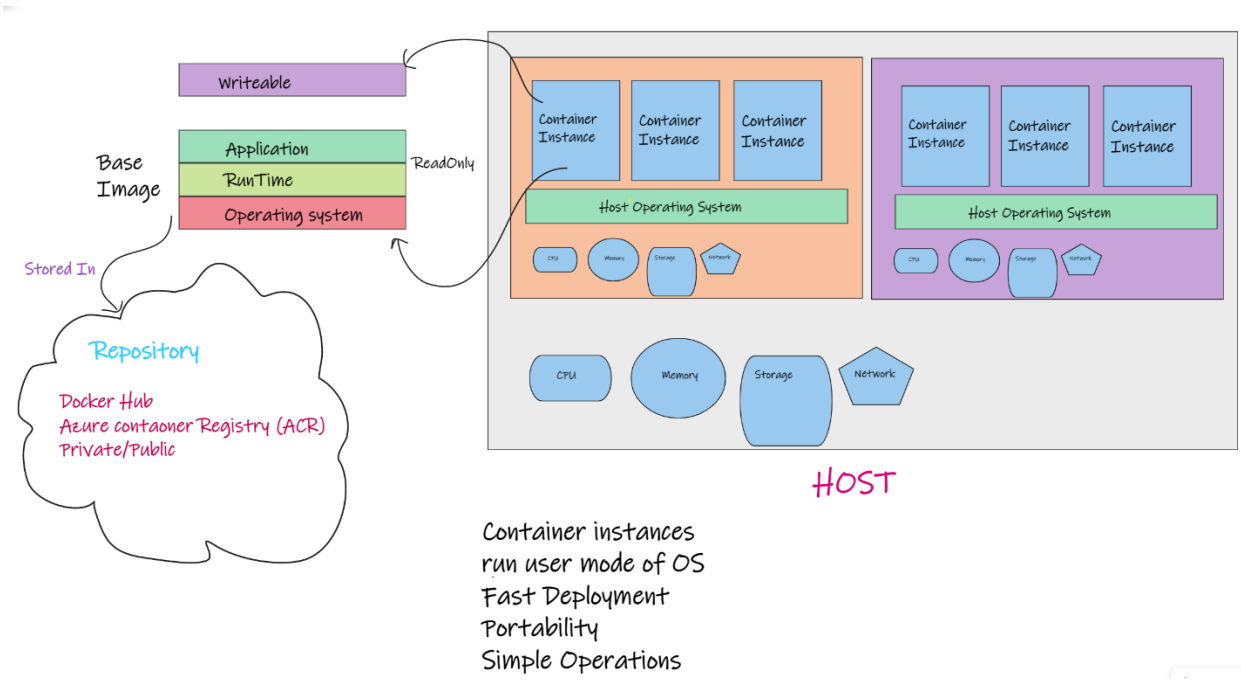
Container:

- It sits on the top of a physical server and its host operating system.
- They share a common operating system that requires care and feeding for bug fixes and patches.
- They are more agile and have high portability than virtual machines.

VM vs Container

SNo.	Virtual Machines(VM)	Containers
1	VM is piece of software that allows you to install other software inside of it so you basically control it virtually as opposed to installing the software directly on the computer.	While a container is a software that allows different functionalities of an application independently.
2.	Applications running on VM system can run different OS.	While applications running in a container environment share a single OS.
3.	VM virtualizes the computer system.	While containers virtualize the operating system only.
4.	VM size is very large.	While the size of container is very light; i.e. a few megabytes.
5.	VM takes minutes to run, due to large size.	While containers take a few seconds to run.
6.	VM uses a lot of system memory.	While containers require very less memory.
7.	VM is more secure.	While containers are less secure.
8.	VM's are useful when we require all of OS resources to run various applications.	While containers are useful when we are required to maximise the running applications using minimal servers.
9.	Examples of VM are: Hyper-V, KVM, Xen, VMware.	While examples of containers are:RancherOS, PhotonOS, Containers by Docker.

Container Architecture

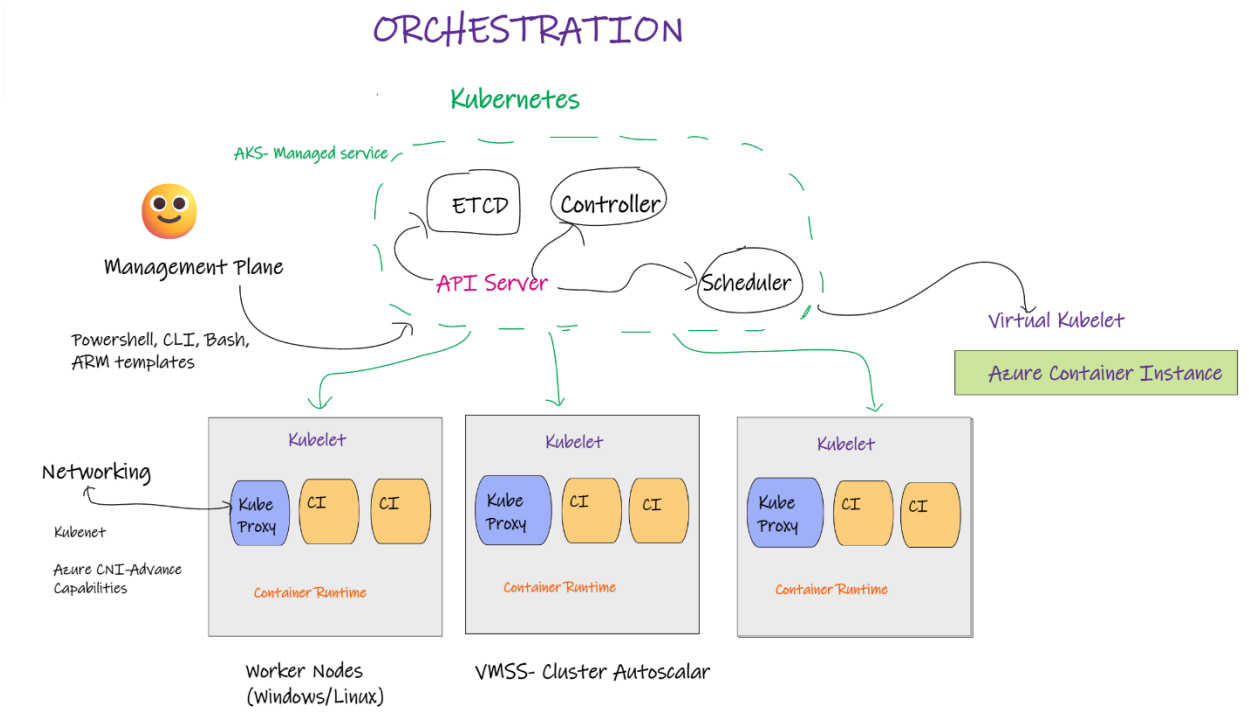


Container instances
run user mode of OS
Fast Deployment
Portability
Simple Operations

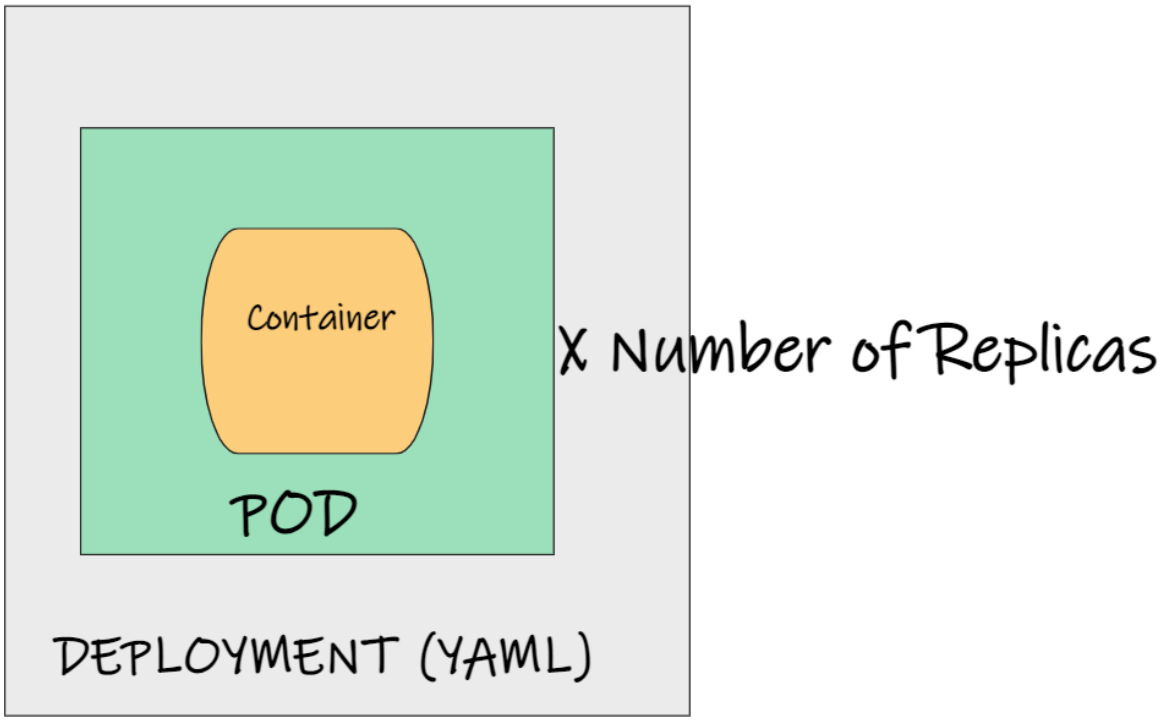
ISSUES

Deployment
Autoscaling
Update
Libraries
Storage
Network

Azure Kubernetes Service



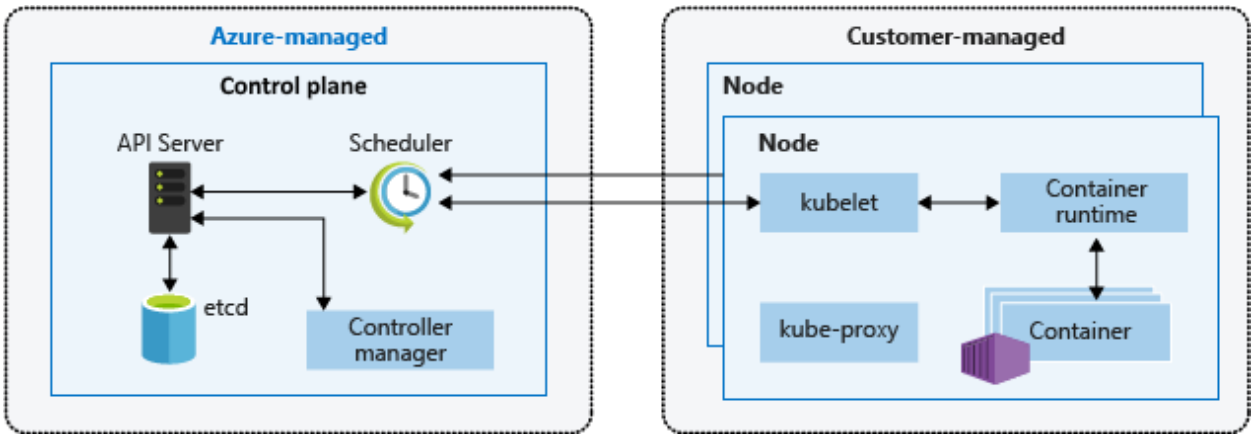
DEPLOYMENT



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
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.

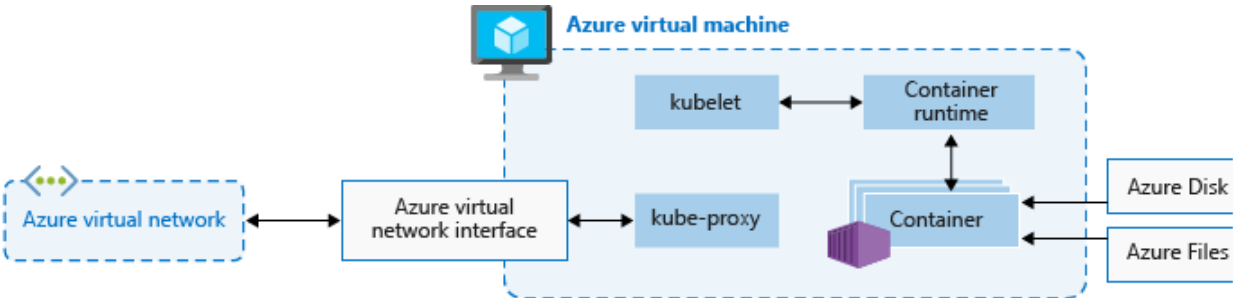
- AKS provides a single-tenant control plane, with a dedicated API server, scheduler, etc.
- You define the number and size of the nodes, and the Azure platform configures the secure communication between the control plane and nodes.
- Interaction with the control plane occurs through Kubernetes APIs,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

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
kubelet	The Kubernetes agent that processes the orchestration requests from the control plane along with scheduling and running the requested containers.
kube-proxy	Handles virtual networking on each node. The proxy routes network traffic and manages IP addressing for services and pods.
container runtime	Allows containerized applications to run and interact with additional resources, such as the virtual network and storage. AKS clusters using Kubernetes version 1.19+ for Linux node pools use containerd as their container runtime. Beginning in Kubernetes version 1.20 for Windows node pools, containerd can be used in preview for the container

runtime, but Docker is still the default container runtime. AKS clusters using prior versions of Kubernetes for node pools use 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.
- If you need advanced configuration and control on your Kubernetes node container runtime and OS, you can deploy a self-managed cluster using Cluster API Provider Azure.

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.
- 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.

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 *"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:
    "kubernetes.io/os": linux
```

Pods

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.

- 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 to 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.

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.

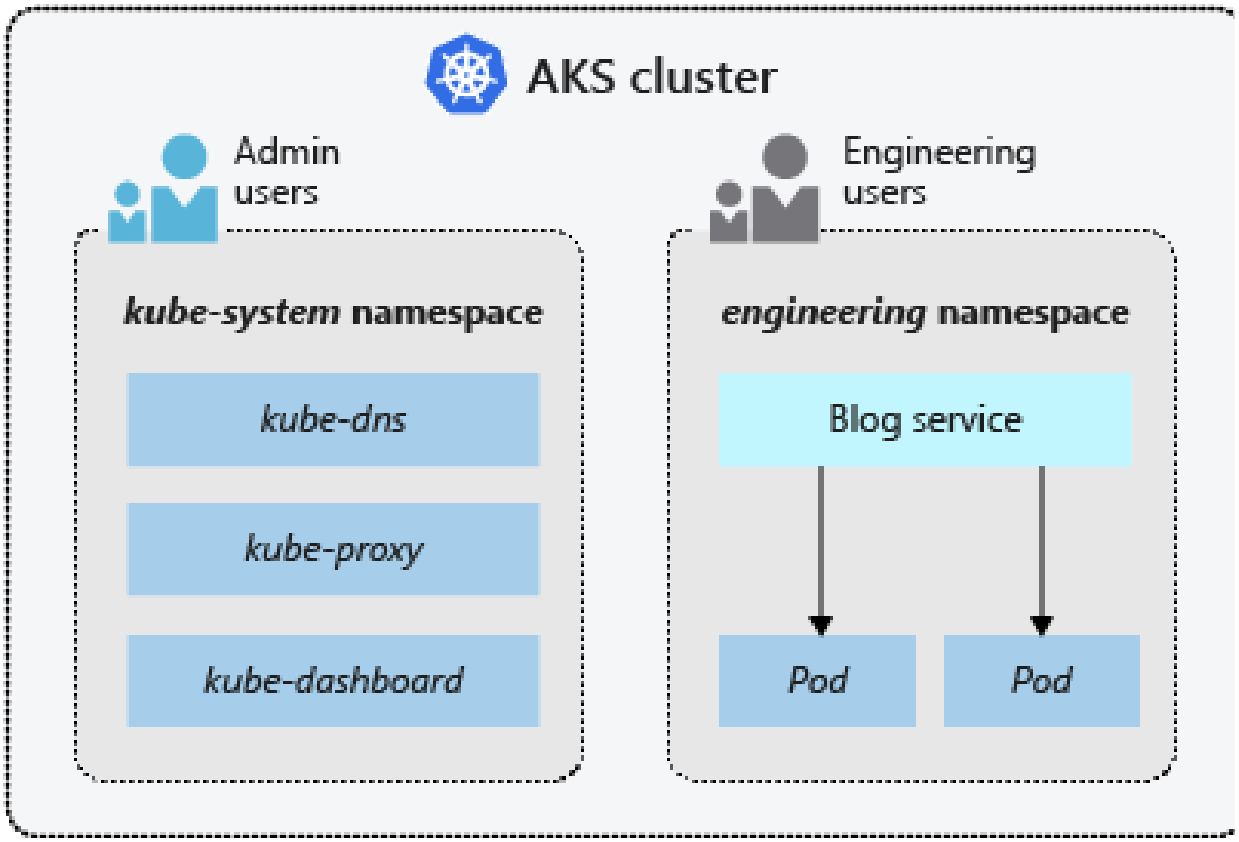
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: mcr.microsoft.com/oss/nginx/nginx:1.15.2-alpine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

```
        memory: 64Mi
      limits:
        cpu: 500m
        memory: 256Mi
```

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.

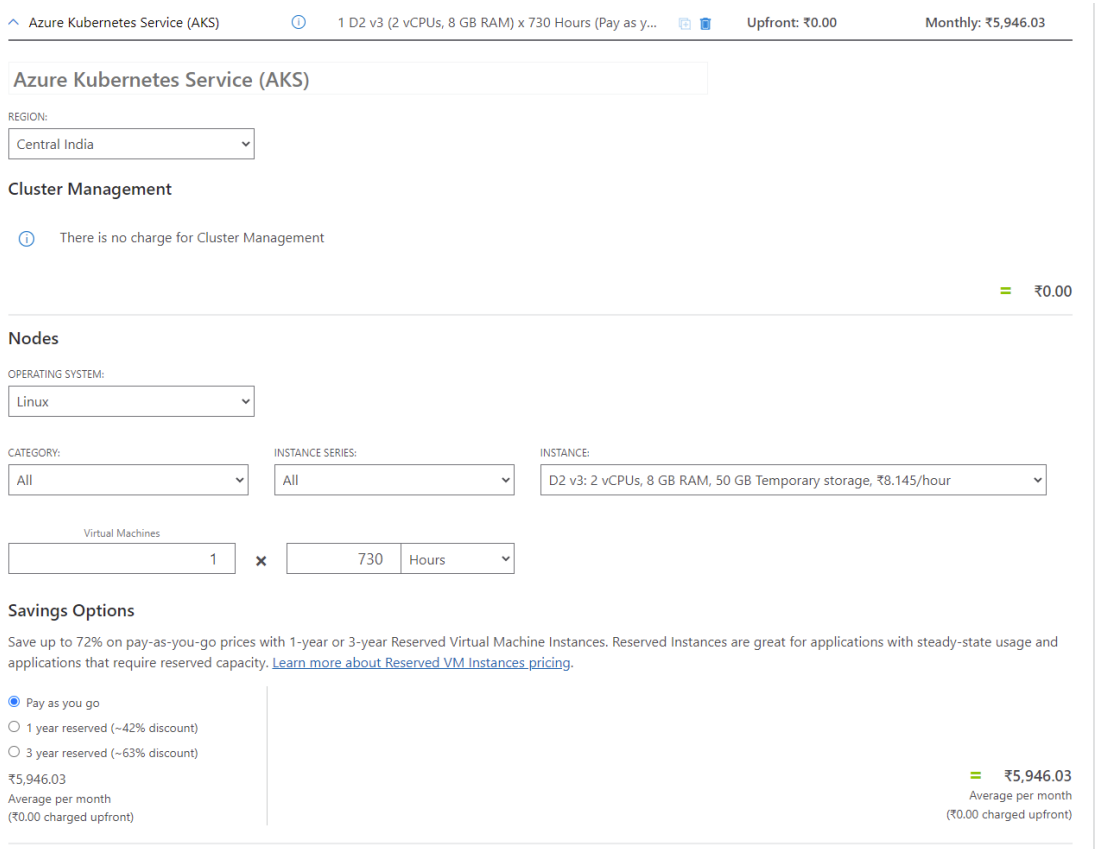
- For example, you can create namespaces to separate business groups.
- Users can only interact with resources within their assigned namespaces.



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

Namespace	Description
default	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 kubectl get pods, the default namespace is used when none is specified.
kube-system	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.
kube-public	Typically not used, but can be used for resources to be visible across the whole cluster, and can be viewed by any user.

Azure Kubernetes Service (AKS) pricing



For latest pricing visit- [AKS Pricing]