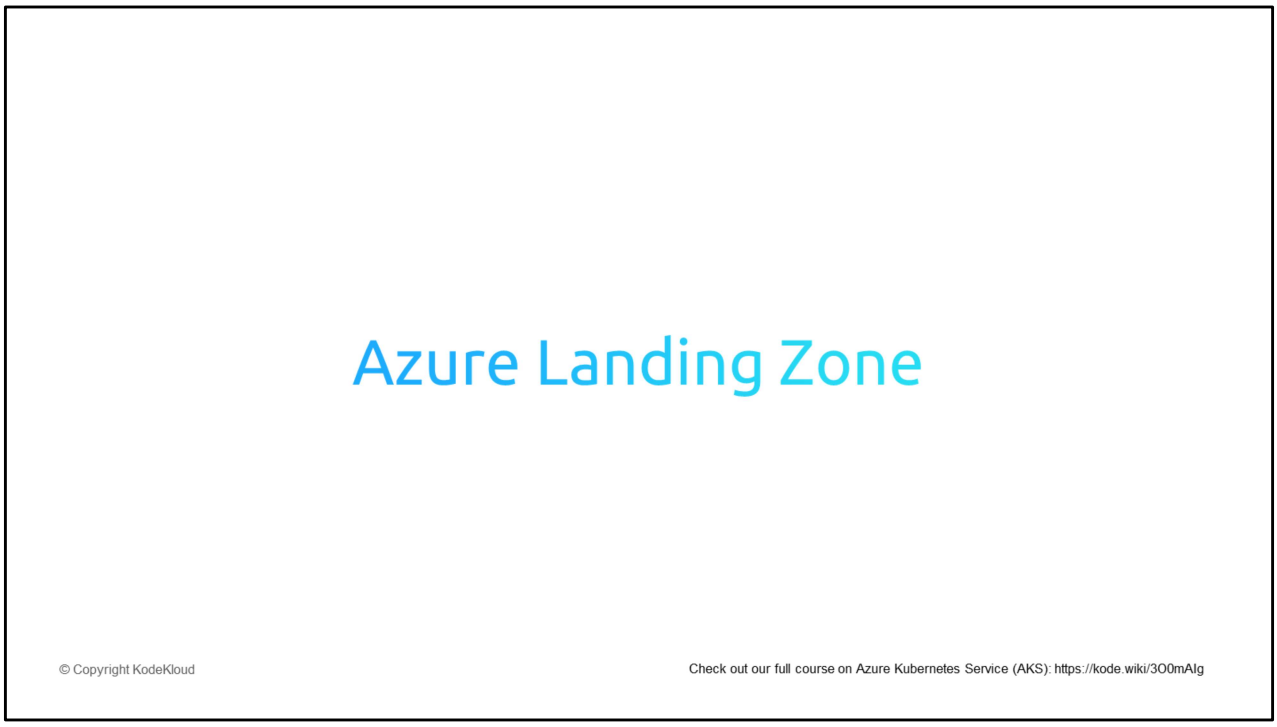
KodeKloud

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

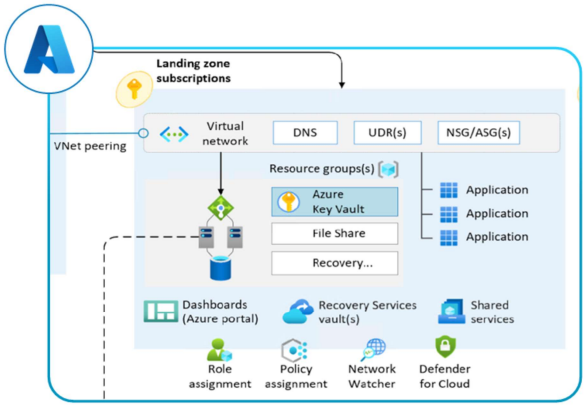
Agenda

- Topic 01: Azure Landing Zone
- Topic 02: Azure Compute
- Topic 03: Azure Storage
- Topic 04: Azure Network & Security
- Topic 05: Azure Database offering



Have you ever written a program but didn’t have an application server to run it on? Maybe you're a developer and you’ve written code that executes a basic task.

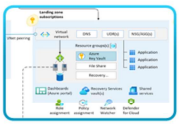
Azure Landing Zone



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

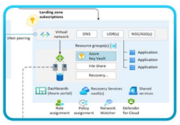
Azure landing zone is a Cloud management system that makes deployment and configuration of cloud applications simple, repeatable, consistent and compliant to your organizational standards and requirements.
No two organizations are same and hence their implementation of landing zones will be unique to meet their requirements. As a Kubernetes engineer or Architect, you might not be required to design the Landing Zone but it will be valuable to know these concepts in general and how your organization has implemented this in particular.

Azure Landing Zone



One of the common analogies for understanding this scaffolding concept, is to compare it to foundational services upon which your suburb or city depends on.

Azure Landing Zone



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Before a new housing complex is built and people move into them essentials services must be deployed. Similar to networking on Azure, a well planed road infrastructure is necessary for people to move around. Similarly, water and sewer facilities need to be in place. power and gas lines must be available and always-on along with the ability to charge back.

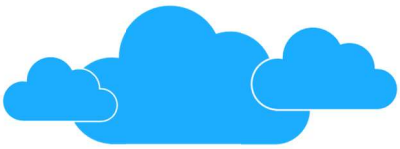
Azure Landing Zone

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

You can think of Azure landing zone as the essential services required for the different types of applications you will host on Azure. Different applications like AKS will require different services, and policies, and therefore will live in different landing zones.

Reference: https://github.com/Azure/AKS-Landing-Zone-Accelerator

Azure Landing Zone



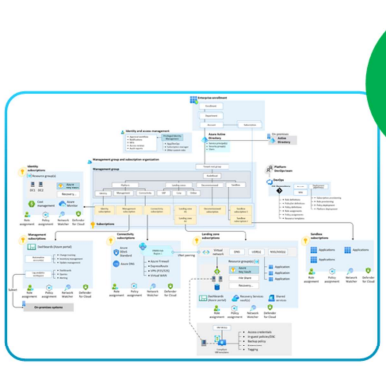
Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

The primary purpose of the landing zone is to ensure when your application lands on Azure the required plumbing is in place providing greater agility and compliance with enterprise security and governance requirements.

Azure Landing Zone

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Azure Landing Zone



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Azure landing zone is a Cloud management system that makes deployment and configuration of cloud applications simple, repeatable, consistent and compliant to your organizational standards and requirements.
No two organizations are same and hence their implementation of landing zones will be unique to meet their requirements. As a Kubernetes engineer or Architect, you might not be required to design the Landing Zone but it will be valuable to know these concepts in general and how your organization has implemented this in particular.

Azure Landing Zone

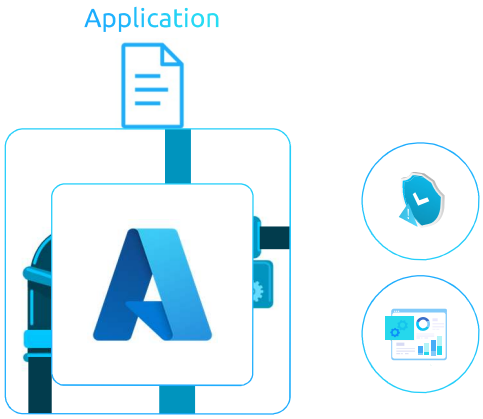
Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

One of the common analogies for understanding this scaffolding concept, is to compare it to foundational services upon which your suburb or city depends on. Before a new housing complex is built and people move into them essentials services must be deployed. Similar to networking on Azure, a well planned road infrastructure is necessary for people to move around. Similarly, water and sewer facilities need to be in place. power and gas lines must be available and always- on along with the ability to charge back. You can think of Azure landing zone as the essential services required for the different types of applications you will host on Azure.

Different applications like AKS will require different services, and policies, and therefore will live in different landing zones.

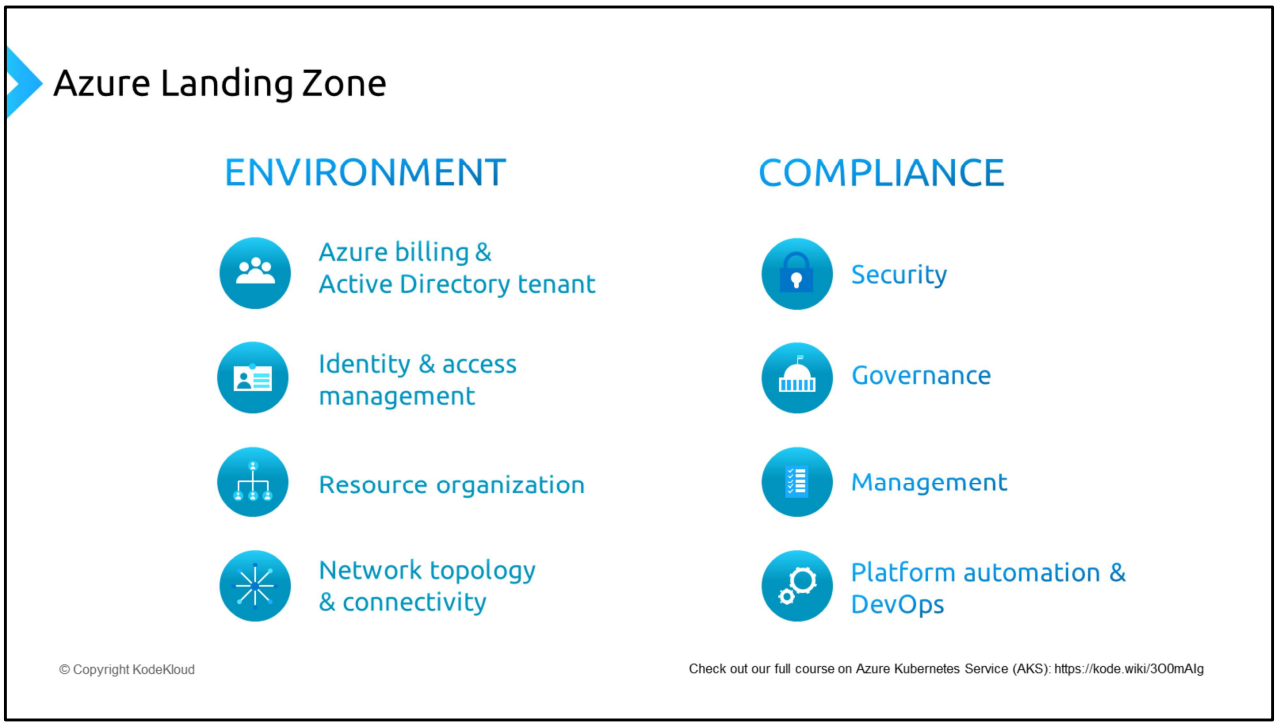
Reference: https://github.com/Azure/AKS-Landing-Zone-Accelerator

Azure Landing Zone



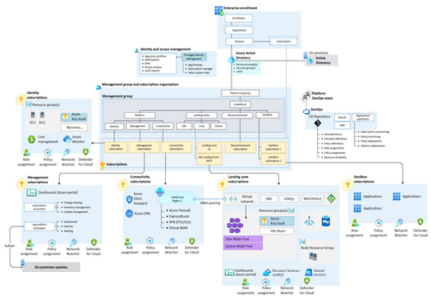
Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

The primary purpose of the landing zone is to ensure when your application lands on Azure the required plumbing is in place providing greater agility and compliance with enterprise security and governance requirements.



While an in-depth discussion of Landing zone is out of scope for this course, at high-level. These 8 areas needs to be covered for a well rounded landing Zone

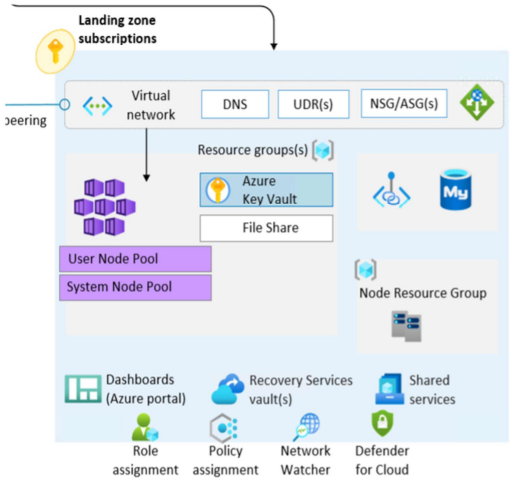
AKS Azure Scaffolding



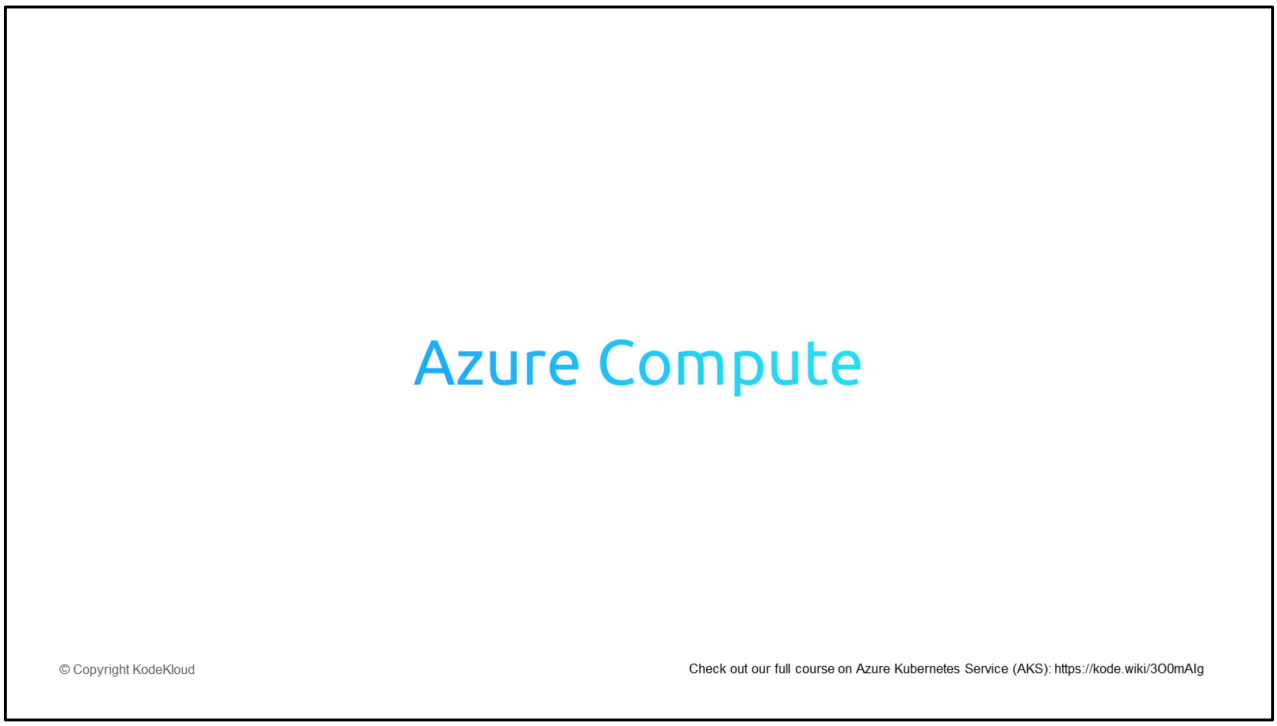
Now that you understand the basic concept of a Landing zone, when you need to deploy an AKS based application, you will have a AKS landing zone similar to the highlighted area here. You application can take advantage of all the foundational services deployed and can connect to other Azure services in a predictable manner.

Reference: https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/scenarios/app-platform/aks/landing-zone-accelerator

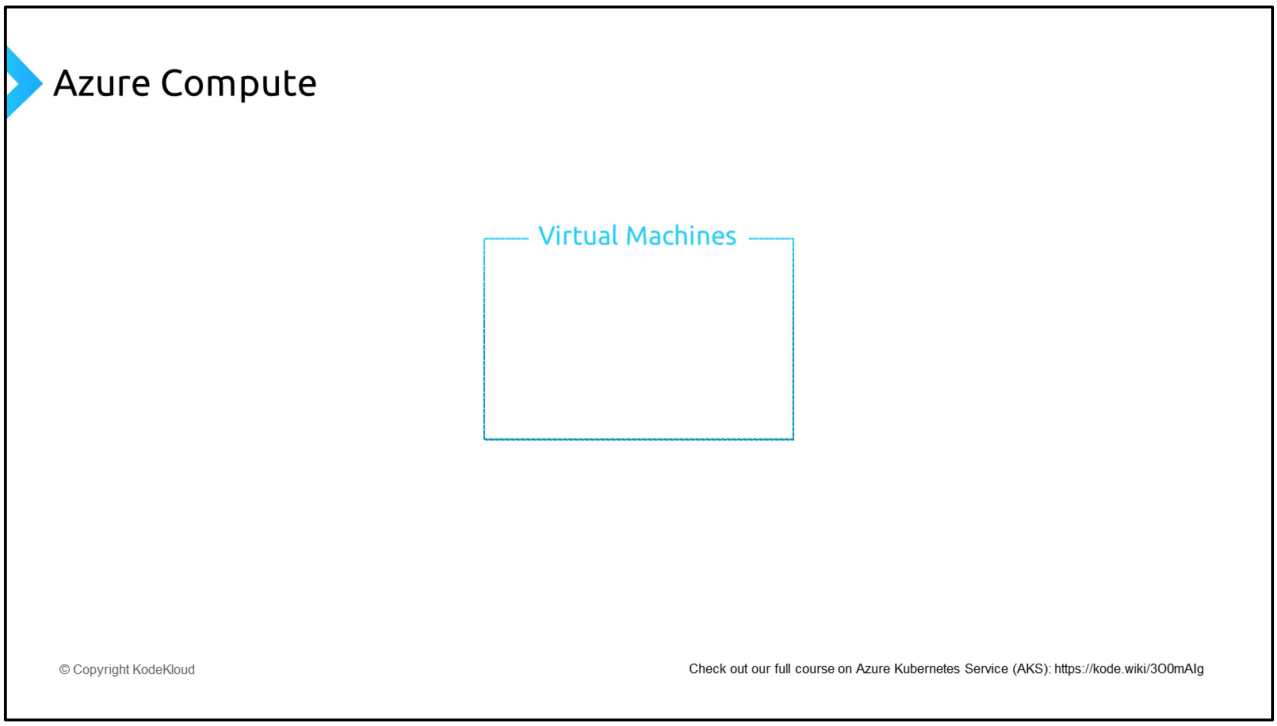
AKS Azure Scaffolding



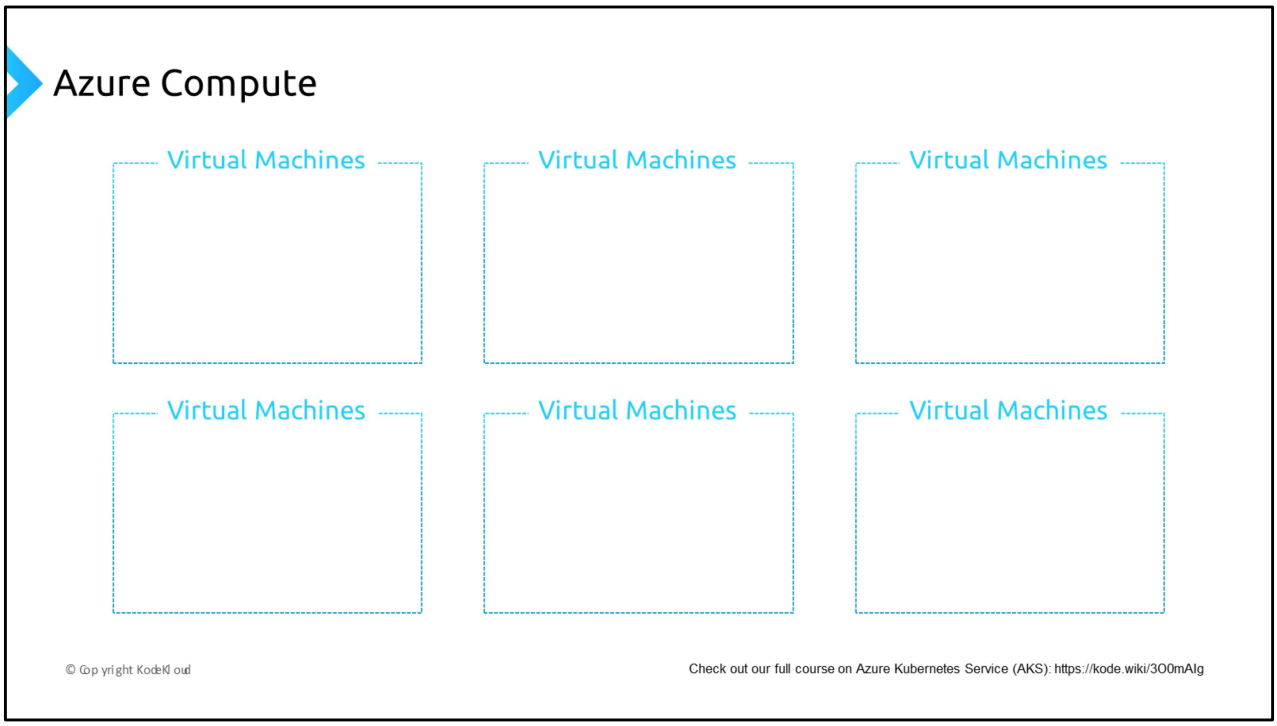
Reference: https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/scenarios/app-platform/aks/landing-zone-accelerator



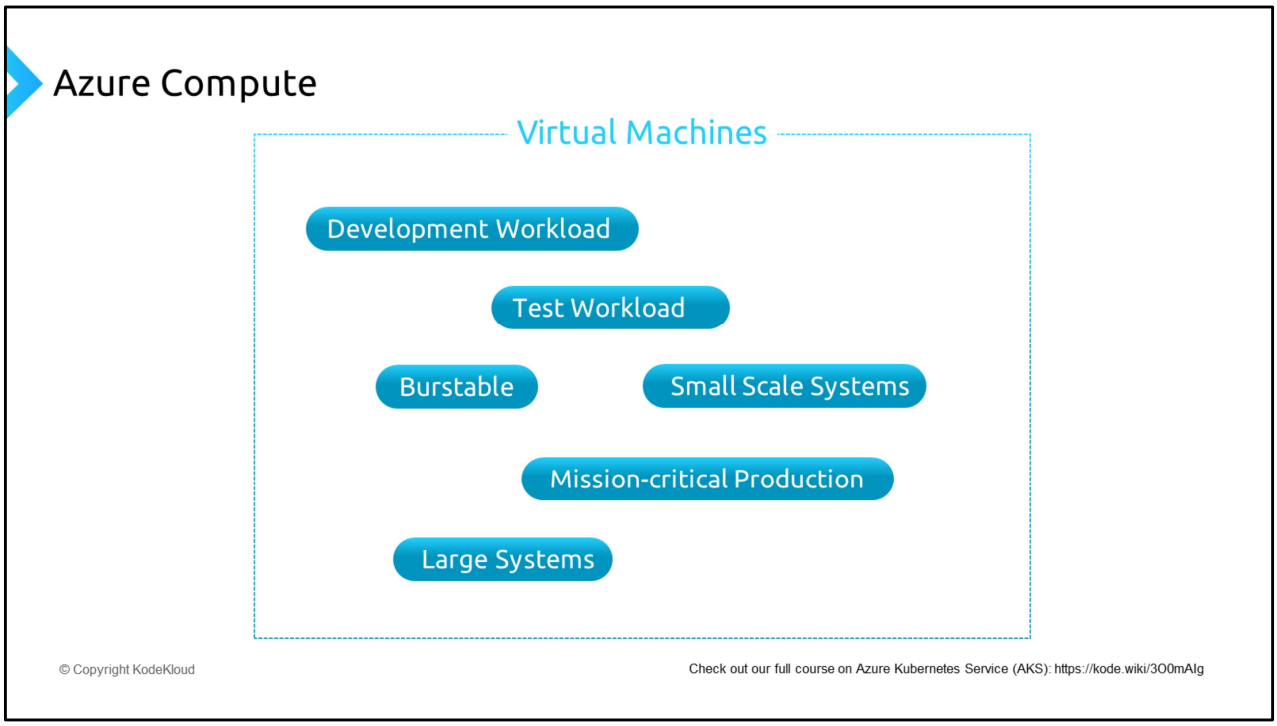
Have you ever written a program but didn’ t have an application server to run it on? Maybe you're a developer and you’ ve written code that executes a basic task.



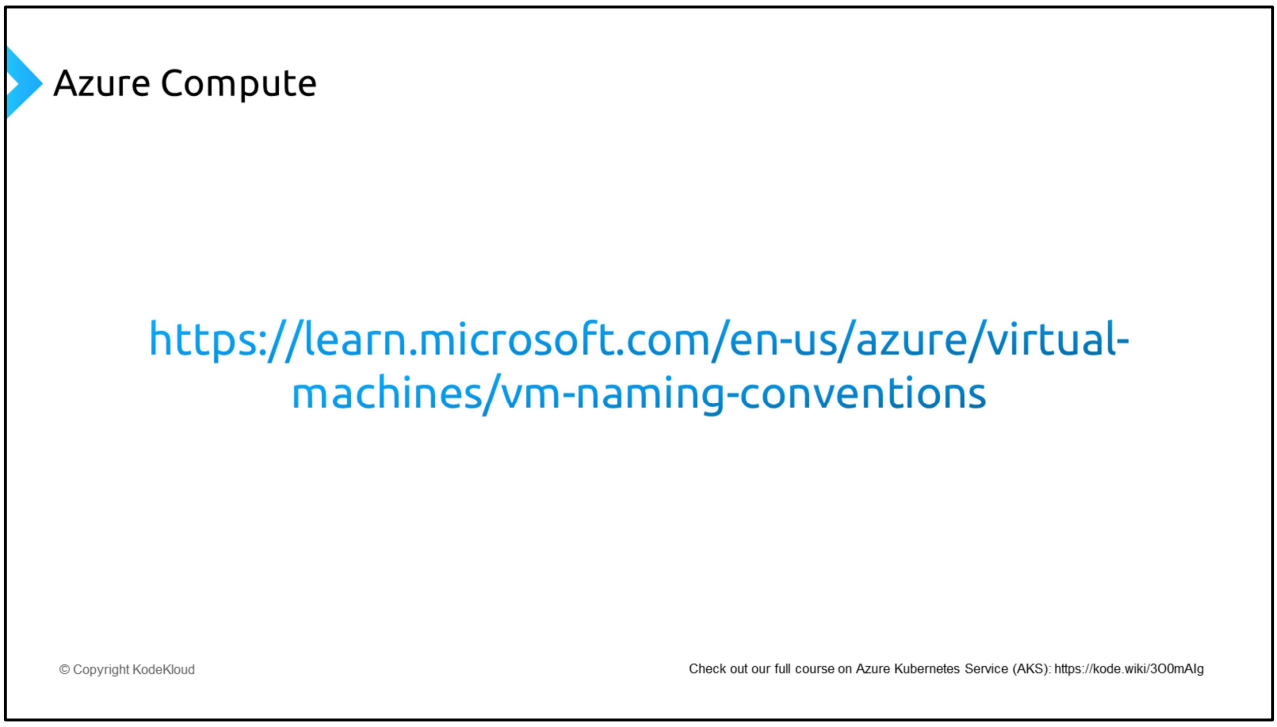
Azure offers Virtual Machine types and sizes that can address almost every type of workload



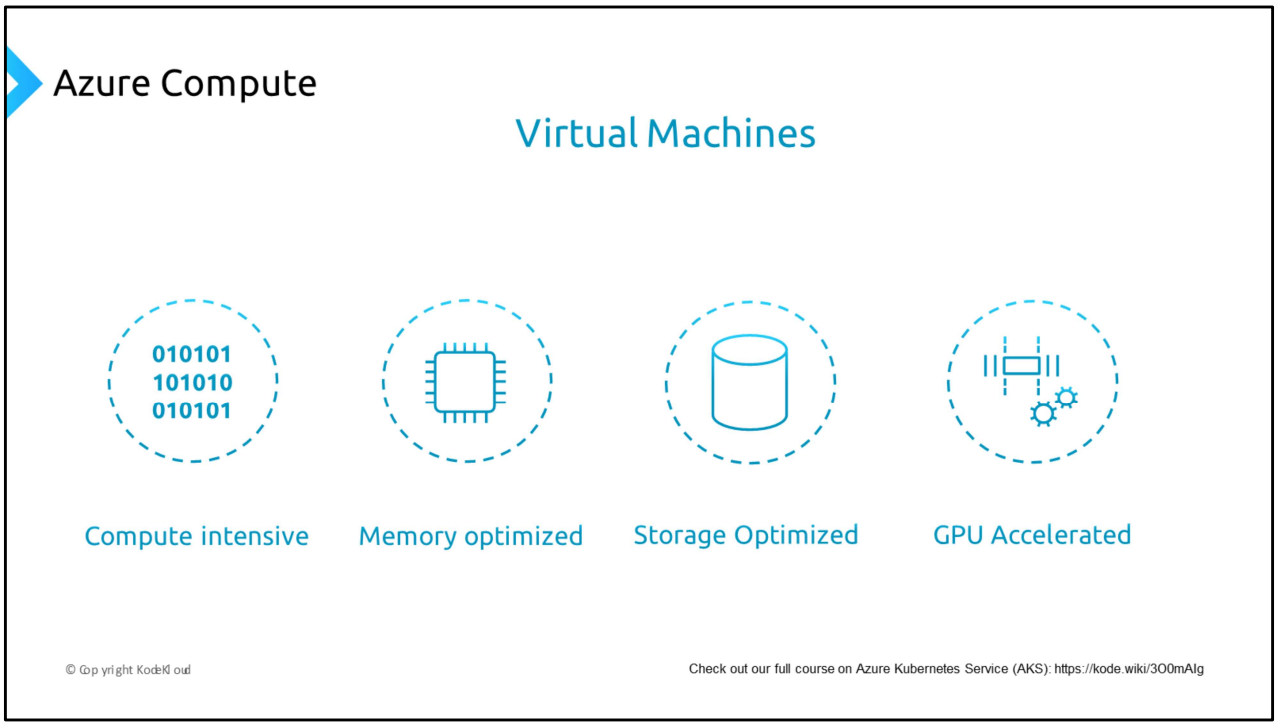
For example, dev/test workloads, burstable, mission-critical production workloads, small scale to large systems and off course almost all of these can be used for AKS clusters. I have listed a few machine types per category, you can check this URL for full list of VMs and their specs.



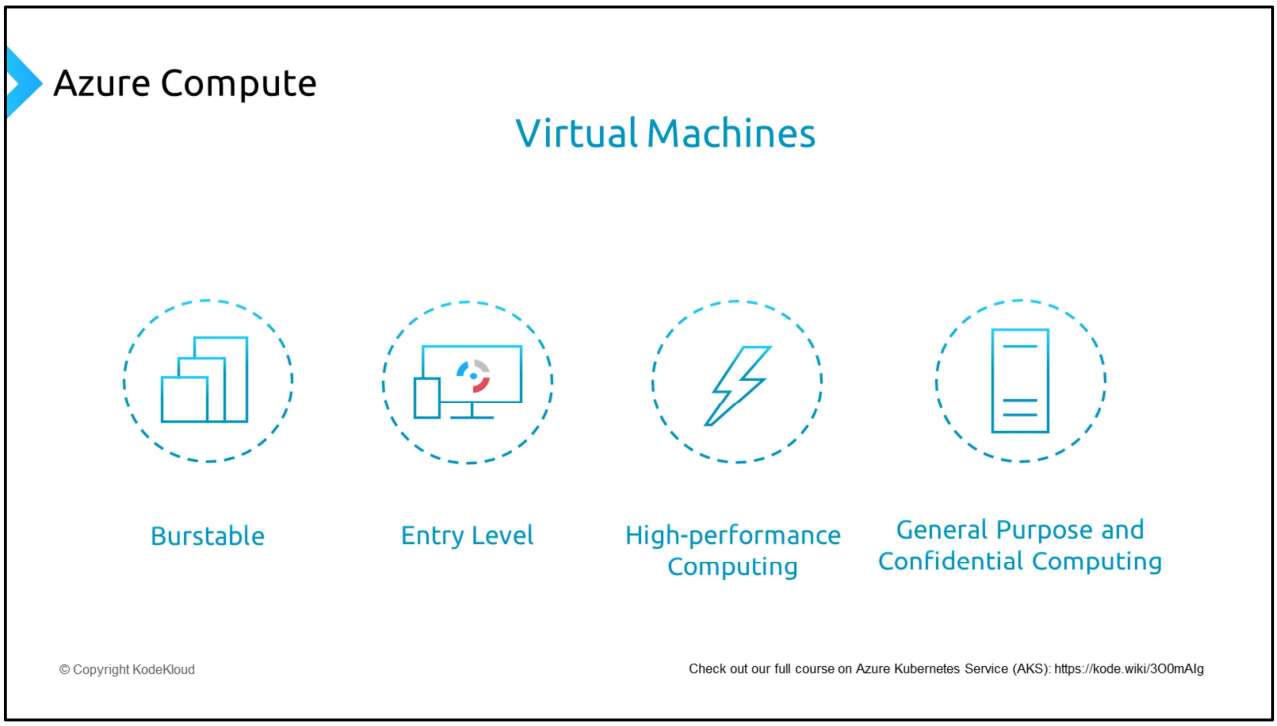
Azure offers Virtual Machine types and sizes that can address almost every type of workload – For example, dev/test workloads, burstable, mission-critical production workloads, small scale to large systems and off course almost all of these can be used for AKS clusters. I have listed a few machine types per category, you can check this URL for full list of VMs and their specs.



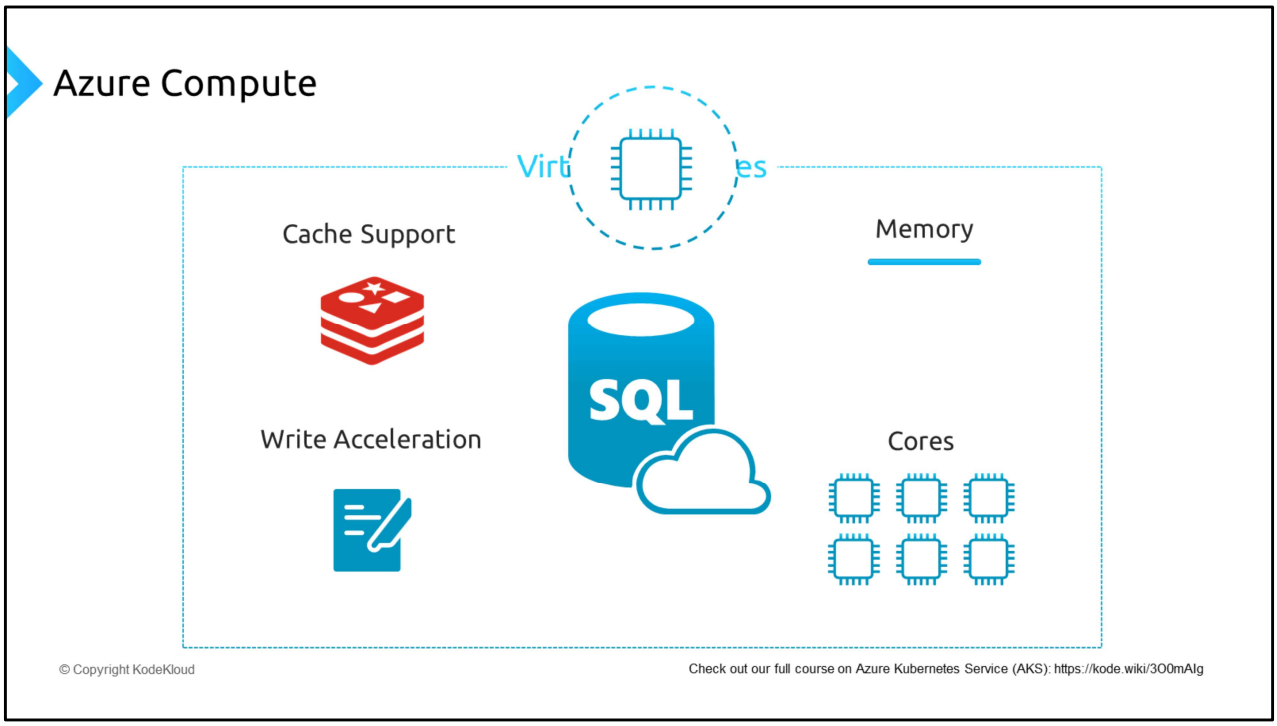
You should also consider getting yourself familiar with Azure Compute naming convention. You will come across names like: Standard_E64-32ads_v5
Every character has a meaning here, you can refer to the link of the screen to demystify the naming convention: https://learn.microsoft.com/en-us/azure/virtual-machines/vm-naming-conventions



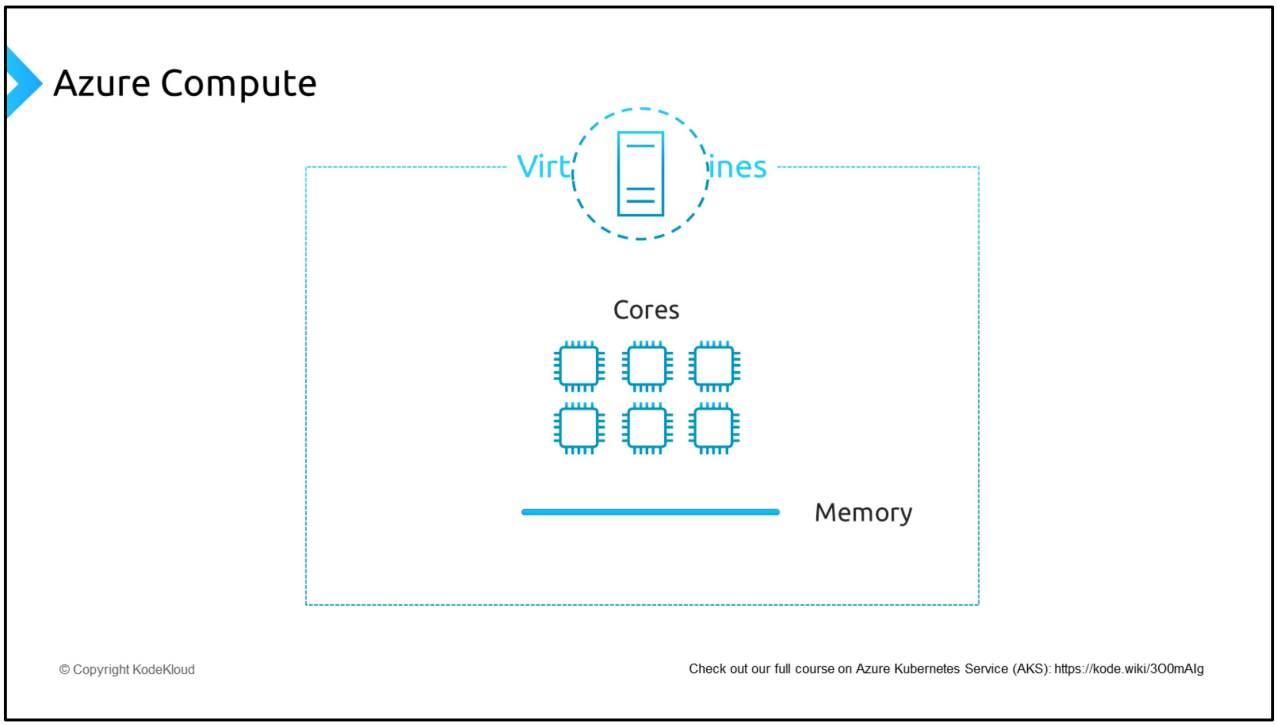
If you are new in Azure, you would need to understand what these series and naming conventions are. There are VM series optimized for compute, memory, disk, GPUs,



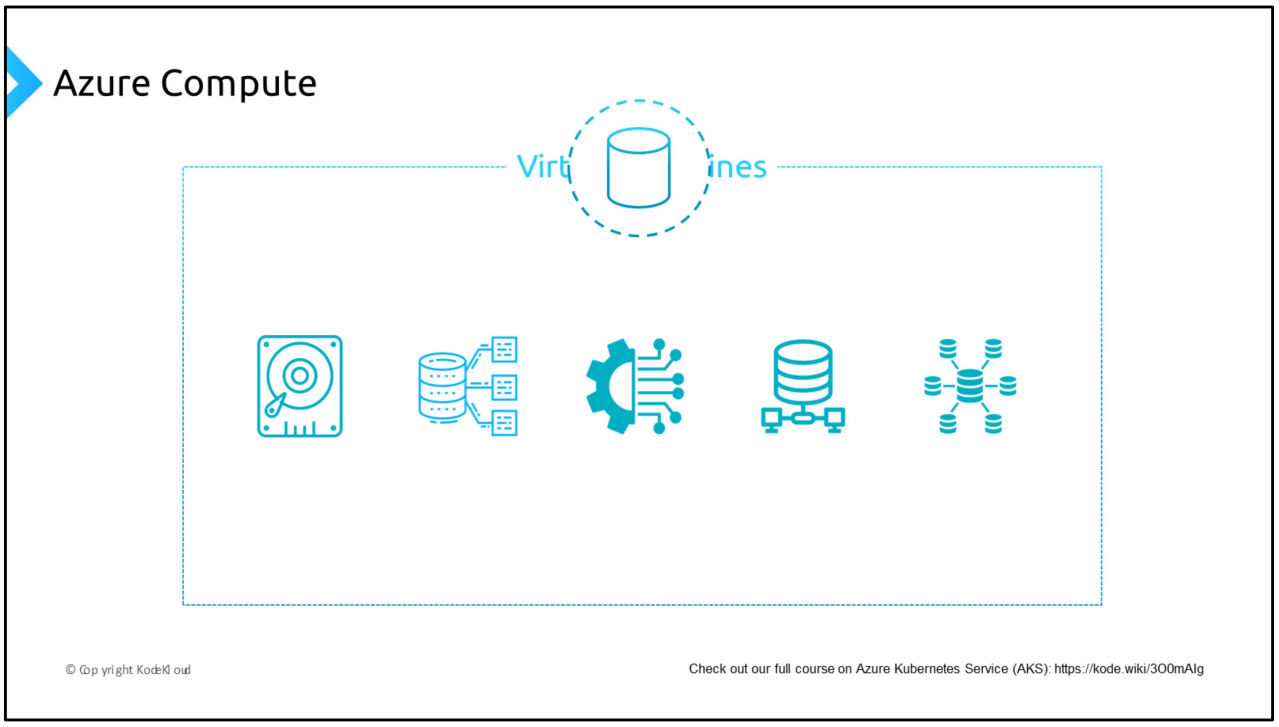
and more and as you could imagine this portfolio is growing constantly.



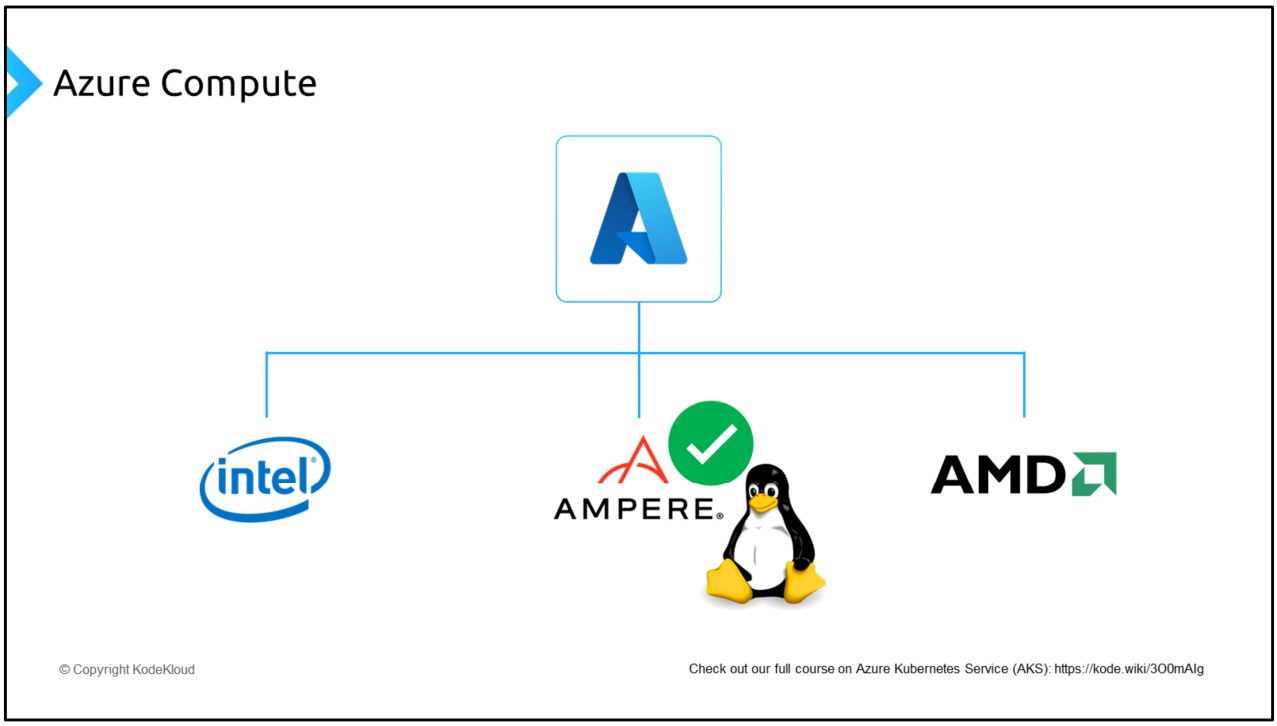
Just to give you a flavor, let me mention couple of these machine types and what criteria you can use to make your selection. Let us say you want to host a critical SQL database on a VM. In general these databases require a lot of memory but comparatively less number of CPU Cores. Then Memory optimized VM sizes should be considered because they offers large amounts of memory, high memory-to-core ratios (which means large amount of memory per core), premium disk and cache support, and write acceleration capabilities.



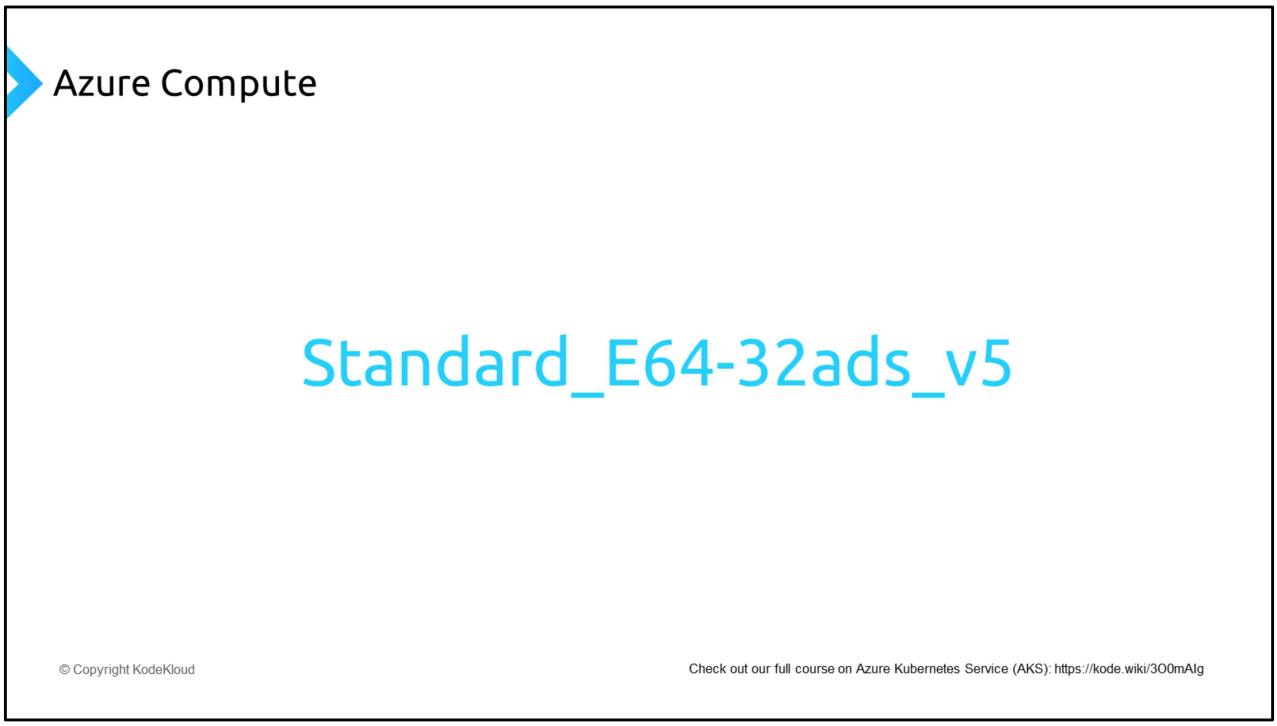
On the other hand, General purpose VM sizes provide balanced, but lower memory-to-core ratio. This might be good for smaller database servers.



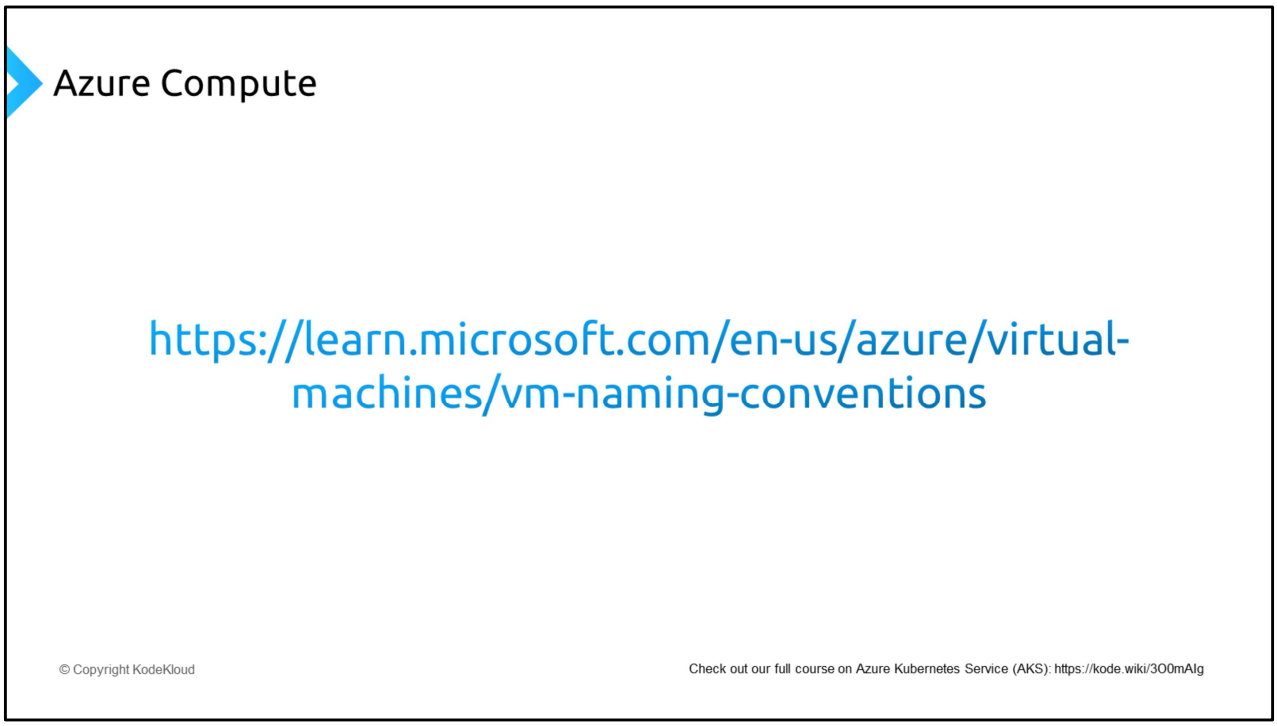
Now let us say, you need high disk throughput and IO for Big Data, ETL, Operational Data Store (ODS), and data warehousing processing, then you should consider storage optimized series. This series can offer very high throughput & low latency storage with directly mapped local NVMe storage.



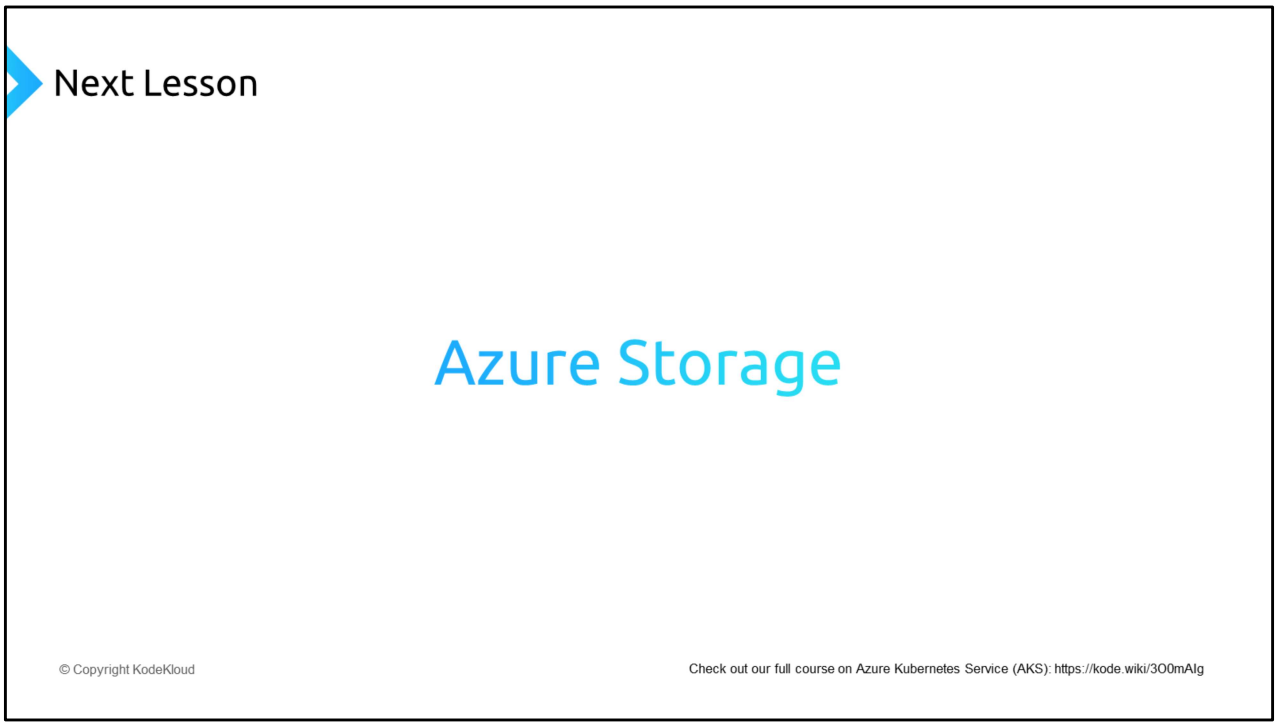
While Azure has both Intel and AMD based machines, they have recently launched Ampere® Altra® Arm-based processors. This would be an excellent choice for Linux based containers as they have the best performance to cost ratio and have the lowest environmental footprint.



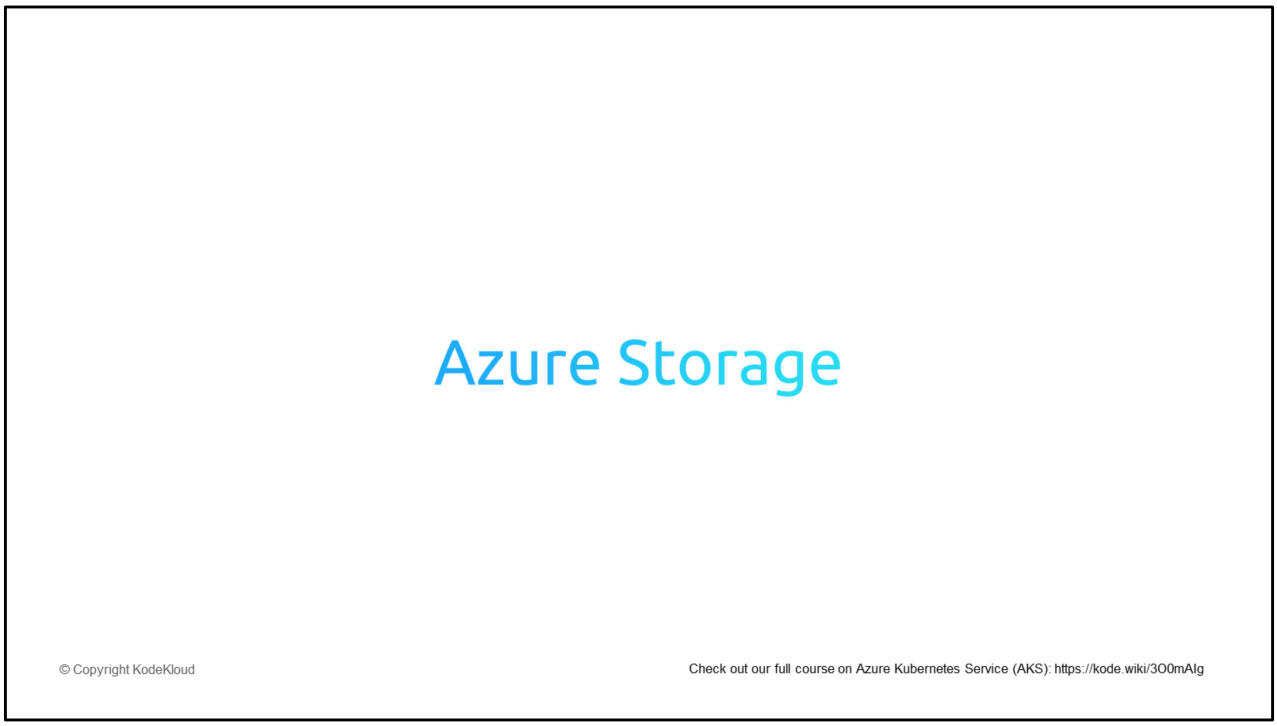
You should also consider getting yourself familiar with Azure Compute naming convention. You will come across names like: Standard_E64-32ads_v5
Every character has a meaning here, you can refer to the link of the screen to demystify the naming convention: https://learn.microsoft.com/en-us/azure/virtual-machines/vm-naming-conventions



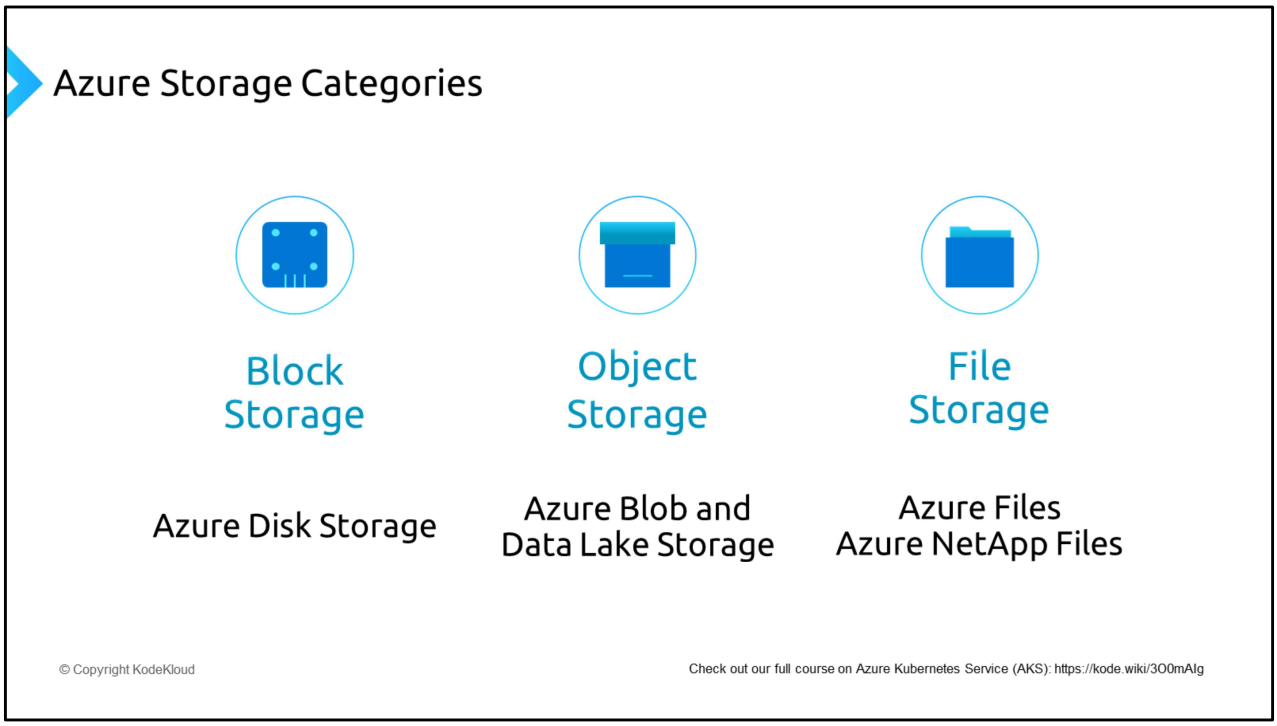
You should also consider getting yourself familiar with Azure Compute naming convention. You will come across names like: Standard_E64-32ads_v5
Every character has a meaning here, you can refer to the link of the screen to demystify the naming convention: https://learn.microsoft.com/en-us/azure/virtual-machines/vm-naming-conventions



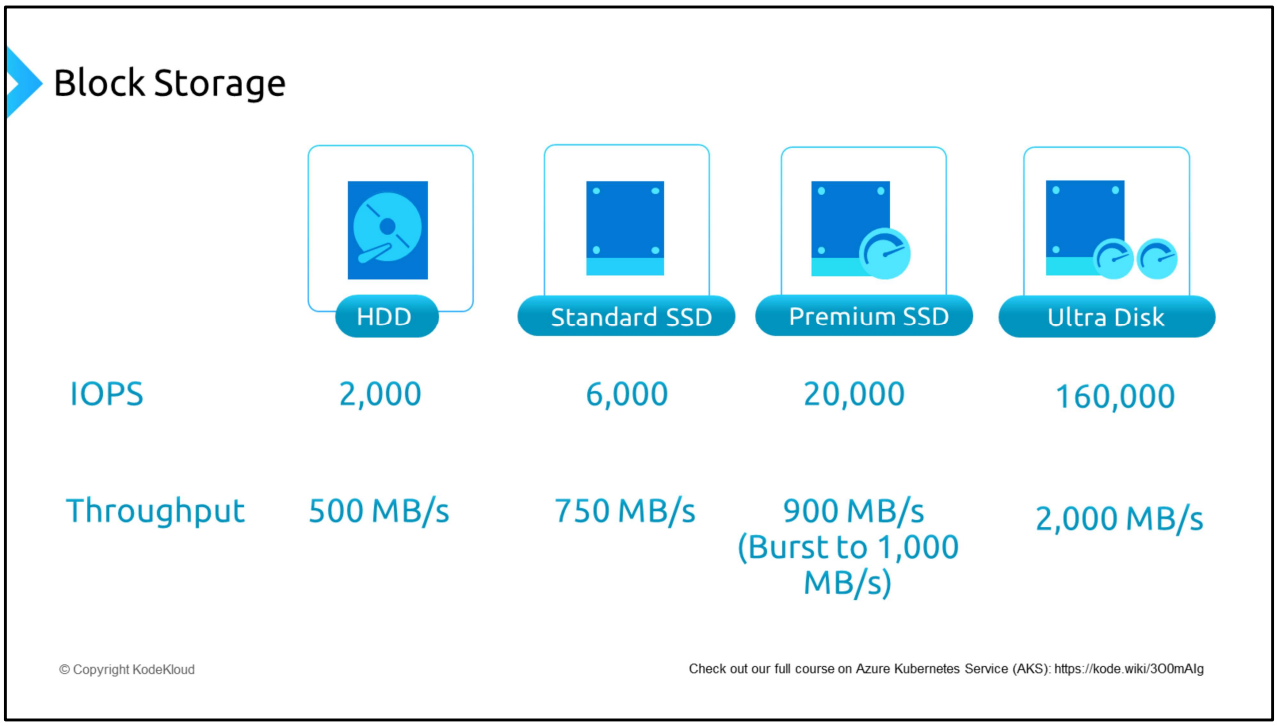
Now lets move on to storage.



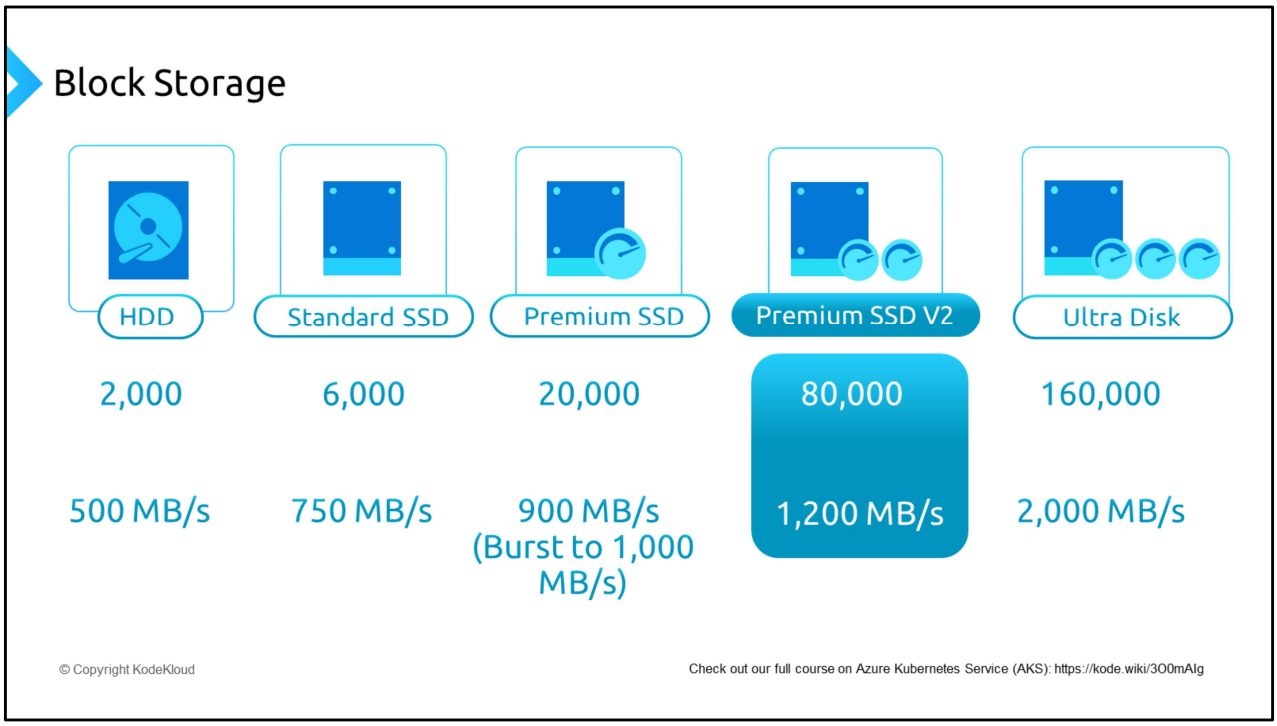
Have you ever written a program but didn’t have an application server to run it on? Maybe you're a developer and you’ve written code that executes a basic task.



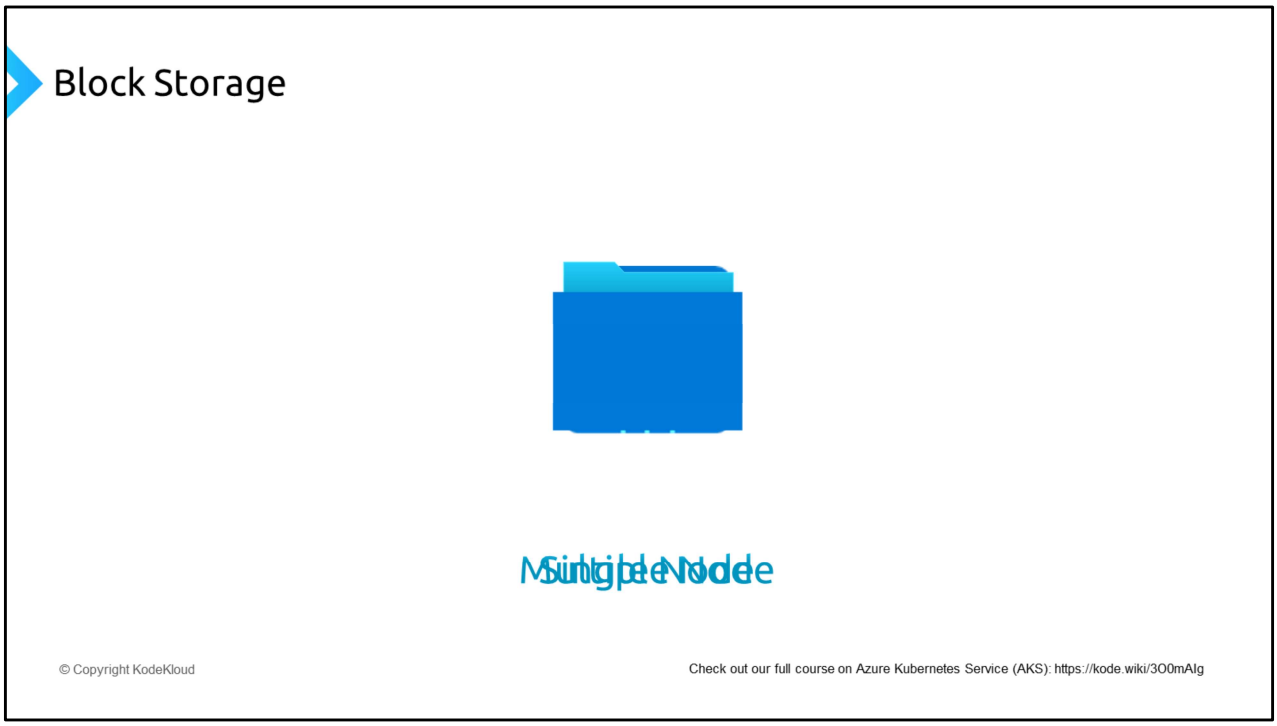
The three most common Azure Storage platform services are Block storage which includes capabilities like disks, Object Storage with Blob and Data lake and File storage with capability to deploy SMB and NFS fileshares.



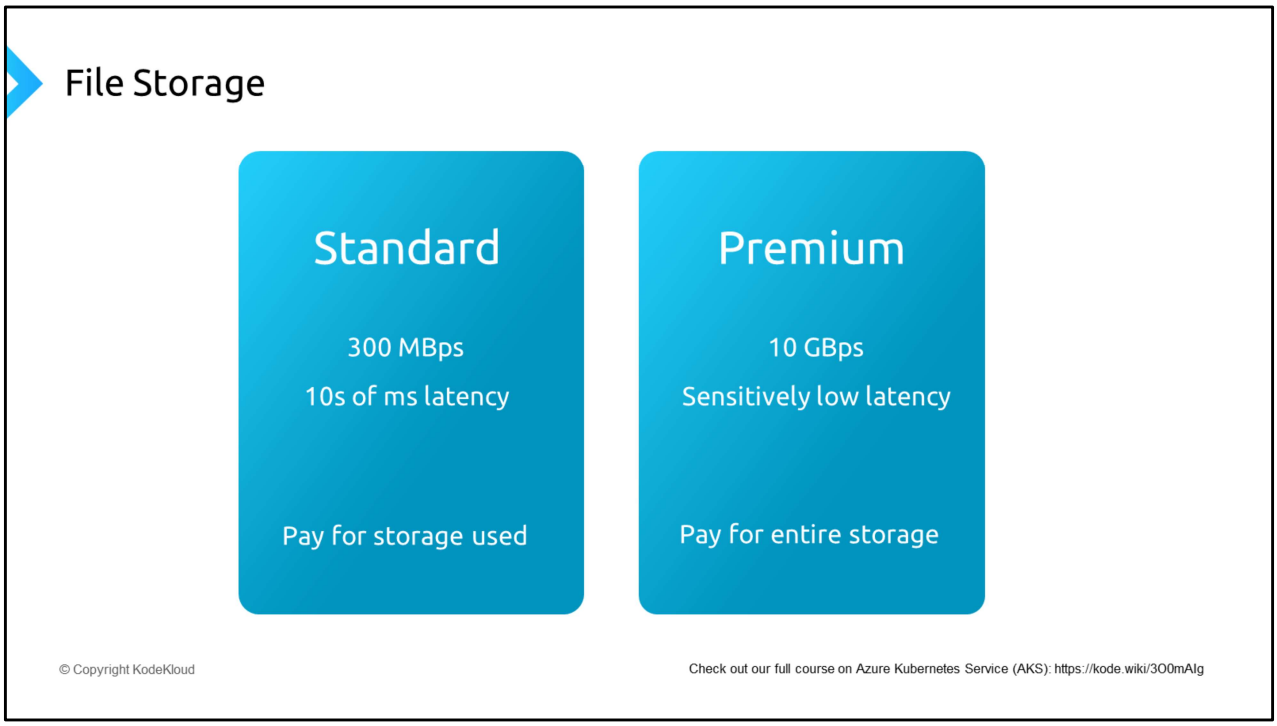
You can use block storage options also called Azure Disks to create a Kubernetes DataDisk resource. Depending on your IOPS and throughput requirements you can select from a number of SKUs. As you can imagine cost profile of these disks increases from left to right as you move from Standard spinning HDDs to NVMe based Ultra disks.



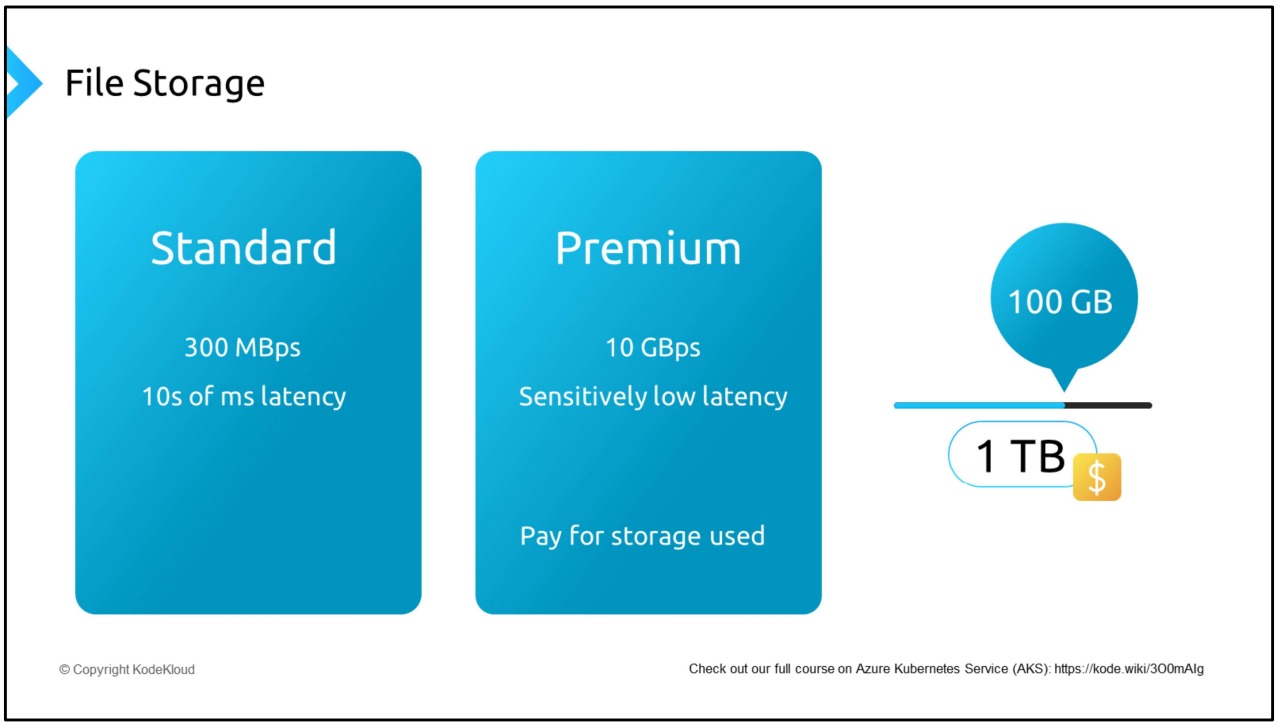
Microsoft has recently added another offering called Premium SSD V2 which provides performance specs between Premium SSD and Ultra disks. At the time of this recording, they do not provide the ability to use this disk as OS disk.



While we will talk more about AKS and Storage in a later section, it is worth noting that because Azure Disks are mounted as ReadWriteOnce, they're only available to a single node. For storage volumes that can be accessed by pods on multiple nodes simultaneously, use Azure Files, which I will describe in the next slide.



File storage service in azure provides serverless file shares functionality with the option to choose either standard SKU with spinning disks or Premium SKU with SSD disks.
As you can see standard files are slower with higher latency while premium files are extremely fast.

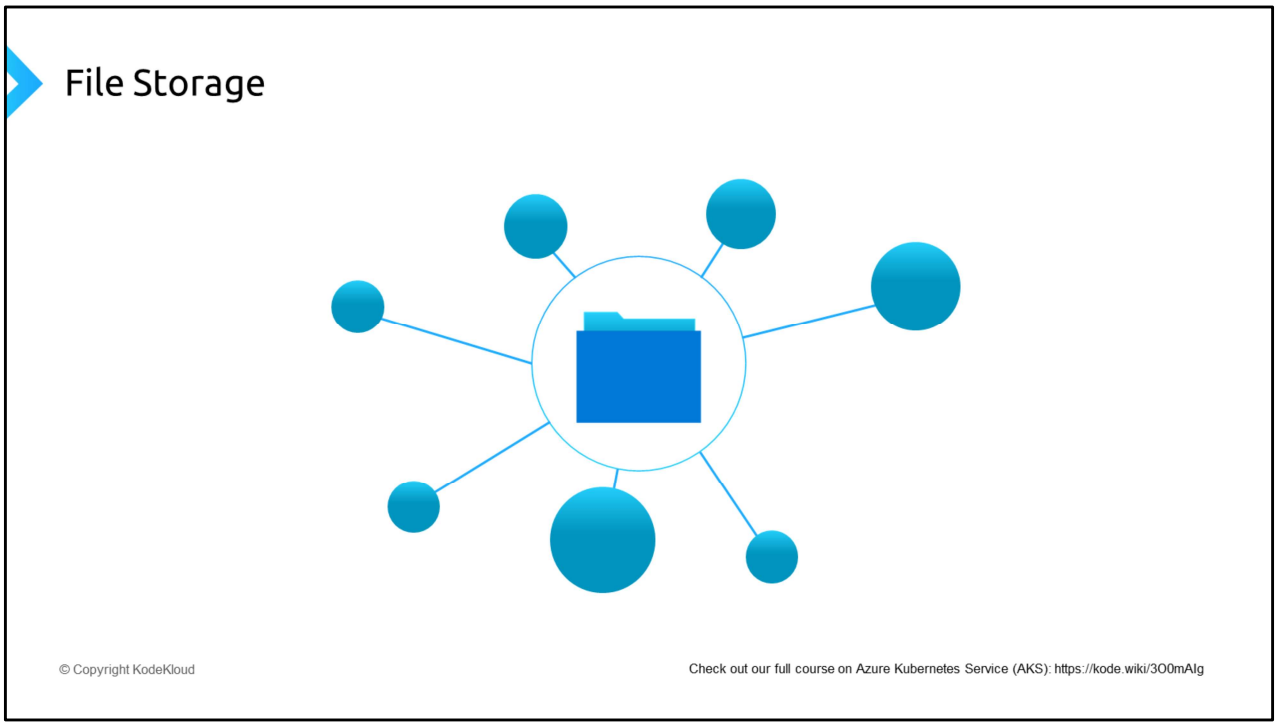


As you start to build you AKS based solutions on Azure, you will need to know which option to pick and why. Cost implications are always critical in such a selection. While I have documented some of the common use cases for these file storage types, it is worth mentioning that while in standard SKU you pay for storage used in a Premium file type, you pay for entire storage provisioned. For example, if you have provisioned a 1 TB of storage used only 100 Gig, in case of standard storage you pay for only 100 gig while in case of premium files you pay for entire 1 TB.

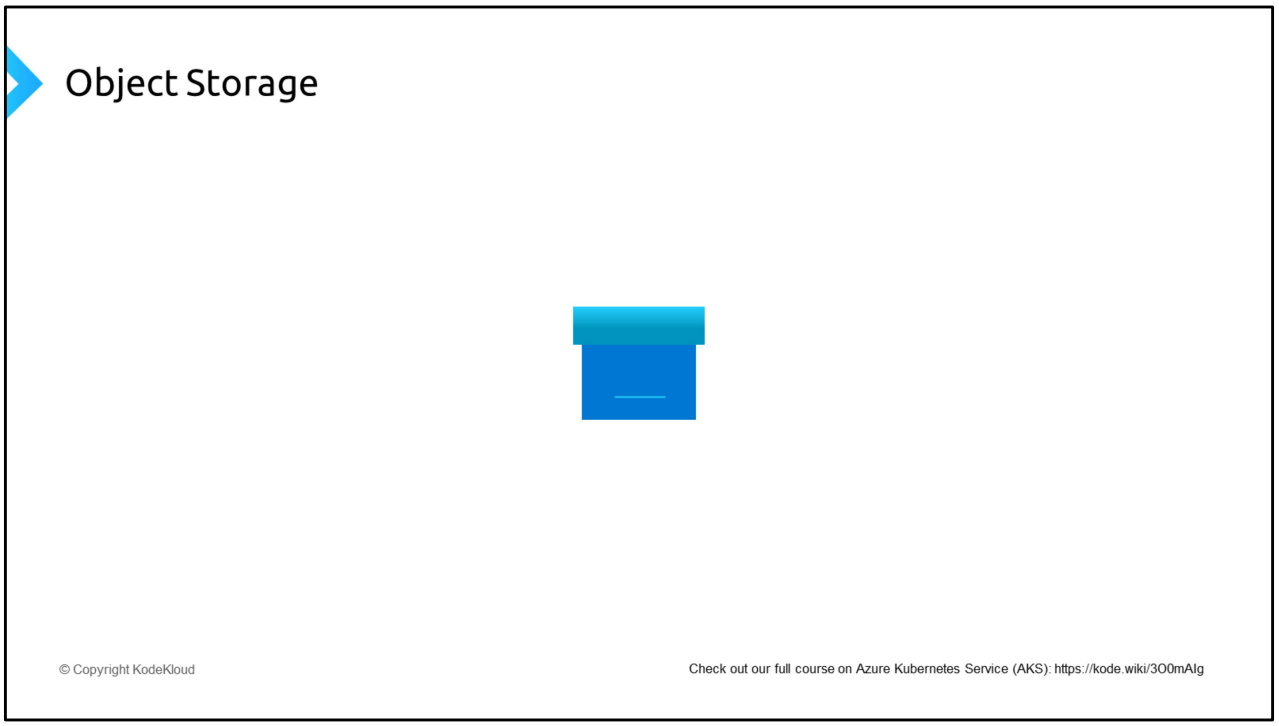
If your performance need are higher than what Premium files has to offer, you can explore Netapp Files. This is a first party solution in Azure but managed by their vendor i.e Netapp.

In AKS world, if multiple pods need concurrent access to the same storage volume, you we use Azure Files to connect using the Server Message Block (SMB) protocol.

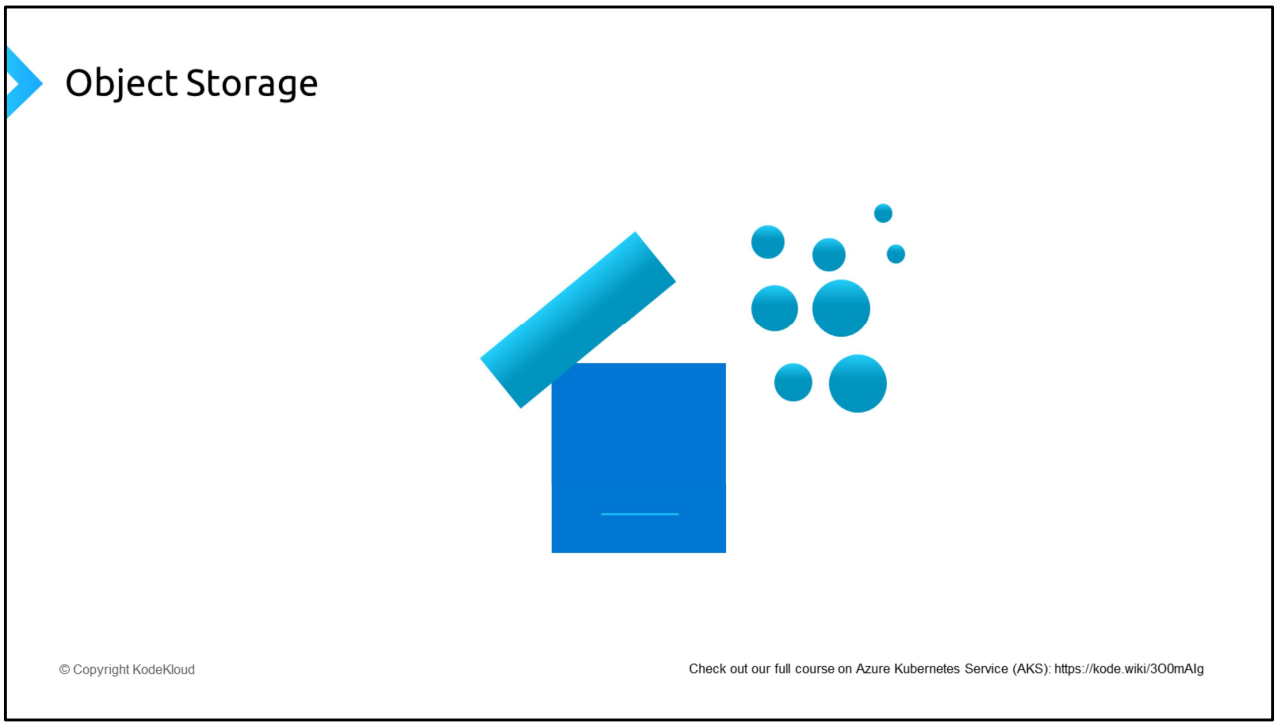
If your performance need are higher than what Premium files has to offer, you can explore Netapp Files. This is a first party solution in Azure but managed by their vendor i.e Netapp.



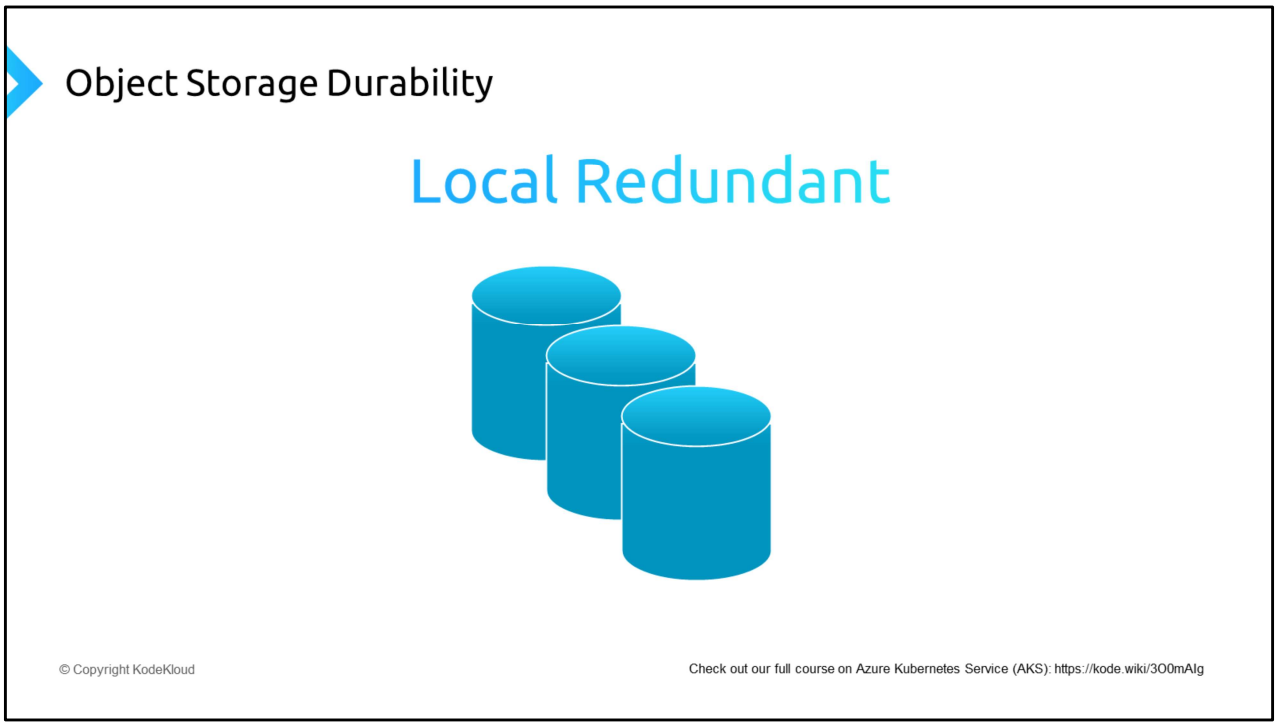
In AKS world, if multiple pods need concurrent access to the same storage volume, you we use Azure Files to connect using the Server Message Block (SMB) protocol.



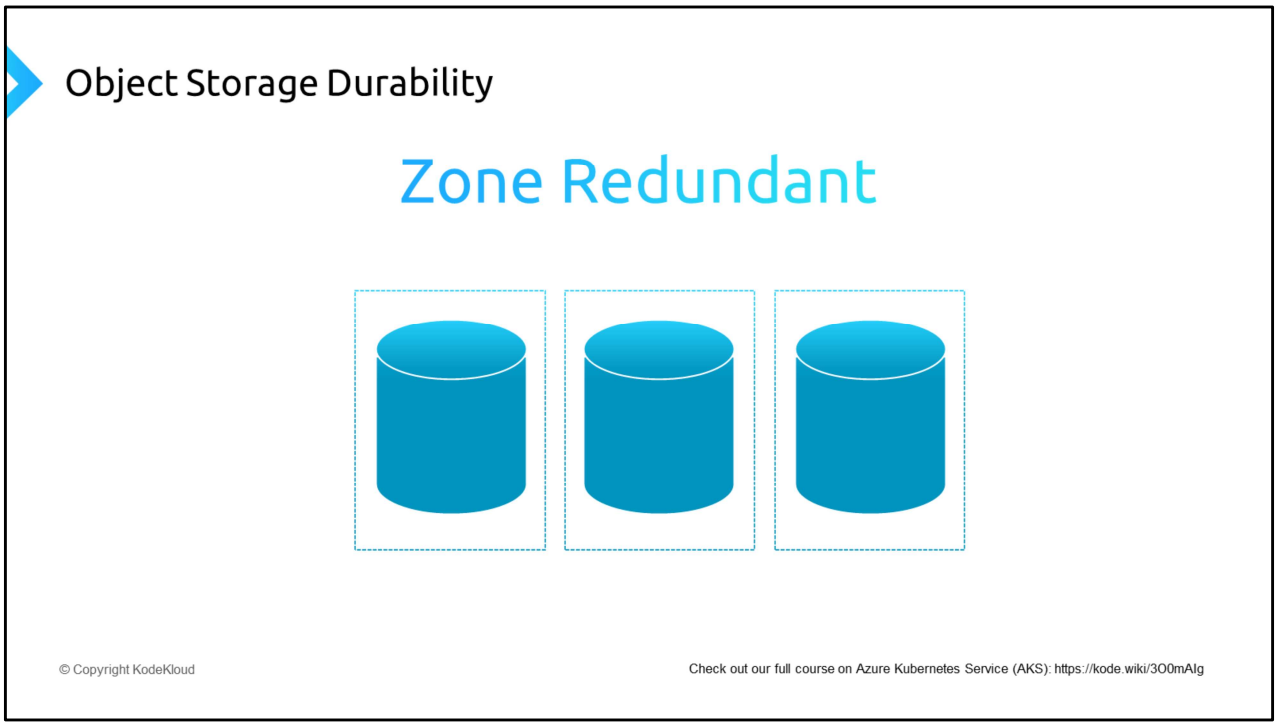
Azure object storage is also called Blob storage and is optimized for storing massive amounts of unstructured data. Similar to file storage, azure blobs also have the ability for you to select between Standard or Premium performance tiers.



If this is the first time you are getting exposed to all the storage options, it can be intimidating. Trust me it get a little better with time
To make matters a little bit more confusing, to have the ability to enable hierarchical namespace which is also called ADLSGen2. It enables a set of capabilities that are dedicated to big data analytics.



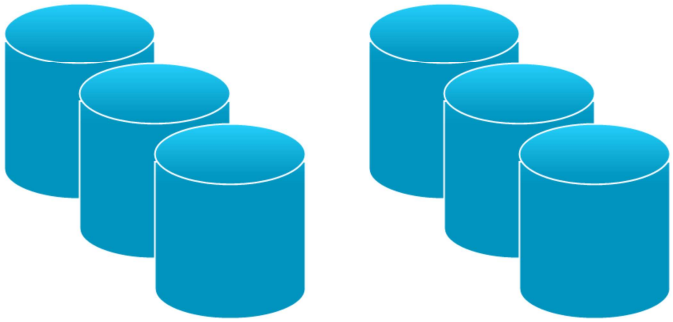
You also have the choice to select one of the four Durability or Redundancy options of Local Redundant, Zone Redundant, Read-Access Geo-redundant or Geo-Zone redundant.



Without making it complicated, as you move from left to right, Azure can make multiple copies of your data in Availability zones, regions or both.

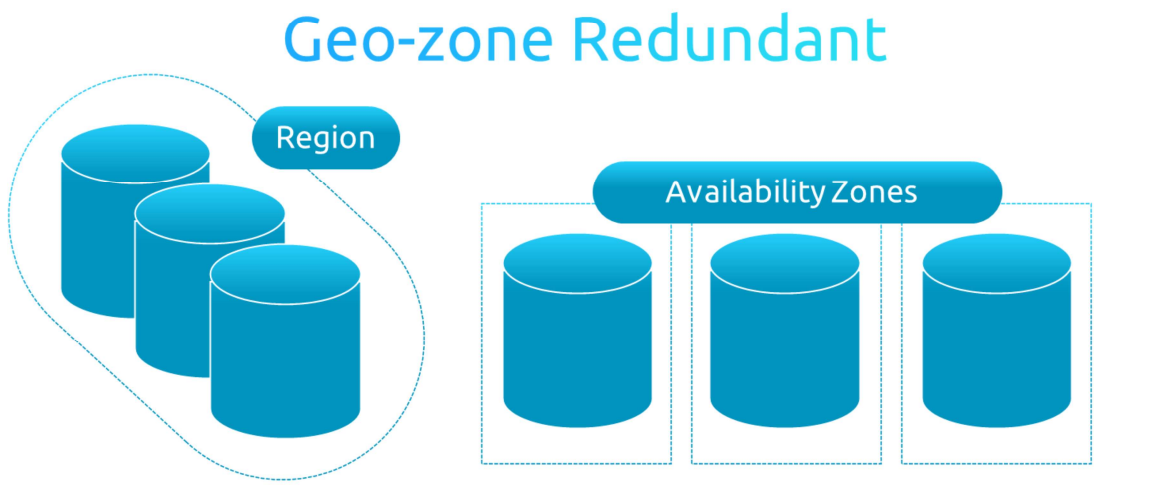
Object Storage Durability

Read-access Geo-redundant



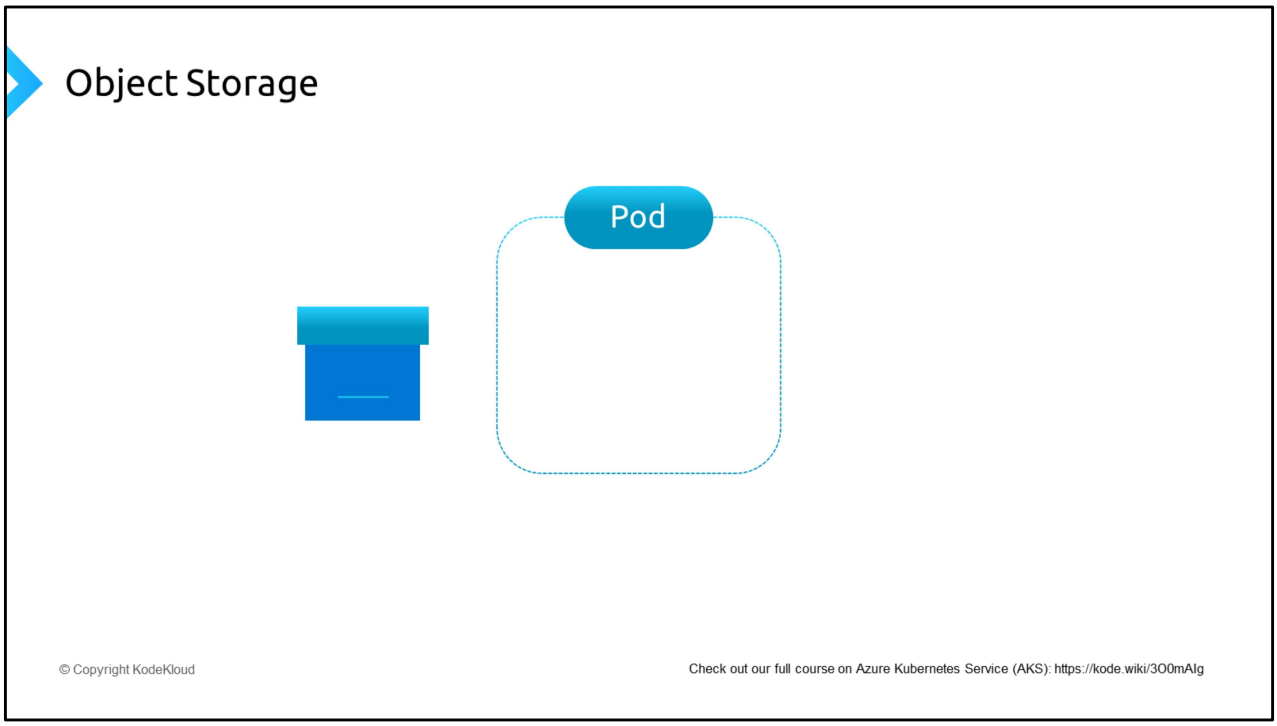
Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Object Storage Durability

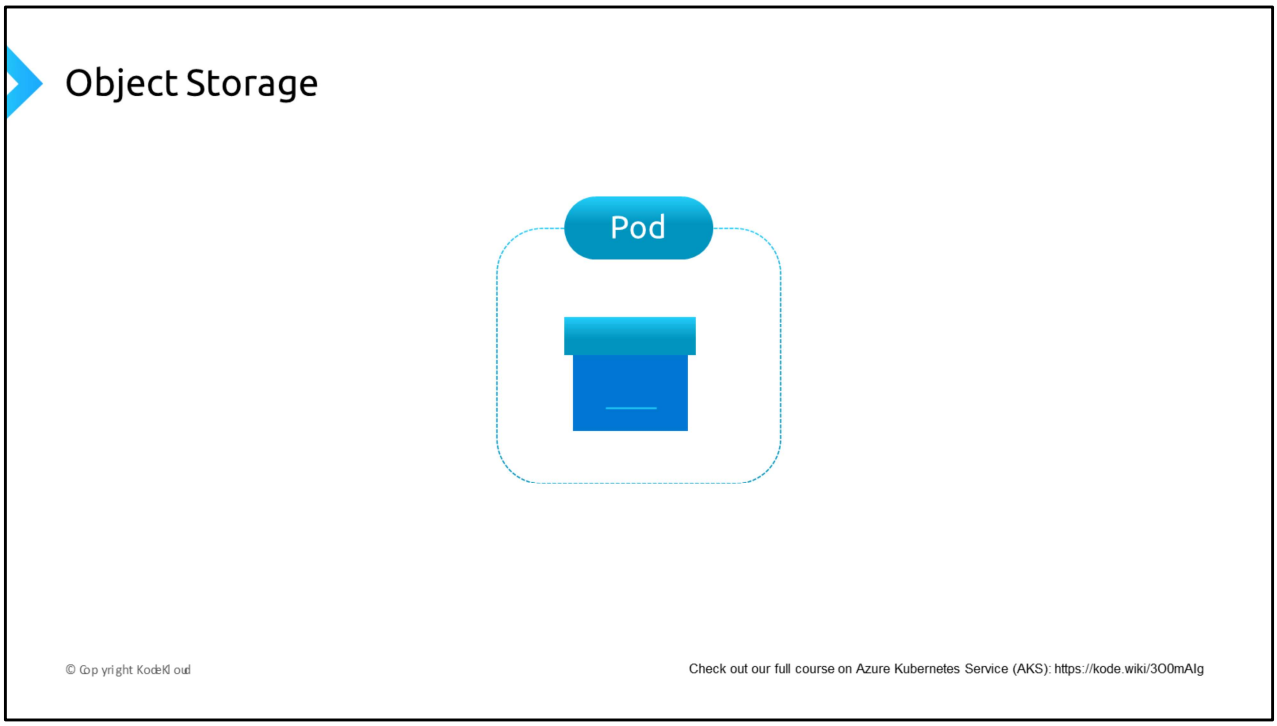


Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

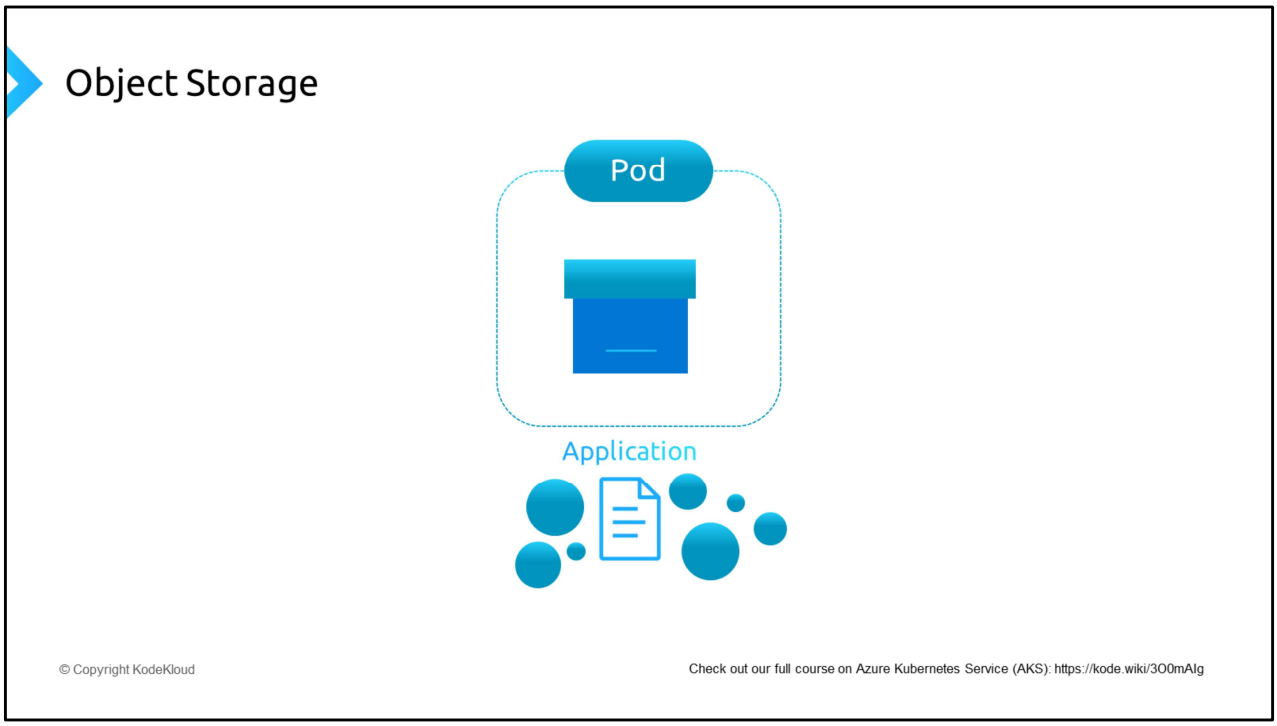
.



In terms of how you can use Blob Storage in AKS – you can mount Blob storage (or object storage) as a file system into a container or pod.

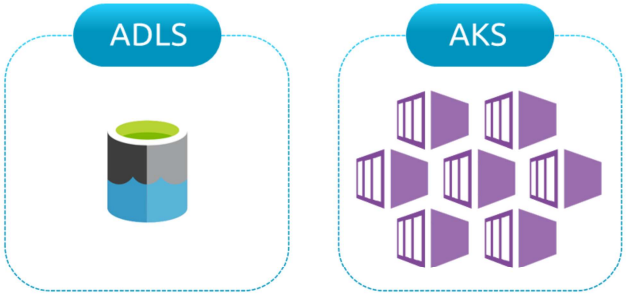


Using Blob storage enables your cluster to support applications that work with large unstructured datasets like log file data, images or documents

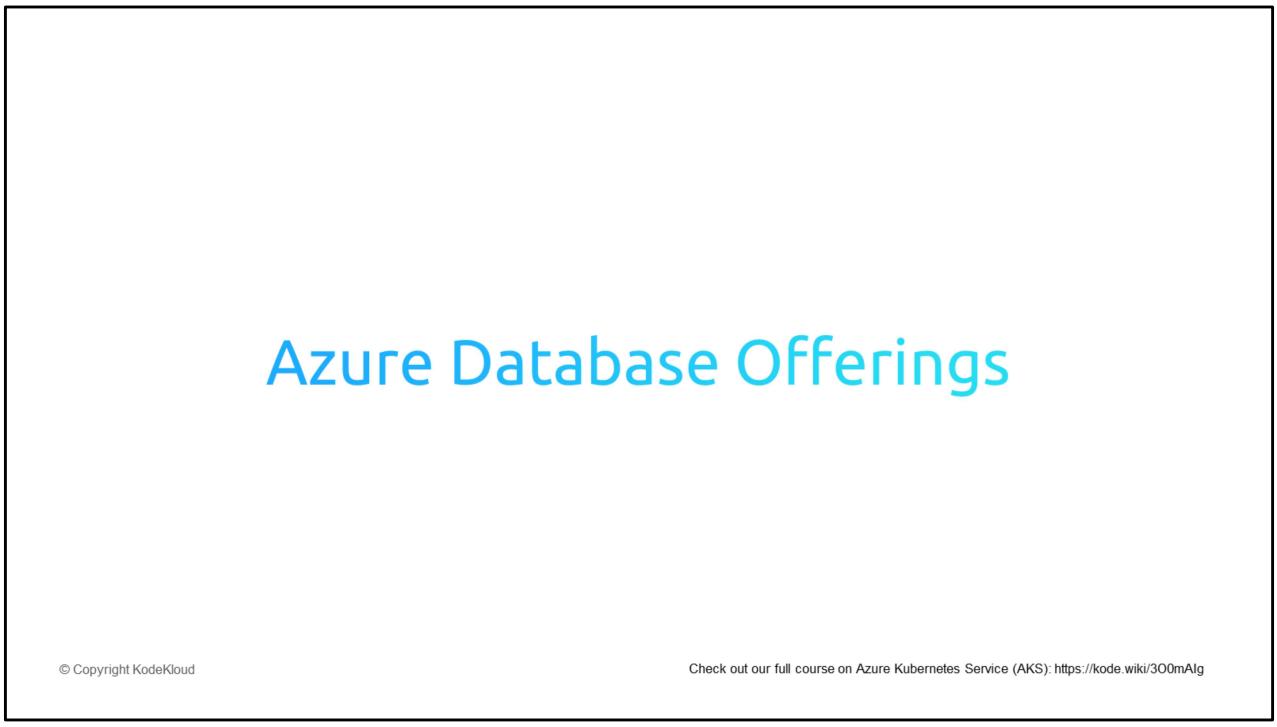


Additionally, if you ingest data into Azure Data Lake storage i.e ADLS Gen2, you can directly mount and use it in AKS without configuring another interim filesystem.

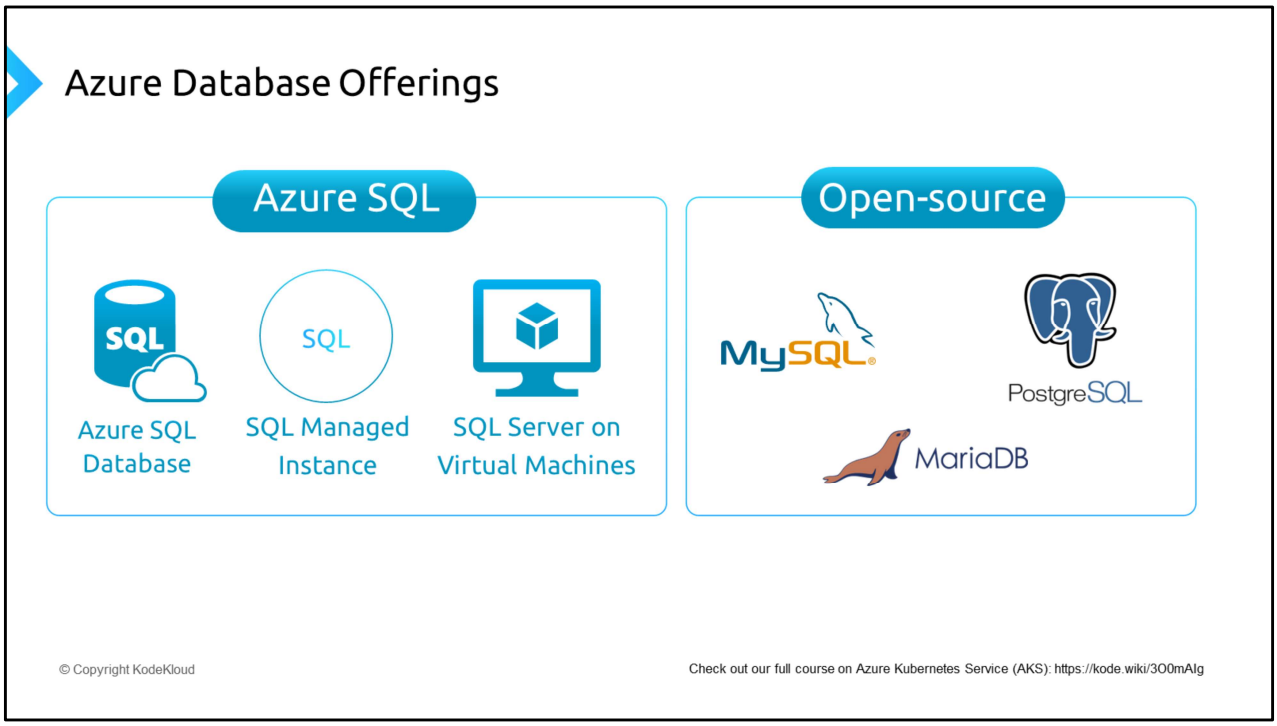
Object Storage



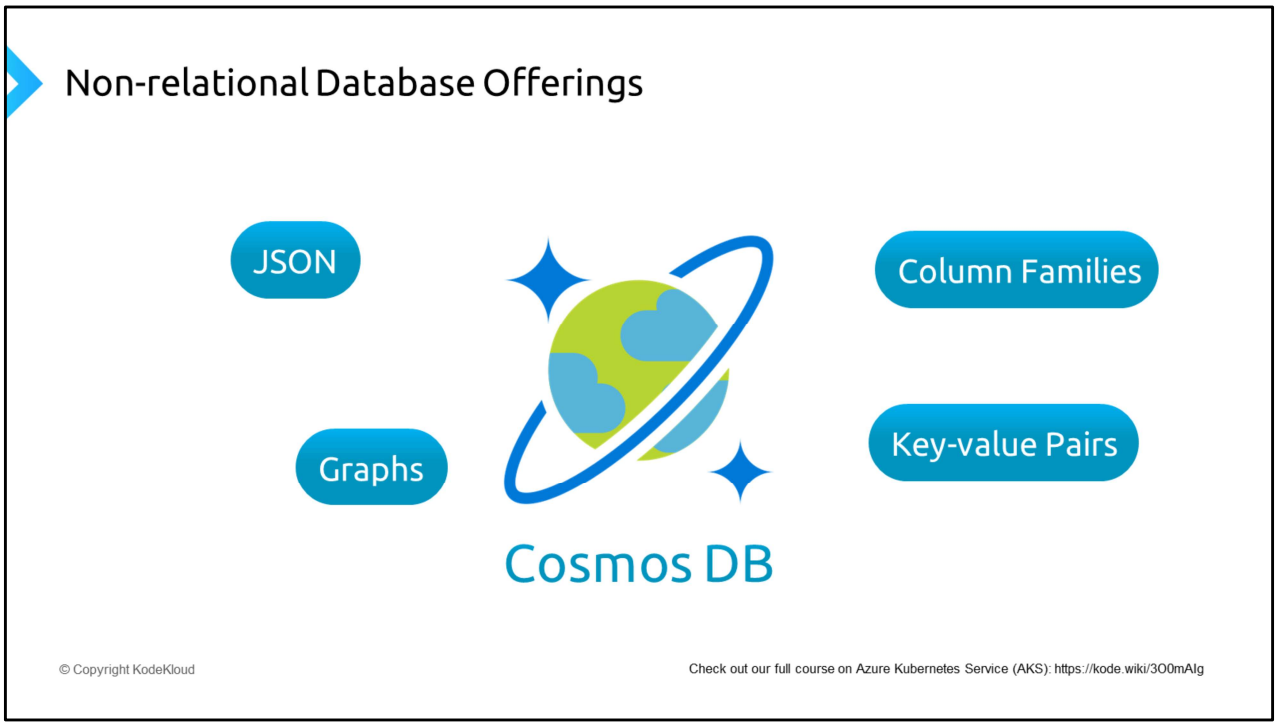
Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg



Have you ever written a program but didn’t have an application server to run it on? Maybe you're a developer and you’ve written code that executes a basic task.



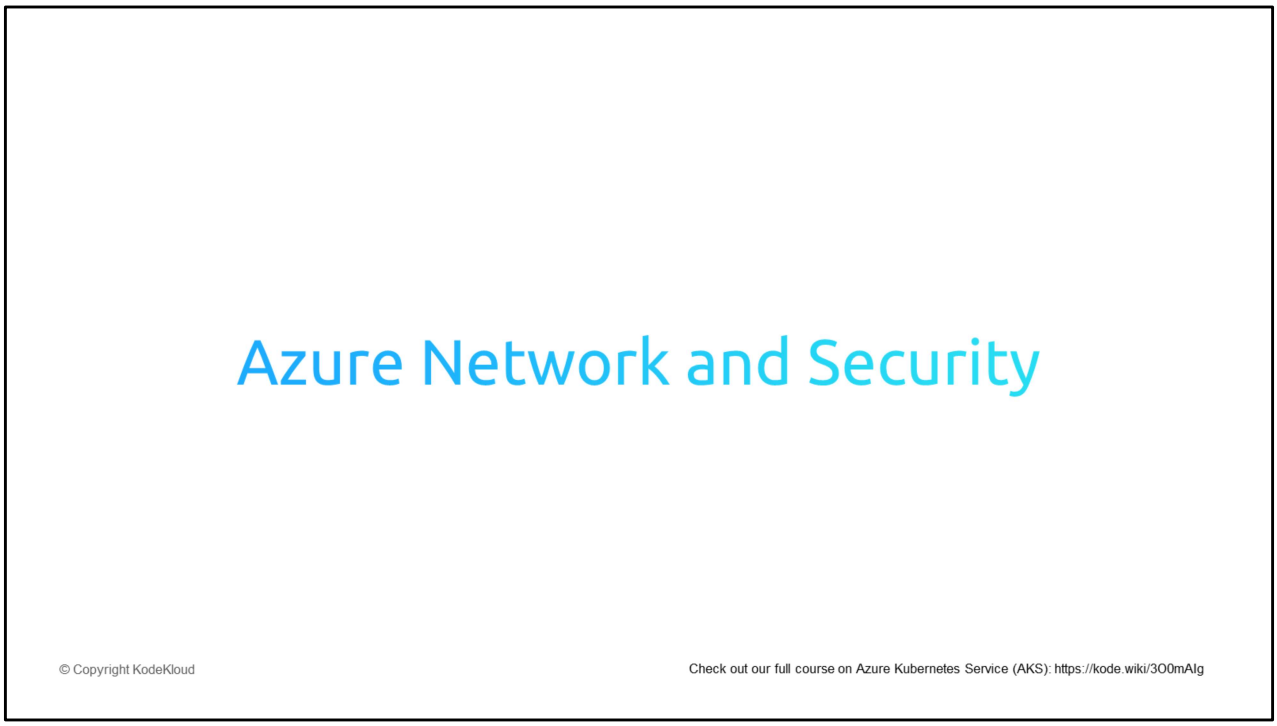
Most of your Kubernetes applications will interact with one or more databases. Azure has a rich of data offerings both in relational and non-relational databases. I have tabulated some of the common use case for these database offerings.
Some of the most common Database offering used on Azure are:



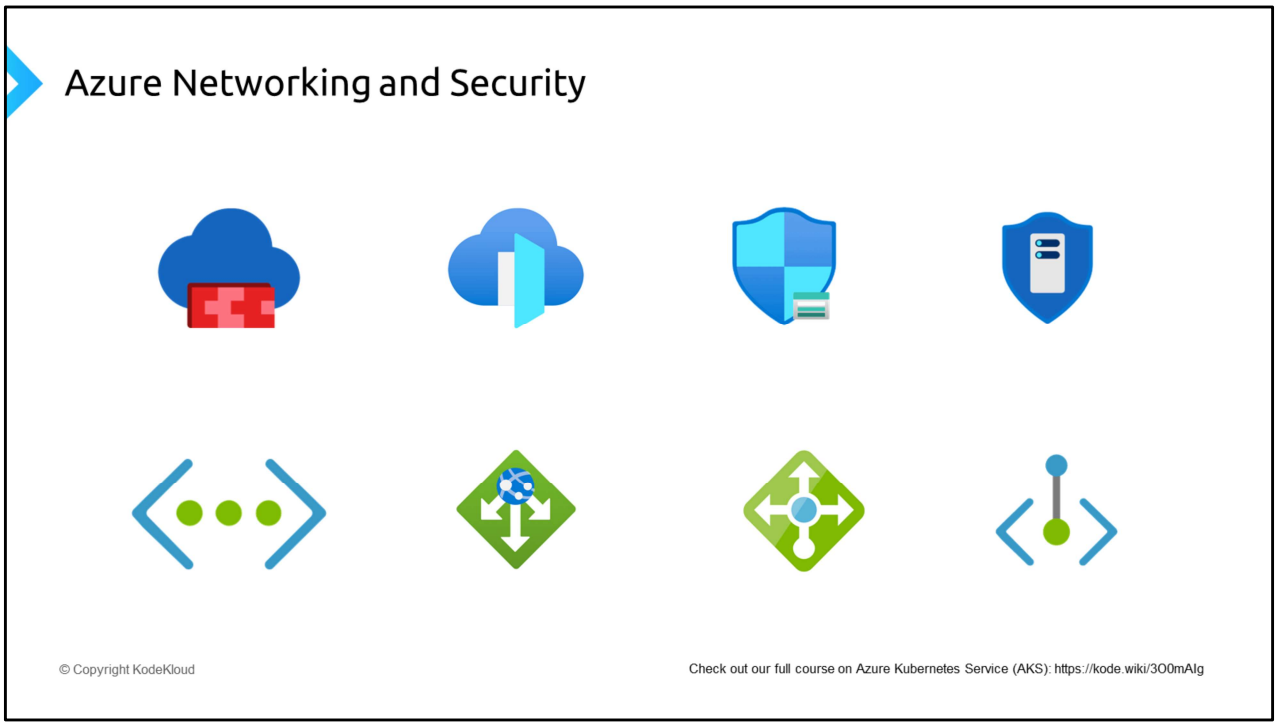
1.) Azure SQL: There are 3 way to deploy SQL databases based on the Microsoft SQL Server database engine - Azure SQL Database which is a PaaS solution, SQL Managed instance which is a hosted instance of SQL server and a traditional SQL on VM option

2.) There are then Azure Database for open-source relational databases like MySQL, MariaDB and PostgreSQL

3.) When it comes to non-relational databases, Azure's no #1 option is Cosmos DB which is a global-scale *NoSQL* database system that supports multiple application programming interfaces (APIs), enabling you to store and manage data as JSON documents, key-value pairs, column-families, and graphs.

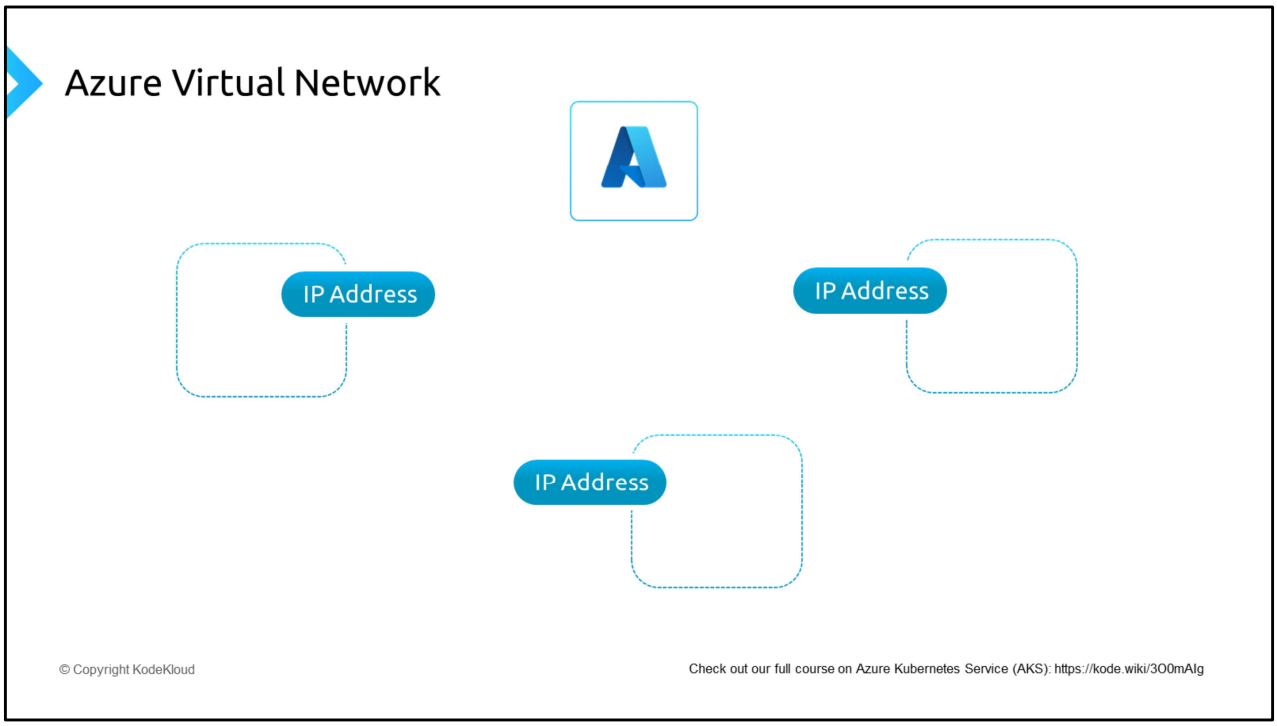


Have you ever written a program but didn’t have an application server to run it on? Maybe you're a developer and you’ve written code that executes a basic task.

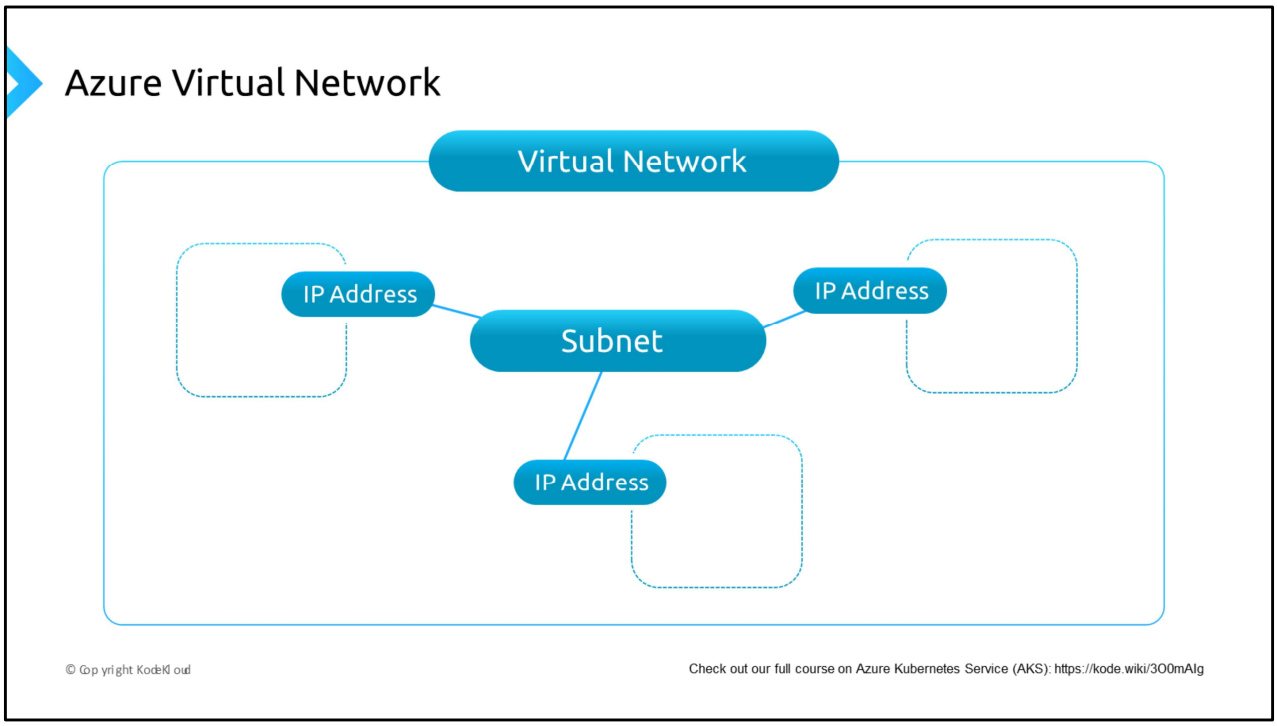


Azure has a strong and extensive set of services when it comes to networking and security.

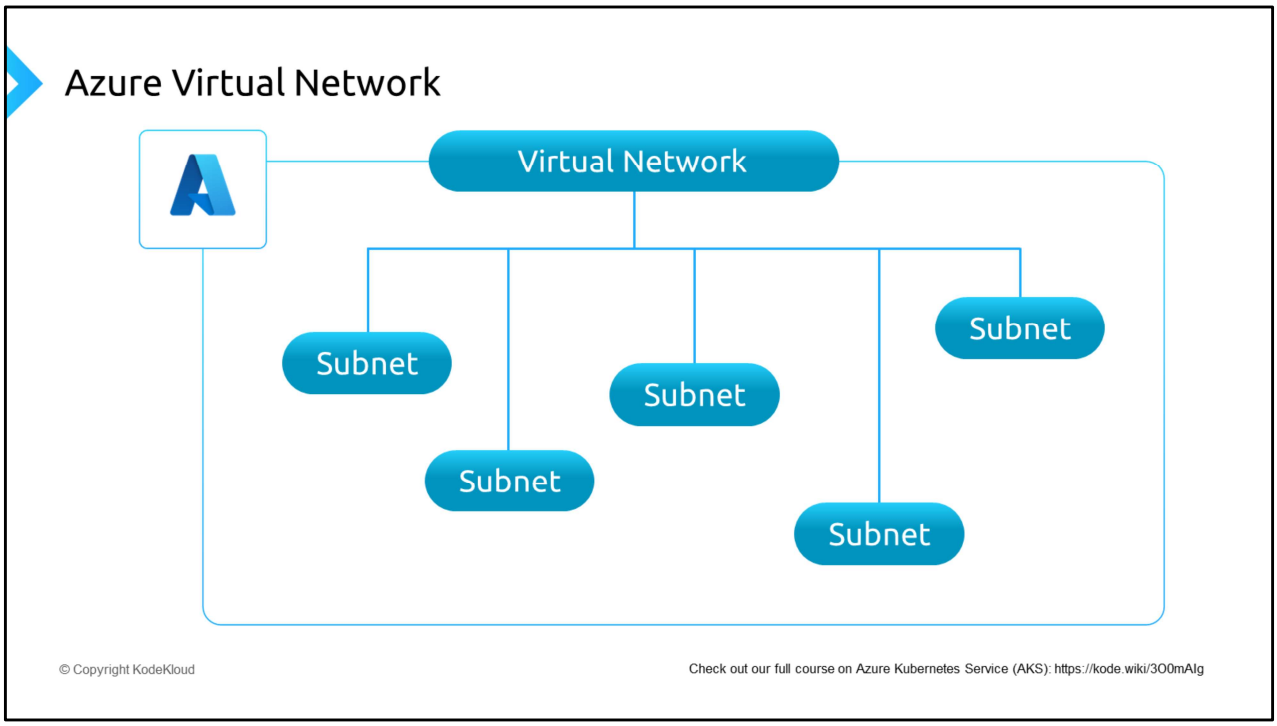
It also provides you with an extensive list of 3rd party services so you can get additional functionality if missing in native products or extend your existing products from on-premises or another cloud. Let us cover some basic connectivity and application delivery services.



Most services on Azure will require an IP address to function and an Azure Virtual Network or VNet is that fundamental building block in Azure that IP addressing depends on.

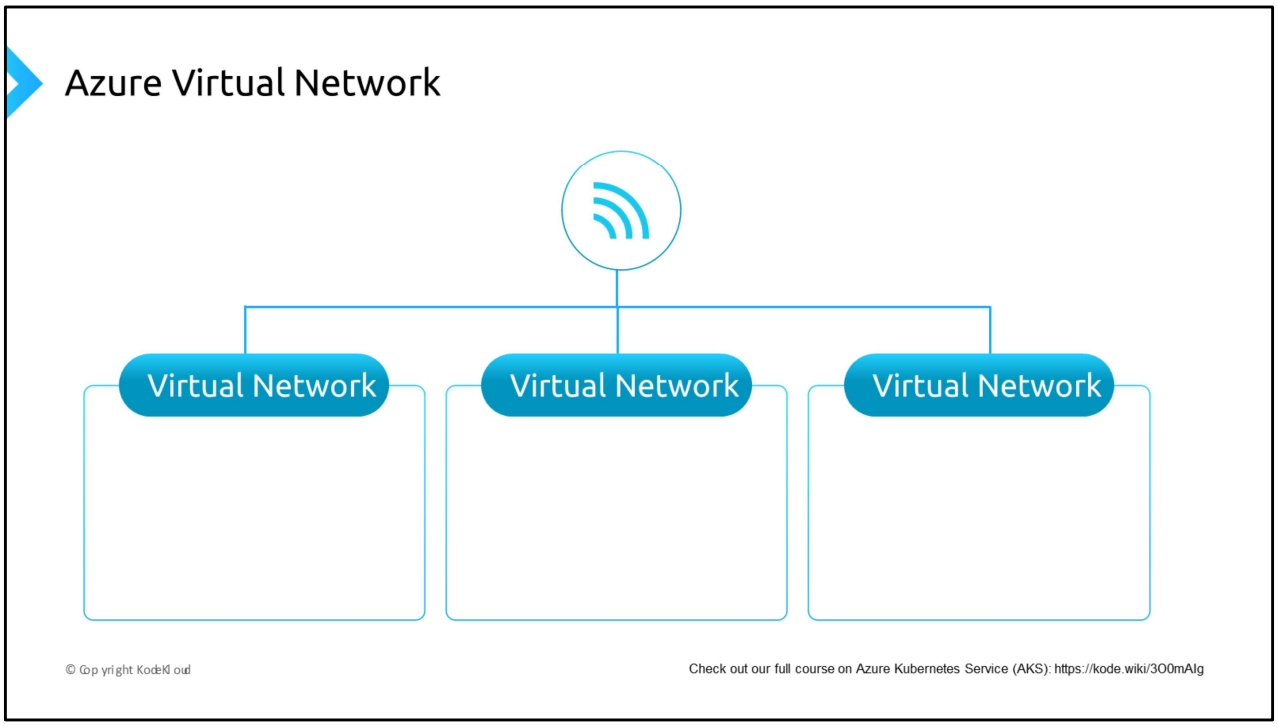


IP addresses are part of a subnet which in turn are part of a vnet.
Azure routes traffic between subnets, connected virtual networks, on-premises networks, and the Internet, by default



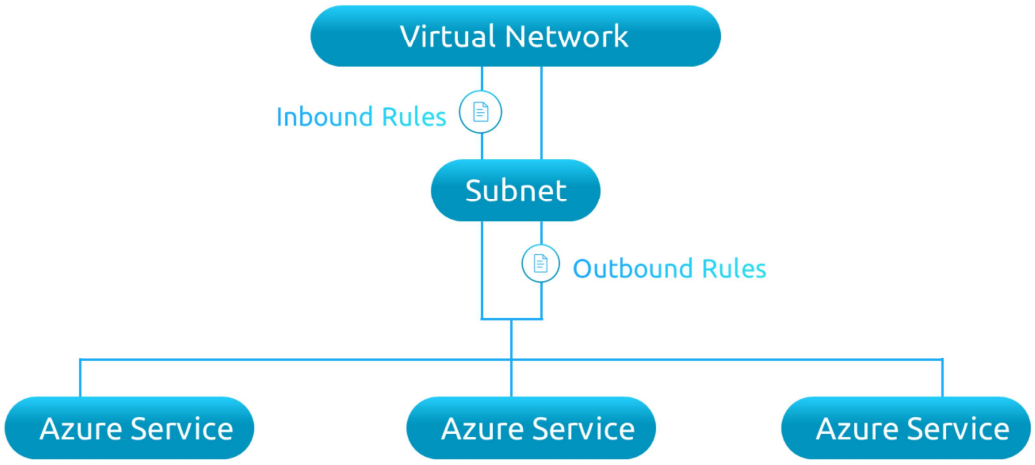
You can protect resources within your subnet via Network security group by adding inbound and outbound rules to allow or deny based on IP addresses and port combination.

By default, a deny all rule is applied as the least priority rule and you will need to explicitly allow traffic to enable communication.

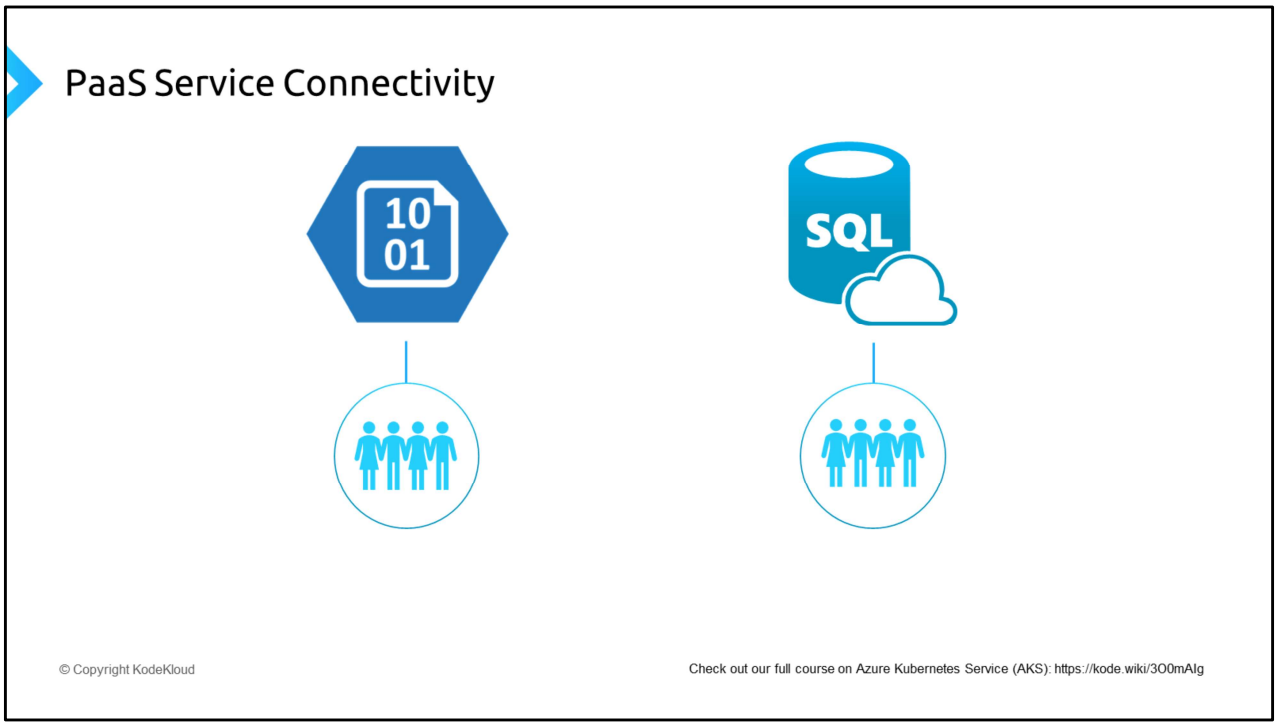


If you are coming from AWS background, it would be worth noting that subnets in Azure can span availability zones, and the same vnet can have private and public IPs.

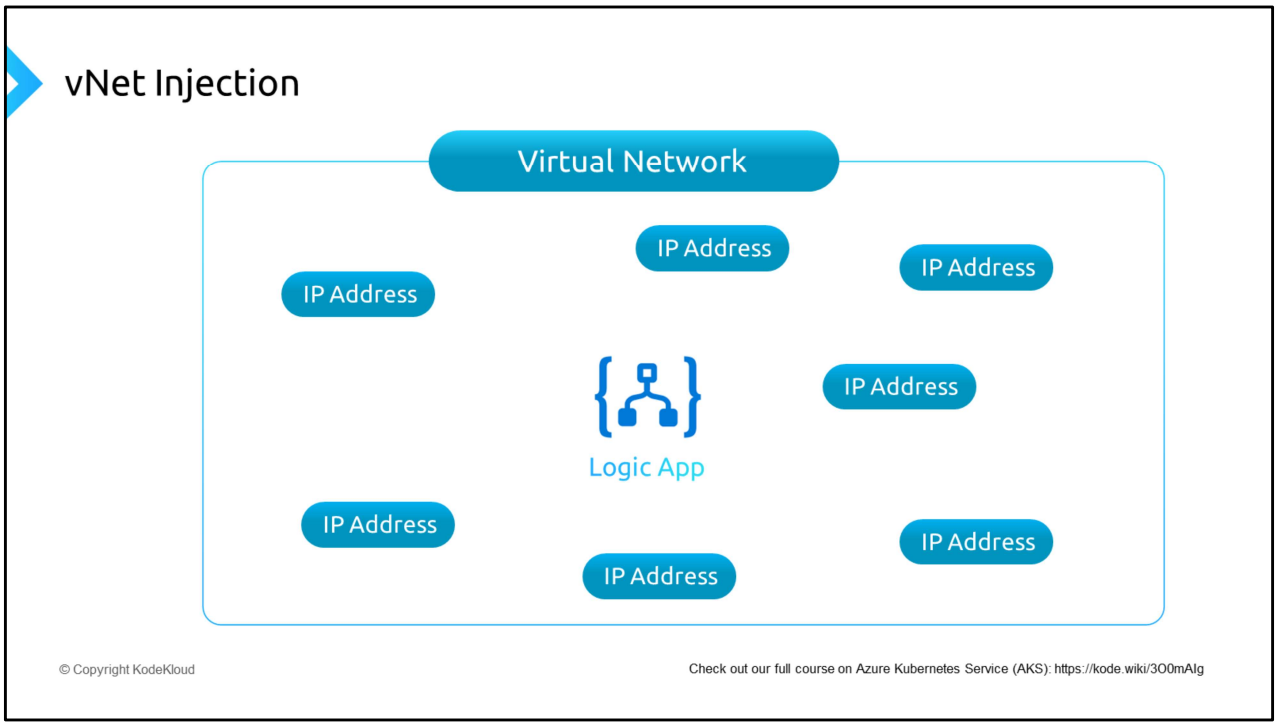
Azure Virtual Network



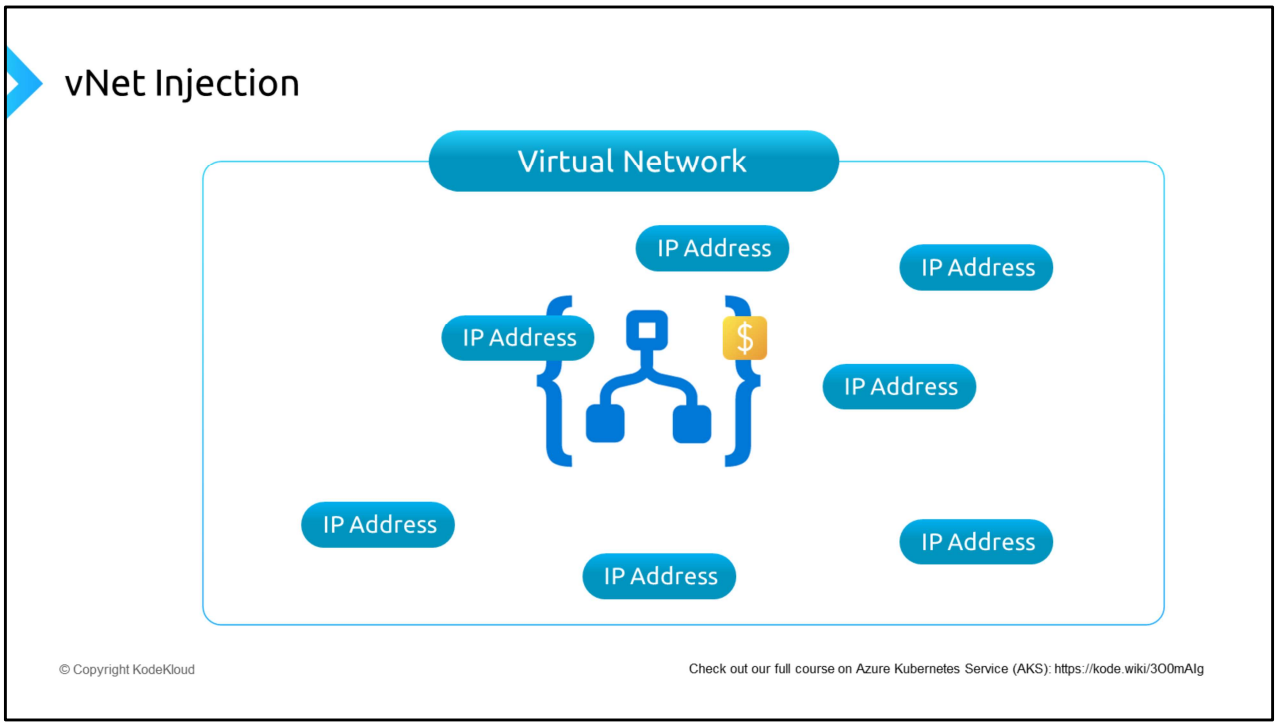
Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg



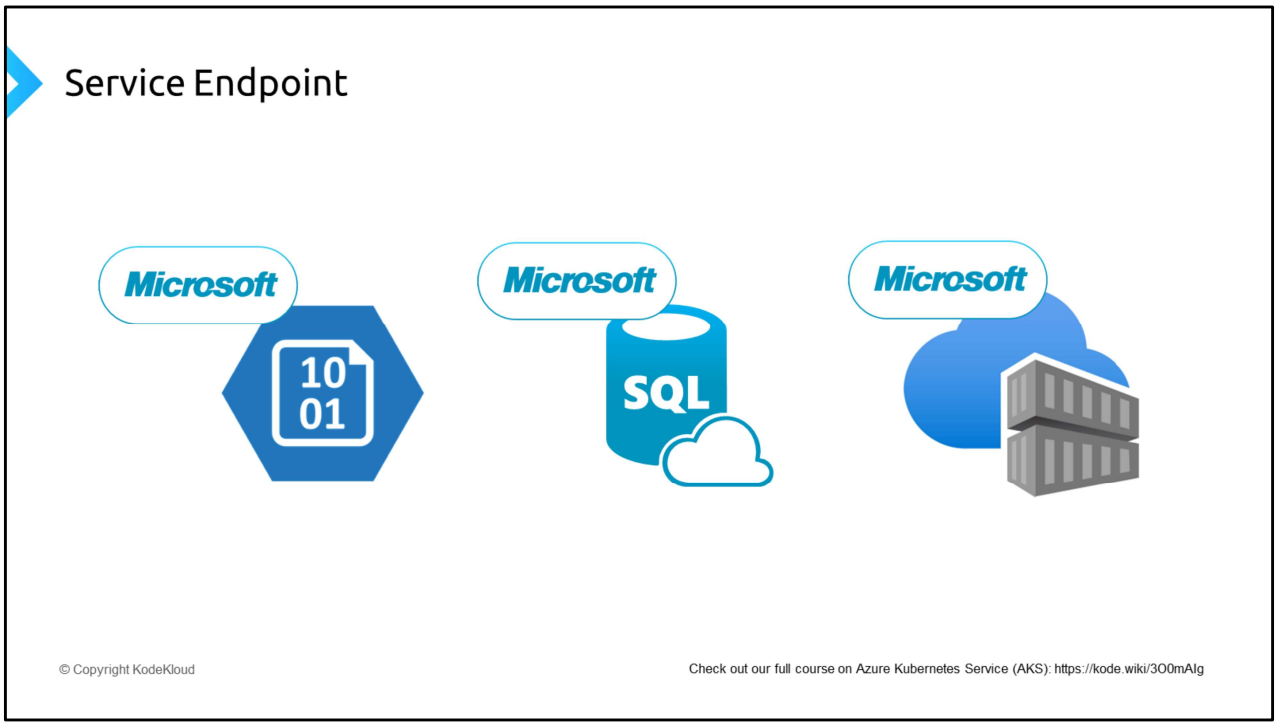
Azure multi-tenant PaaS Services like Azure Storage/SQL, etc. have public endpoints. In order to deploy and access these services privately and securely, there are three options.



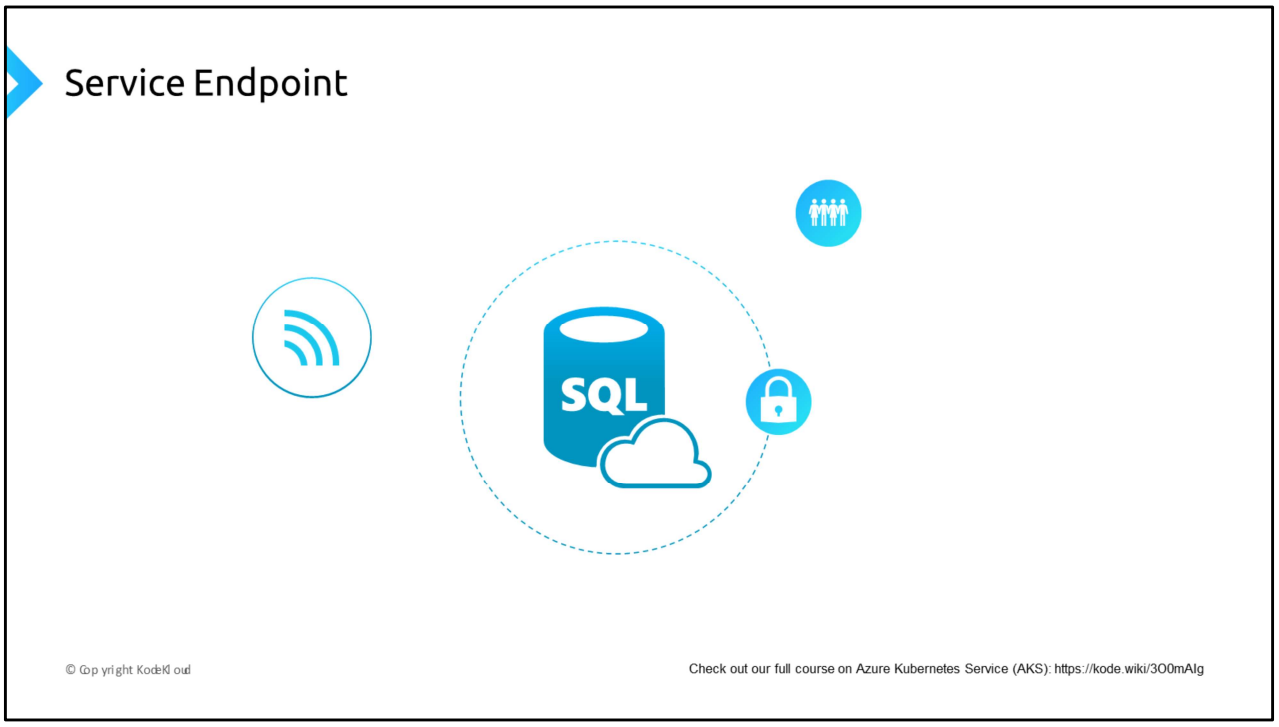
vNet Injection : in which an instance of the service like Logic App is deployed in your private vNet. Because the entire instance is dedicated to you, you can assign a private IP from within you range. Not all PaaS services are vNet injection enabled and where you have the ability to do it, it is a very expensive option.



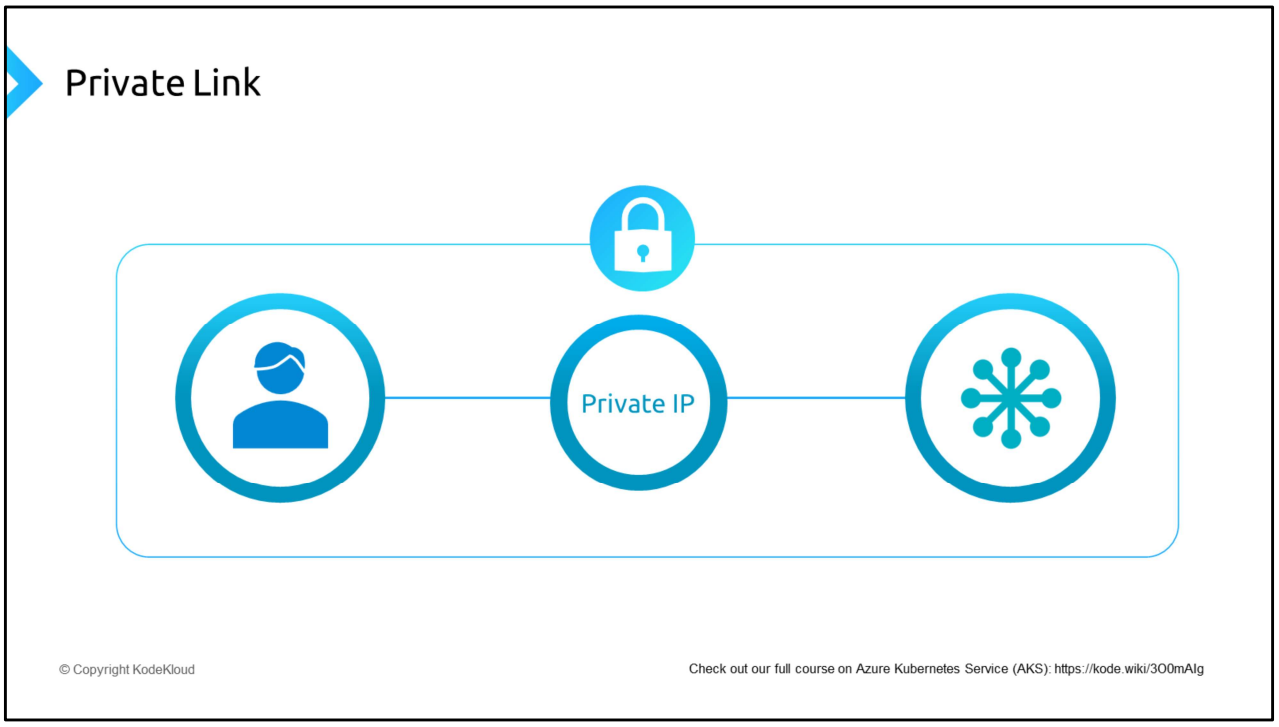
The second method is Service Endpoint where you access multi- tenant PaaS Service like storage, SQL or Azure Container Registry using Microsoft backbone and have the ability to deny all access from public internet, This option is cost effective as the service remains multi-tenanted but you can secure it by denying public access



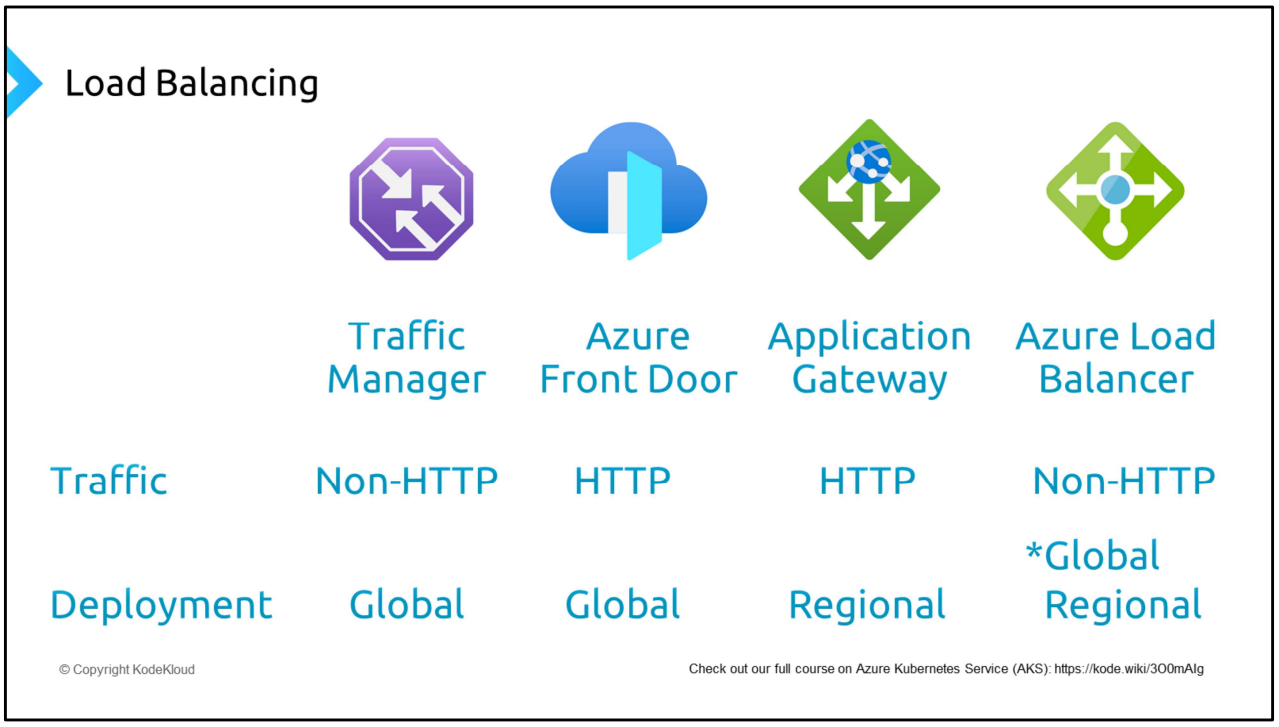
The last option it to use private link, where you can create a private IP within your IP ranges and link it to an instance of PaaS service that you deploy.



For example, you can create a private endpoint for AKS i.e Private cluster, or Azure SQL and access it via a private IP. This is by far the most secure way to access your PaaS Services



The last option it to use private link, where you can create a private IP within your IP ranges and link it to an instance of PaaS service that you deploy. For example, you can create a private endpoint for AKS i.e Private cluster, or Azure SQL and access it via a private IP. This is by far the most secure way to access your PaaS Services

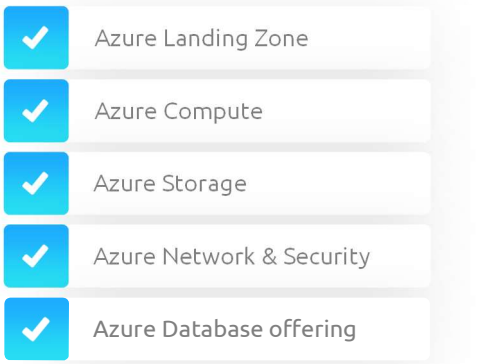


There are four native load balancing options on Azure and all of them can distribute traffic to backend compute resources based on a defined criteria. What you end up using depends on the traffic type ie. Layer 7 or HTTP traffic or non-http traffic. For example, if you need to make routing decisions based on an HTTP request, you cannot use an Azure Load balancer for that.

The other important criteria is whether you need to load balance across azure regions or within a region. For example, if you need to host an active-active application with nodes across multiple azure regions, and Azure front door could be used.

At the time for this recording, Azure loadbalancer has a global option in preview and can be used for global non-HTTP traffic.

Azure Kubernetes Services (AKS) - Summary

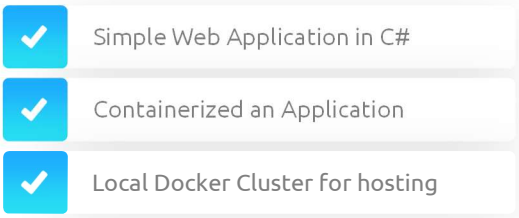


Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

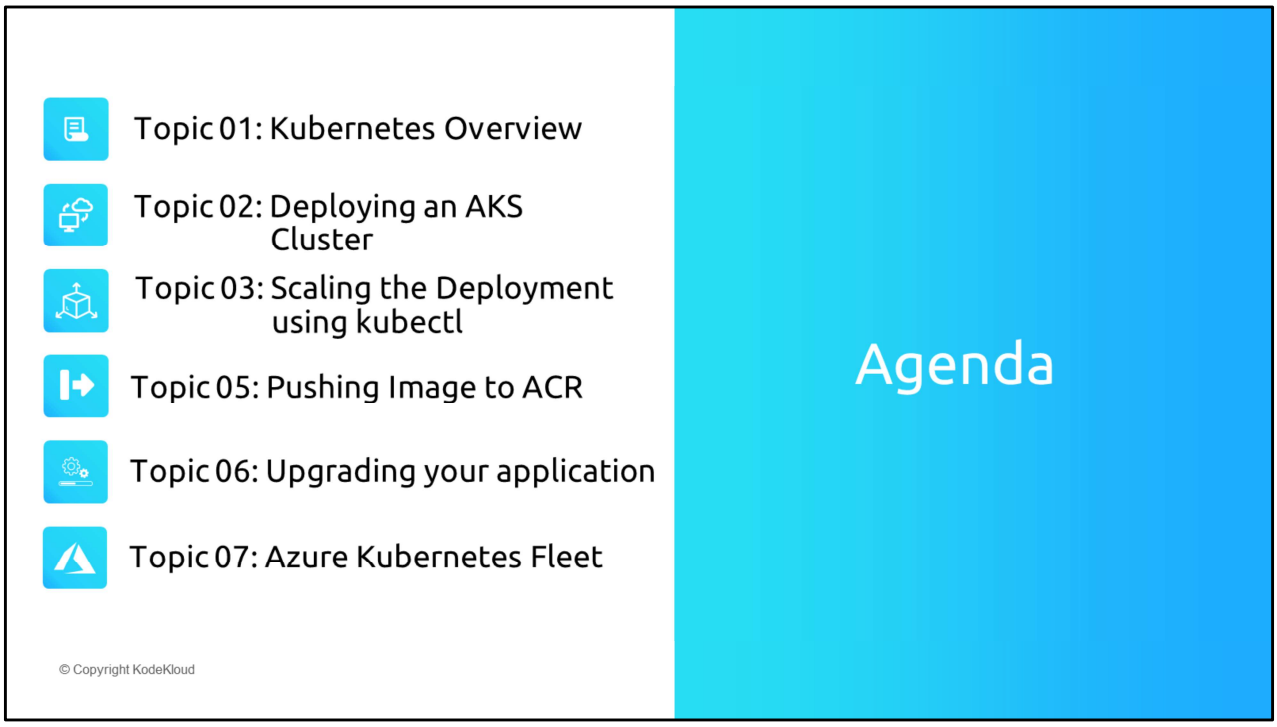
KodeKloud

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Building and Containerizing Sample Application

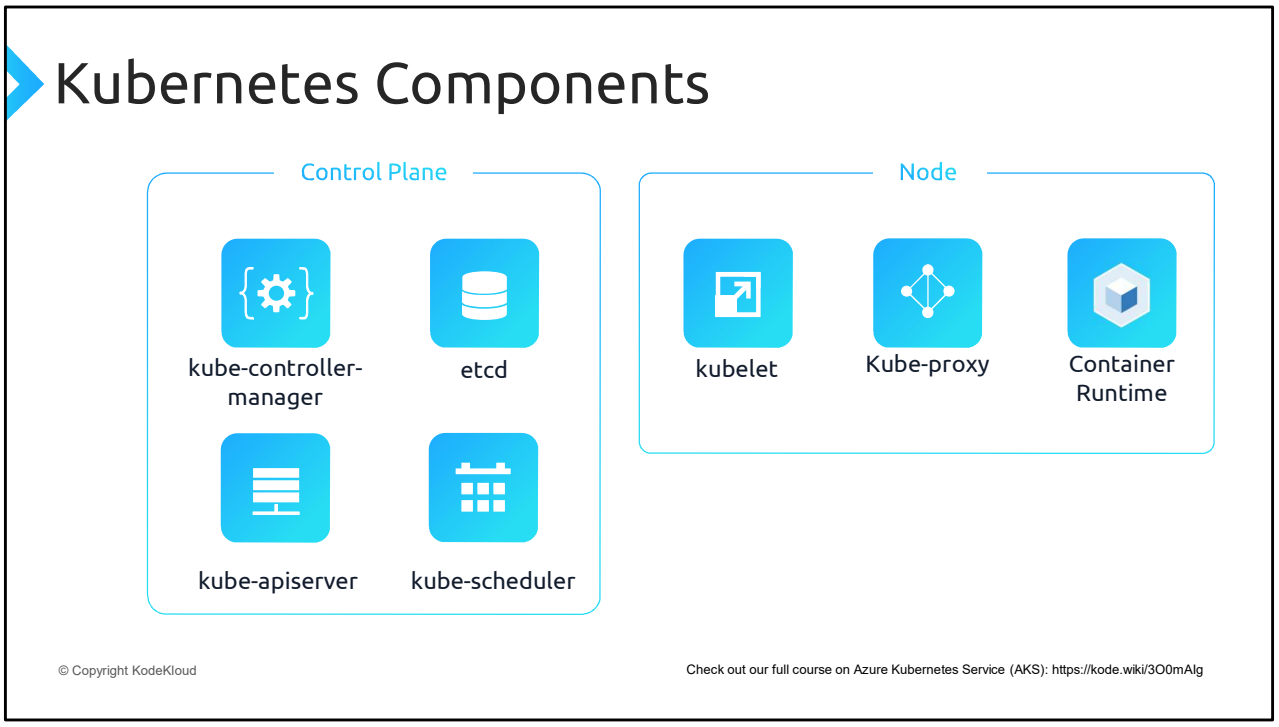


Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg


Topic 01: Kubernetes Overview
Topic 02: Deploying an AKS Cluster
Topic 03: Scaling the Deployment using kubectl
Topic 05: Pushing Image to ACR
Topic 06: Upgrading your application
Topic 07: Azure Kubernetes Fleet
© Copyright KodeKloud
Agenda

Kubernetes Overview

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

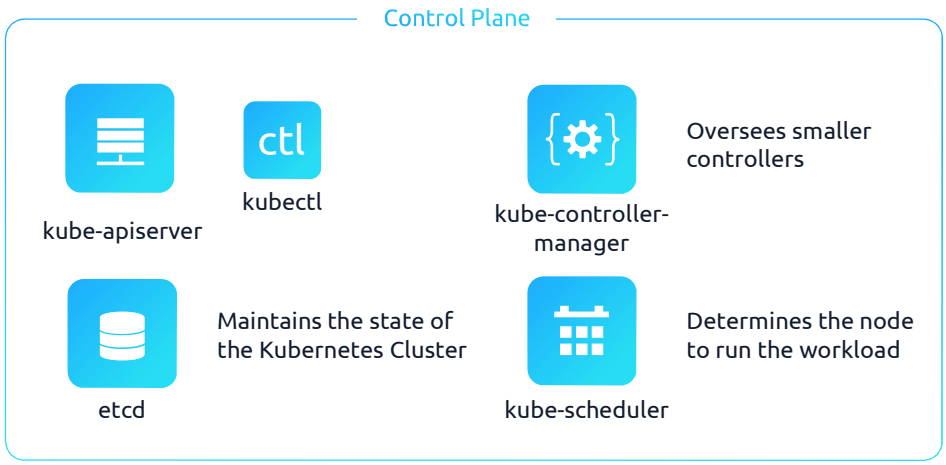


While this course is not designed to make you a Kubernetes expert, I'll cover some Kubernetes fundamentals so you can follow along the rest of the course. We, at KodeKloud, have some of the best Kubernetes courses across any platform. So do check them out later if you want to learn more about Kubernetes.

K8s architecture can be broken down into two parts: The control plane components which include the Controller manager, API Server, etcd, and scheduler
And the node components which include Kubelet, the kube proxy, container runtime

The control plane components provides the core K8s services and orchestration of the application workloads.
While the actual ..application workloads run on the node component. In Azure K8s service control plain is automatically created, configured and managed by azure.

Kubernetes Components



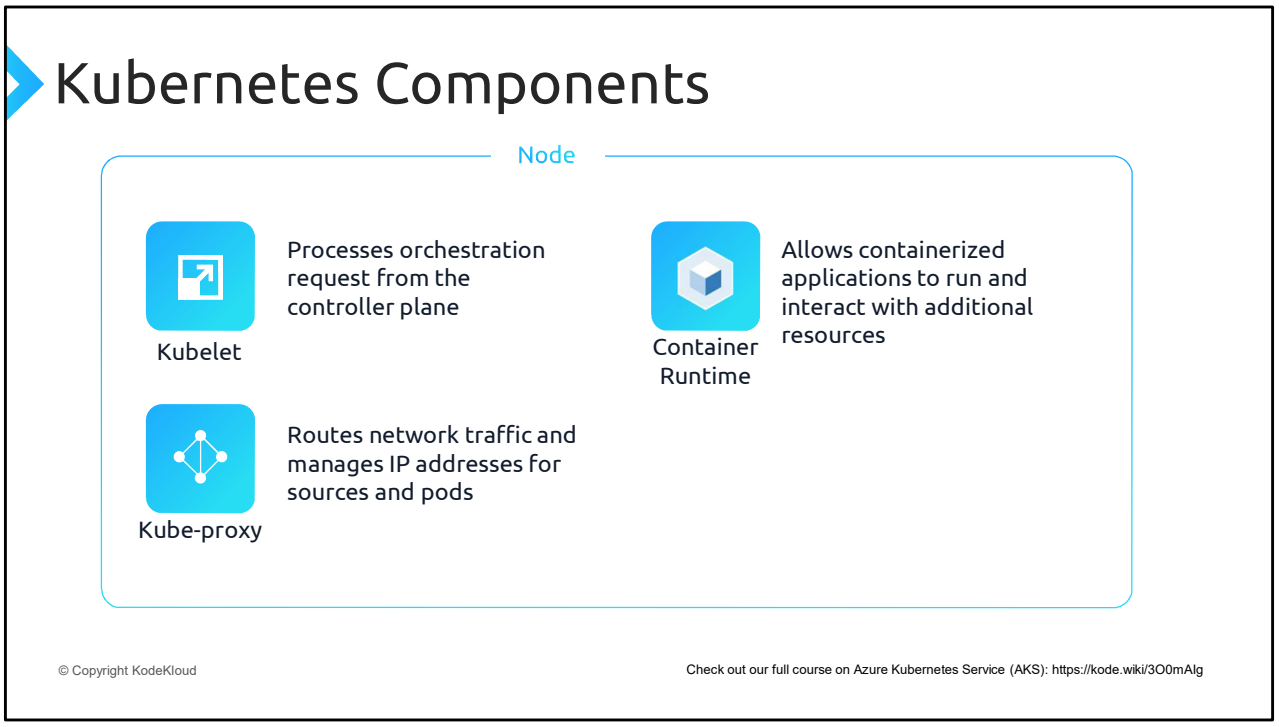
Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

There are 4 main services running in the control plane.

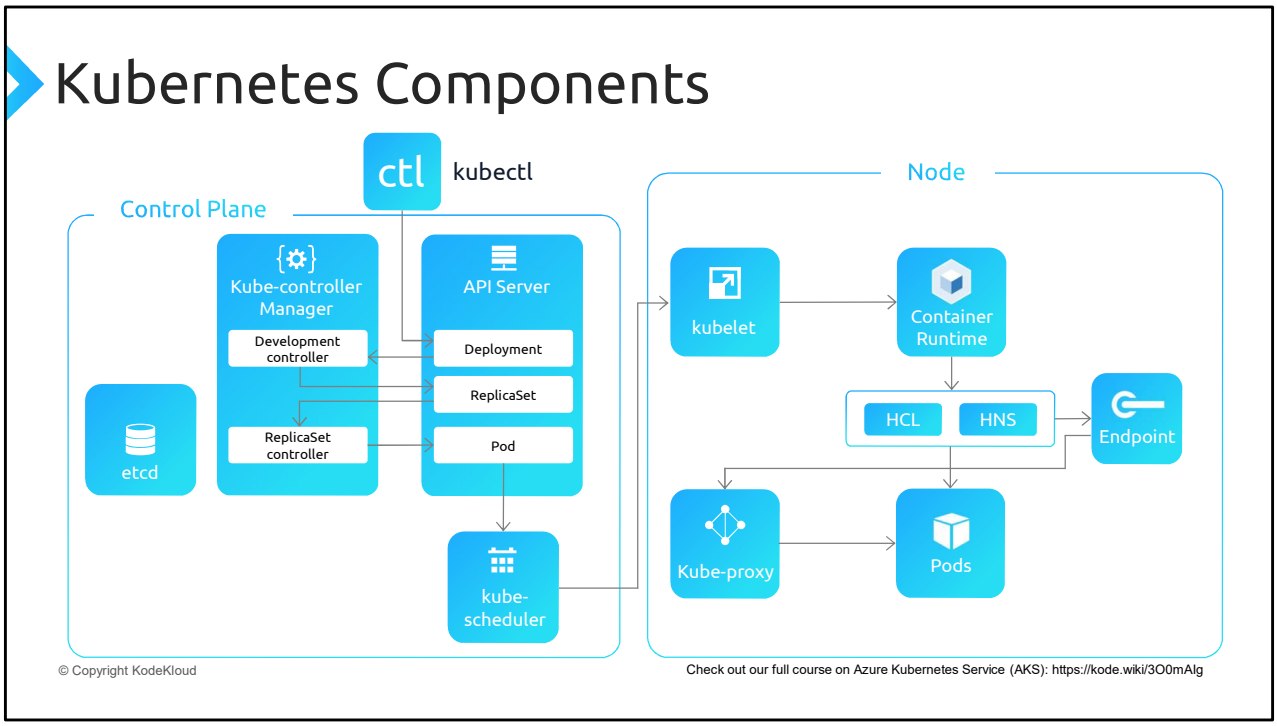
kube-apiserver is how the underlying K8S APIs are exposed. This component provides the interaction for management tools, such as kubectl.
Etcd : is used to maintain the state of your K8s cluster and configuration. This highly available service is a key value store within K8s.
kube-controller-manager : overseas a number of smaller controllers that perform actions such as replicating pods and handling node operations.
When you create or scale applications, the kube-scheduler determines what nodes can run the workload and starts them
On the agent node components side, we have

K8s architecture can be broken down into two parts: The control plane components which include the Controller manager, API Server, etcd, and scheduler
And the node components which include Kubelet, the kube proxy, container runtime

The control plane components provides the core K8s services and orchestration of the application workloads.
While the actual ..application workloads run on the node component. In Azure K8s service control plain is automatically created, configured and managed by azure.



Kubelet : which is a k8s agent that processes the orchestration request from the control plane and schedule 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 virtual network and storage.



Now that you know some of the components and their high-level functionality, let us see how these components talk to each other when you issue a command to create a new Kubernetes deployment.

Do not worry if this ends up looking very complicated, you do not need to remember all of this. Just remember there are multiple subcomponents with a well-defined function for each of them.
1: KubeCTL gives a POD Deployment manifest to the API Server.
2: API Server Creates Deployment Resource and that info is saved in etcd.
3: Deployment Controller watches for any new deployment resource and creates the replicaset resource.
4: Replicaset controller watches for any new replicatSet resource.
5: Replicaset controller creates the POD resource based on how many replicaset and actual pods are present.
6/7: Kube Scheduler watches for unbound POD resource and schedules them on to a node.
8: Kubelet calls Container Runtime Interface (CRI) and (Container Network Interface) to create the POD/Containers with networking.
9&10: CRI calls Host Compute Layer (HCS) to create the Container and Host

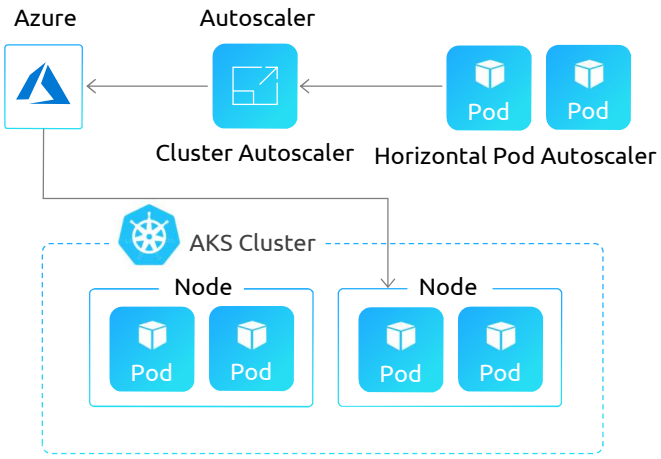
Networking Service (HNS) to create the Endpoint.
11&12: Kubeproxy watches for any new Endpoint and programs HNS with Load Balancer and Access Control List (LB and ACL) Policies.

Now that we have covered some k8s fundamentals, let us start deploying a cluster in Azure.

Scaling the Nodes using Azure CLI

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Cluster Autoscaler

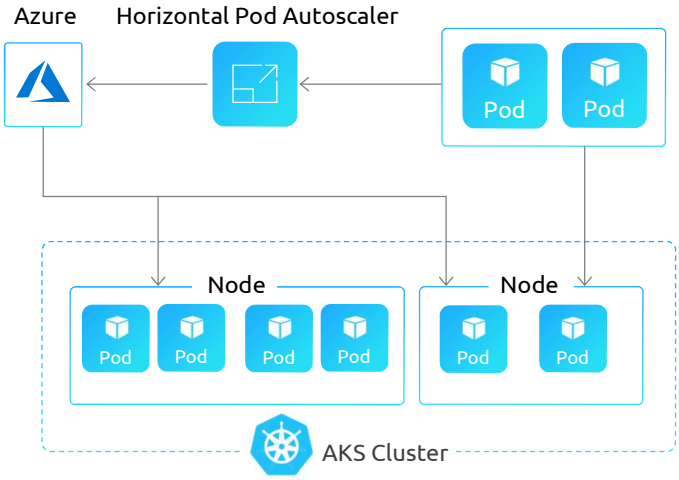


Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

We ended the last video with a cluster that has been configured to run 30 pods and is running only 16.

Cluster Autoscaler

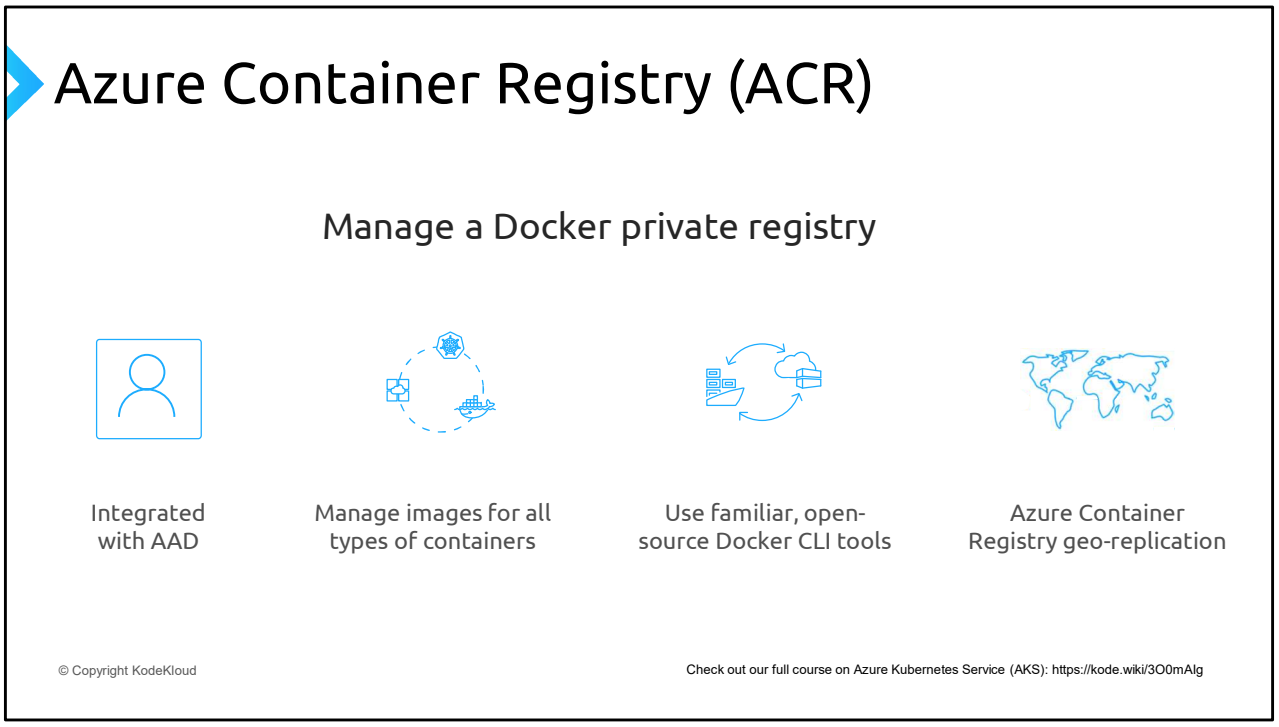
1. Horizontal Pod Autoscaler (HPA) obtains resource metrics and compares them to user-specified threshold
2. HPA evaluates whether user-specified threshold is met or not
3. HPA increases/decreases the replicas based on the specified threshold
4. Deployment controller adjusts the deployment based on increase/decrease in replicas



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAlg

As I mentioned in the previous slide, Kubernetes uses the horizontal pod autoscaler (HPA) to monitor the resource demand and automatically scale the number of pods. HPA is updated every 60 seconds with key metrics like CPU utilization to take scale out or scale in decisions.

This is 4 step process.. In step 1..



A container *registry* like Docker hub is a service that stores and distributes container images and related artifacts. In the previous module we containerized an application and pushed it to Docker hub.

Azure Container Registry is a private registry and provides users with direct control of their container content, with integrated authentication, geo-replication, distribution and, virtual network configuration with Private Link, and many other enhanced features.

In addition to Docker-compatible container images, Azure Container Registry supports a range of content artifacts including Helm charts and Open Container Initiative (OCI) image formats.

The *tag* for an image or other artifact specifies its version. A single artifact within a repository can be assigned one or many tags and may also be "untagged." That is, you

can delete all tags from an image, while the image's data (its layers) remain in the registry.

Azure Container Registry (ACR) Tiers

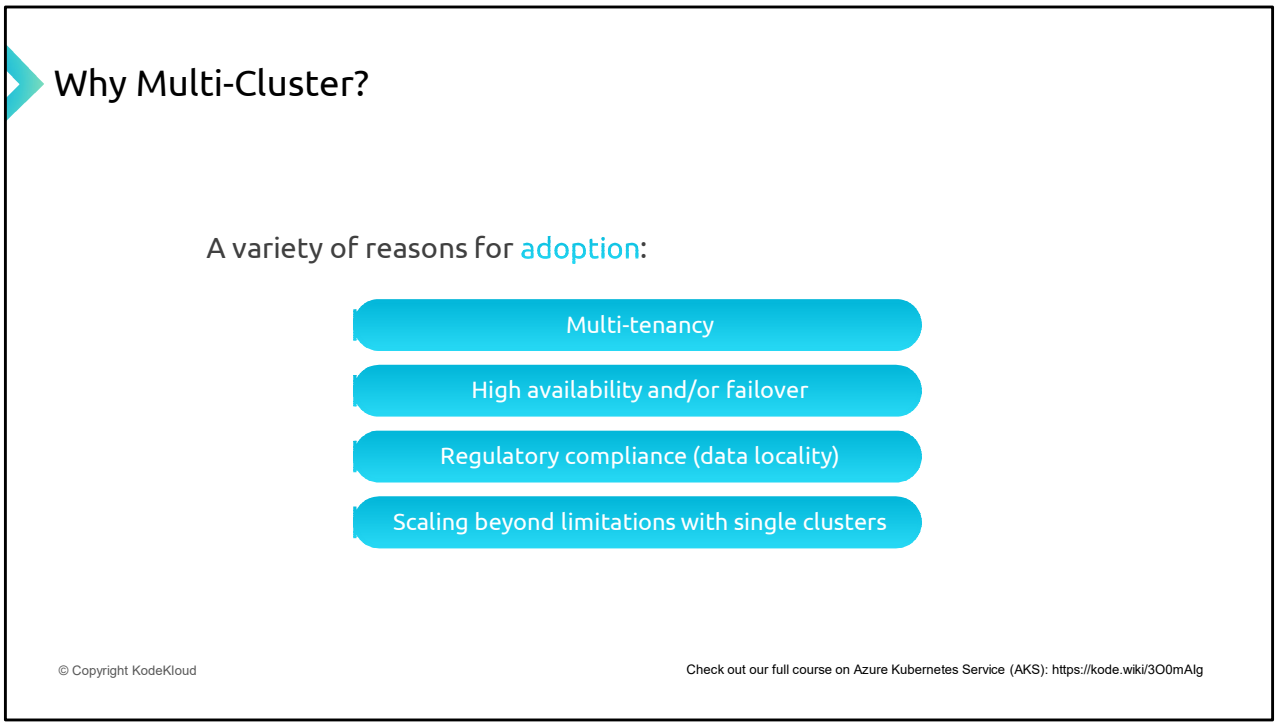
Resource	Basic	Standard	Premium
Included storage (GiB)	10	100	500
Storage limit (TiB)	20	20	20
Maximum image layer size (GiB)	200	200	200
Maximum manifest size (MiB)	4	4	4
ReadOps per min	1000	3000	10000
WriteOps per min	100	500	2000
Download Bandwidth (Mbps)	30	60	100
Upload Bandwidth (Mbps)	10	20	50
Private endpoints	n/a	n/a	200
Geo-replication	n/a	n/a	Supported

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Azure Container Registry is available in multiple SKUs. I have added some differences in the table here. This should help you choose a tier that can work for you. In case you want to upgrade, there is a command to upgrade. You can move freely, both up and down, between tiers as long as the tier you're switching to has the required maximum storage capacity. For example, if you are using 200 GB in a premier tier and run a command to downgrade ACR to standard, that change will fail as the maximum storage included in Standard tier is 100 GB.

If you recall, we had created a ACR when creating AKS cluster. Now lets push our

image to that registry.

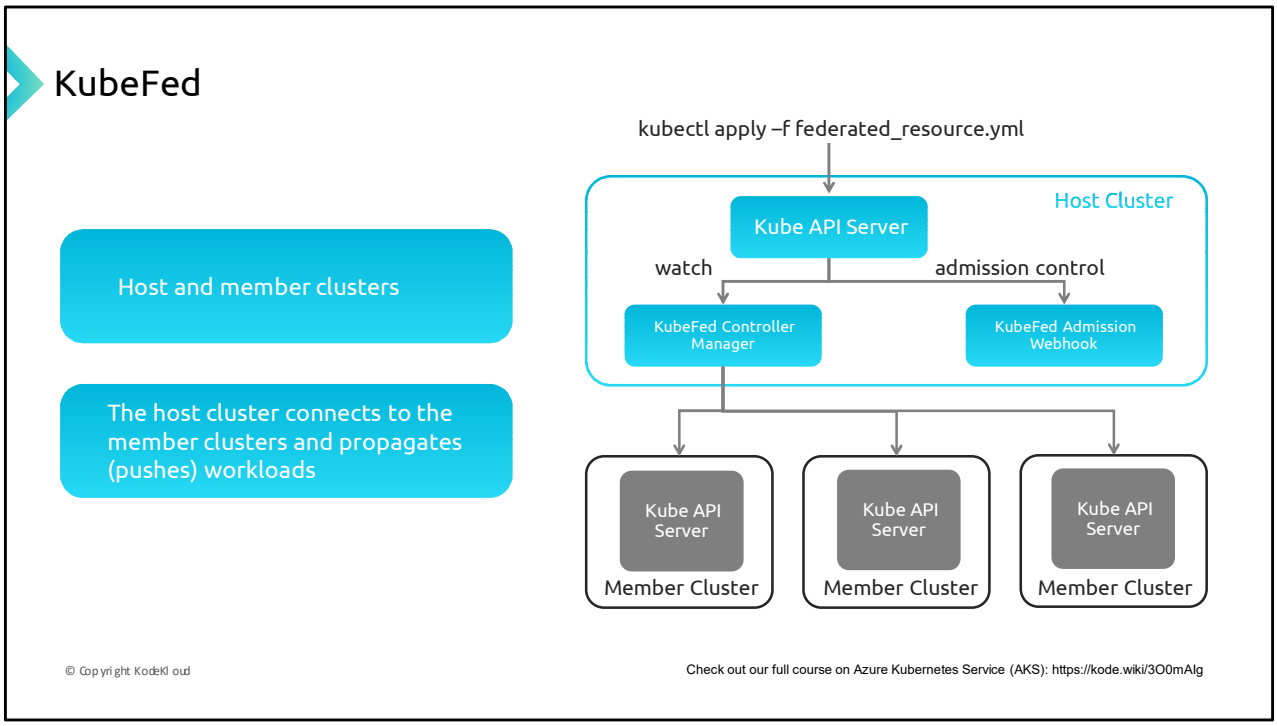


Þ until now, we have seen how to manage a single Kubernetes cluster in AKS. In real life, you will be dealing with 100s if not 1000s of AKS clusters.

Multi cluster system, by itself is not a new concept as you can imagine many large organisations would have hundreds of Kubernetes clusters spread across the world to host their applications.
A logical question to ask here, is why do I need multi cluster systems in the first place ? Can I not host all my applications in a single cluster across Pods and namespaces ?

Multi- cluster pattern are used for variety of reasons – Some use it to achieve multi tenancy, other use it as it provides a convenient way to achieve high availability or failover, making sure that a failure of a single cluster region or even cloud provider will not impact your application' s availability.
If there is nothing else, you can always depend on compliance as a reason to complicate matters, haha but honestly there are real compliance reasons requiring your application to be hosted on separate Kubernetes clusters across geographic boundaries. For example, in US if you are hosting a betting app, by law you are required to host that app in each state boundary.

If your application is huge, you can also use multi cluster system to scale beyond prescribed limits.

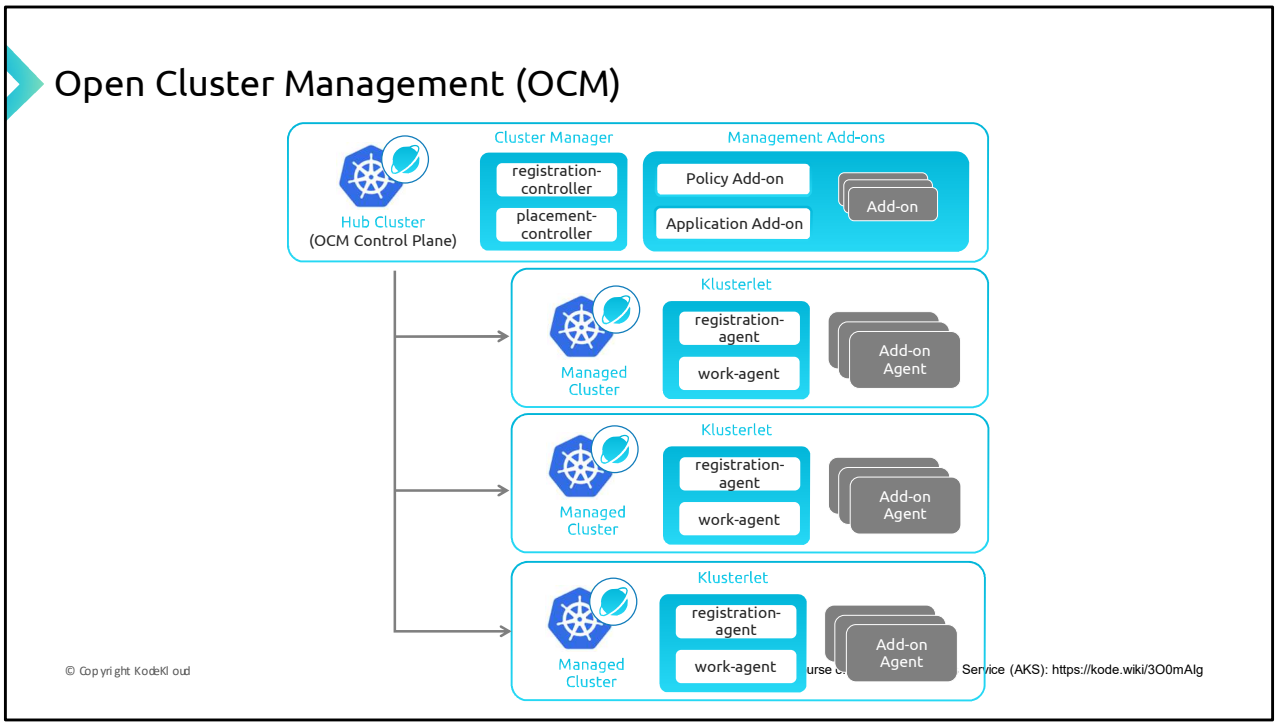


So lets look some of the open source solutions for multicluster management.

The first one is Kubernetes Cluster Federation aka KubeFed, the official multicluster solution sponsored by Multicluster Special Interest Group.
It is built on top of the concept of Kubernetes federation, the core idea of which is to provide a way to coordinate services among a group a Kubernetes clusters. This project has a long and complicated history and has two major versions so far – Kubefed v1 and kubefed V2. V1 is already deprecated.

Multi-cluster pattern are used for variety of reasons – Some use it to achieve multi tenancy, other use it as it provides a convenient way to achieve high availability or failover, making sure that a failure of a single cluster region or even cloud provider will not impact your application's availability.
If there is nothing else, you can always depend on compliance as a reason to complicate matters, haha but honestly there are real compliance reasons requiring your application to be hosted on separate Kubernetes clusters across geographic boundaries. For example, in US if you are hosting a betting app, by law you are required to host that app in each state boundary.

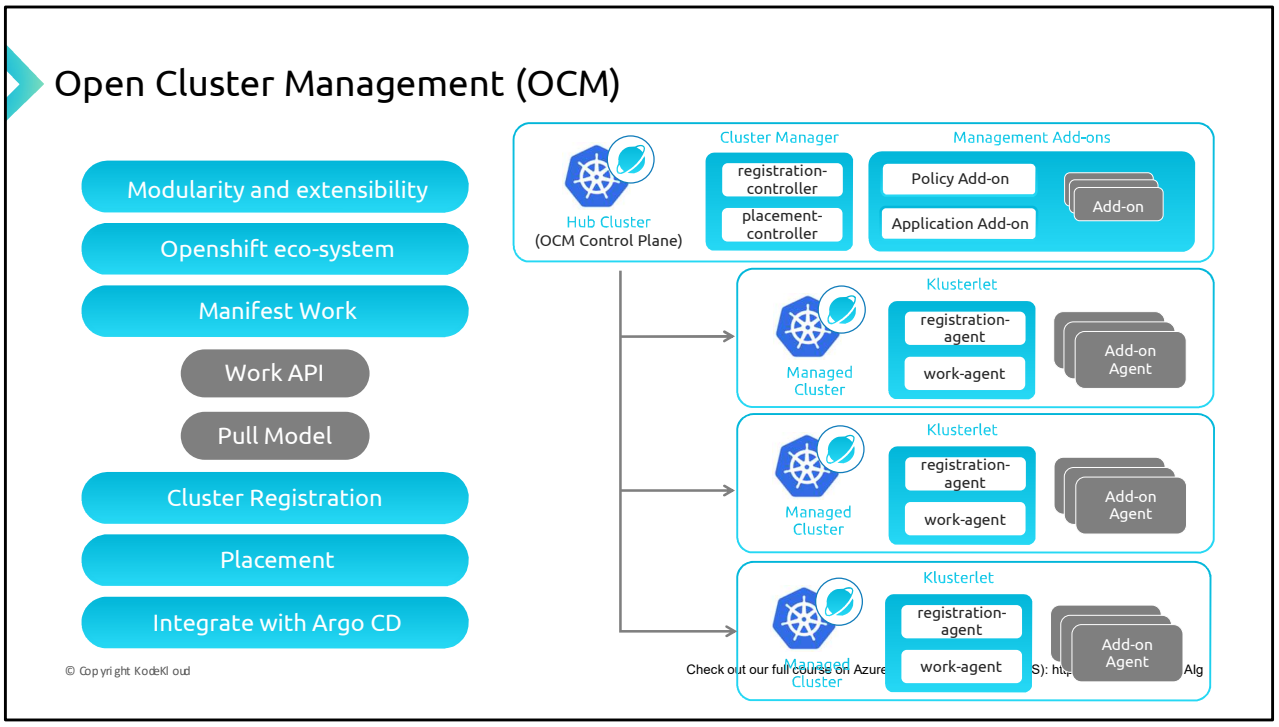
If your application is huge, you can also use multi cluster system to scale beyond prescribed limits.



Open Cluster Management (OCM)
is a community driven project focusing on multicluster and muticloud deployments. Interesting fact - this project is backed by redhat and you will see them actively backing this project.
https://open-cluster-management.io/concepts/

The orchestration in OCM is similar to K8s, in a sense that they have a hub cluster that mimics K8s control plane and spoke clusters have an agent called Klusterlet similar to kubelet in K8s. Klusterlet in turn have registration and work agents.

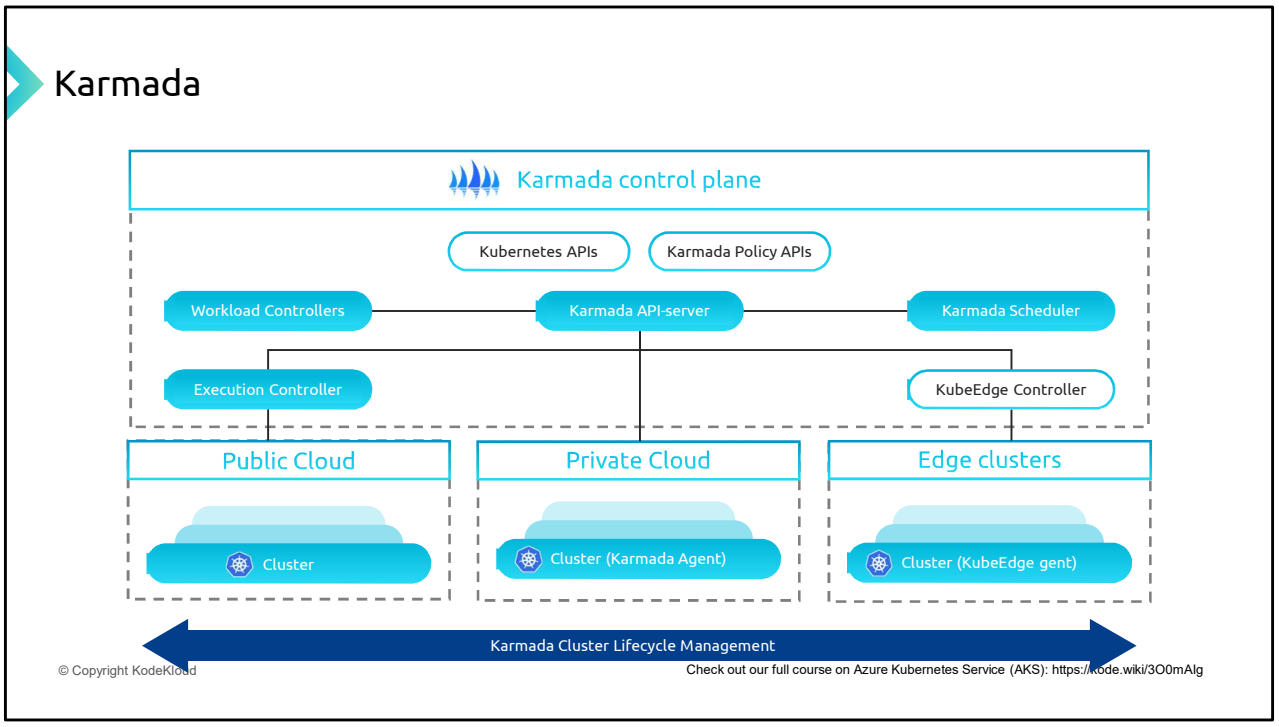
If you want to know more about this system, you can access their repo via the link on the screen. Key concepts to understand before you can use OCM is their Cluster Registration process that you see on the screen now and how they use placement and scheduling. If you or your organisation is using OCM I would love to get your feedback on how its working in AKS. Please share your comments in this module.



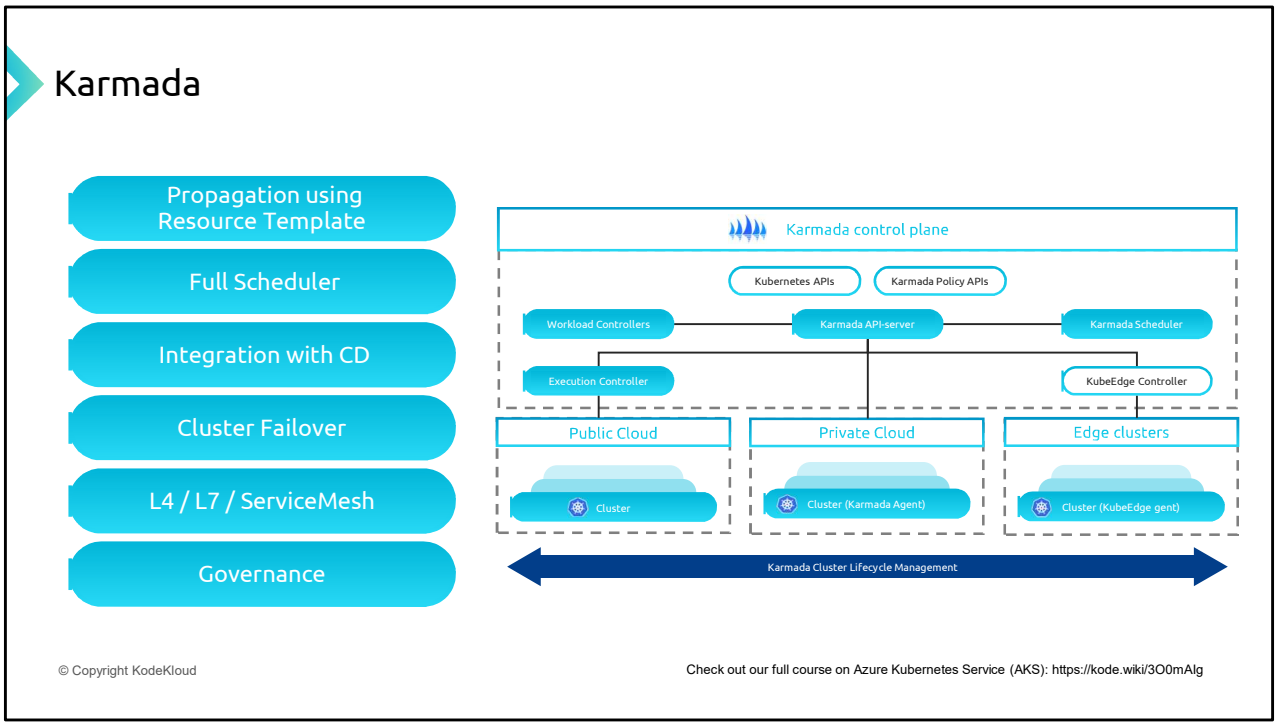
Open Cluster Management (OCM)
is a community driven project focusing on multicluster and muticloud deployments. Interesting fact - this project is backed by redhat and you will see them actively backing this project.
https://open-cluster-management.io/concepts/

The orchestration in OCM is similar to K8s, in a sense that they have a hub cluster that mimics K8s control plane and spoke clusters have an agent called Klusterlet similar to kubelet in K8s. Klusterlet in turn have registration and work agents.

If you want to know more about this system, you can access their repo via the link on the screen. Key concepts to understand before you can use OCM is their Cluster Registration process that you see on the screen now and how they use placement and scheduling. If you or your organisation is using OCM I would love to get your feedback on how its working in AKS. Please share your comments in this module.



The next opensource solution for multi cluster management that we will touch upon is Karmada.
On the screen, you can see Karmada's design that is taken from their website and some aspirations and features they have. They have couple of really interesting features: if you look at the karmada control plane they use their API server. This two tier approach is different than OCM in that Karmada has another API server instead of using K8s API server implementation. You also need to install another K8s control plane in the clusters that join Karmada which is the karmada agent.
You can also observe that Karmada Workload controllers and scheduler talks to Karmada API-server and not to K8S native API servers.
If you want to know more about this implementation, check out the link of the screen: https://karmada.io/docs/

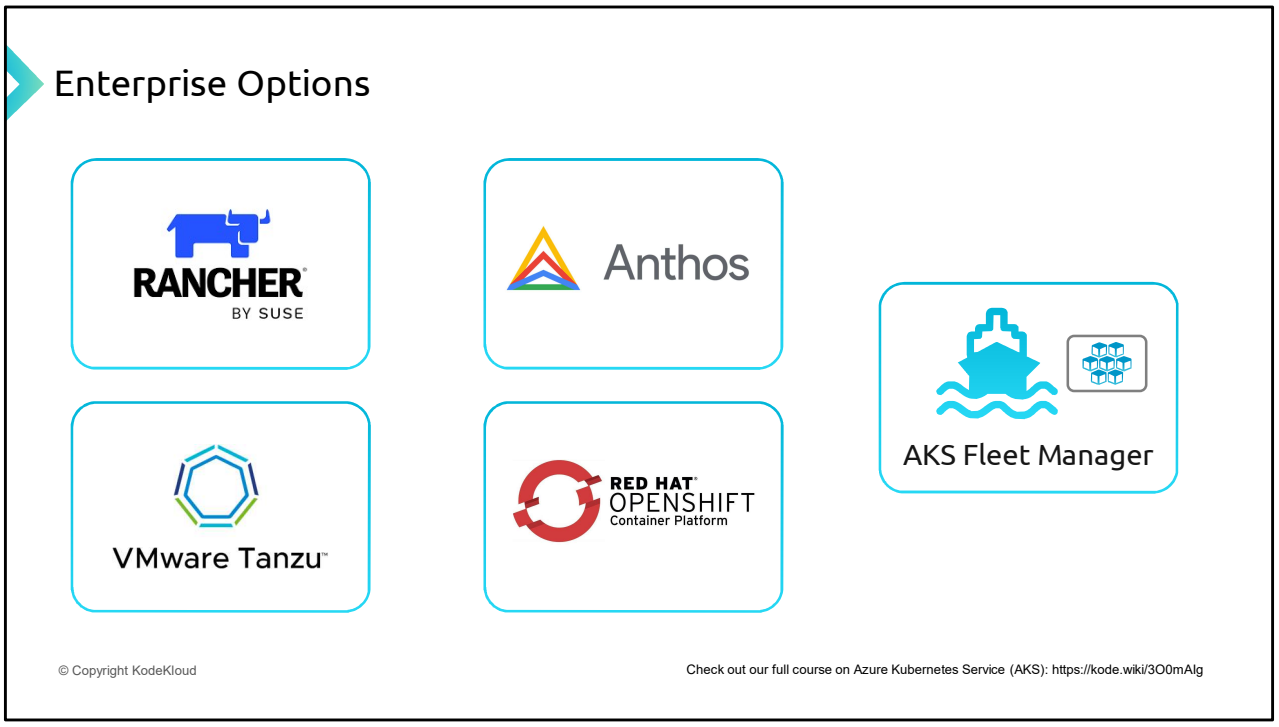


The next opensource solution for multi cluster management that we will touch upon is Karmada.

On the screen, you can see Karmada's design that is taken from their website and some aspirations and features they have. They have couple of really interesting features: if you look at the karmada control plane they use their API server. This two tier approach is different than OCM in that Karmada has another API server instead of using K8s API server implementation. You also need to install another K8s control plane in the clusters that join Karmada which is the karmada agent.

You can also observe that Karmada Workload controllers and scheduler talks to Karmada API- server and not to K8S native API servers.

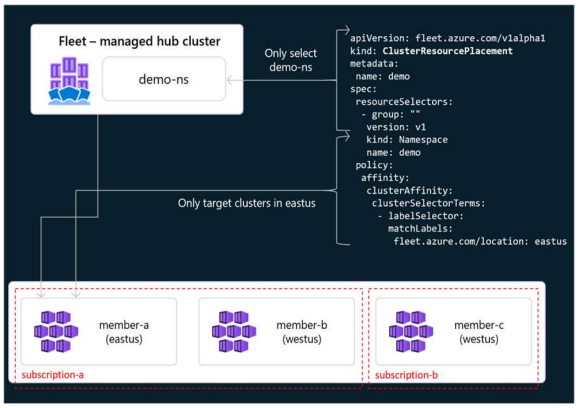
If you want to know more about this implementation, check out the link of the screen: https://karmada.io/docs/



When it comes to enterprises, there are few of notable Multi-cluster management tools that you will come across and can see on your screen. Azure AKS Fleet manager is a new entrant and

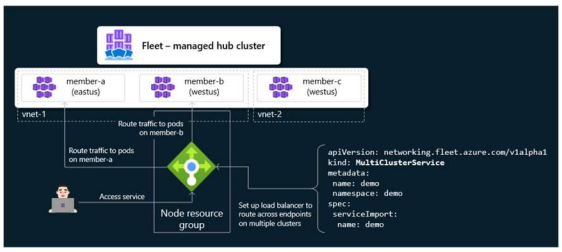
AKS Fleet Manager

Centralised Management



© Copyright KodeKloud

North-south load balancing



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAlg

Azure Kubernetes Fleet Manager (Fleet) is a managed service that helps you centrally manage multiple Kubernetes clusters at scale. Fleet provides a single pane of glass for managing your clusters, and it can automate tasks such as cluster provisioning, upgrades, and configuration management.

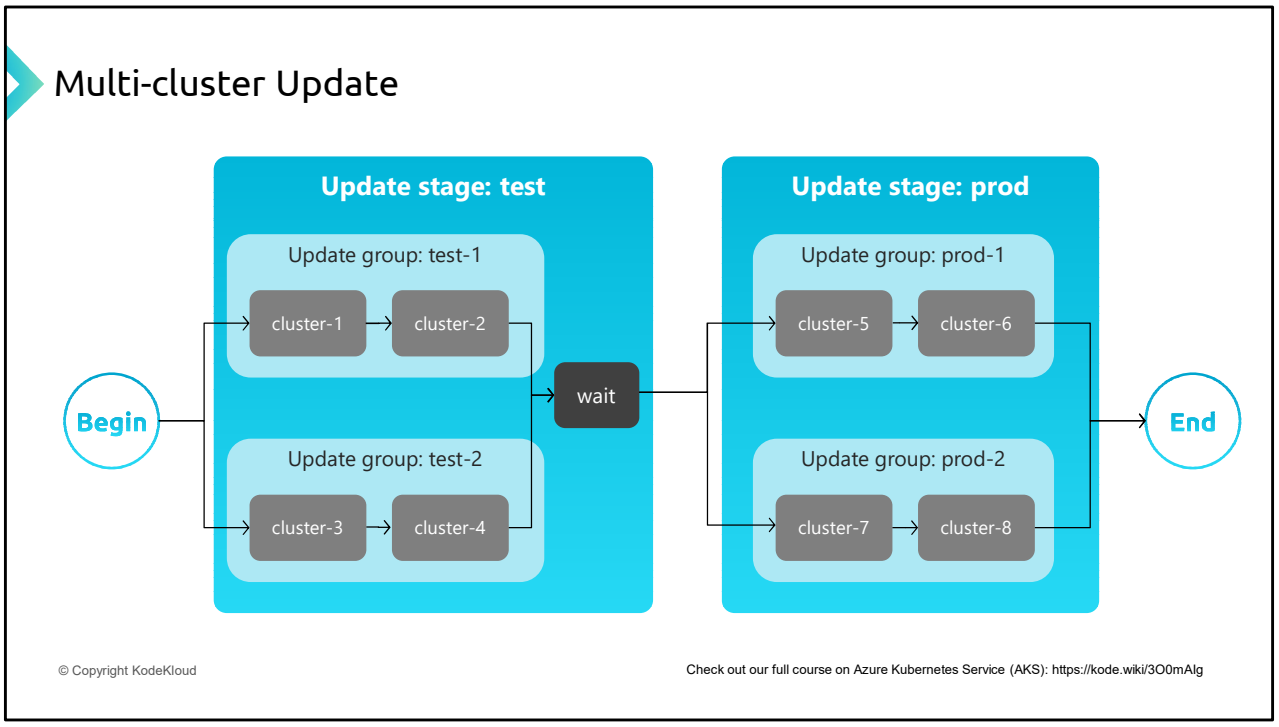
There are 4 key features of Azure Kubernetes Fleet Manager:

•Centralized management: Fleet provides a single pane of glass for managing your Kubernetes clusters. You can view all of your clusters in one place, and you can perform operations on them in bulk.

•Automated tasks: Fleet can automate tasks such as cluster provisioning, upgrades, and configuration management. This can help you reduce operational overhead and improve the reliability of your Kubernetes deployments.

•Policy-based management: Fleet supports policy-based management. This means that you can define policies that govern how your Kubernetes clusters are managed. For example, you can define policies that specify the minimum version of Kubernetes that your clusters must run, or the maximum number of pods that a cluster can have.

•North-south load balancing: Fleet can provide north-south load balancing for your Kubernetes clusters. This means that you can distribute traffic across multiple clusters, which can improve the performance and availability of your applications.



You can also use Kubernetes Fleet Manager to orchestrate multi-cluster updates. This means that you can update your clusters in a planned manner across multiple clusters and multiple environments to achieve a consistent environment across the set of clusters. This can also help test the updates in lower environments before deploying to production. This can be fully automated using Update groups.

Summary

- [x] Fundamental Concepts and Benefits
- [x] How to deploy AKS Cluster
- [x] Deploying application to AKS Cluster
- [x] Scaling
- [x] Image Management and Pushing Image to ACR
- [x] Upgrading Application
- [x] Azure Kubernetes Fleet

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

KodeKloud

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

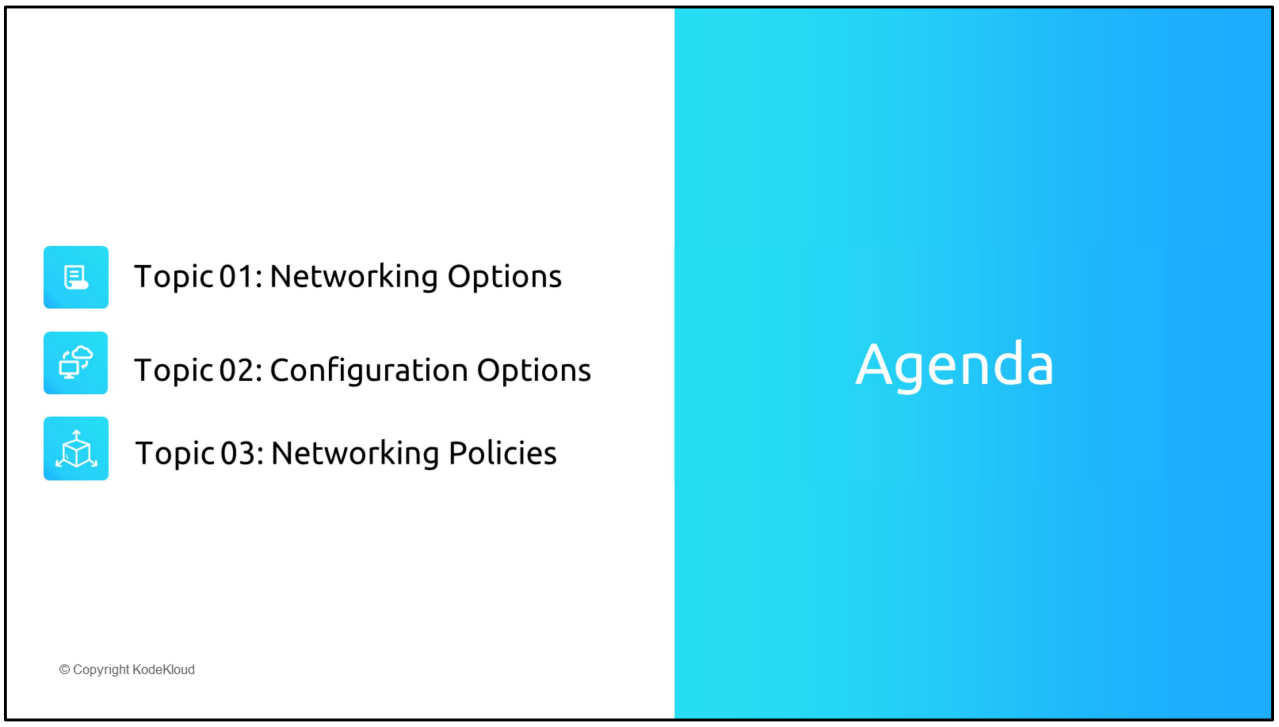
In this networking module, we will cover Networking options in AKS including configuration options like Kubenet, Azure CNI and networking policies. When we deployed AKS cluster in the previous module, we did not change the default networking options. It is now time to look a little deeper on available options and how you can use them to design your deployments.

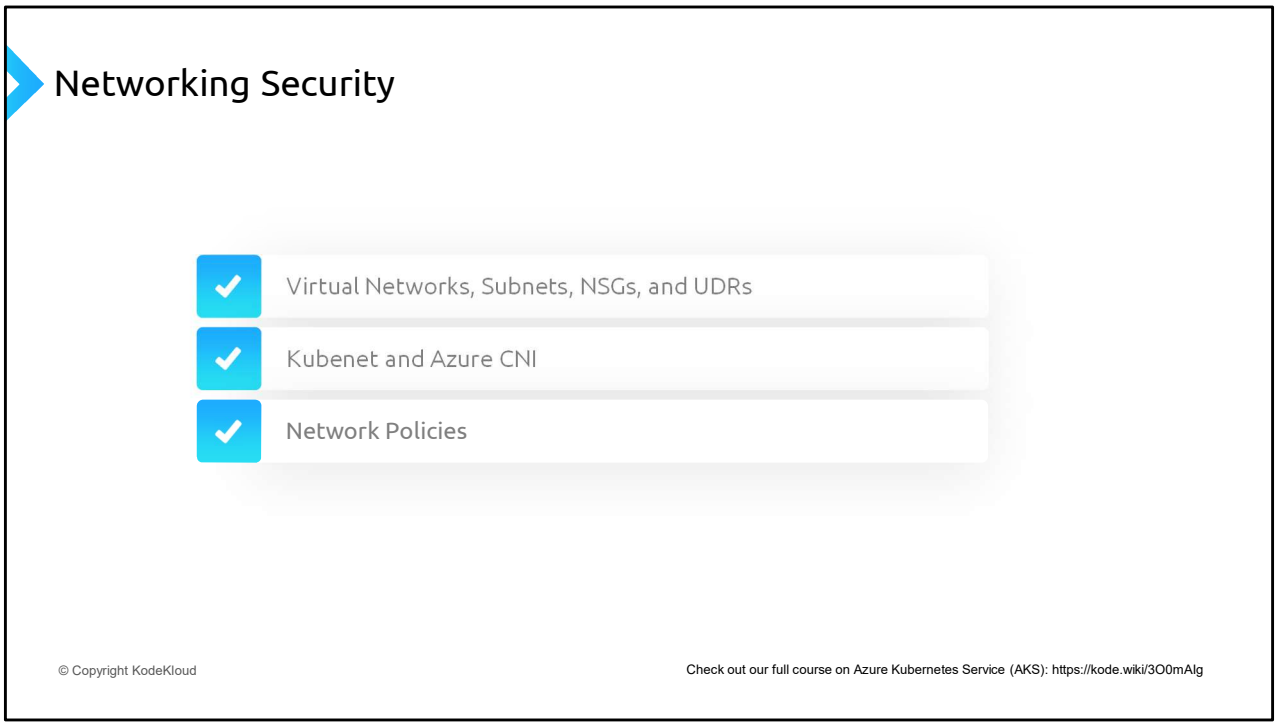
There are three videos on this topic, in the first video I'll talk about some basics like virtual network, Subnets, NSGs and UDRs

I'll follow that up with details around Kubenet and

CNI.

The final video is around Network policies.. Lets get started


Topic 01: Networking Options
Topic 02: Configuration Options
Topic 03: Networking Policies
© Copyright KodeKloud
Agenda



Ok, we have now reached the culmination of this video course on Azure Kubernetes Service (AKS). Our journey began by delving into the fundamental aspects of Azure, encompassing essential concepts such as Compute, Storage, and Networking. With a some foundation in place, we transitioned towards the creation and containerization of a basic C# web application. Throughout that module, we focused on leveraging Docker desktop and crafting Dockerfiles to facilitate containerization.

Subsequently, we navigated to the Azure portal to establish an AKS cluster to deploy our world class application onto that cluster. Harnessing the power of kubectl, we effectively executed various imperative actions, including scaling of deployments. we explored the deployment of container images within the Azure Container Registry (ACR), unearthing its potential for application enhancement and version management. By harnessing the image hosted on ACR, we facilitated a seamless application upgrade, only to effortlessly revert to a previous version when needed. Concluding this module, we bestowed upon you a overview of the Kubernetes Fleet, empowering you with a holistic understanding of this powerful toolset.

The subsequent module, entirely dedicated to AKS networking options and policies,

showcased the indispensable importance of robust network management within AKS deployments. Furthermore, we delved into security-centric topics, delving into the intricacies of Azure AD integration, defender for AKS, and Azure policy for AKS. Empowered with this knowledge, you are poised to fortify your AKS infrastructure with optimal security measures and governance practices.

As we progressed, we ventured into the realm of CI/CD patterns within AKS, looking at the art of managing AKS using declarative techniques. By adopting these practices, you can streamline the continuous integration and delivery pipelines within your AKS environment.

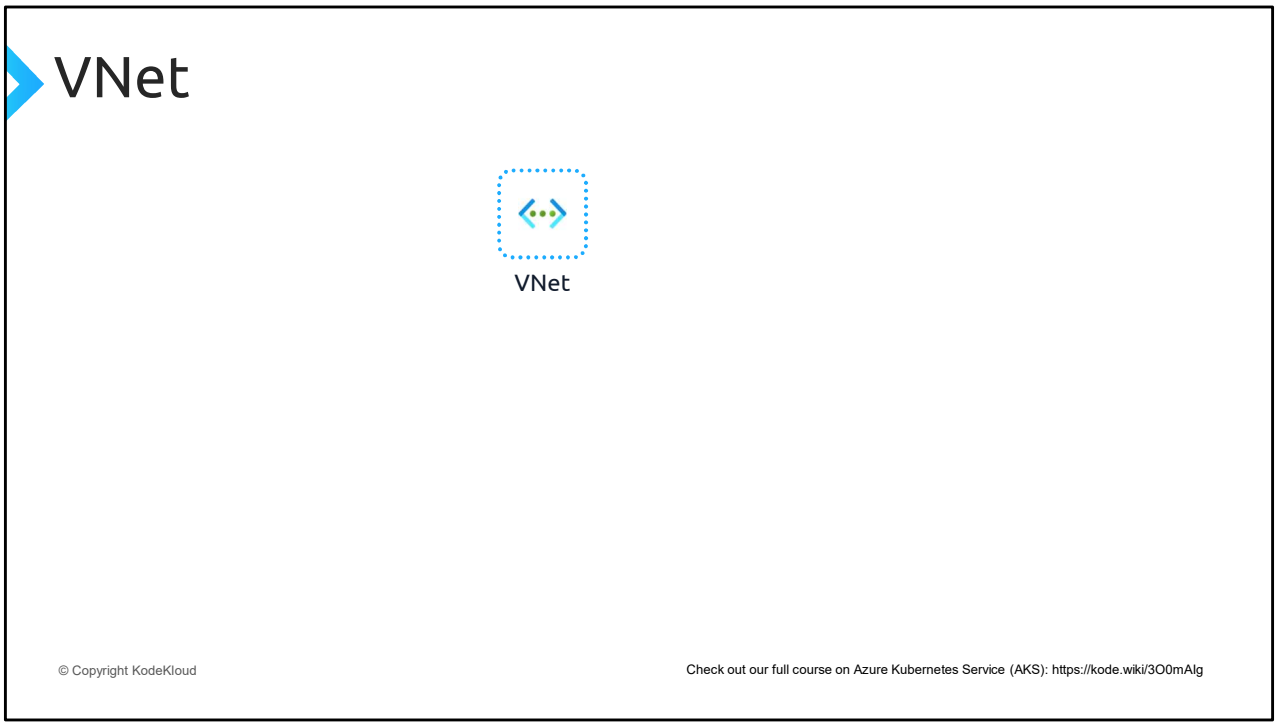
An indispensable component of any infrastructure, observability received some attention in module seven. We explored observability within AKS, equipping you with the necessary tools and techniques to monitor and gain valuable insights into your AKS deployments.

Finally, before we brought this course to a close, we presented you with an overview of alternative platforms that embrace the power of containers. By expanding your horizons and exploring these platforms, you can unlock a world of possibilities and further enhance your containerization endeavors.

We sincerely hope that your journey throughout this course has been as fulfilling as our commitment to its production. It has been an absolute pleasure to guide you through the intricacies of AKS, empowering you with the knowledge and skills needed to excel in your containerization journey. This is your host Pranav, signing off for now.

Azure Networking Fundamentals

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

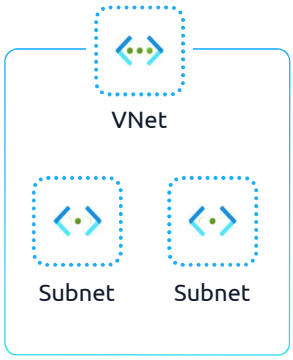


Ok, before we go any further, let us familiarize ourselves with some Azure networking nomenclature.

A Virtual network or vnet is an Azure construct which defines one or more private IP Address prefixes that you intent to use in your environment. For example, (Subnet screen), I can define a vnet CIDR for vnet1 as 10.2.0.0/16, which can allow me to use about 65534 IP addresses. CIDR, by the way, stands for Classless Inter-Domain Routing. You can use this or many IP address management portals to plan your IP ranges. In simple terms, we define Network address and Mask in this format to define what range of Ips we can use.

Azure Vnets support both IPv4 and IPv6 address spaces, for simplicity we will use IPv4 here.

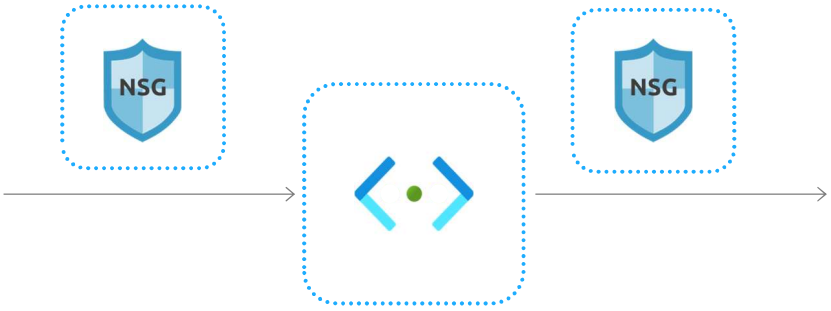
Subnet



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Before you can allocate any of the IP addresses from a vnet to a container, VM or any other resource, you need to define atleast one subnet, again in CIDR format. Let us define two. They offcourse, need to be a subset of what we defined in the vnet and cannot overlap.

Network Security Group (NSG)

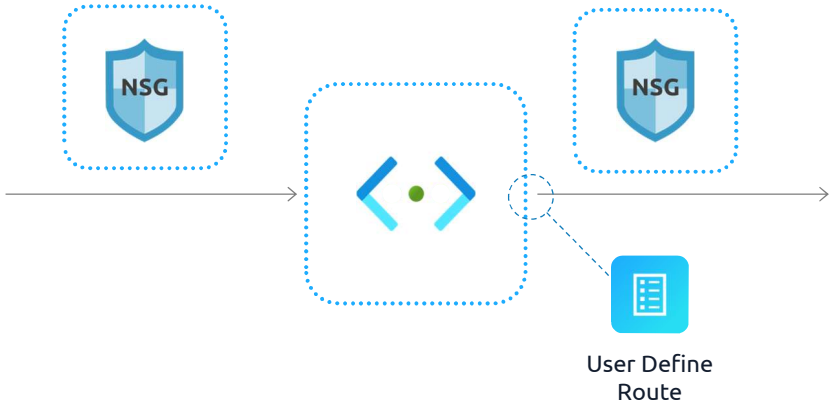


Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Next constructs to understand are Network Security group or NSG and User Defined route or UDR.

NSGs are used to filter network traffic between Azure resources that traverse a subnet. A NSG contains security rules that allow or deny inbound or outbound network traffic. For each rule, you can specify source and destination, port, and protocol. Think of this as a lightweight firewall.

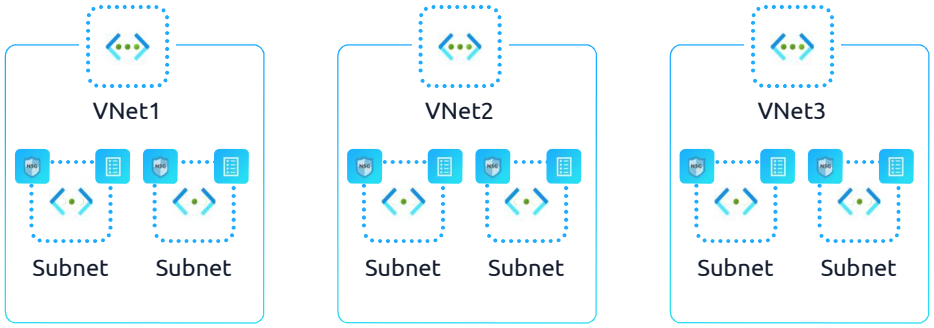
User Define Route (UDR)



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

UDR defines how traffic initiated in that subnet is routed. If your pod needs to talk to another pod hosted in a node in another subnet or vnet, azure needs to know where to move that traffic. A route table is a collection of individual routes used to decide where to forward packets based on the destination IP address. There are three ways to populate route tables: User defined routes, that you see in the screen, BGP routes based on data received via Border gateway protocol and System routes.

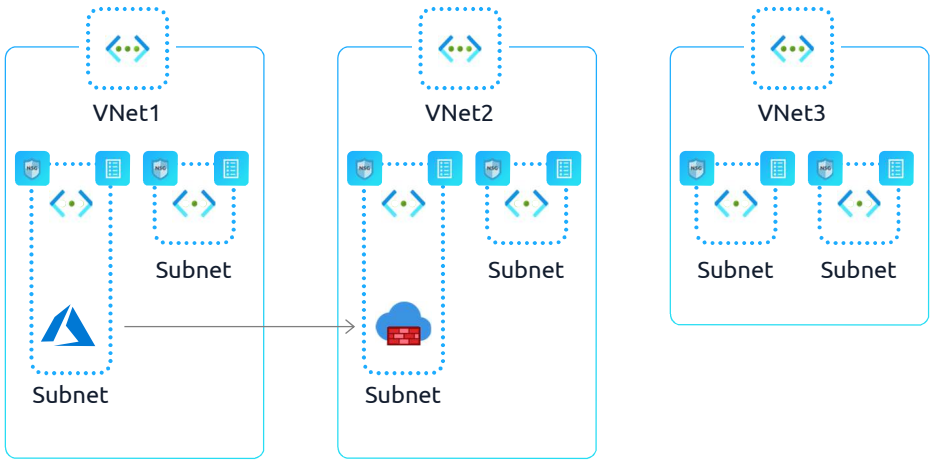
Subnet



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Once you define your NSGs and UDRs, both of which are optional, you start deploying resources in you subnet.

Subnet



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

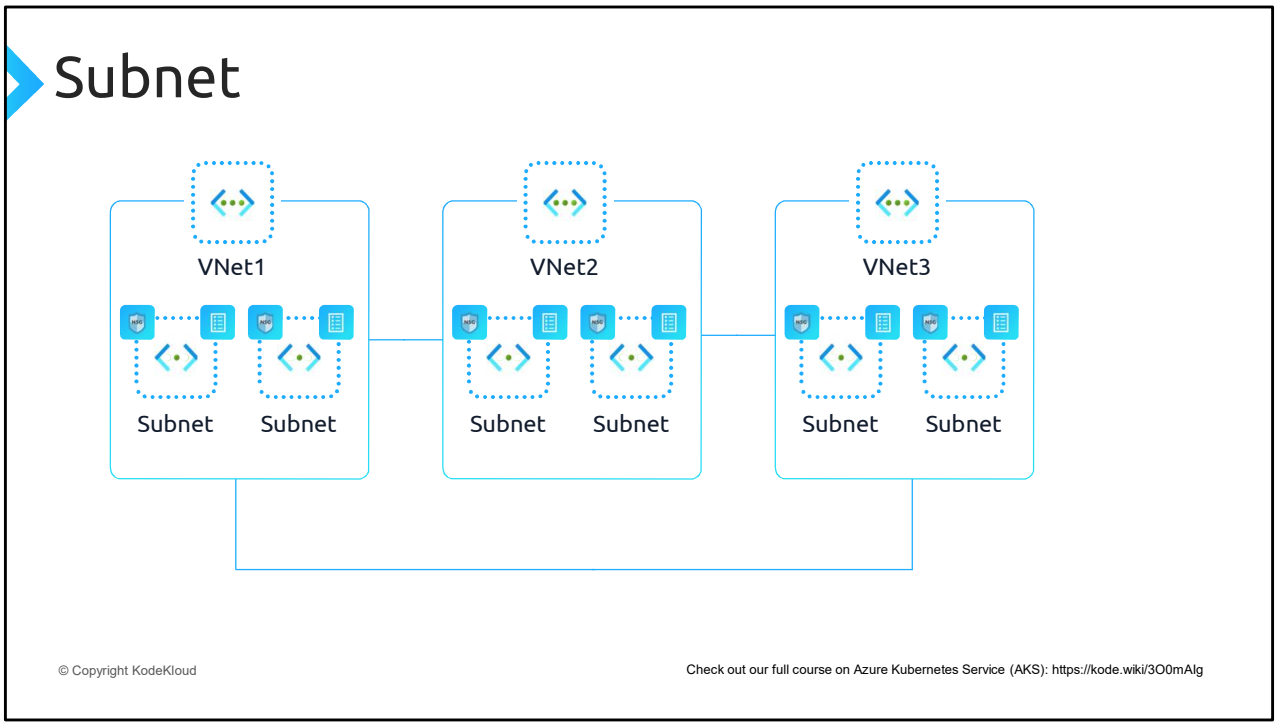
And as you deploy more services, you will have multiple vnets and a defined method to route the traffic. For example, you might decide to push all traffic from your frontend subnet in vnet1 to a firewall in vnet 3 before it goes anywhere else.

If you have more than one vnet, you will need to peer them if you need resources in one vnet to talk to resources in another vnet.

Now that you know these Azure networking concepts lets jump to AKS networking.

AKS Networking Options

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg



There are two networking options to chose from when you deploy K8s – Kubnet and Azure CNI. these used to be called basic and advanced options respectively.

Depending on the option you choose, that plugin gets deployed on every VM of the AKS cluster providing networking for the cluster and to the nodes and pods inside them.

With *kubenet*, nodes get an IP address from the Azure subnet defined by you while Pods receive an IP address from a logically different address space to the subnet of the nodes. Network address translation (NAT) is then configured so that the pods can reach resources on the Azure virtual network.

By default, AKS assigns a /24 address space to each VM, you can optionally specify what that address pool to use.

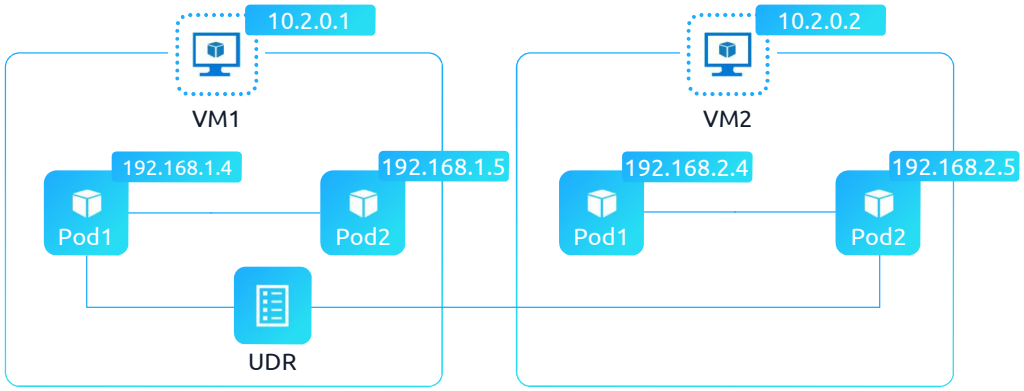
Pods on the same VM can communicate with each other via a bridge. For Pods to communicate across VMs, aks uses UDR to route traffic. You know what UDRs are already, so if Pod 1 from VM1 need to talk to Pod1 from VM2, there will be a UDR on

subnet1 that will force traffic to VM2 so it can reach the destination pod.

If your Pod needs to communicate to another resource outside the aks cluster, AKS uses SNAT i.e Source Network Address Translation.

Lets say our first pod with IP 192.168.1.4 needs to talk to a VM on another Vnet with IP 10.10.1.1, AKS will use the VMs IP where the pod is hosted to SNAT the traffic to the destination VM. Since SNAT is being used the destination will use Source VM IP and pre designated port combination to respond back to any requests.

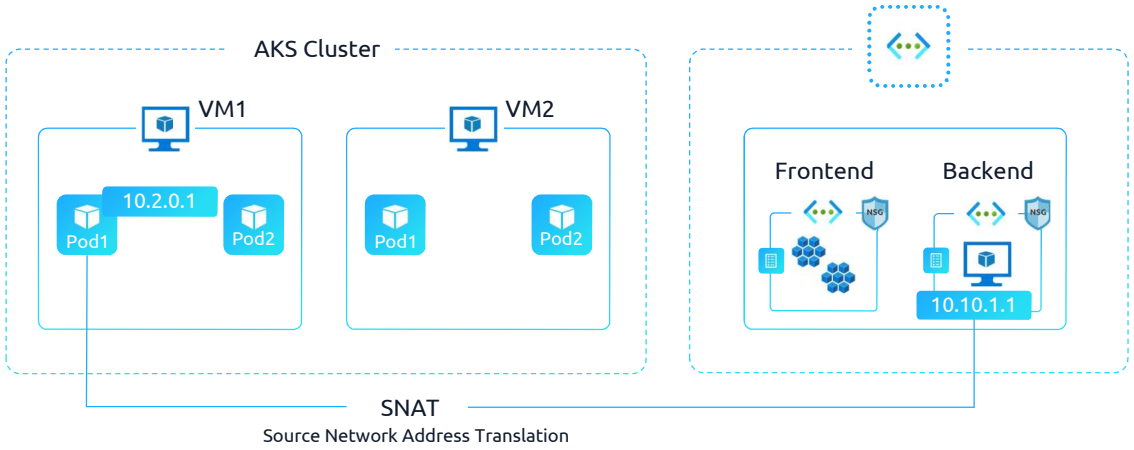
Networking Options for AKS - KubeNet



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Pods on the same VM can communicate with each other via a bridge. For Pods to communicate across VMs, aks uses UDR to route traffic. You know what UDRs are already, so if Pod 1 from VM1 need to talk to Pod1 from VM2, there will be a UDR on subnet1 that will force traffic to VM2 so it can reach the destination pod.

Networking Options for AKS - KubeNet

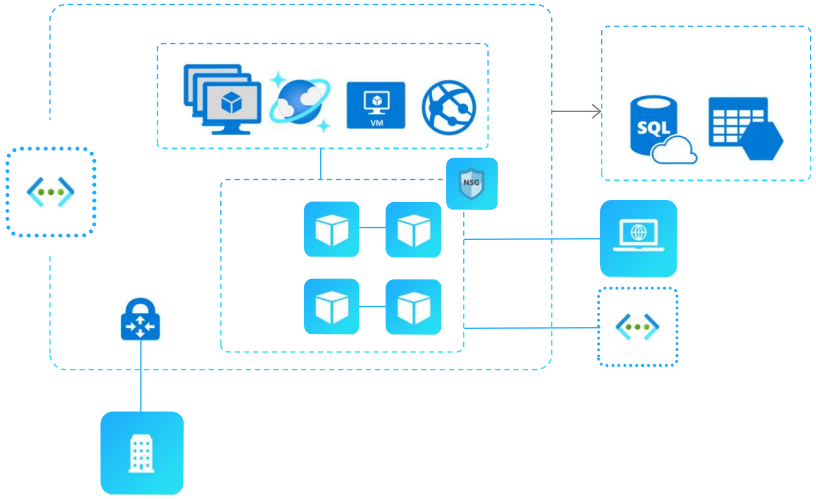


Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAlg

If your Pod needs to communicate to another resource outside the aks cluster, AKS uses SNAT i.e Source Network Address Translation.

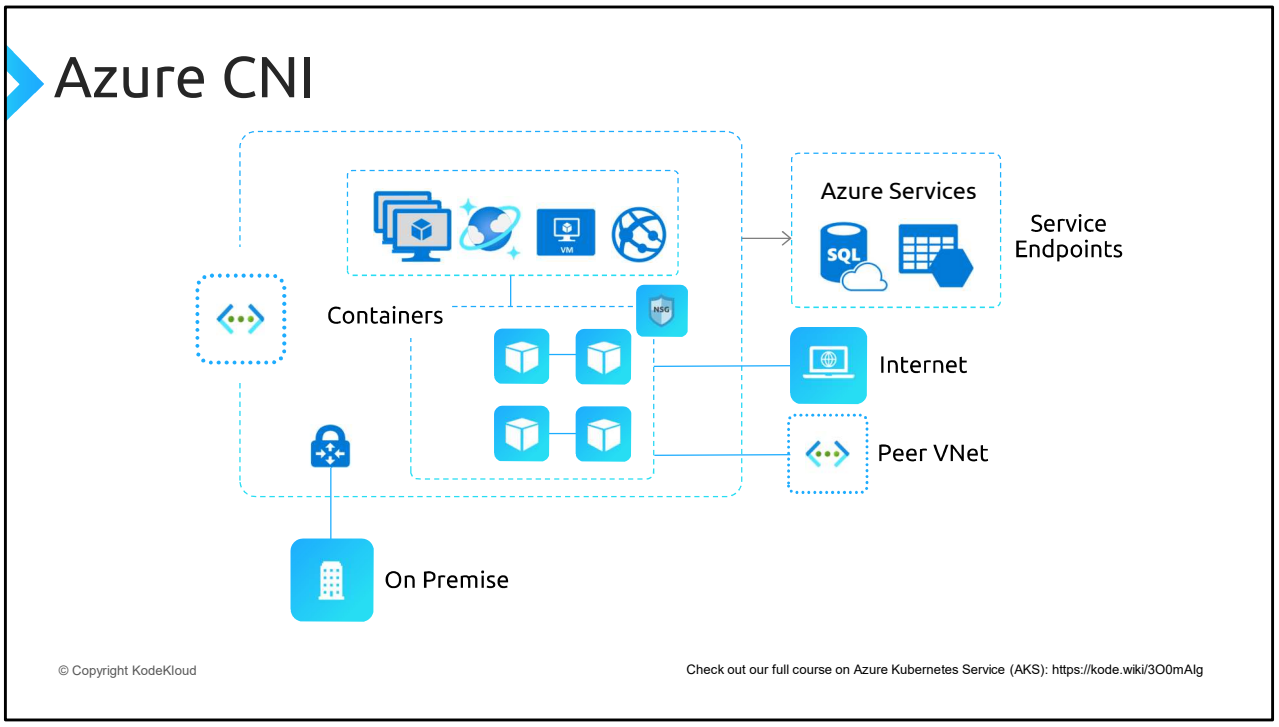
Lets say our first pod with IP 192.168.1.4 needs to talk to a VM on another Vnet with IP 10.10.1.1, AKS will use the VMs IP where the pod is hosted to SNAT the traffic to the destination VM. Since SNAT is being used the destination will use Source VM IP and pre designated port combination to respond back to any requests.

Azure CNI



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

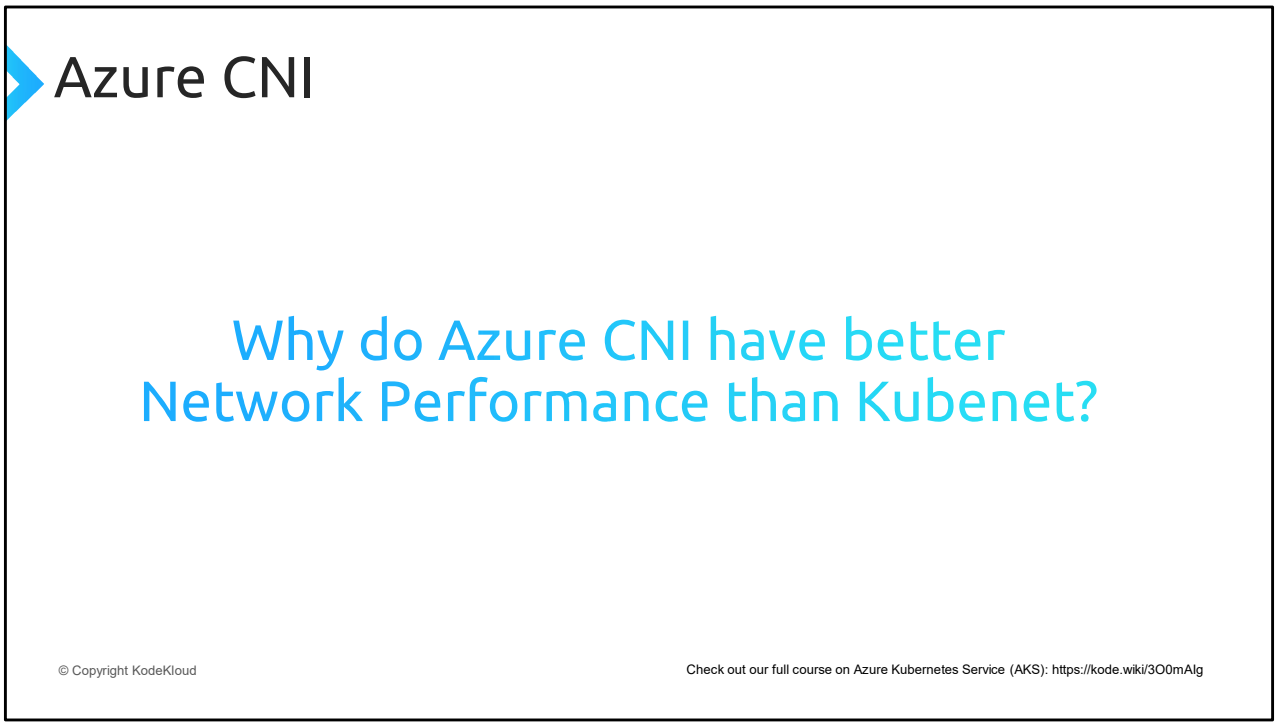
As I mentioned before if you choose Azure CNI, the CNI plugin gets deployed on every VM and unlike Kubenet, this can assign an IP from your vnet to your pods.
When you deploy a cluster with Azure CNI, you specify which of your vnet and subnet you want to deploy the cluster in. When you do that, not only the VMs in the cluster get IPs from your subnet, the pods inside the cluster also get Ips from subnet controlled by you.
This makes the pod, directly routable from any other system, be it another AKS pod or a VM or in fact from your on-premise system. This provides seamless and directly connectivity to your pods. And this also means that your pod can also talk to any other service on azure and on-premise as along as firewalls allow that to happen..



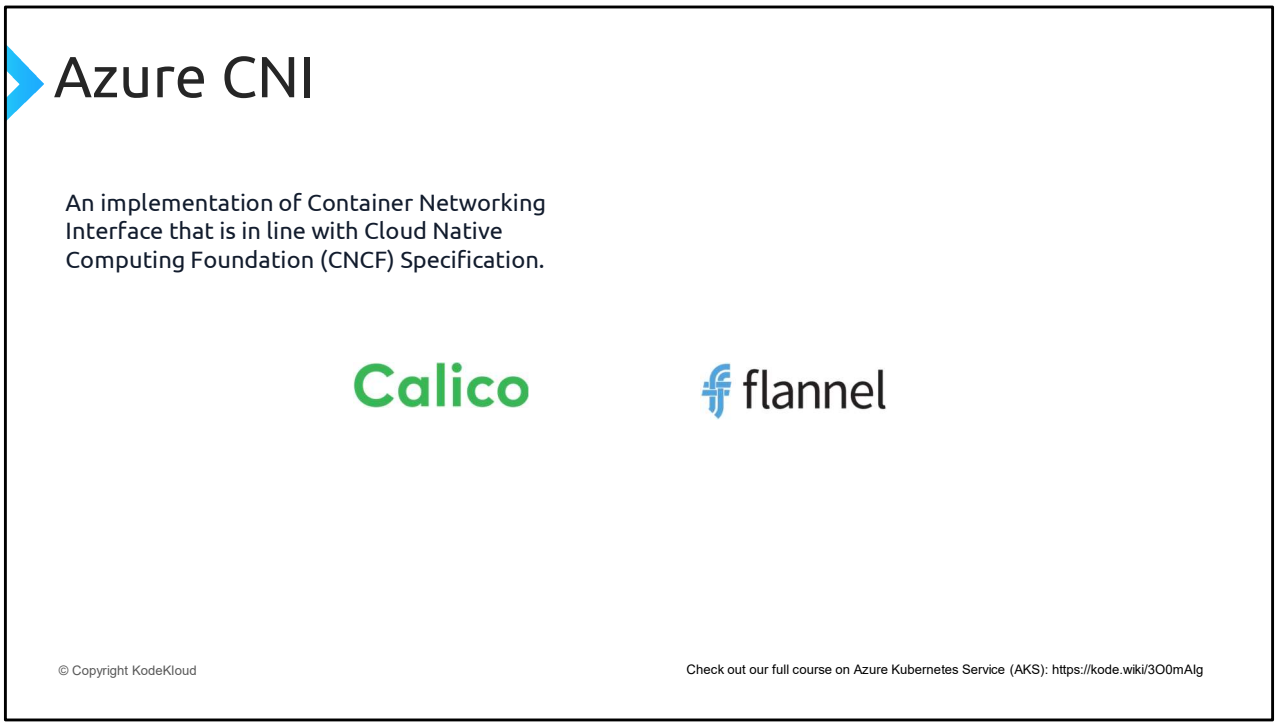
NSGs and UDRs that you would have defined for your subnets would automatically apply to the pods in your cluster. In the next video, we will talk about network policies, I'l have more to talk about NSGs and UDRs when it comes to aks pods.

Kubernetes also have lot of networking capabilities that it offers natively like ingress controllers, DNS etc so all these will also work seamlessly when you use it via CNI plugin.

Can you think of a reason why Azure CNI wil have much better network performance than Kubenet ? I'll answer this question at the end of this video.



Can you think of a reason why Azure CNI wil have much better network performance than Kubenet ? I' ll answer this question at the end of this video.



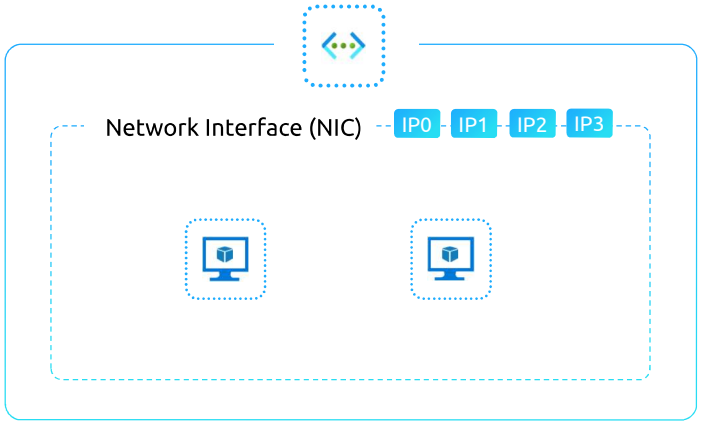
Lets take a deeper look at what CNI is doing. CNI is an implementation of container networking interface that inline with CNCF specification. There are many vendors which have deployed a flavor of this specification. like Calico, falnel etc.

CNI Modules

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Any CNI have two modules to it – one is networking that is when a container is initiated it get a NIC card assigned to it so it has connectivity to talk to other containers and resources. The other function is IP Address management or IPAM. After a NIC card is assigned, it needs an IP address from your subnet. The function to allocated an IP from a pool and deallocating it when the container is terminated is done by this IPAM module.

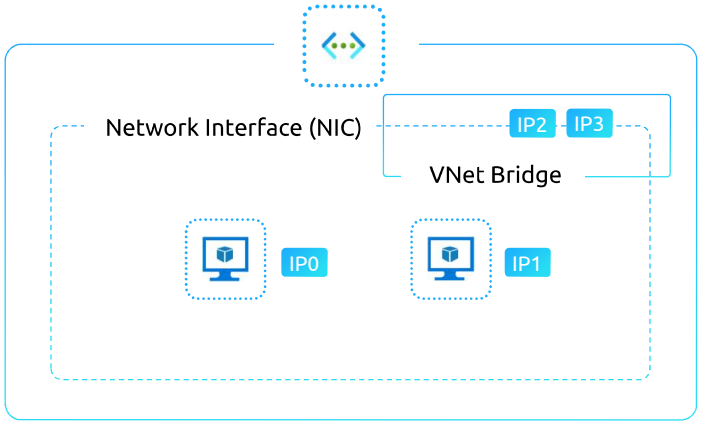
IP Management



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

The way the IP address management works in Azure CNI is when you provision your cluster, Azure assigns multiple set of Ips to the NIC. Let us say i have one cluster with 2 VMs and each of those VMs have a single NIC card. Every NIC has a primary IP, which is the address of the VM. Apart from that IP, Azure adds another set secondary of IPs to that NIC. The number of Ips assigned is the same that you would have configured as max pods that you want to deploy per VM. When you are deploying a cluster via CLI, you have the option to specify how many maximum pods you want to deploy. Based on that, number of secondary Ips are assigned to the NIC.

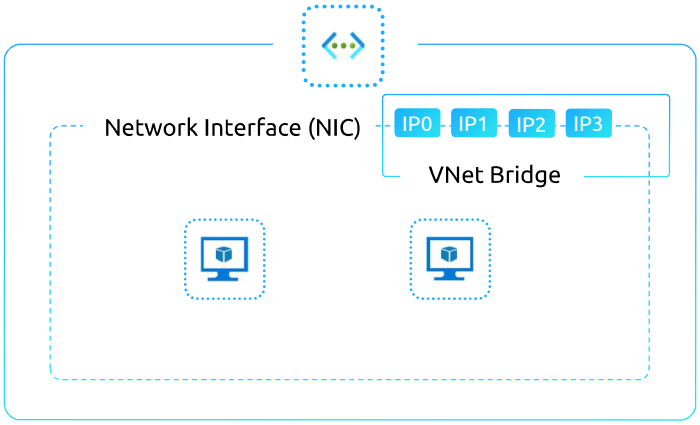
IP Management



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

As your containers get deployed, the CNI on that node will draw an IP from that pool of Ips and assign it to the container. When the container terminates CNI puts it back to the pool and available Ips. Vnet Bridge is used by CNI to manage these interactions.

IP Management



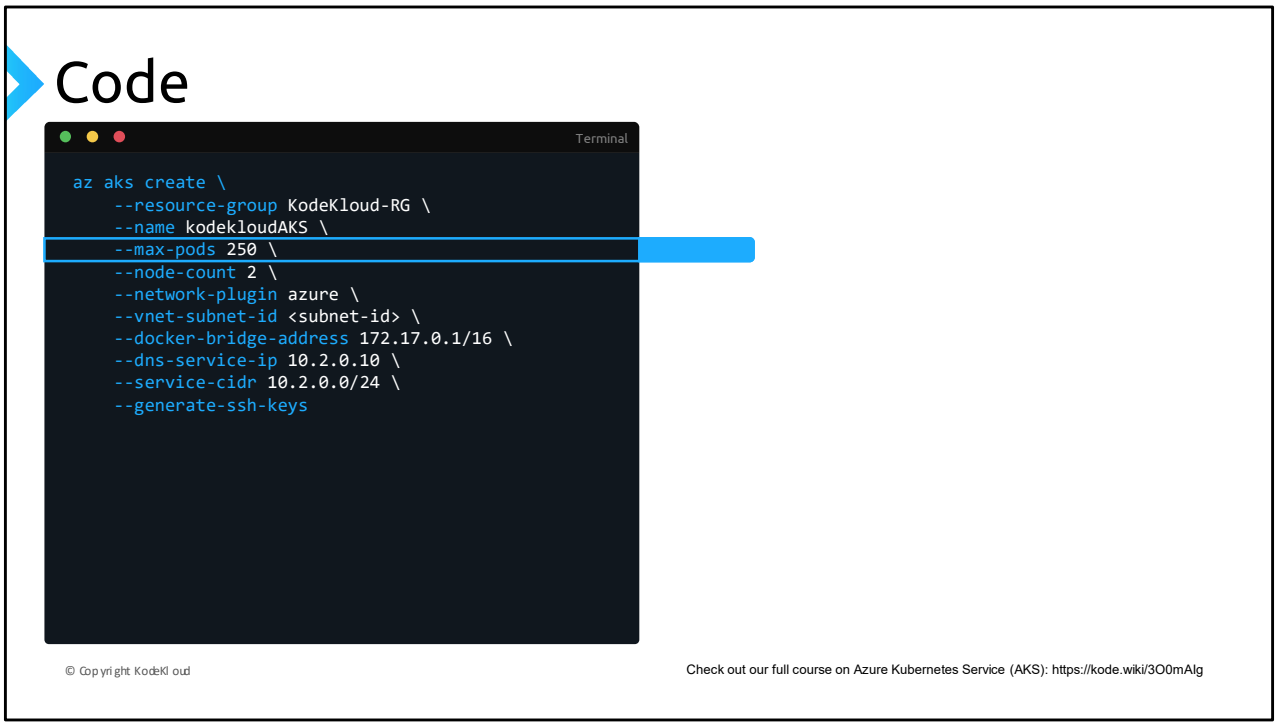
Azure Supports:

250 Containers / pod / VM

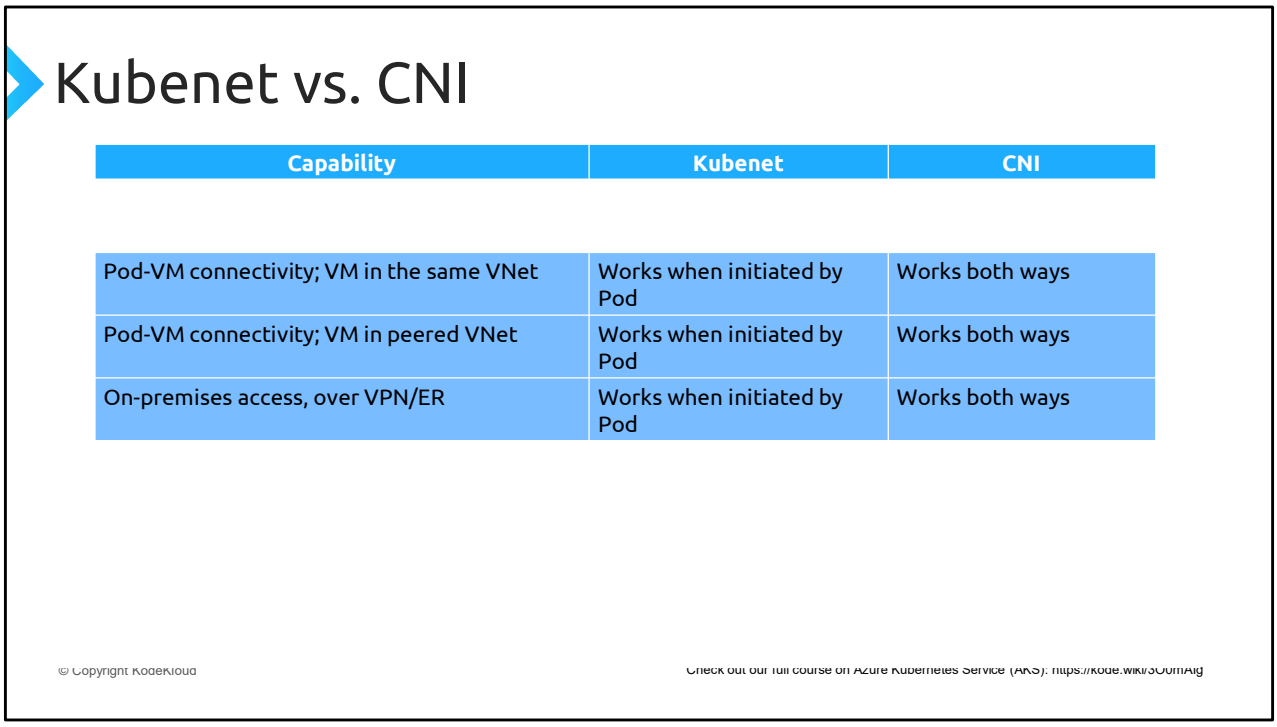
64,000 Ips / VNet

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

At the time of this recording, Azure support 250 containers or Pods per Vm and 64K IP per vnet i.e a /16 range.



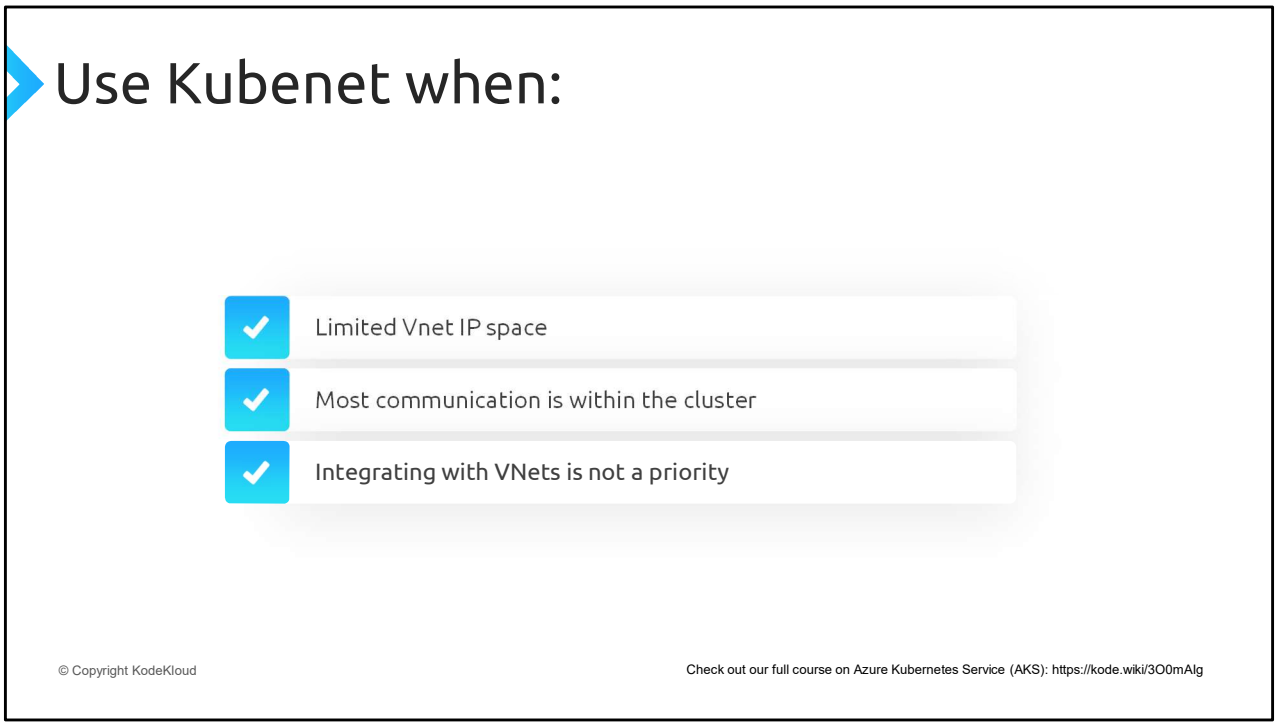
We can use labels to select the type of pods and then there is a separate section in the yaml file that lets you specify what the policies are which are essentially filtering rules that are defined based on labels on names or namespaces or IPBlocks as shown in this screen.



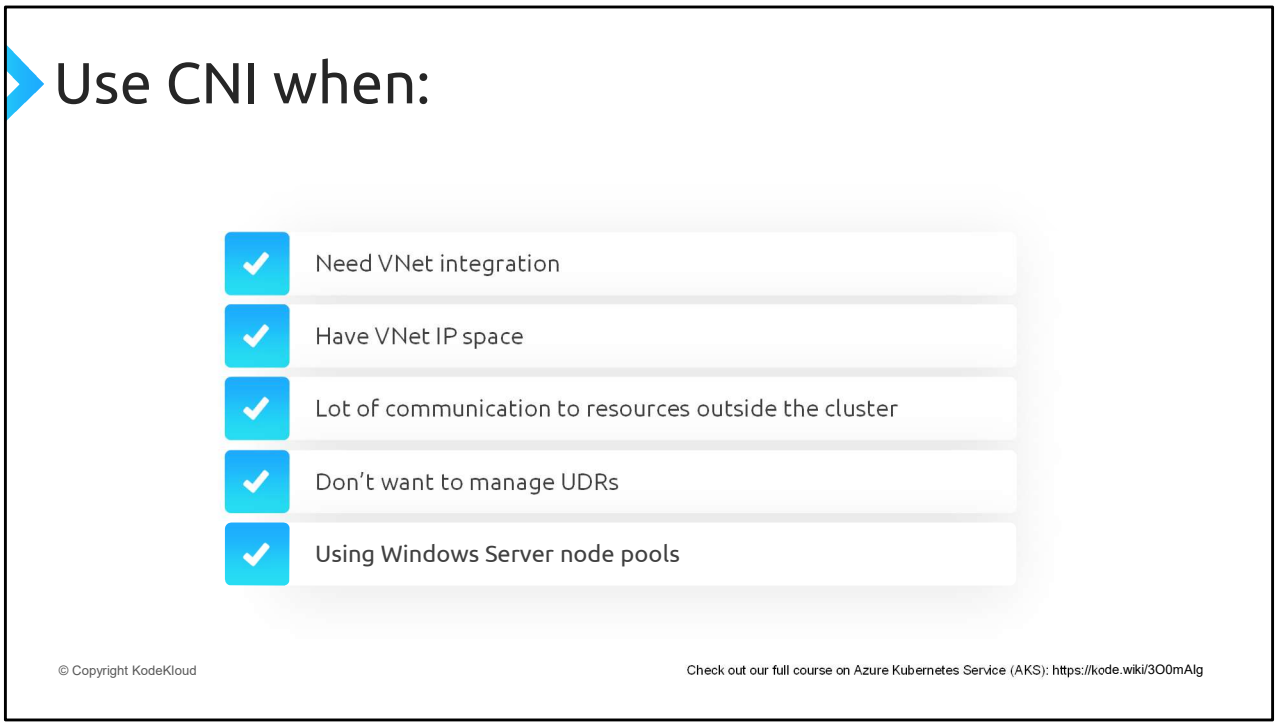
Just to contrast both the options, while CNI is giving you direct connectivity to your vnet, it can also potentially use up lot of Ips. I have seen organization where this can be a challenge especially if you intent to deploy lots and lots of containers. Kubenet on the other hand use very few of your Ips at the cost of not giving you direct connectivity. As highlighted on this screen, if you are using kubenet, there is no native way for a resource on another vnet to access your pod directly they can only respond back if Kubenet based pod initiates a connection.

There are couple of other differences in that Kubenet does not support Virtual endpoints that is bursting nodes on Azure Container instances ACIs and you cannot have multiple clusters sharing the same subnet when using Kubenet.

If you use Kubenet you will not be able to use Azure network policies. We will discuss network policies in the next video.



In summary, use Kubenet when you have limited VNET space and when resources outside the cluster do not need to initiate traffic to the pods running on Kubenet. Also recall Kubenet uses SNAT to communicate out of the cluster. This can have a performance overhead if the quantum of traffic going out of the cluster is large. And hence Kubenet is a slower option of the two.



If you have enough Ips to current and future AKS clusters, its best to use CNI. Also if you happen to use windows server node pools, CNI is the only available option.

Network Policies

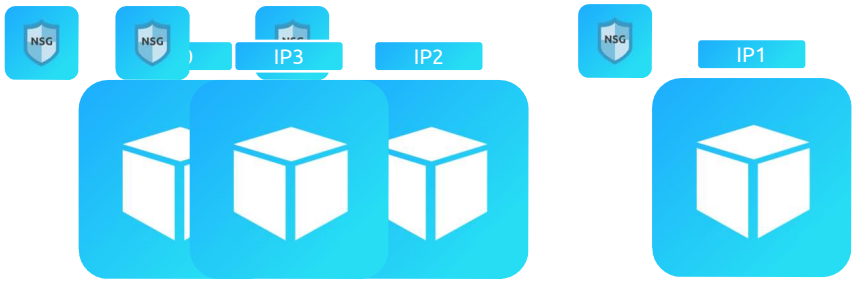
Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Why use Network Policies?

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

In the previous videos, we talked about network security groups or NSGs to filter traffic that are applied to a subnet so you can filter both incoming and outgoing traffic. You can offcourse use NSGs to secure your AKS pods when using CNI. But this the not a best solution, and the reason for that is Pods are dynamic and they would come and go on the fly so if you use an IP specific NSG the utility of that will expire the moment a pod is terminated and when a new one comes up its likely to have another IP.

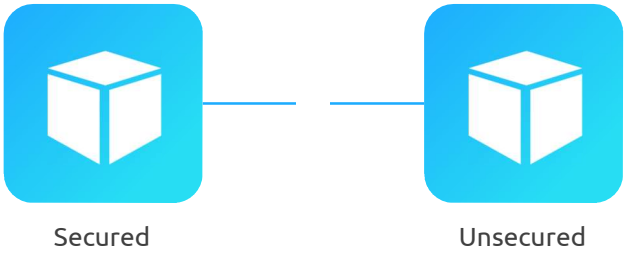
Why use Network Policies?



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

so if you use an IP specific NSG the utility of that will expire the moment a pod is terminated and when a new one comes up its likely to have another IP.

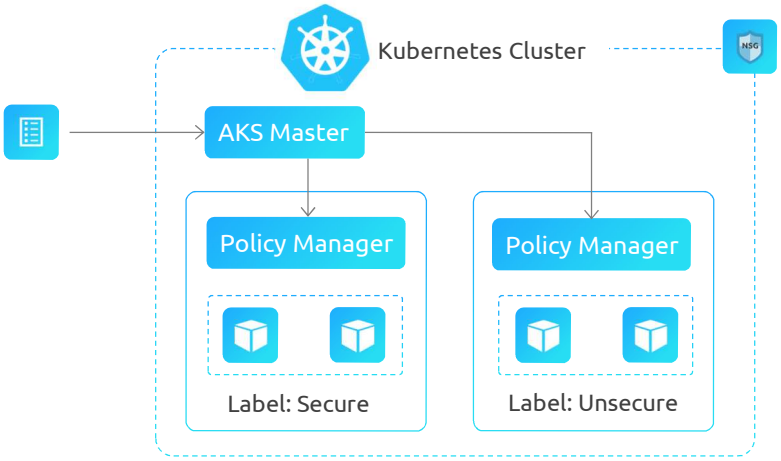
Why use Network Policies?



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

There will also be scenarios where you would like to define rules based on Kubernetes constructs like pod labels. For example, you might like to block traffic from pod labeled Secure to another pod labled unsecured. That is not something an NSG can address. As you can imagine, we need a fine-grained method to define these policies.

Kubernetes Network Policies



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

This is where Kubernetes network polices come in. This is a specification of Kubernetes that is used to filter traffic between pods and specification defines a YAML file and the YAML file has a format that lets us choose labels and help define policies around those labels.

Code

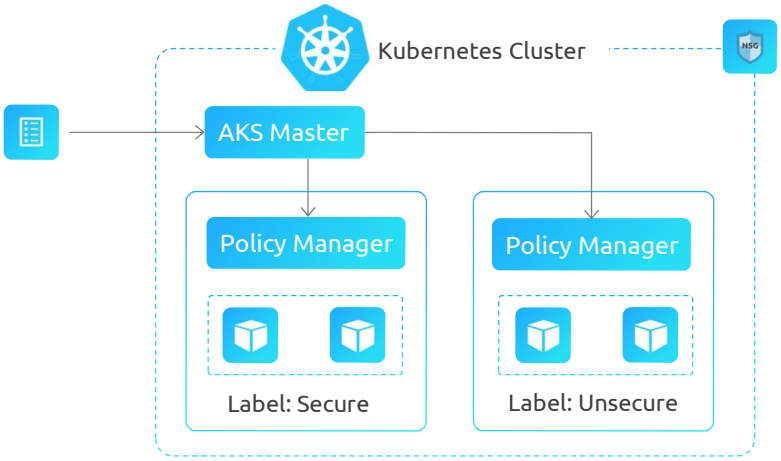
Terminal

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-policy
  namespace: demo
spec:
  podSelector:
    matchLabels:
      role: server
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
    egress:
    - to: 80
     - ipBlock:
       cidr: 10.2.0.0/22
```


Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

We can use labels to select the type of pods and then there is a separate section in the yaml file that lets you specify what your netwok policies are, which are essentially filtering rules that are defined based on either labels or names or namespaces. You can also use IPBlocks as is shown in this screen.

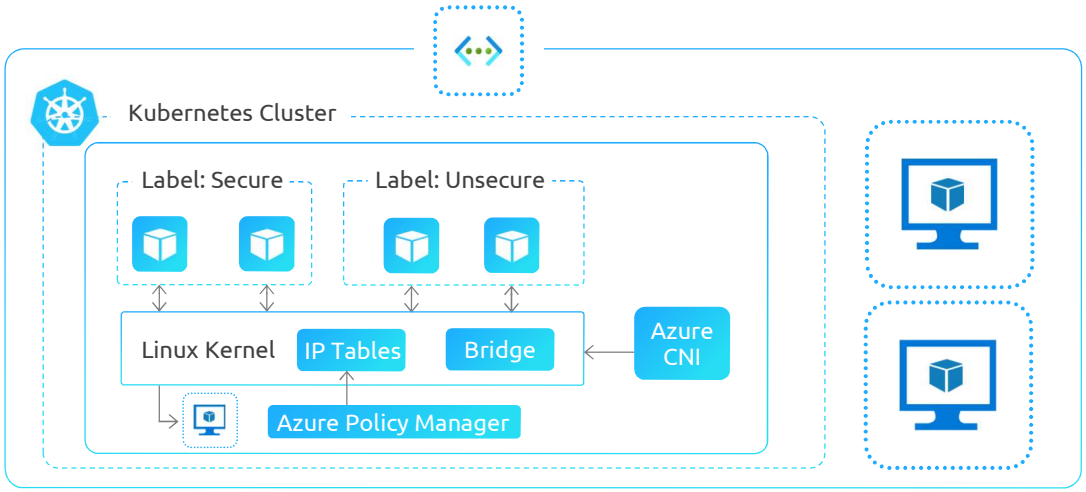
Azure Network Policies



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

So Azure has deployed their own version of Network policy specification that deploys a demonset which goes and sits on every node in the cluster and anytime you go and apply a yaml file through the k8s API the policy engine which is running on the vms can intercept the file and enforce the policies defined by you. The most common way to enforce these rules is via IPTable rules or other filtering capabilities that are part of the linux . If you are using windows-based container aks leverages *Host Network Service (HNS) supported ACLPolicies* which is part of windows kernel

Azure Network Policies



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

When you deploy the yaml file, Azure Network policy engine will translate the policy file into IP filtering rules provisioned into linux kernel in the form of IPTable rules. This policy engine works in collaboration with Azure CNI, which in turn depends on linux bridge functionality.

This is offcourse the default policy option in AKS

You also use this engine outside Azure if you want a consistent policy experience or if you decide to built K8S cluster in VMs on azure in DIY style.

Please note that Azure network policies in available both for linux and windows hosts but is in preview for windows. I am sure it will be generally available in due course.

Azure vs. Calico Policies

Capability	Azure	Calico
Supported platforms	Linux, Windows 2022	Linux, Windows 2019, 2022
Supported networking options	Azure CNI	Azure CNI (Windows & Linux), KubeNet (Linux)
Compliance with Kubernetes specification	All policy types supported	All policy types supported
Additional features	None	Extended policy model consisting of Global Network Policy, Global Network Set, and Host Endpoint.
Support	Supported by Azure support and Engineering team	Calico community support. For more information on additional paid support, see Project Calico support options.
Logging	Logs available with kubectl log -n kube-system <network-policy-pod> command	Information on component logging available at Calico component logs page.

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Calico network policy is another option available as first party that you can deploy on AKS clusters. Calico an open-source network and network security solution founded by Tigera. You can see their website for implementation details.

A quick comparison for both Azure Network policy manager and Calico network policy are listed on your screen. If you are planning to use Kubenet then Calico would be your goto option. Also note that if you choose Calico, Microsoft support engineers will not help troubleshoot if they determine that Calico is a potential cause of any incidents.

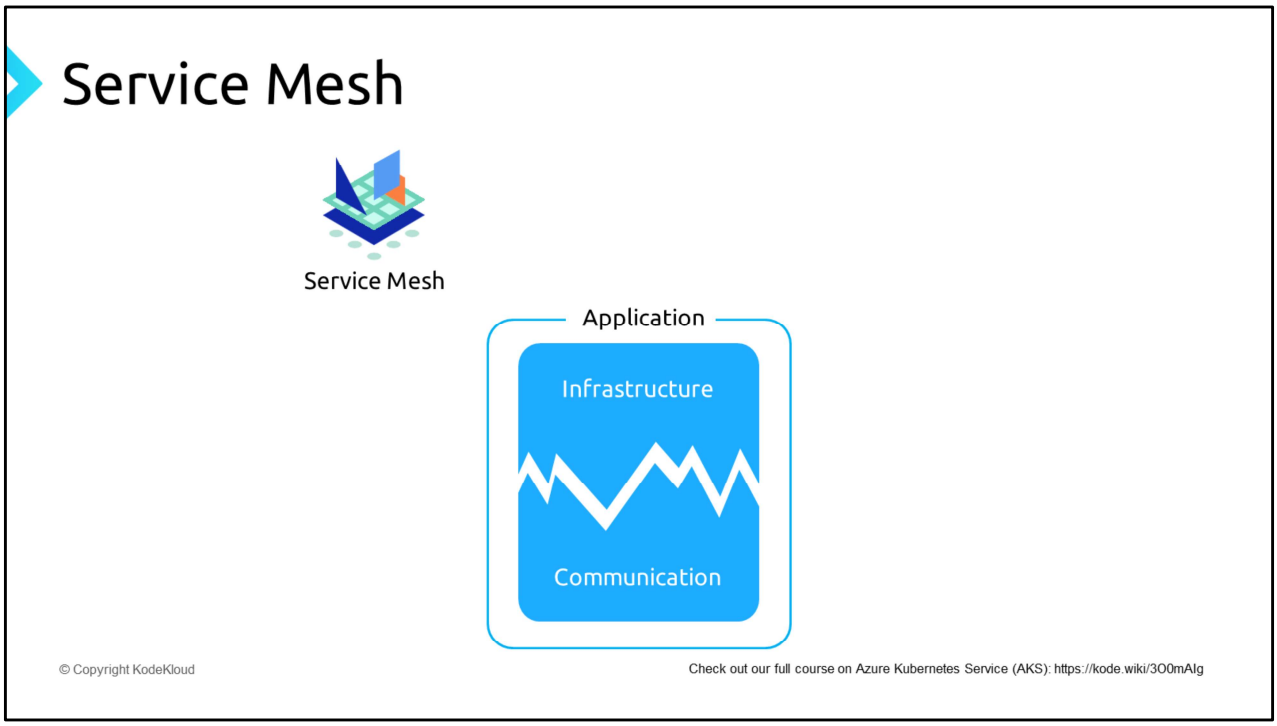
This brings to the end of this section on networking and policies. Hope you found this useful and you can now jump on to the associated labs to get your hands dirty.

KodeKloud

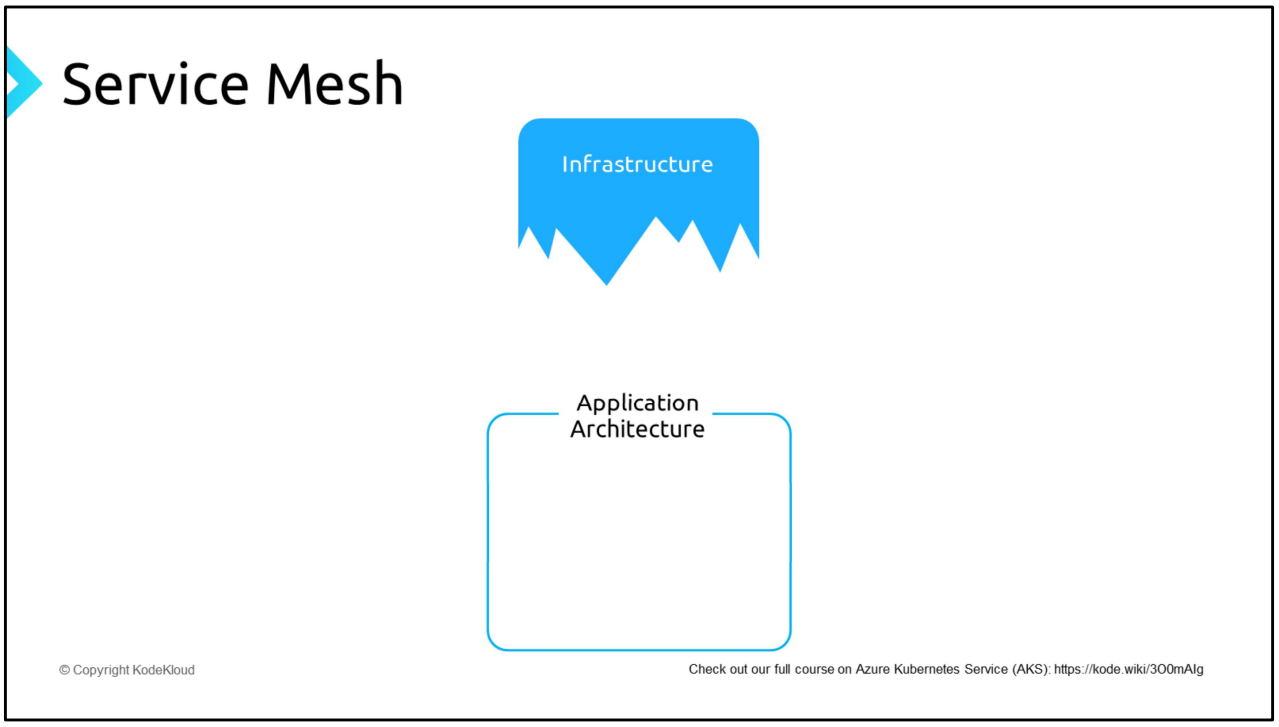
Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Open Service Mesh

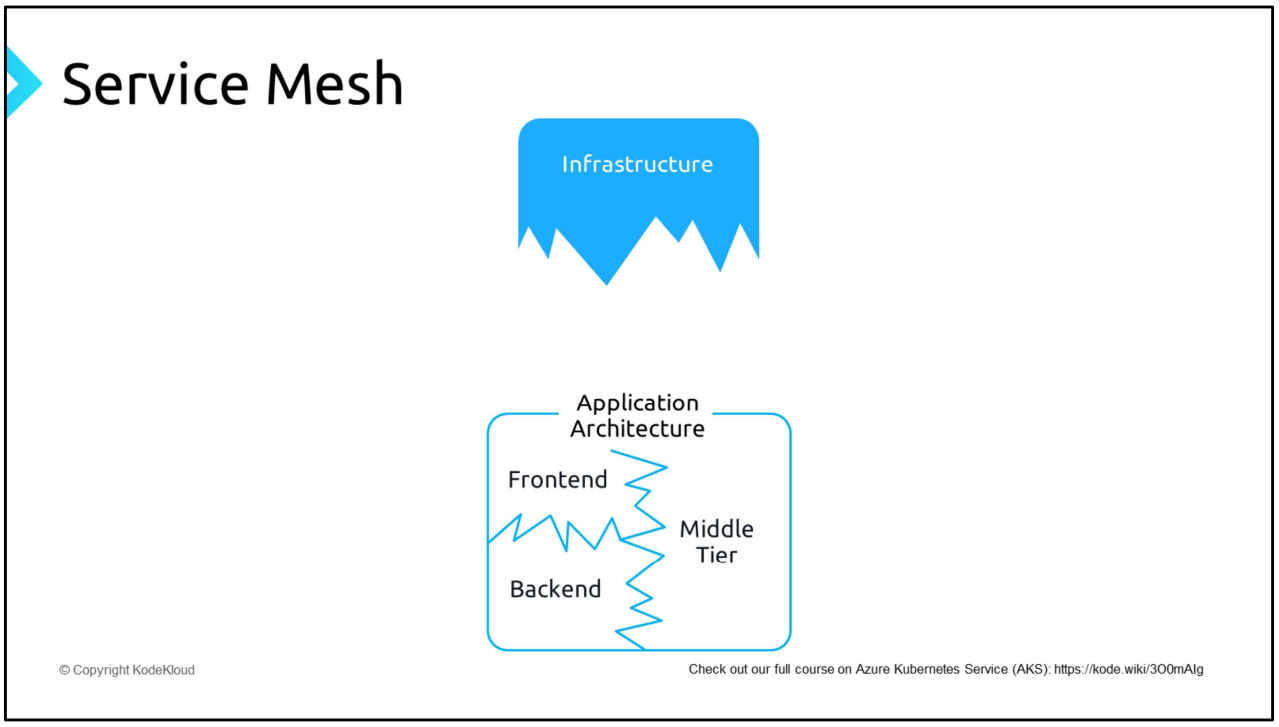
Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg



Service mesh, In Kubernetes, provides a configurable communication layer that decouples infrastructure and communication concerns from the application itself.



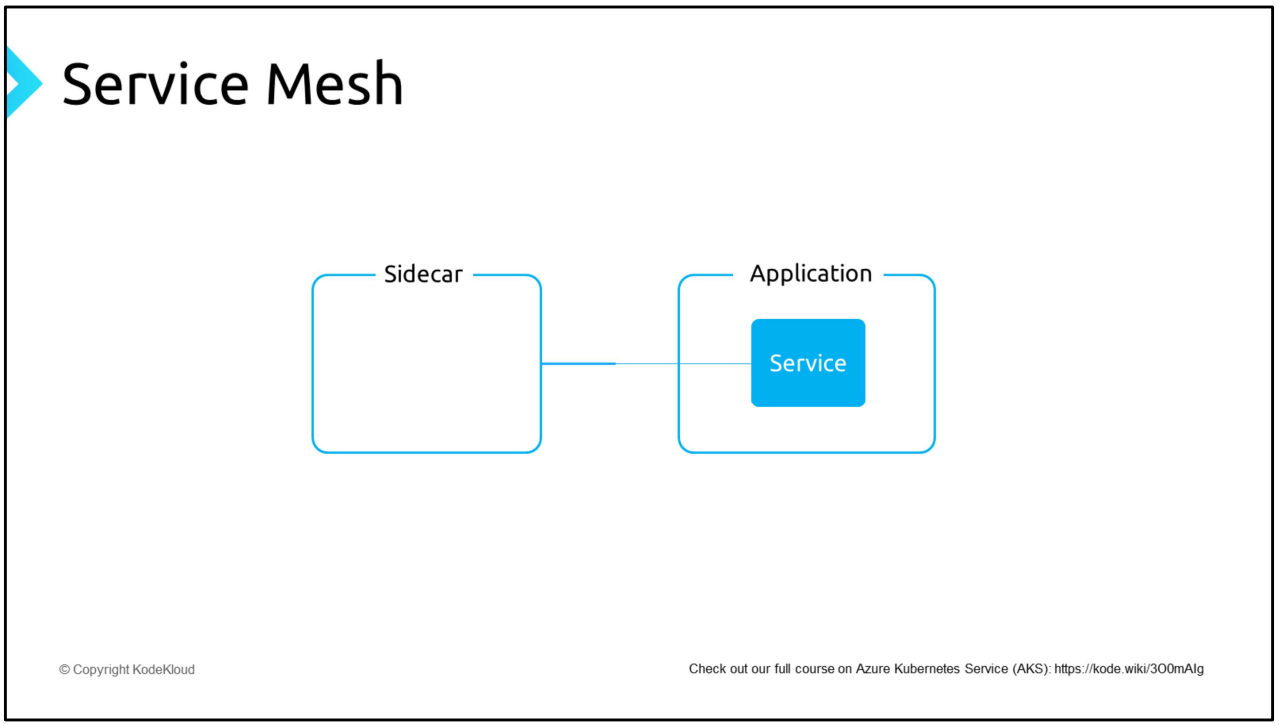
It achieves this by moving the intricacies of infrastructure plumbing to a separate proxy container or process, commonly known as a sidecar. This sidecar proxy is injected alongside each application service, intercepting and managing the communication between them.



This application service can be broken down based on your application architecture, you can call it Service, A,B or C or Frontend, Middle tier and bankend. Names does not matter here.

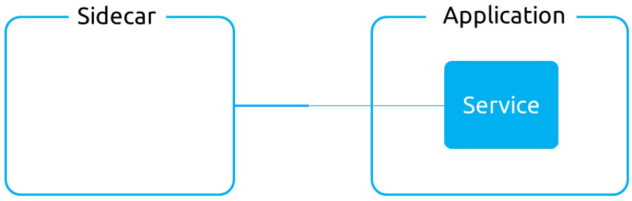
Importantly, the sidecar proxy remains loosely-coupled to the application service, meaning it is created together with the service, shares its lifecycle, and can be managed independently.

This architecture allows for seamless integration of essential service mesh functionalities while keeping the application codebase clean and free from communication logic.



By adopting this sidecar approach, Kubernetes service mesh empowers developers to focus on building resilient and scalable applications, while the infrastructure layer handles the complexities of service communication.

Service Mesh



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Service Mesh

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

By adopting this sidecar approach, Kubernetes service mesh empowers developers to focus on building resilient and scalable applications, while the infrastructure layer handles the complexities of service communication.

Service Mesh

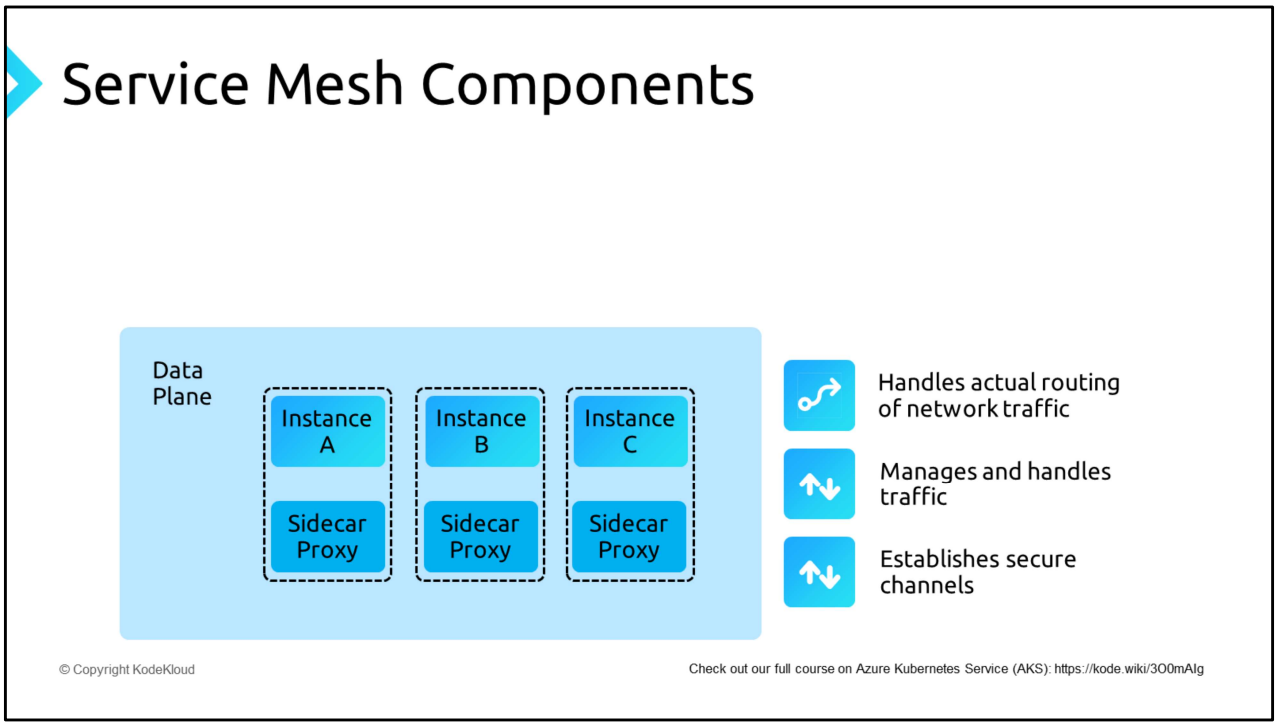
Developers

Resilient

Scalable

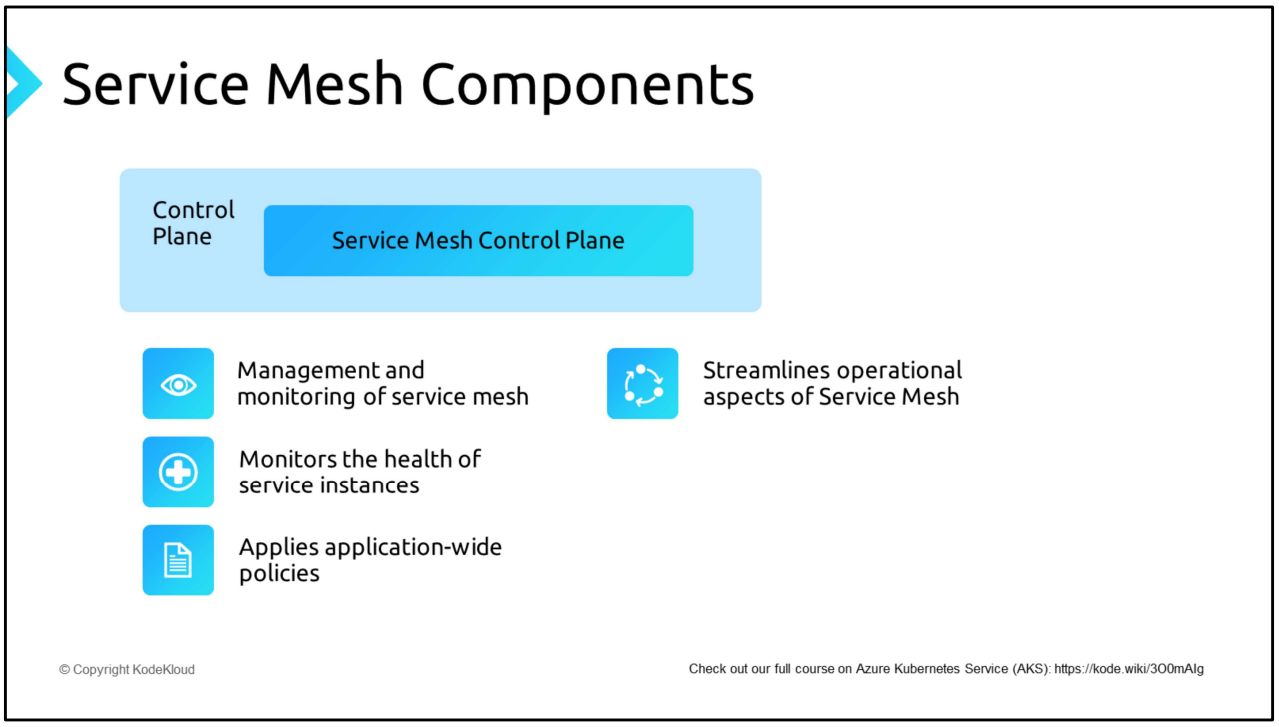
Service Mesh

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

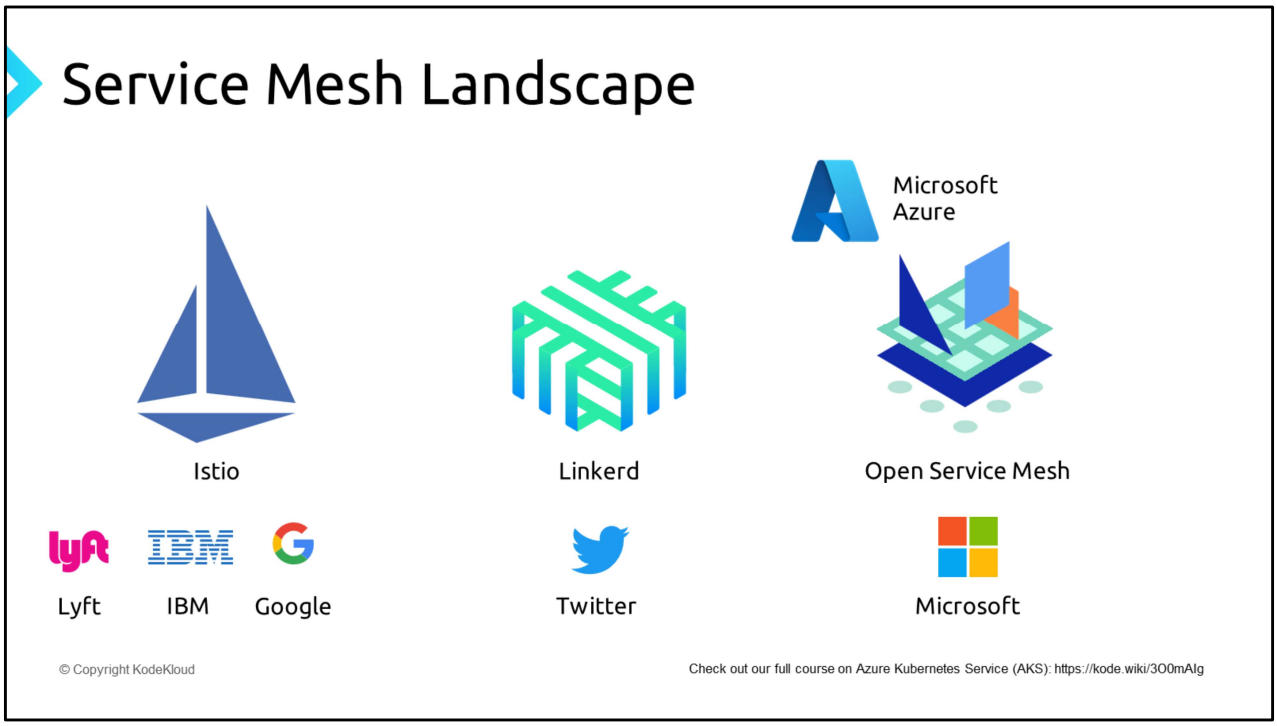


The Kubernetes service mesh consists of two essential components: the Data Plane and the Control Plane.

The Data Plane handles the actual routing of network traffic to service instances within the mesh. It performs critical functions such as service discovery, which enables services to dynamically locate and communicate with each other. The Data Plane also manages traffic, ensuring that requests are efficiently routed to the appropriate service instances. It handles traffic resiliency features like circuit breakers and retries, which help maintain service availability and reliability. Additionally, the Data Plane establishes secure channels for communication, managing identity and access management (IAM) as well as encryption to ensure secure communication between services within the mesh. Overall, the Data Plane focuses on the low-level networking and traffic management aspects, enabling seamless and secure communication between services.



On the other hand, the Control Plane is responsible for the management and monitoring of the service mesh. It takes care of provisioning new instances of services as needed, ensuring that the mesh can scale and handle increased traffic demands. The Control Plane constantly monitors the health of service instances, probing and terminating unhealthy instances to maintain overall system reliability. It also applies application-wide policies, such as rate limiting or access control, to enforce consistent behavior across the mesh. By providing centralized management and monitoring capabilities, the Control Plane streamlines the operational aspects of the service mesh, allowing operators to efficiently manage and maintain the overall health and performance of the application.



There are many Service mesh implementations that you can use on K8s. I'll mention three most popular ones.

Istio is widely regarded as a full-featured, customizable, and extensible service mesh. It has gained immense popularity and is backed by a collaboration between Lyft, IBM, and Google. Written in Go, Istio is designed to be application-agnostic. It offers a comprehensive set of features and provides robust capabilities for traffic management, security, observability, and more.

Linkerd, on the other hand, is known for its ease of use and lightweight nature. It is feature-rich and cross-platform, making it suitable for various environments. Linkerd builds upon early work from Twitter and is written in Scala, running on the JVM. It focuses on simplicity and ease of adoption, making it an attractive option for teams seeking a straightforward service mesh solution.

Open Service Mesh (OSM) is a cloud-native service mesh developed by Microsoft. It is designed to be a lighter-weight alternative to Istio. OSM prioritizes simplicity, aiming to provide a service mesh that is easy to understand, install, maintain, and operate. It

leverages the Service Mesh Interface (SMI) specification, allowing for standardization and easier integration with existing Kubernetes environments.

Microsoft Azure supports Open Service Mesh as its developed by them. Historically that was the default option on AKS, at the time of this recording Microsoft has started to offer native Istio service as preview. It remains to be seen how this will change their recommendation.

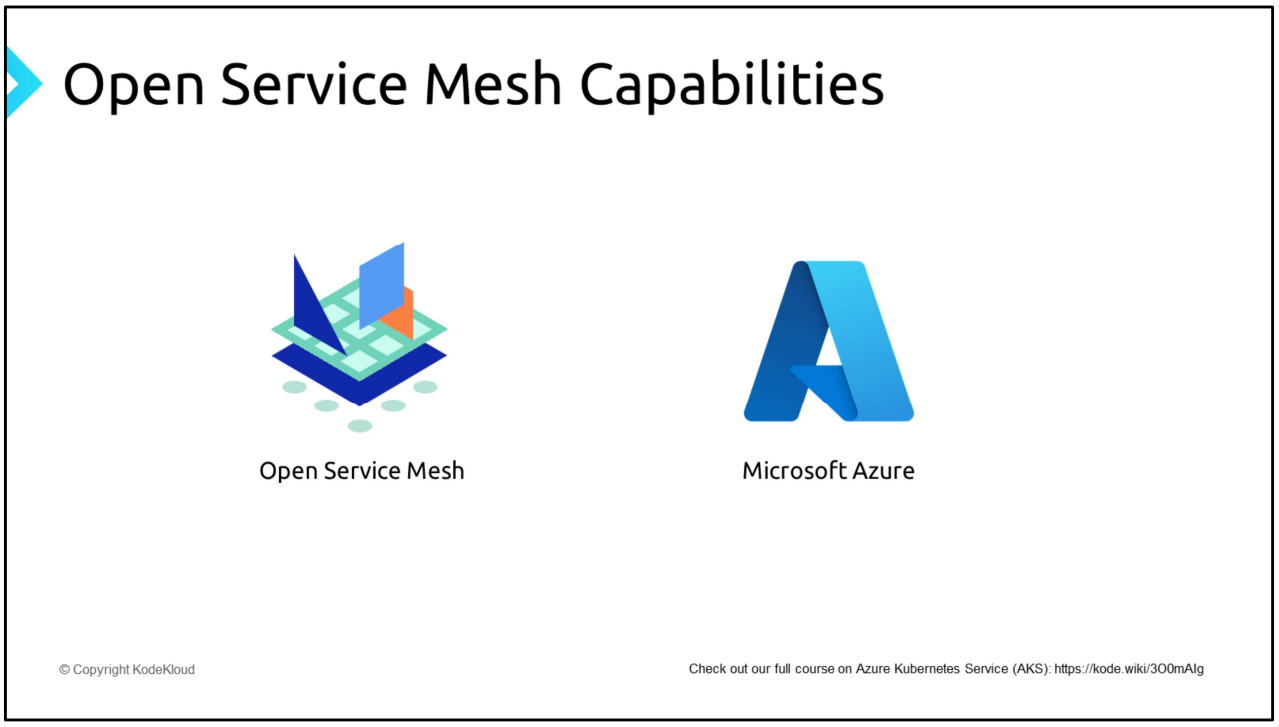
We will look into the OSM implementation in the rest of this video.

Open Service Mesh Capabilities

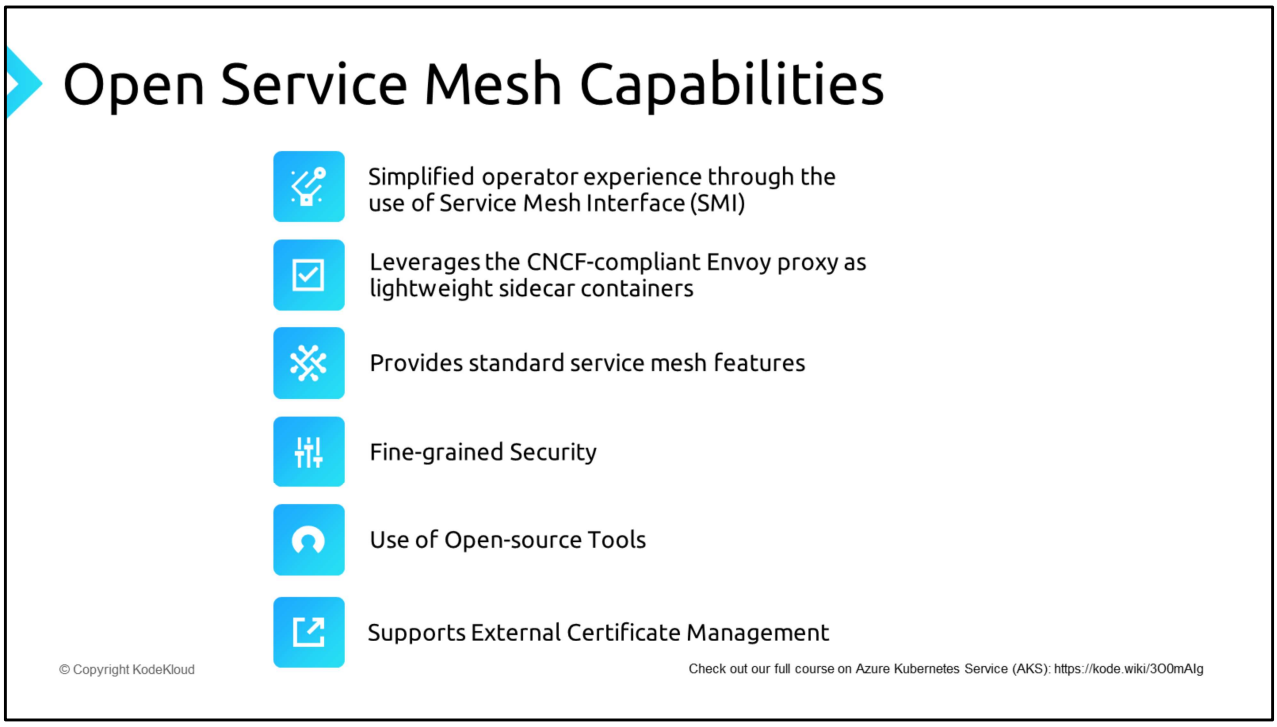
Open Service Mesh

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

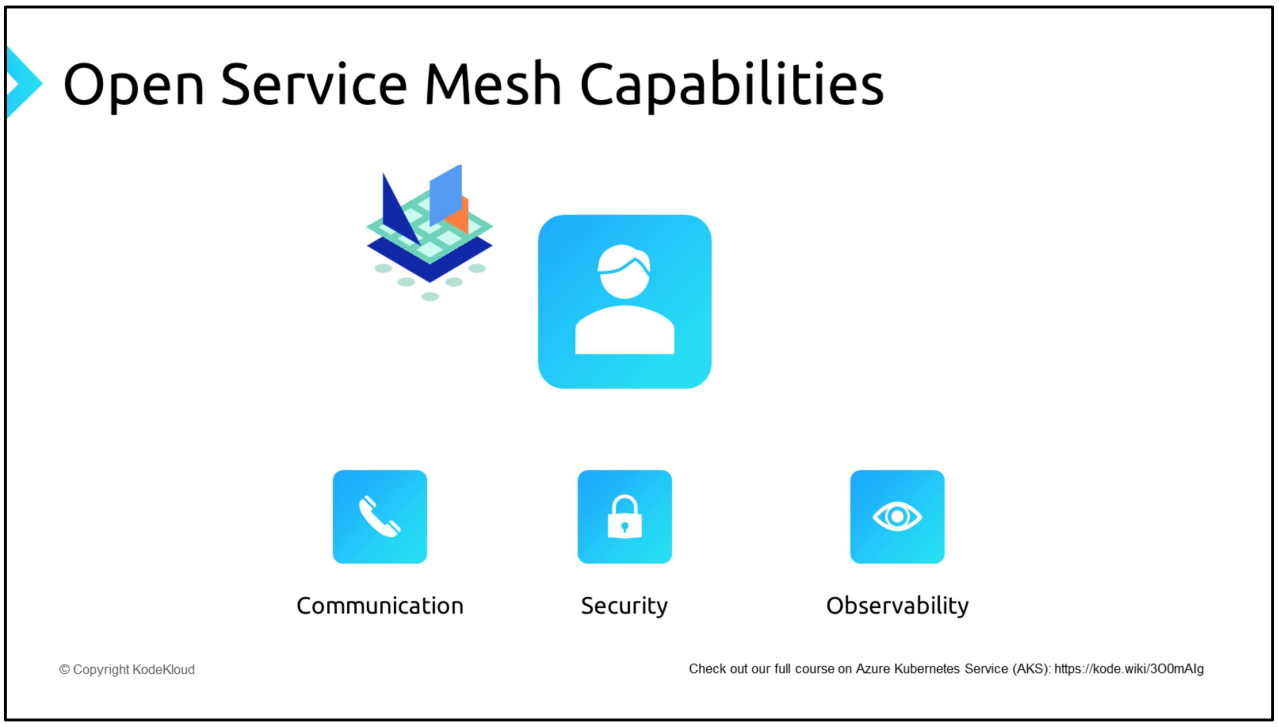
So lets look at some of the features of OSMas is implemented in Azure



Open Service Mesh (OSM) is a powerful service mesh solution that seamlessly integrates with Microsoft Azure, providing numerous benefits for managing microservices-based applications.

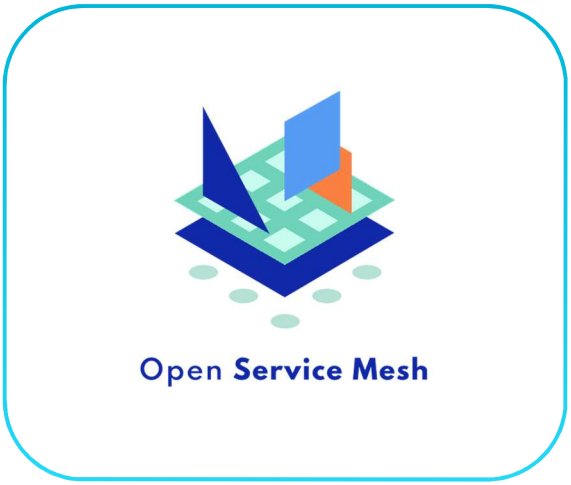


With OSM, Azure users experience a simplified operator experience through the use of Service Mesh Interface (SMI), which offers standardized APIs for easy mesh configuration. OSM leverages the CNCF-compliant Envoy proxy as lightweight sidecar containers, ensuring efficient and secure communication between services. It provides standard service mesh features such as traffic access control, traffic splitting, and traffic metrics. Additionally, OSM enables fine-grained security with mutual TLS (mTLS) encryption for secure service-to-service communication.

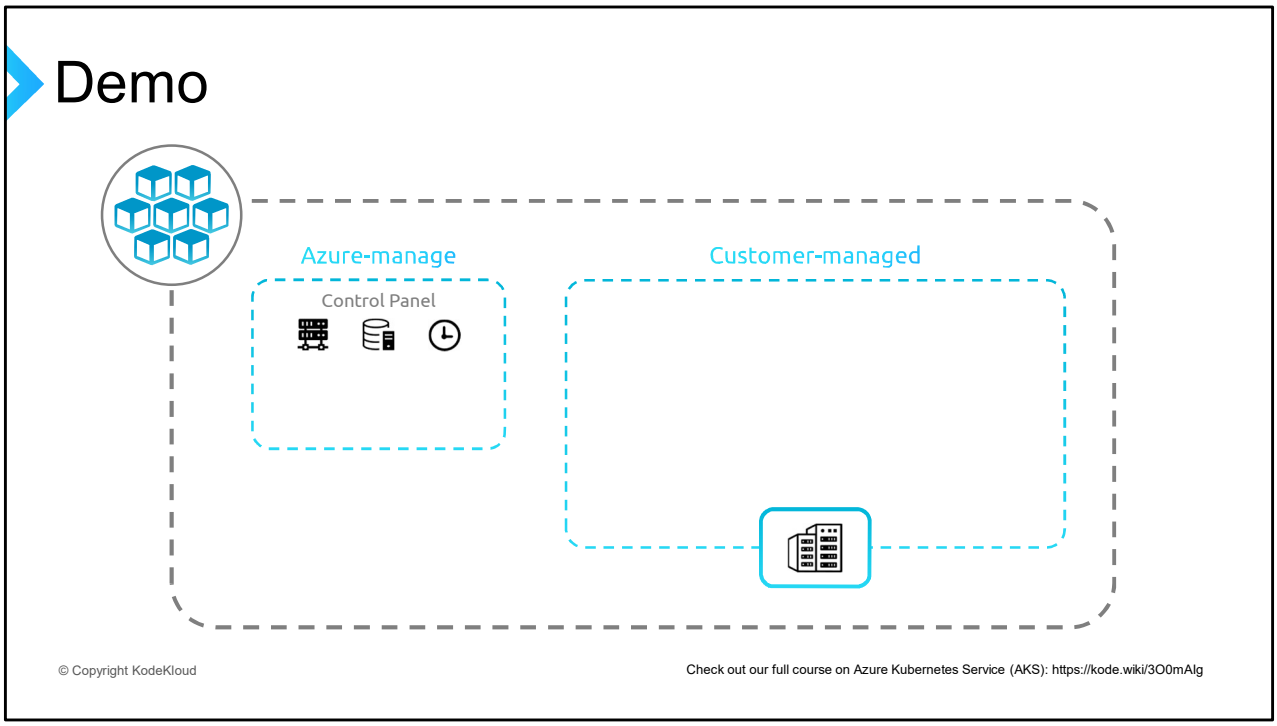


Monitoring and tracing capabilities are facilitated through open-source tooling, allowing users to gain insights into application behavior. Furthermore, OSM supports external certificate management, making it convenient to handle security certificates for enhanced protection. Overall, OSM's implementation on Azure empowers developers with a robust service mesh solution that streamlines communication, security, and observability within their applications.

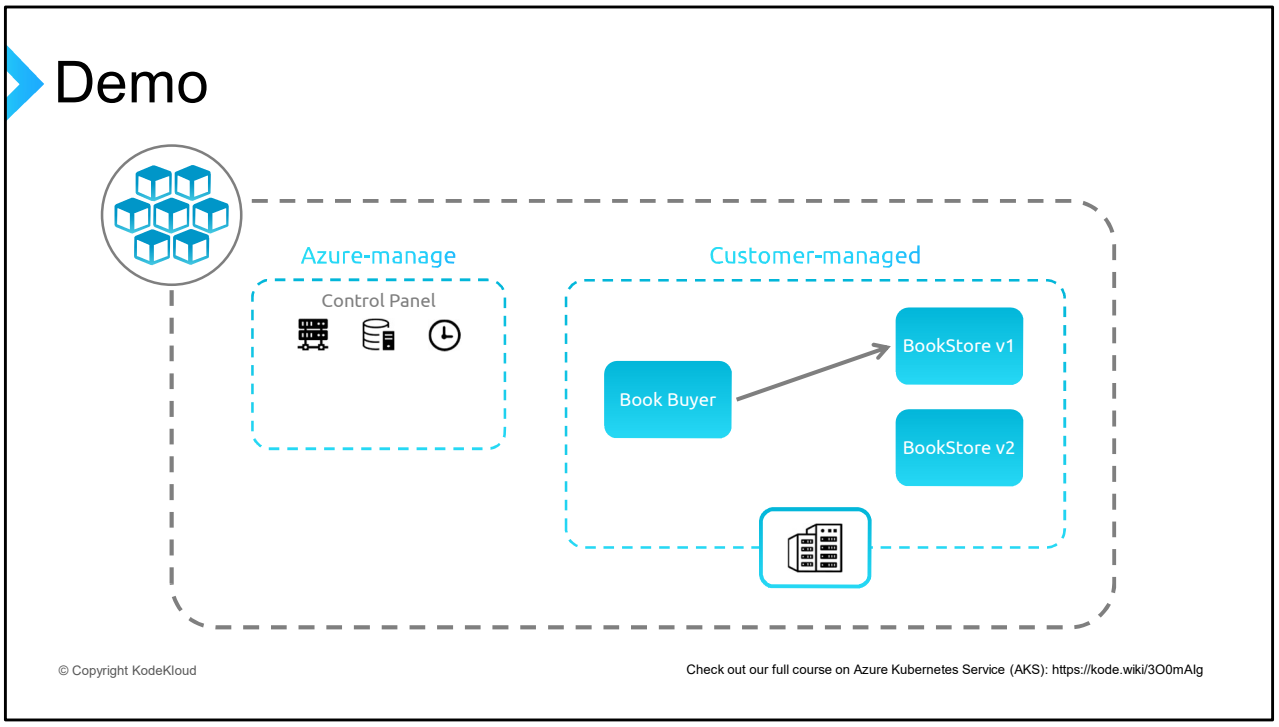
Demo



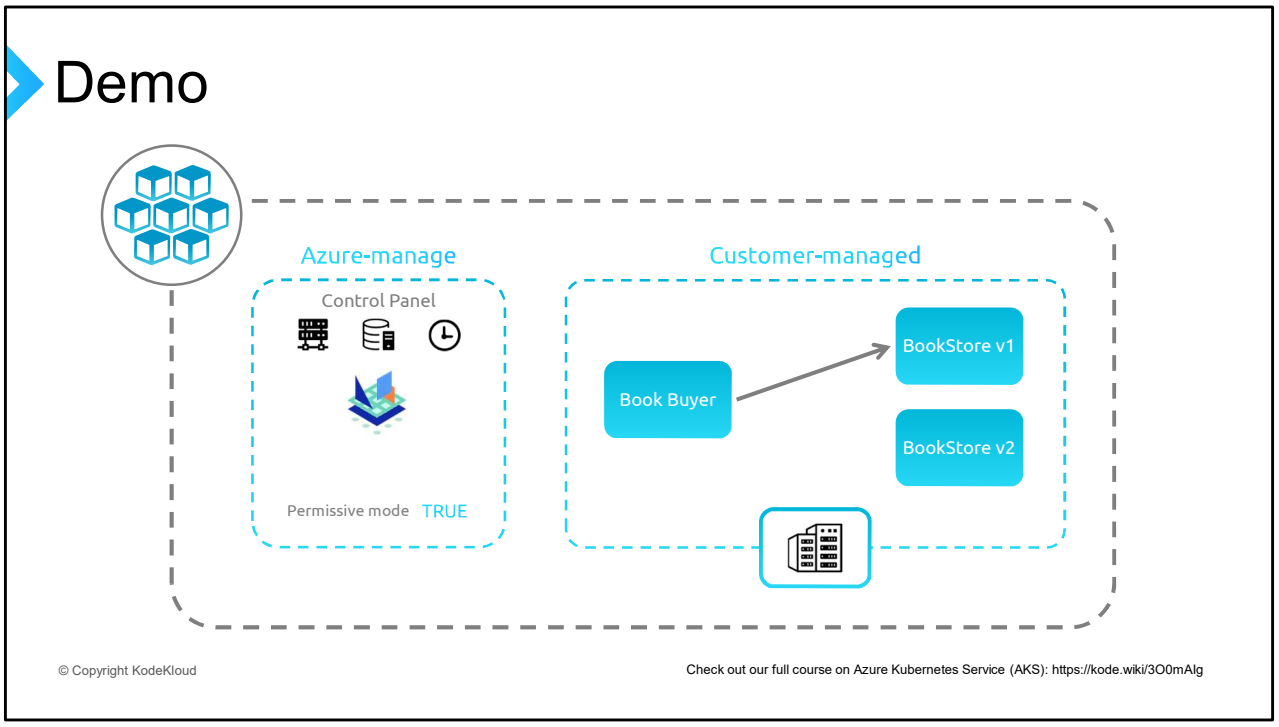
Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg



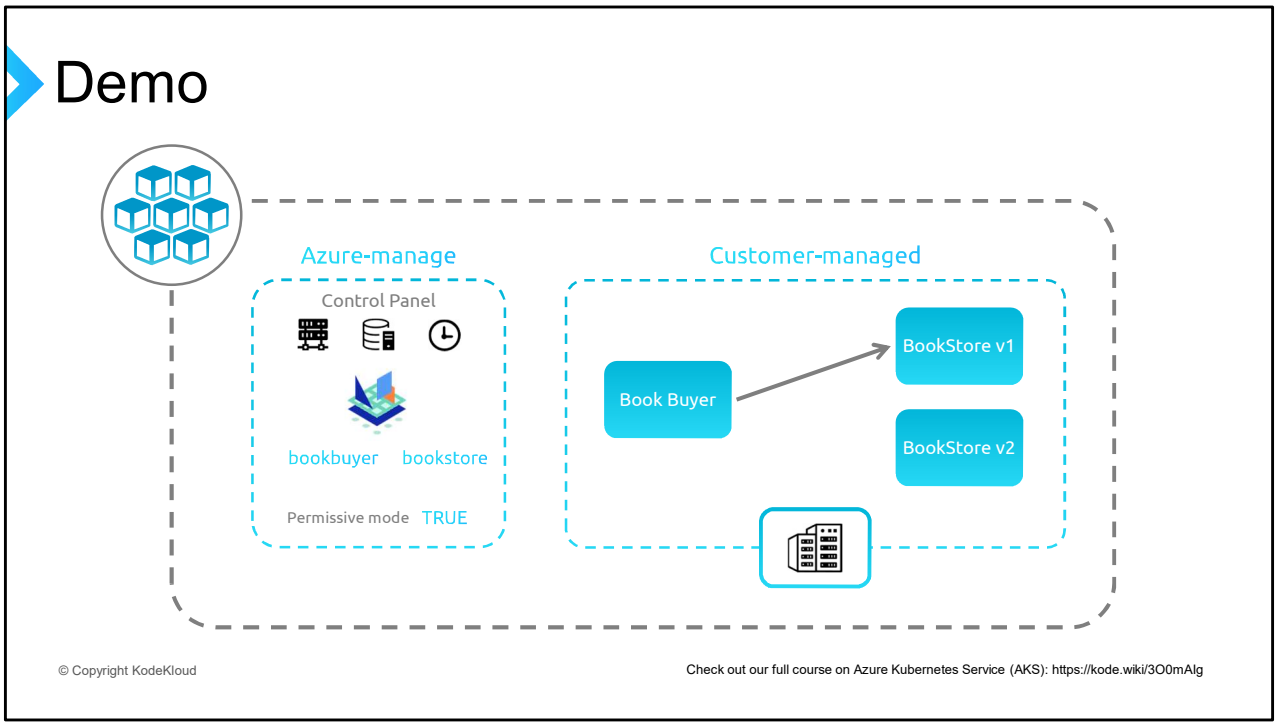
In this demo, we will showcase the power of Open Service Mesh (OSM) by demonstrating its key capabilities. To begin, we'll deploy a sample bookstore application on your AKS cluster. This application comprises two services that will be automatically deployed within the cluster.



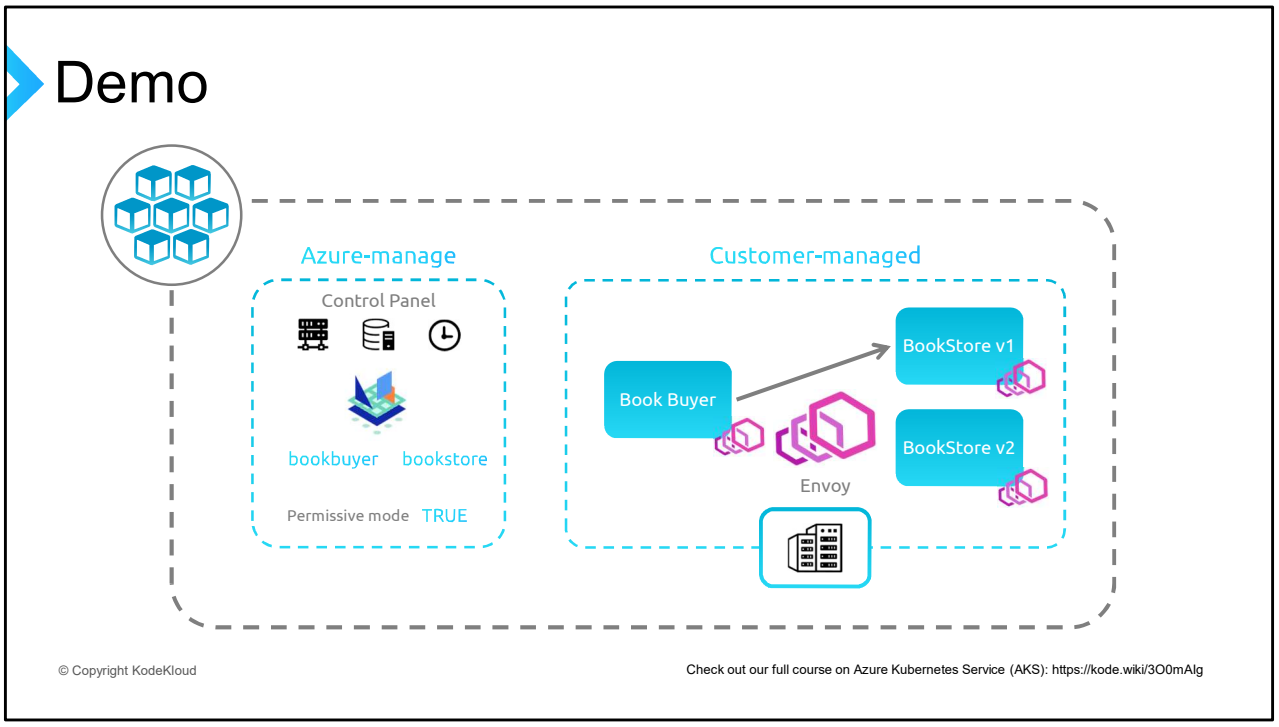
The two services included in the sample bookstore application are the book Buyer and bookstore v1. Initially, the book Buyer service will establish communication with the bookstore v1 service. Later in the demo, we will deploy bookstore v2 to demonstrate the functionality of traffic splitting within OSM.



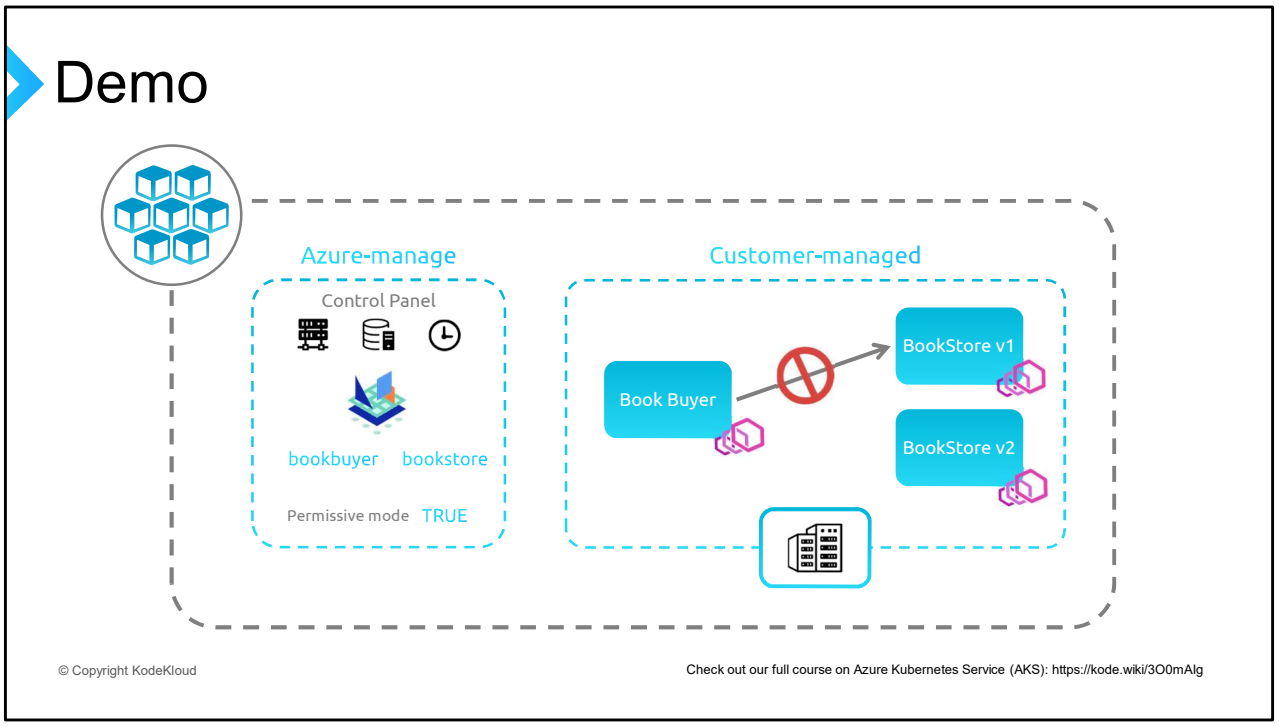
Next, we will enable the OSM add-on for AKS cluster, which will default to the permissive traffic mode. In this mode, all traffic will be allowed by default, serving as our starting point.



Afterward, we will proceed to onboard the bookbuyer and bookstore namespaces to be managed by OSM. This will involve configuring OSM to handle and govern the services within these namespaces.

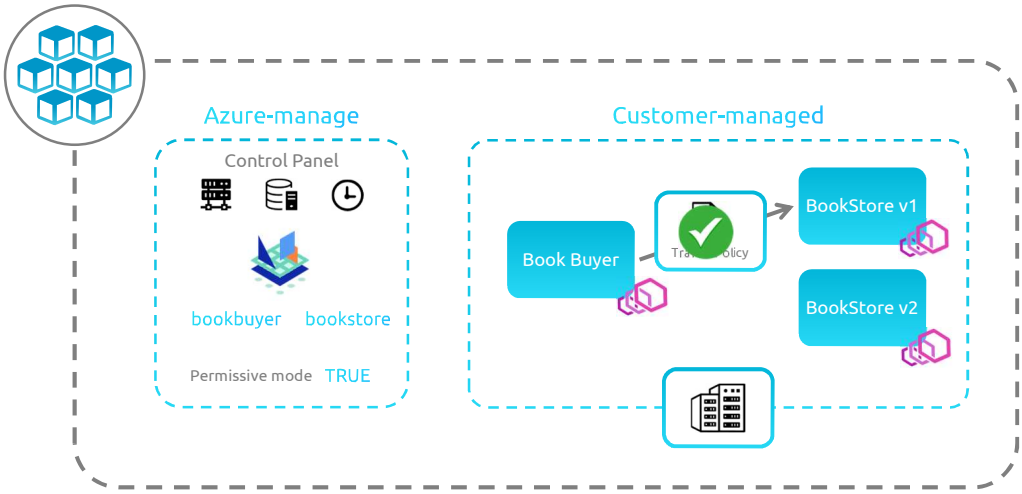


This process will enable the injection of Envoy sidecar proxies alongside the application containers within the pod.



At this stage, the book buyer service should retain its ability to communicate with the bookstore v1 service. Once we confirm this communication, we will proceed to change the permissive mode of OSM to "false." This change will result in the cessation of communication between the book buyer and bookstore v1 services.

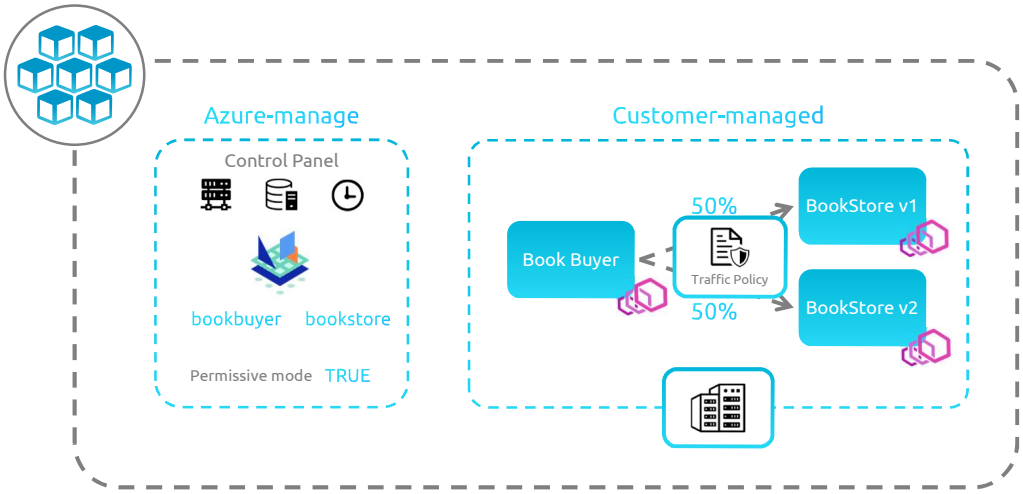
Demo



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAlg

Next, we will deploy a traffic policy to enable traffic between the services, thereby establishing communication and allowing the desired interaction.

Demo



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Lastly, we will deploy a traffic split policy to evenly distribute the traffic between both versions of the service. This policy will ensure a balanced distribution of requests, allowing us to showcase the capabilities of traffic splitting within OSM.

AAD and AKS

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Identity in Azure AD

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Identity in Azure AD

Used for:

- Authentication
- Access Control

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Identity in Azure AD

Enables:

- Applications to authenticate
- Access resources

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Identity in Azure AD

Acts

As an identity of Credentials for the application to interact with Azure Services

Allows:

- Application all services to authenticate
- Access resources in Azure

Often Use:

Non-interactive Scenarios

- Automations
- Service-to-service Communications

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Identity in Azure AD

Simplifies
authentication process for resources

Eliminates
the need of managed explicit credentials

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Identity in Azure AD

Service Principal Identity

Use either a client ID and client Secret or a certificate to authenticate and obtain access tokens

Note:

- AKS clusters that use service principal eventually expires.
- Service principal must be renewed to keep the cluster working.

Managed Identity

Obtain tokens directly from azure AD, eliminating the need for explicit authentication details like secrets or services

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Kubernetes RBAC

Identities

- Providers
- Users, Groups
- Service Accounts

Access Control

- Namespace scoped Roles and RoleBindings
- ClusterRoles and ClusterRoleBIndings

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Kubernetes Identity Providers

X509 Certificates

Manually Issued

Configurable trusted CAs

Kubernetes Service Accounts

Automatically generated by a built-in Admission Controller

OpenID Connect (OAuth2)

AAD (and many others)

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAlg

This how a K8s role looks like.

The role named ”kodekloud-clusterole" is Clusterwide Role that grants permissions to perform get and patch operations on daemonsets and deployments resources within the apps API group. It also allows get operations on configmaps resources in the core Kubernetes API group.

Sample Role Binding

Terminal

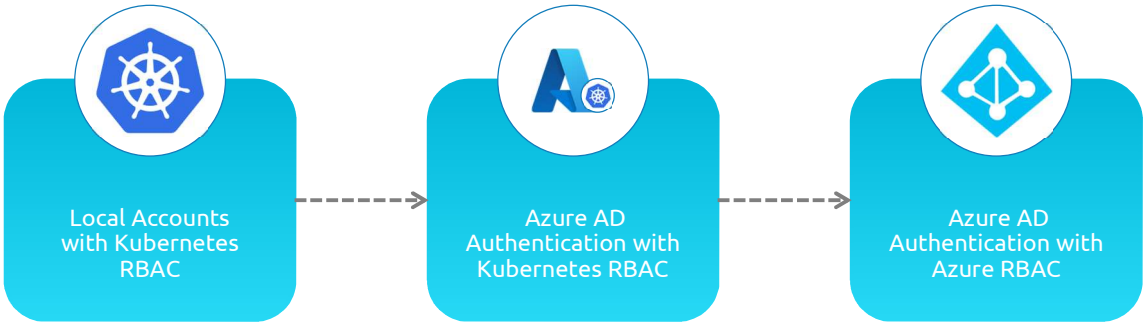
```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: cluster-reader-rolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: kodekloud-clusterrole
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: 8e9a4227-0e29-4d2a-9fd3-cacae5a6821e
```


Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Once the role is created we can use Rolebiding to assign that role .

In this sample, we have a clusterrolebinding that binds the ClusterRole showed in the previous slide, to a specific group of subjects identified by their unique identifier. This binding provides the subjects with the permissions and access defined by the ClusterRole shown before.

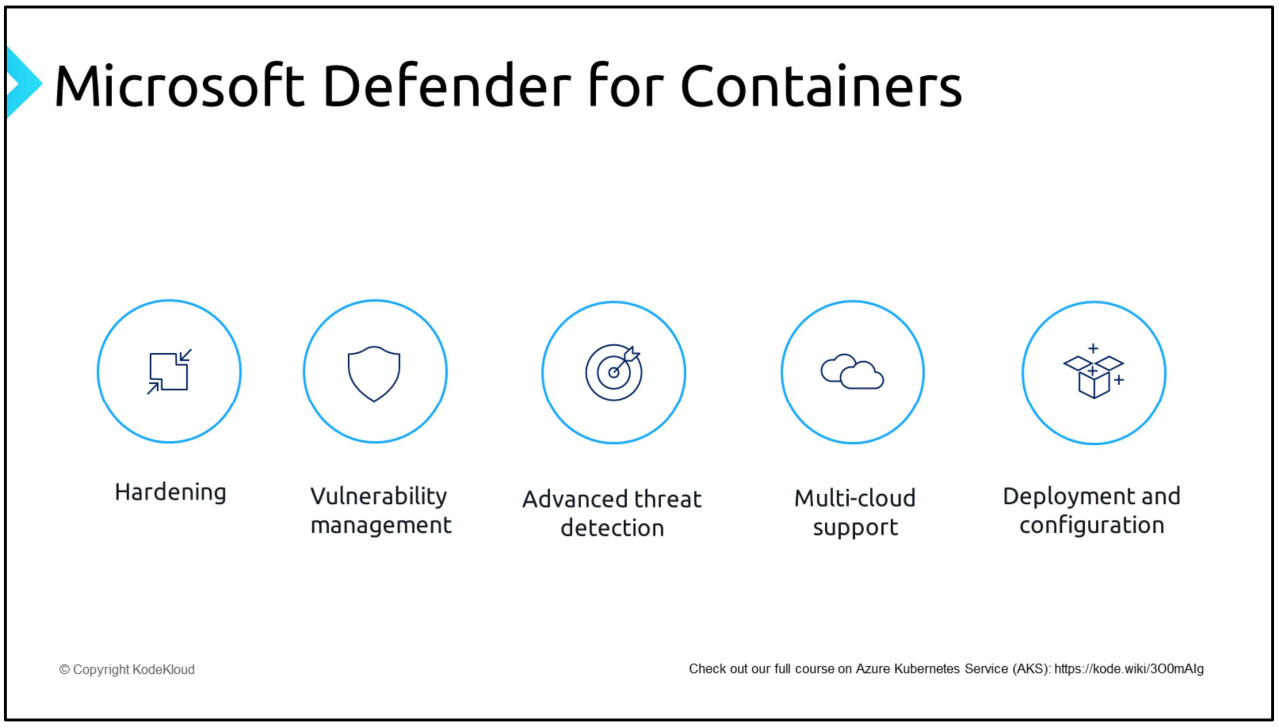
Authentication and Authorization in AKS



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

AKS Defender

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg



Continuously assess and improve the security posture of your containerized environments and workloads

Identify runtime threats with prioritized, container-specific alerts – using powerful insights from Microsoft Threat Intelligence

Single container security solution for Kubernetes clusters, across Azure, AWS, GCP and on-premise

Hybrid security approach with agentless capabilities

Frictionless at scale deployment with easy onboarding and support for standard Kubernetes monitoring tools

Native integration in AKS

Defender Profile

Azure Policy add-on

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

What do we need for the defender plan, we are looking at 2 key components:

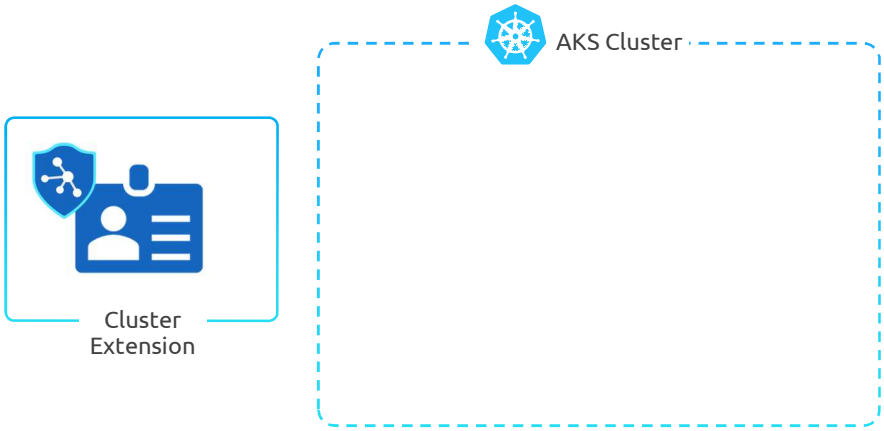
- The defender profile
- The azure policy add on

The Defender Profile

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

You will see both of these components are natively integrated to Azure Kubernetes services.

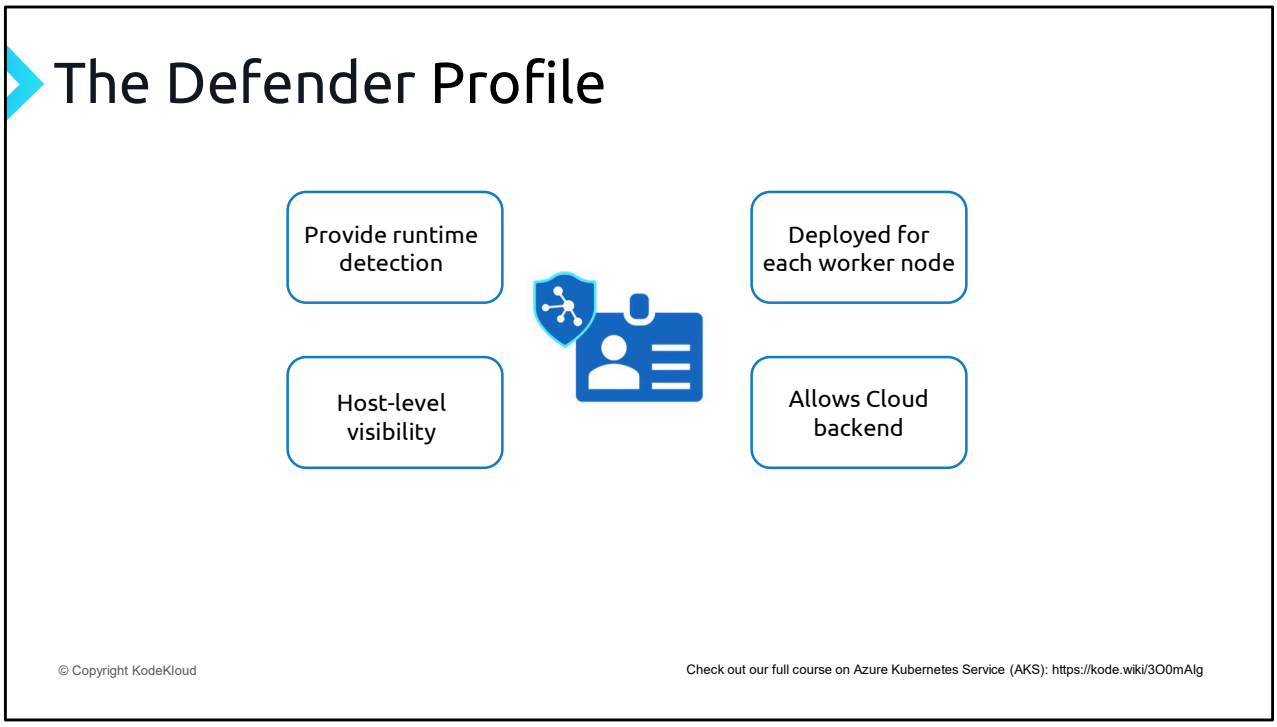
The Defender Profile



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

- The **defender profile,** when you are working in Azure it is integrated as azure security profile, and when you are working outside of azure this is integrated as a cluster extension.

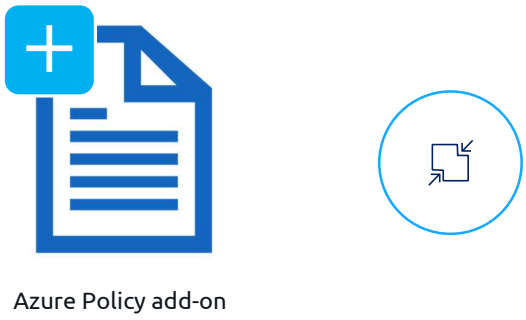
The Defender Profile



This allows us to:

- Provide run time detection
- And get host level visibility to the cluster
- The defender profile as you can see is deployed for each worker node and with the defender profile, we collecting security data events and inventory and sending it to the defender for cloud backend.

Azure Policy Add-on



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

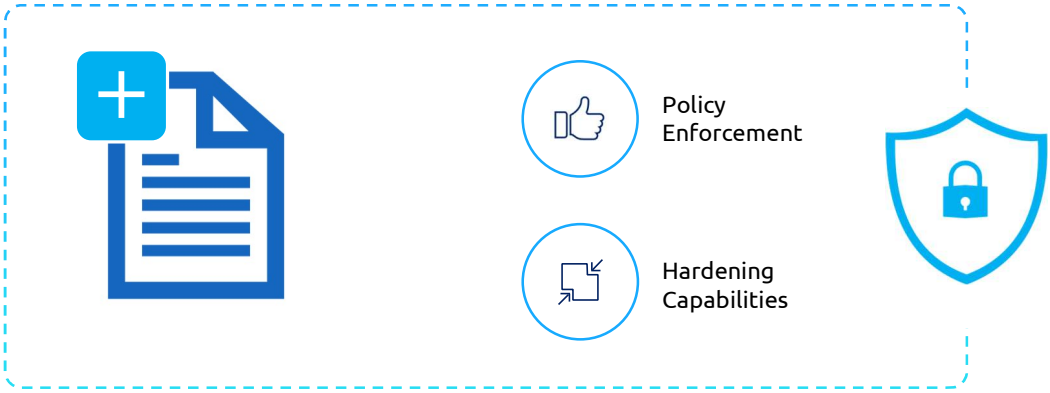
- The second component is the **azure policy add on**, this component is extending the gate keeper, to apply Kubernetes data plane policies enforcements and hardening capabilities, essentially every API request to the Kubernetes API server is been monitoring against best practices, we use this for different recommendations you can see in portal.

Azure Policy Add-on

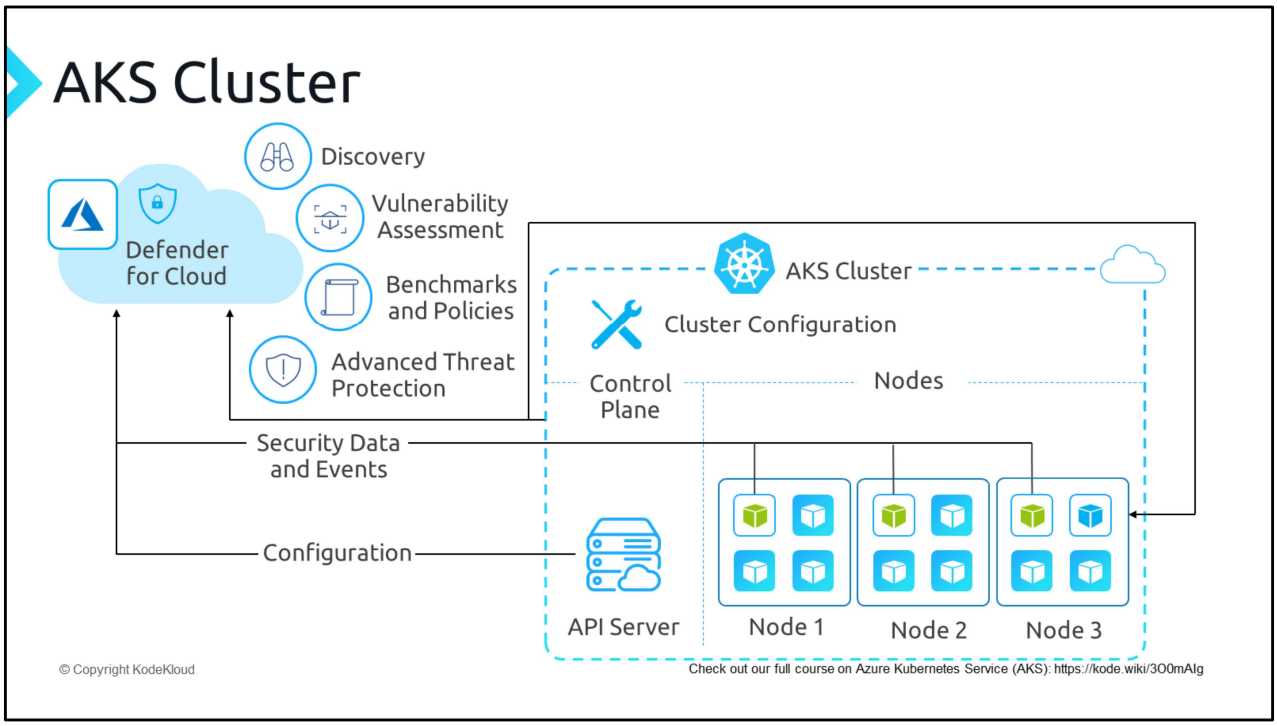
Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

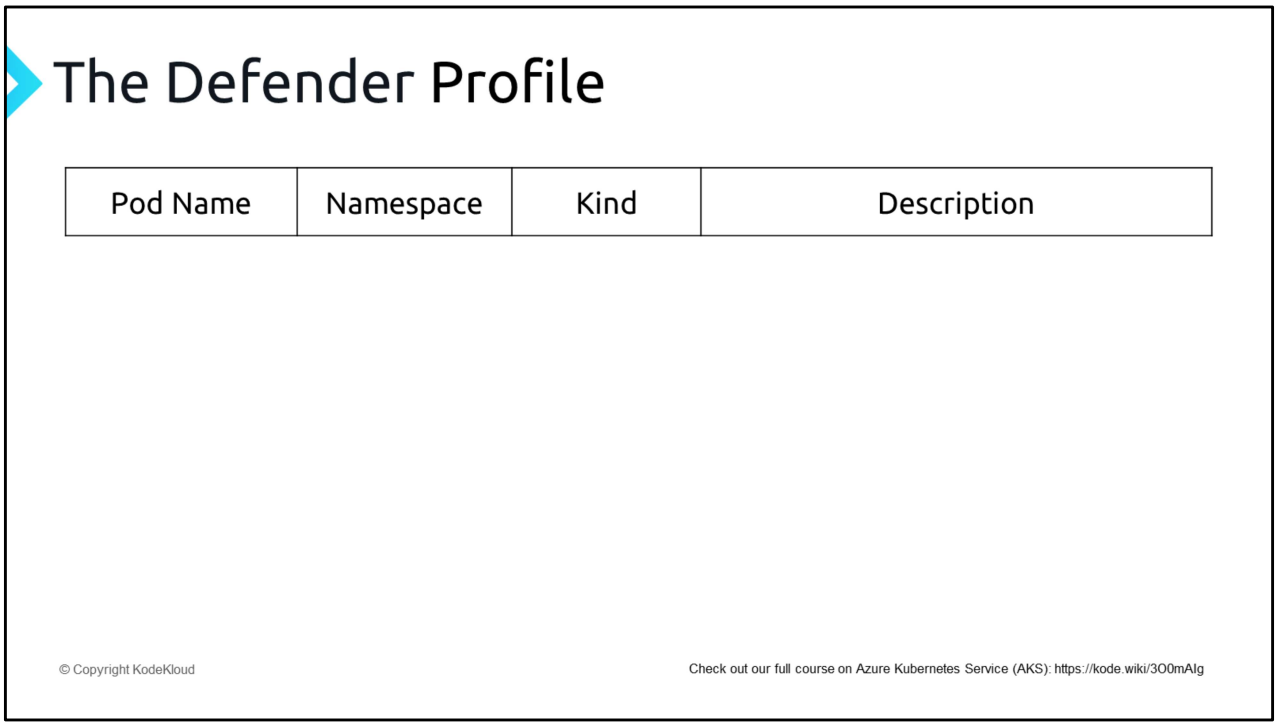
- To enable frinction less deployment, The deployment options are:
  - From the MDC portal – auto provisioning
  - From the MDC portal – recommendations quick fix
  - With Rest API
  - From Arm (azure resource manager)
  - From CLI

Azure Policy Add-on

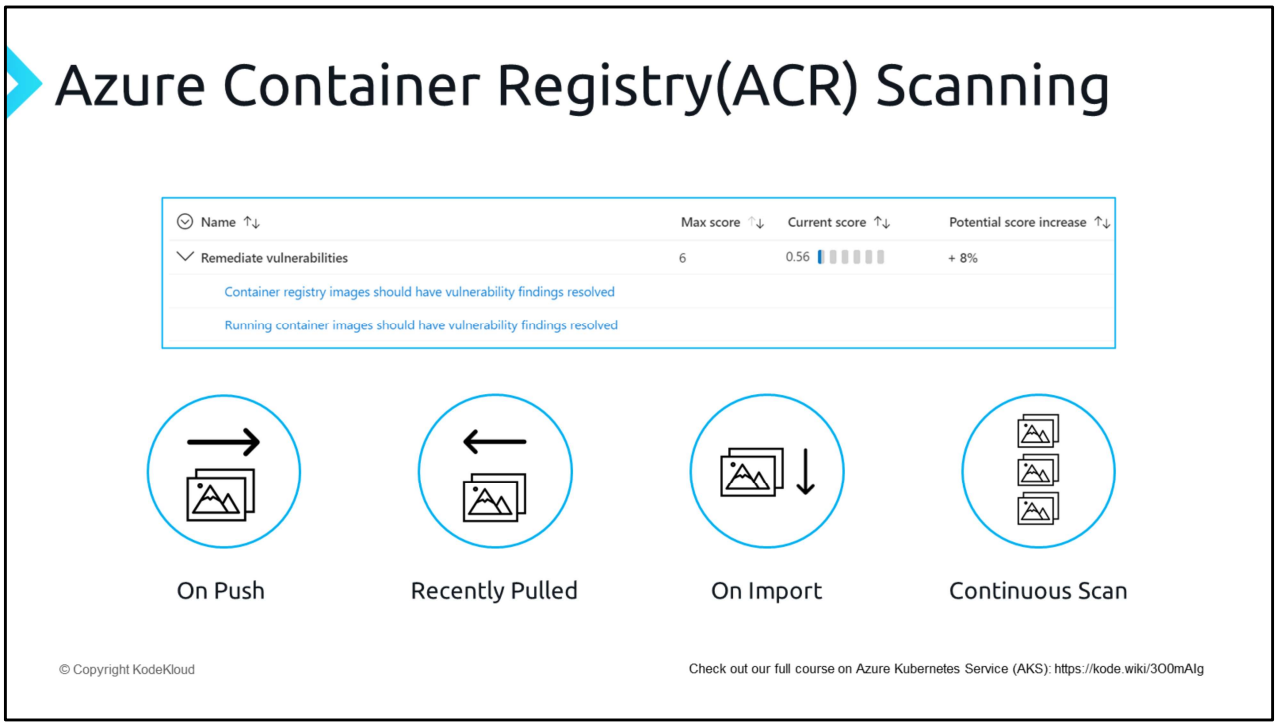


Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg





As we said the defender profile is deployed as a daemon set, so it gets to be deployed on every single node in the cluster, as a pod.

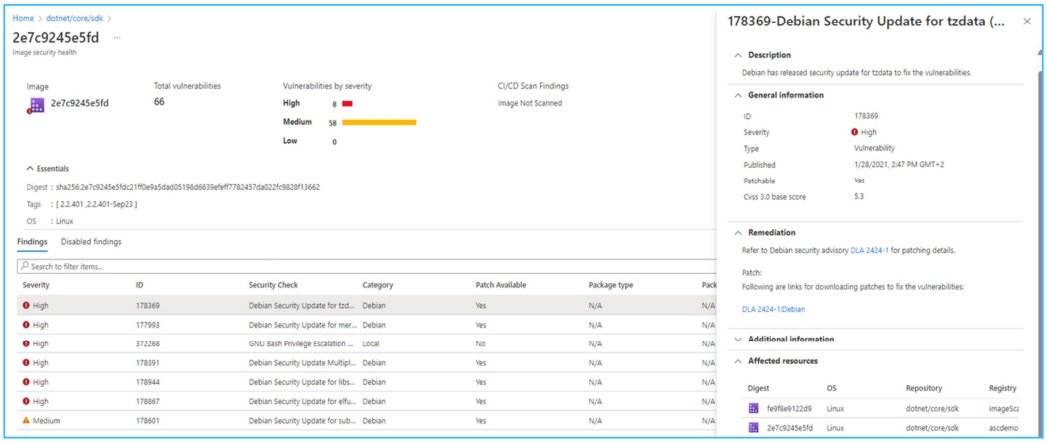


Triggers for an image scan:

On push: whenever and image is pushed to the registry
Recently pulled: any image that been pulled within the last 30 days is scanned on a weekly basis
On import: when an image is imported into ACR
Continous scan

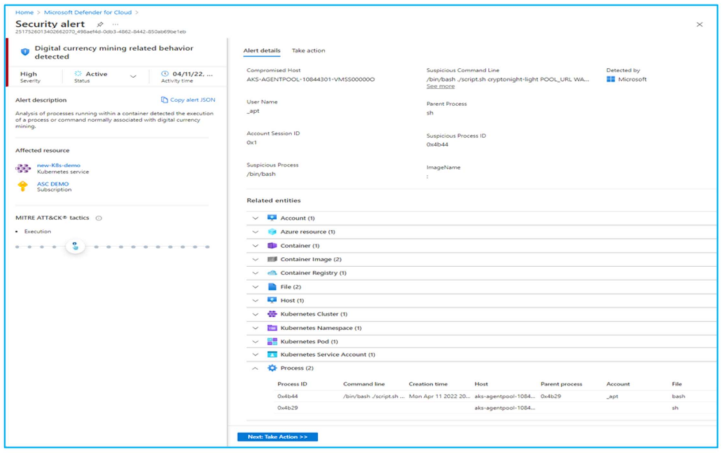
Every 7 days after an image is pulled. Does not require the security profile
Continous scan for running images. Every 7 days as long as the image runs. Runs instead of the above when Defender profile is enabled

Advanced threat detection

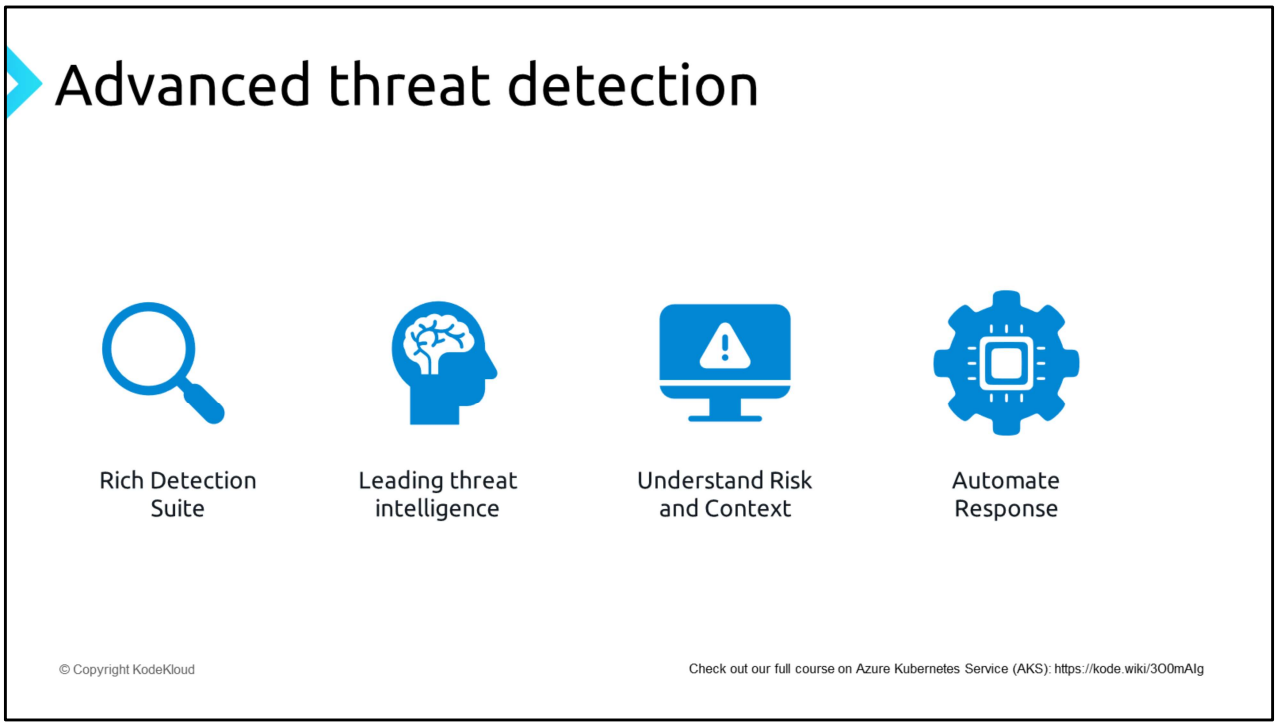


Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Advanced threat detection



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg



Rich detection suite

- Control plane and workload level detections
- Deterministic, AI, and anomaly-based alerts to identify threats

Leading threat intelligence

- Microsoft's global threat intelligence with honeypot networks, research malware feeds, in addition to memory forensic techniques to identify fileless attacks

Understand risk and context

- Prioritized alerts mapped to MITRE ATT&CK® tactics to easily understand the Kubernetes context effect across the attack lifecycle and to identify response action

Automate response

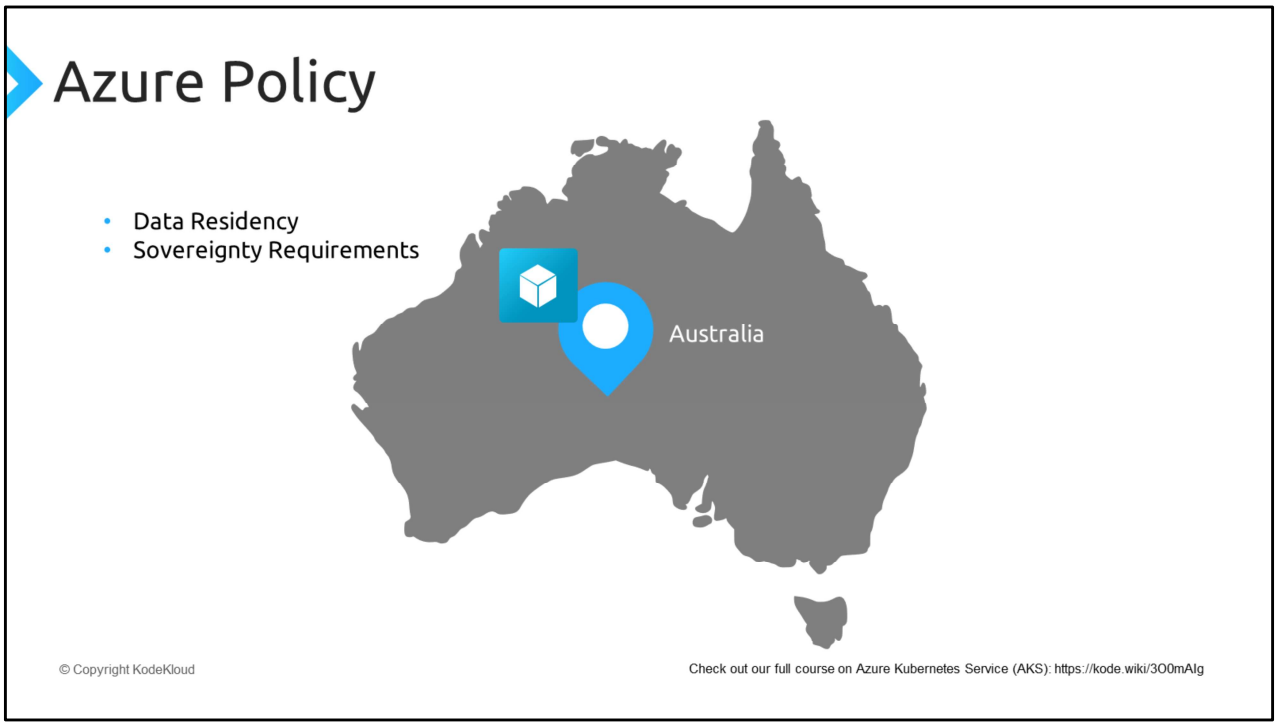
- Automate actions with tools of your choice: SIEM

integration, email notifications, workflow automations and continuous export

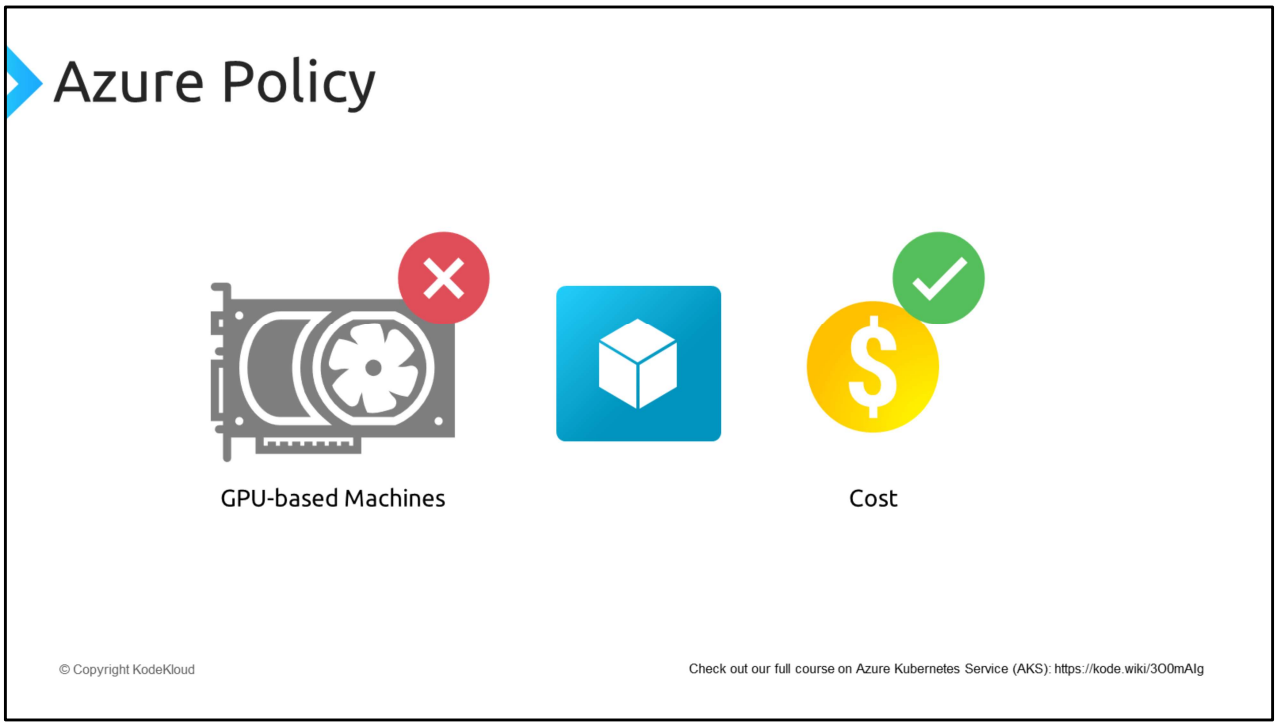
Azure Policy for AKS

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Before we talk about Azure Policy for AKS, lets quickly see what is Azure Policy and why and how do you use it.

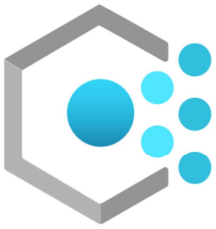


As an organization, you will have many requirements or intents that you would like to enforce or atleast report agaist. For example, lets say you a business critical application and you would like to deploy your resources for that application only in Australia because of data residency and sovereignty requirements. Lets take another scenario where you would like to make sure that none of the AKS deployments use GPU based machines in order to control costs.



Azure Policy is a service that enables you to define and enforce governance and compliance rules for your resources. It provides a centralized and consistent way to apply and enforce policies across Azure subscriptions, resource groups, or individual resources. With Azure Policy, you can establish rules that govern resource configurations, access controls, and adherence to specific compliance requirements. These policies are defined using JSON-based policy definitions and can be customized to align with your organization's unique needs. By implementing Azure Policy, you can ensure that your Azure environment remains secure, compliant, and well-governed, promoting best practices and minimizing risks associated with misconfigurations or non-compliant resources.

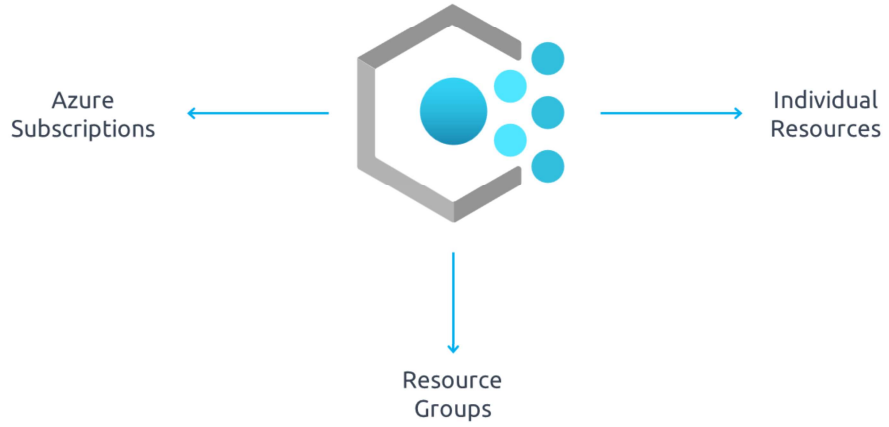
Azure Policy



A service that enables you to define and enforce governance and compliance rules for your resources

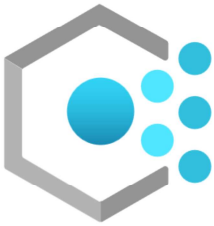
Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Azure Policy



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

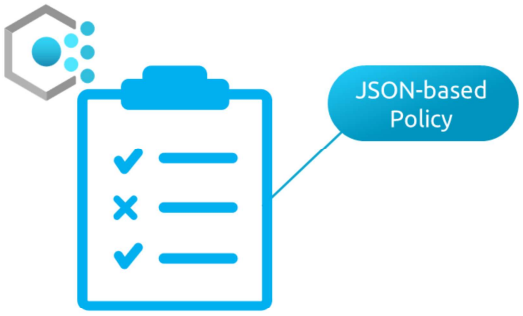
Azure Policy



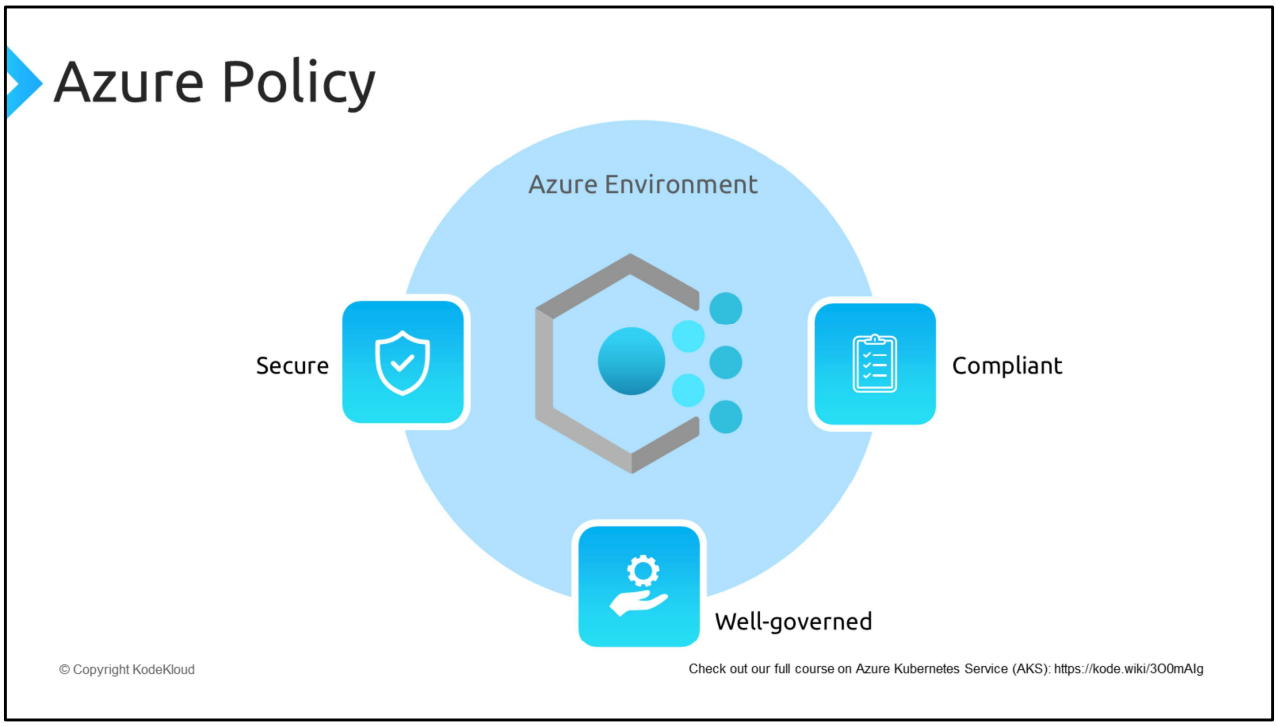
- Resource Configurations
- Access Controls
- Adherence to specific compliance requirements

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Azure Policy



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

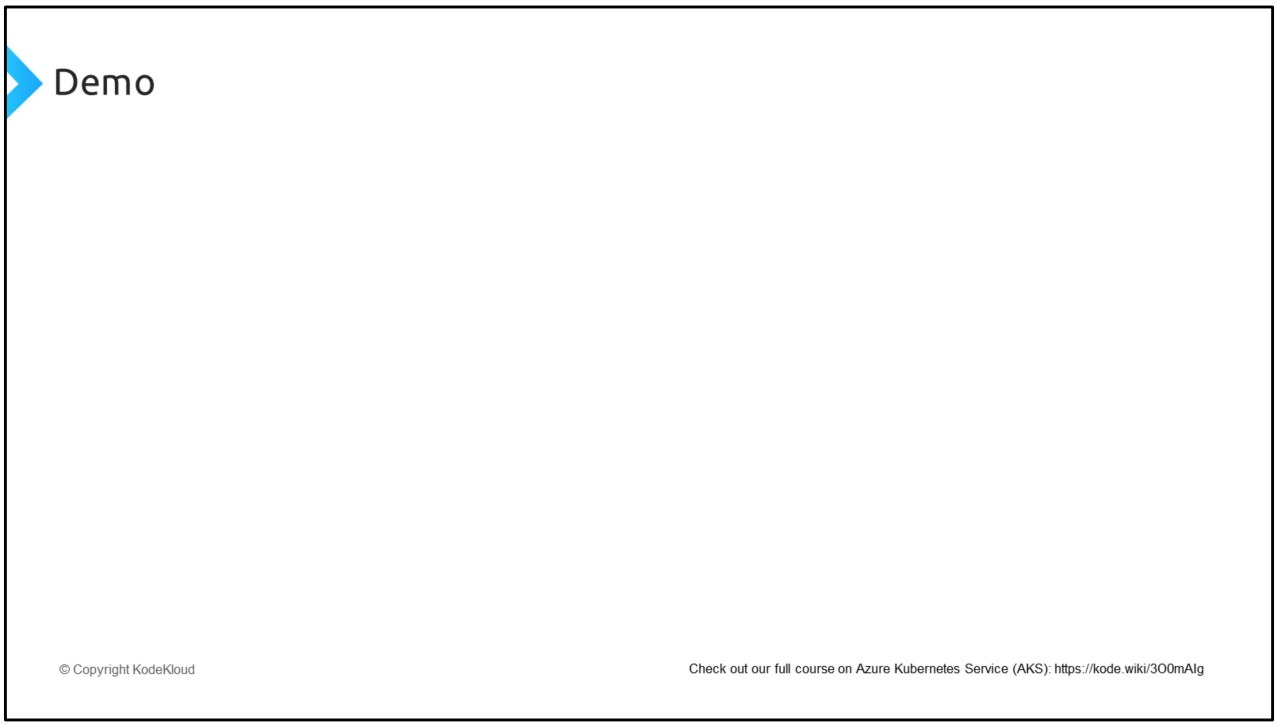


Before we talk about Azure Policy for AKS, lets quickly see what is Azure Policy and why and how do you use it.

as an organization, you will have many requirements or intents that you would like to enforce or atleast report agaist. For example, lets say you a business critical application and you would like to deploy your resources for that application only in Australia because of data residency and sovereignty requirements. Lets take another scenario where you would like to make sure that none of the AKS deployments use GPU based machines in order to control costs.

Azure Policy is a service that enables you to define and enforce governance and compliance rules for your resources. It provides a centralized and consistent way to apply and enforce policies across Azure subscriptions, resource groups, or individual resources. With Azure Policy, you can establish rules that govern resource configurations, access controls, and adherence to specific compliance requirements. These policies are defined using JSON-based policy definitions and can be customized to align with your organization's unique needs. By implementing Azure Policy, you can ensure that your Azure environment remains secure, compliant, and well-governed,

promoting best practices and minimizing risks associated with misconfigurations or non-compliant resources.



Lets jump on Azure portal and deploy a built-in policy to make sure that none of the AKS admins can deploy resources outside Australia.

On the Azure portal, lets open Policy blade. Under Authoring, I'll go to Definitions. Here you can see 100s of pre-defined definitions that you can use out of box. There are two definition types – Policy and Initiative. Initiative is a bucket with multiple policies inside it. It is useful where you have a bunch of related policies. For example I can create a KodeKloud AKS policy initiate with all AKS polices clubbed together . As you can see there are a number of pre-defined initiates like this PCI DSS ones which I can use in case I am deploying a workload that need to report against PCI DSS standards.

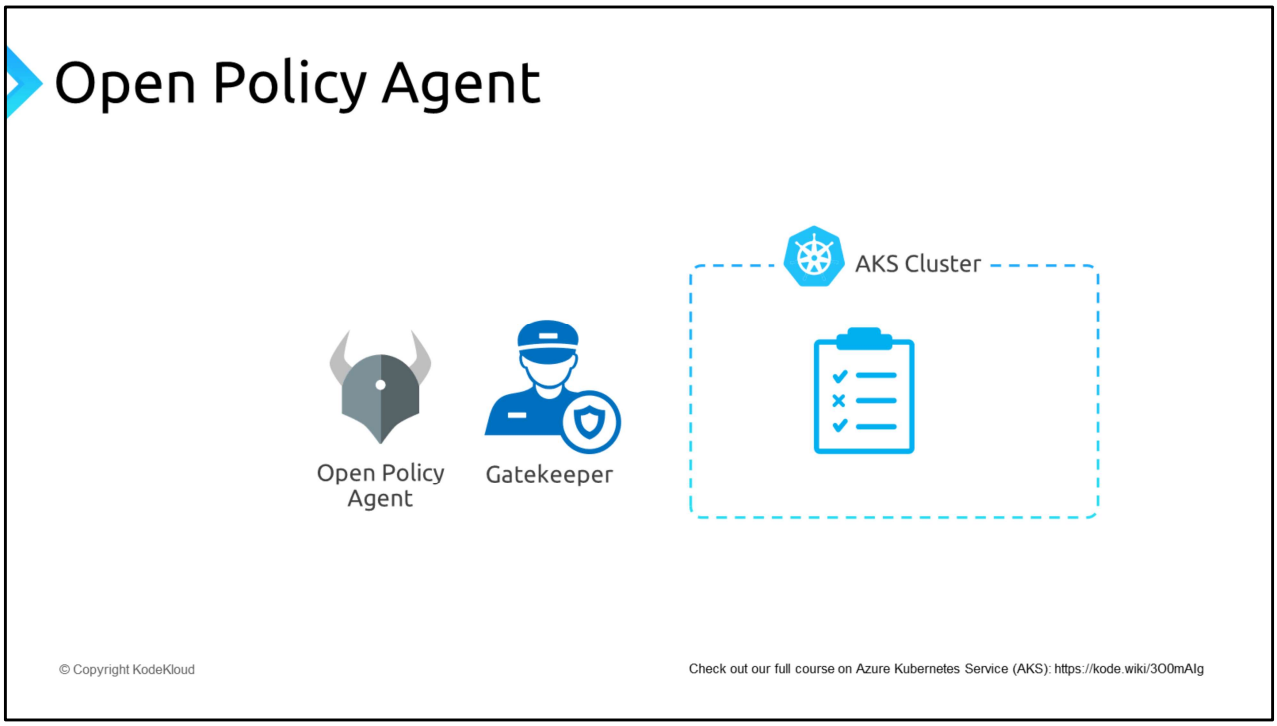
Coming back to the task at hand, I'l looking for a location based policy, so lets search with the keyword location.. And there.. Allowed locations policy is what we need. Lets click on Assign. The scope is the level at which this policy will be assigned. It can either be at a subscription level or you select a resource group under a subscription. I'll select just my subscription. You can optionally change the Assignment Name and add any description. I'l click Next. Here I'll select regions that will be in my allowed list. Any location outside this list will be blocked.

In the remediation tab, you can create a managed identity to move any resources created before this policy to an allowed region. If you need to apply this to your environment, be mindful of any impact this might cause. I'l click next. Lets add a message which admins will see if they run a non-compliant deployment.

Message from Kode Kloud Admin: You are allowed to deploy to only Australian regions for this subscription

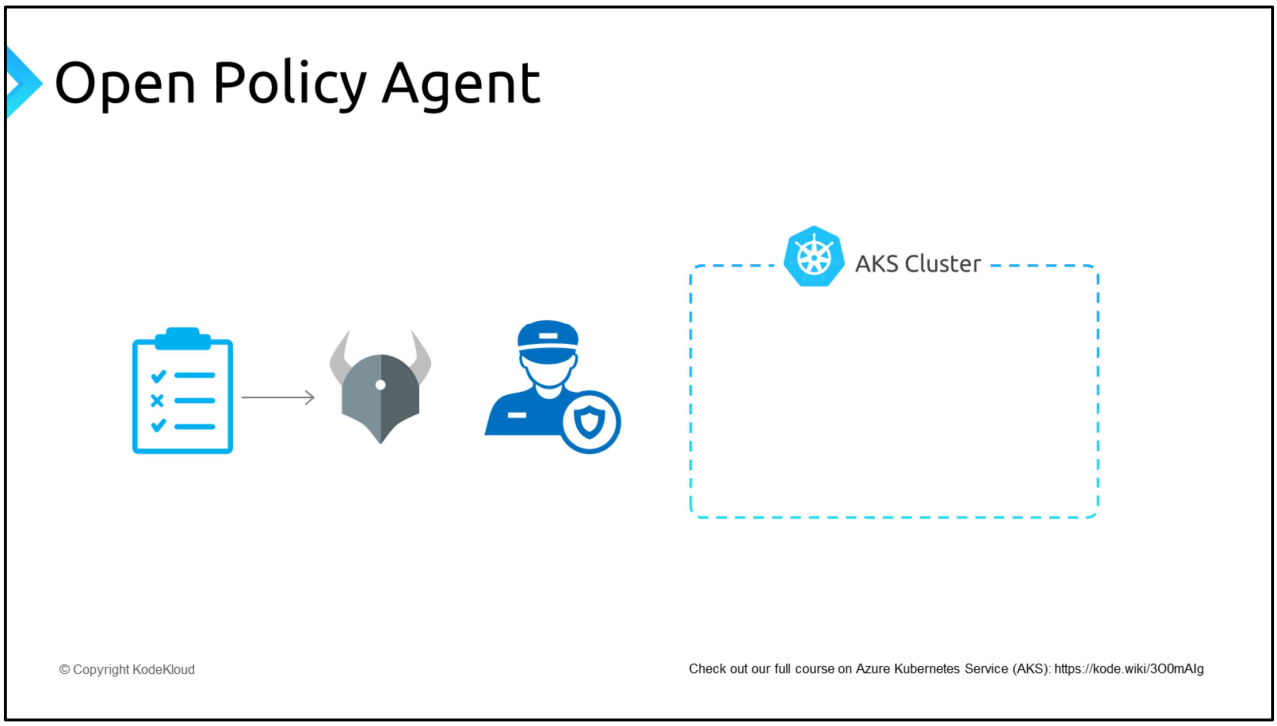
Then lets click create.

In order to validate if our policy works as expected, lets try to create a new AKS cluster. I'll give the cluster a name and choose a location outside Australia. Since we do not need to make any other changes, click Review and create. Azure resource manager engine runs the validation to determine if the deployment will succeed as as expected the validation has failed. If we look at the details, we can see our super helpful message that we had added in the policy. Now that you some background on Azure policies, lets see how Policies work in AKS

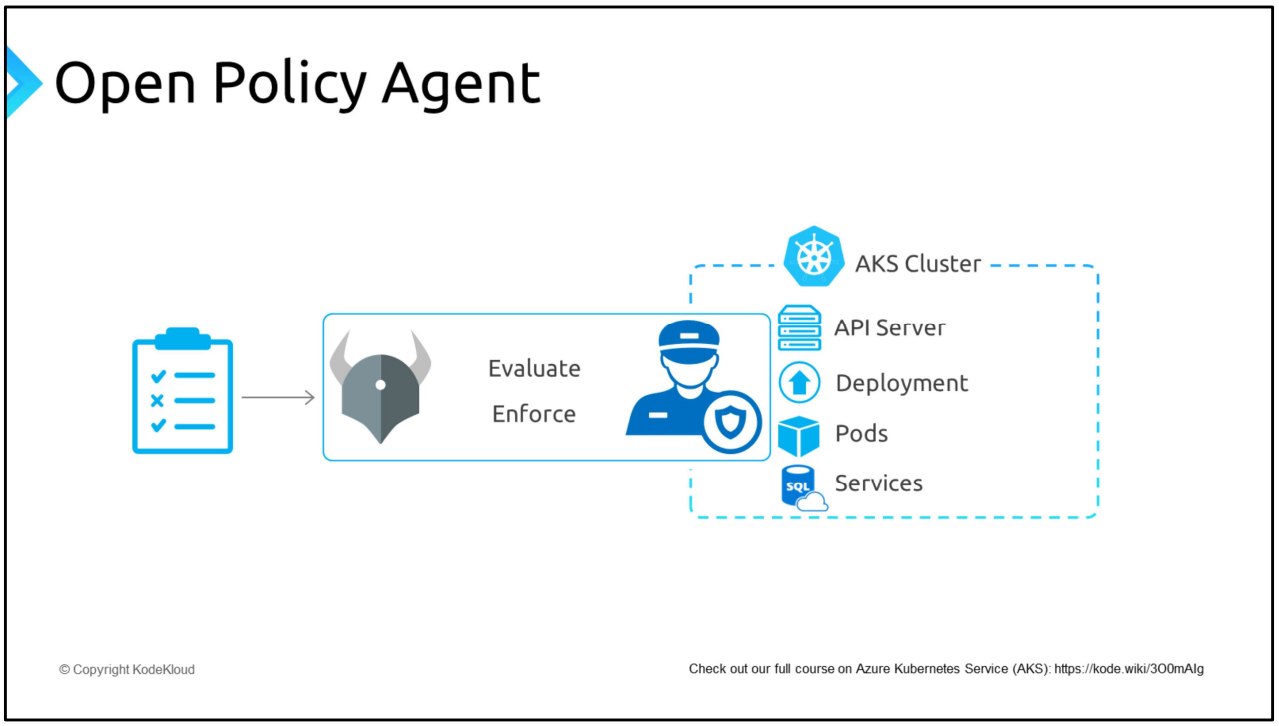


Native Azure policy does not work directly on AKS Clusters. This is where Open Policy Agent or OPA and Gatekeeper come into picture. OPA is an open-source policy engine that is used for policy enforcement

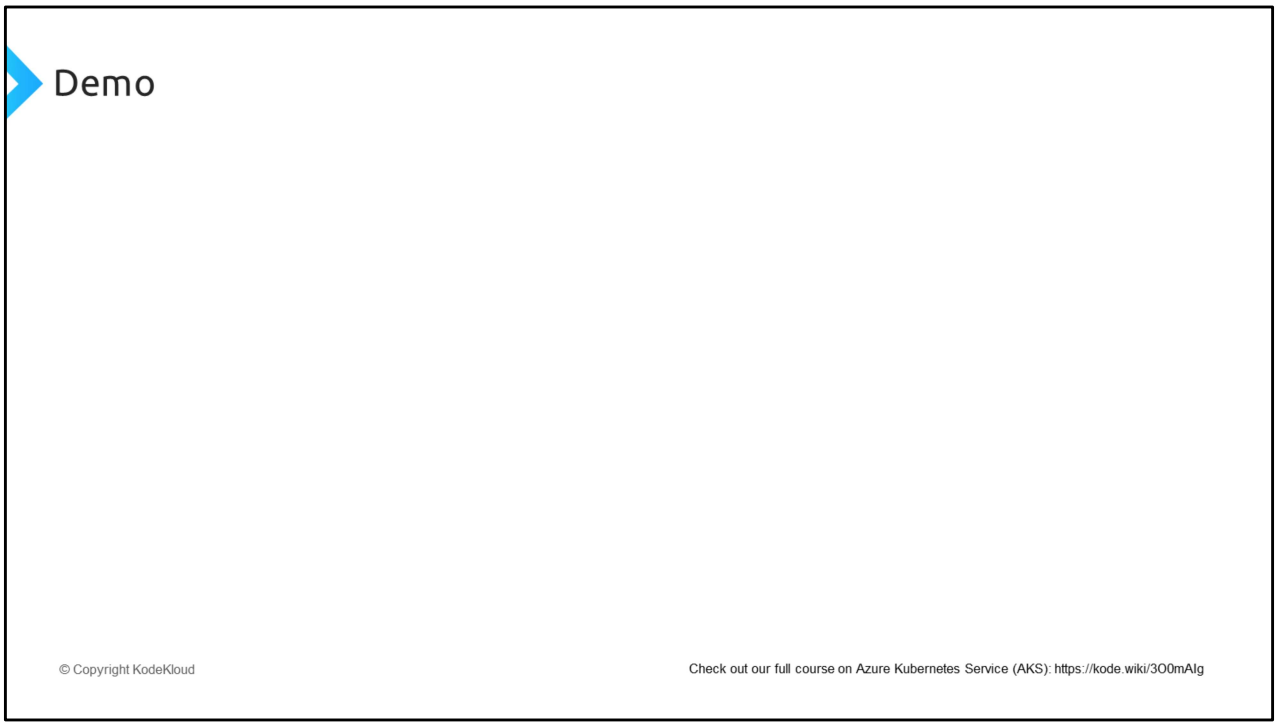
Gatekeeper extends the capabilities of OPA by providing a Kubernetes-native integration for policy enforcement.
Gatekeeper leverages the OPA engine to evaluate policies and enforce them as admission control policies in Kubernetes clusters. It integrates with the Kubernetes API server and acts as a validating webhook, intercepting and evaluating requests made to the cluster. By using Gatekeeper, you can define and enforce custom policies to validate and control various aspects of Kubernetes resources, such as deployments, pods, services, and more.



Native Azure policy does not work directly on AKS Clusters. This is where Open Policy Agent or OPA and Gatekeeper come into picture. OPA is an open-source policy engine that is used for policy enforcement



Gatekeeper extends the capabilities of OPA by providing a Kubernetes-native integration for policy enforcement.
Gatekeeper leverages the OPA engine to evaluate policies and enforce them as admission control policies in Kubernetes clusters. It integrates with the Kubernetes API server and acts as a validating webhook, intercepting and evaluating requests made to the cluster. By using Gatekeeper, you can define and enforce custom policies to validate and control various aspects of Kubernetes resources, such as deployments, pods, services, and more.



So lets see the Azure Policy for AKS in action . I have created a brand new AKS cluster with default options and I have called it AKSPolicyDemo. If I click on the Policies blade in the cluster, you can see that I have an option to enable the add-on from here. This also indicated at the moment its disabled. Lets also verify this using command line.

Lets run Az aks show and query the addonprofiles

Az aks show –g AKSPolicyDemo_group –n AKSPolicyDemo –query addonProfiles.azurepolicy

And this shows that the azure policy addon is currently disabled.

So lets enable Azure policy for aks via command line using az aks enable-addons comand

az aks enable-addons --addons azure-policy --n AKSPolicyDemo --g AKSPolicyDemo

This will install azure-policy pod is in kube-system namespace and gatekeeper pod is installed in gatekeeper-system namespace

Lets quickly validate that too by running

kubectl get pods -n kube-system | grep policy

Here you can see the two components running – one is Azure policy and another one the associated webhook. These are orchestrators that sit behind the Azure policy addon that we just enabled. These pod synchronize the policy that we deploy from Azure policy to Kubernetes. It is worth noting that the add-on checks in with Azure Policy service for changes in policy assignments every 15 minutes.

And
kubectl get pods -n gatekeeper-system

The gatekeeper is an opensource component that runs inside the cluster to do the enforcement of the policies. So while we assign the policies via azure policy it is the gatekeeper inside AKS that is doing the hard work of making sure that those policies are applied inside aks.

Now that Policy add-on is installed, lets go to Azure policy and apply a simple policy to ensure that AKS can only be configured to listen to allowed ports.

I'll go to Policy, then click on Definitions
I'll then search for Search for Kubernetes Policy Definitions.

I have selected Policy Definition **Kubernetes cluster services should listen only on allowed ports** which allows us to restrict ports to be used for Cluster Services. You have been this process before, I'll Assign the Policy and complete the necessary fields, allowing only port 80 and 443.

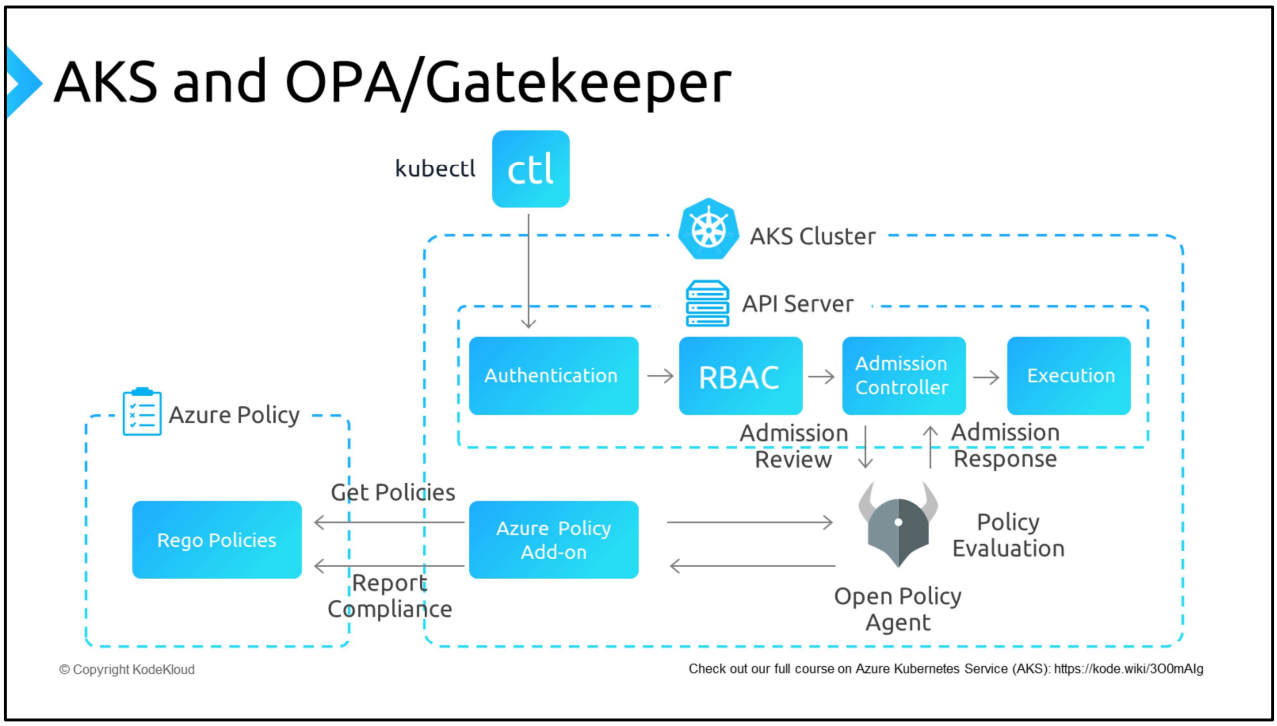
Now lets this deploy this simple yaml file to our AKS cluster. Note that this will deploy nginx and use TCP port 80

Do you this gatekeeper will allow this deployment?

Yes offcourse, we have allowed ports 80 and 443

Lets now try to deploy ngnix but this time lets use port 8080 and see the result.

As you can the see the deployment has failed with the custom error message we have set when we deployed the policy.

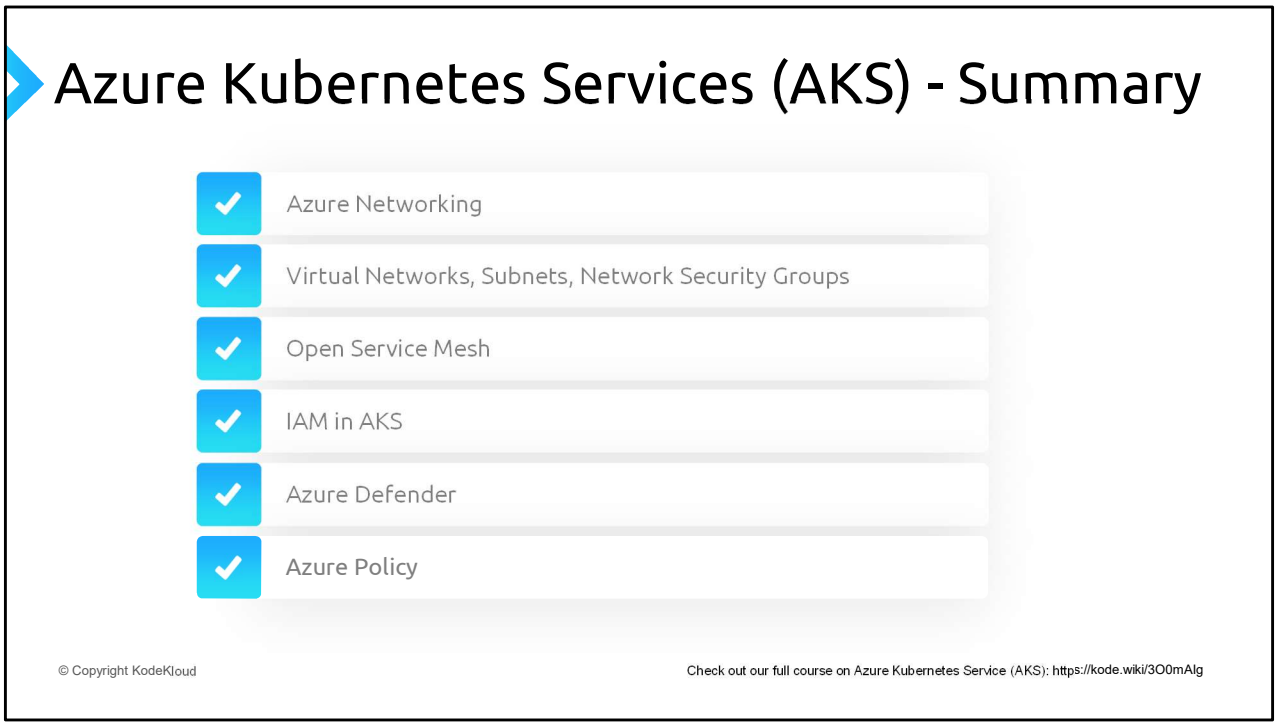


Let's take a look at how these components work together. The Azure managed Kubernetes control plane uses OPA Gatekeeper to enforce policies on behalf of Azure Policy.

Inside the Kubernetes (K8s) cluster, we have the API server. We can use kubectl to interact with this component by sending requests to it. When we do that, the request goes through a pipeline of components. First, it goes through authentication, followed by authorization or RBAC. Next, it passes through optional admission controllers, which can be multiple in the cluster. Only after these three stages have passed, the command is executed.
Azure Policy works alongside Gatekeeper and integrates with the Admission Controller module. On the left side, we have Azure Policy. When we enable the policy add-on, Azure Policy and Gatekeeper are deployed in the cluster.
When we assign a policy or initiative to a subscription or resource group where the AKS cluster is hosted, the Azure Policy add-on synchronizes the policy details from Azure Policy into Gatekeeper. Gatekeeper is configured to integrate with the admission controller. So, when a request comes into the API server, the admission controller sends the request to Gatekeeper. Gatekeeper then validates if the request complies with the synchronized policies. It sends

the response back to the admission controller, and based on the compliance status, the request will proceed accordingly.

This brings us to the end of the video on Azure policy for AKS.



Ok, we have now reached the culmination of this video course on Azure Kubernetes Service (AKS). Our journey began by delving into the fundamental aspects of Azure, encompassing essential concepts such as Compute, Storage, and Networking. With a some foundation in place, we transitioned towards the creation and containerization of a basic C# web application. Throughout that module, we focused on leveraging Docker desktop and crafting Dockerfiles to facilitate containerization.

Subsequently, we navigated to the Azure portal to establish an AKS cluster to deploy our world class application onto that cluster. Harnessing the power of kubectl, we effectively executed various imperative actions, including scaling of deployments. we explored the deployment of container images within the Azure Container Registry (ACR), unearthing its potential for application enhancement and version management. By harnessing the image hosted on ACR, we facilitated a seamless application upgrade, only to effortlessly revert to a previous version when needed. Concluding this module, we bestowed upon you a overview of the Kubernetes Fleet, empowering you with a holistic understanding of this powerful toolset.

The subsequent module, entirely dedicated to AKS networking options and policies,

showcased the indispensable importance of robust network management within AKS deployments. Furthermore, we delved into security-centric topics, delving into the intricacies of Azure AD integration, defender for AKS, and Azure policy for AKS. Empowered with this knowledge, you are poised to fortify your AKS infrastructure with optimal security measures and governance practices.

As we progressed, we ventured into the realm of CI/CD patterns within AKS, looking at the art of managing AKS using declarative techniques. By adopting these practices, you can streamline the continuous integration and delivery pipelines within your AKS environment.

An indispensable component of any infrastructure, observability received some attention in module seven. We explored observability within AKS, equipping you with the necessary tools and techniques to monitor and gain valuable insights into your AKS deployments.

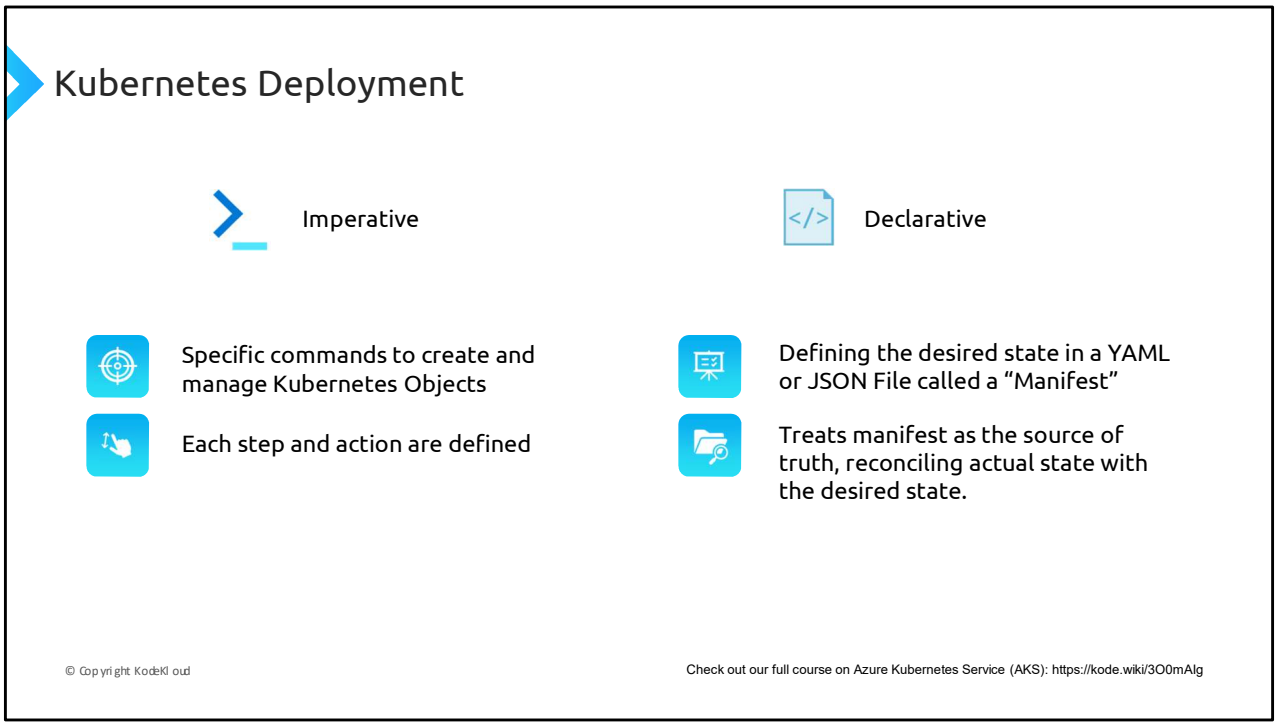
Finally, before we brought this course to a close, we presented you with an overview of alternative platforms that embrace the power of containers. By expanding your horizons and exploring these platforms, you can unlock a world of possibilities and further enhance your containerization endeavors.

KodeKloud

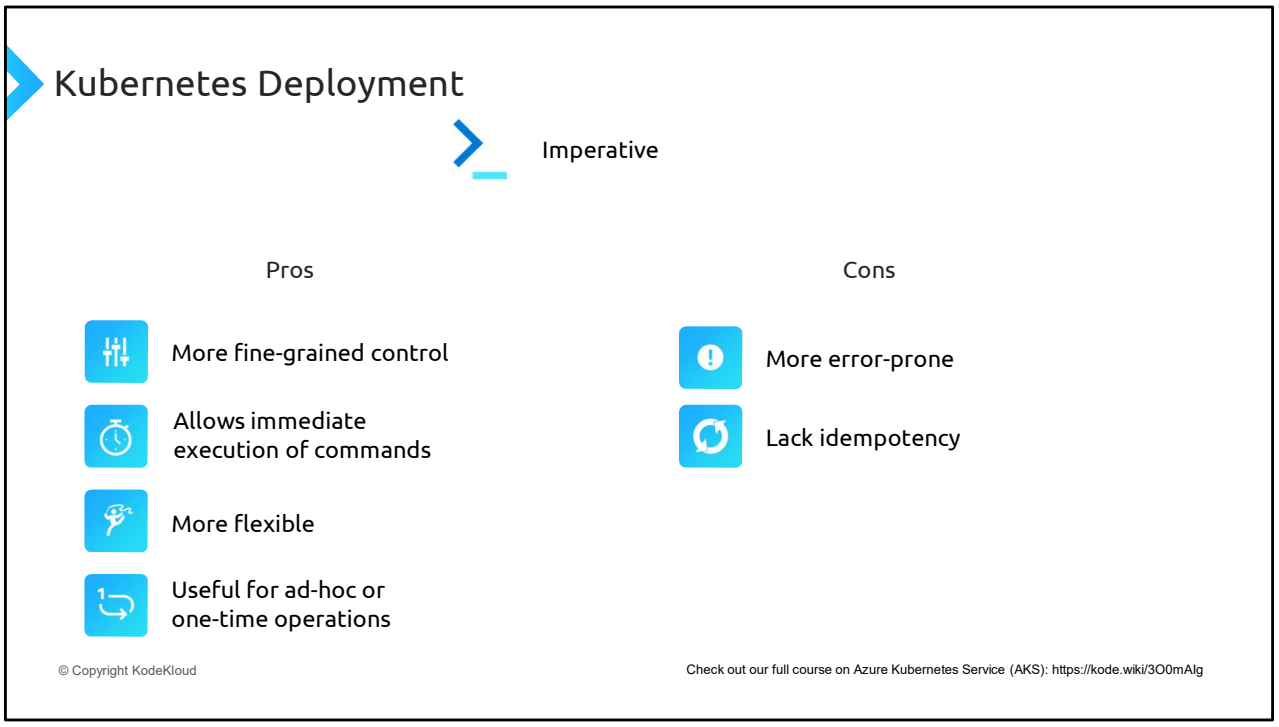
Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

CI/CD Workflow for AKS

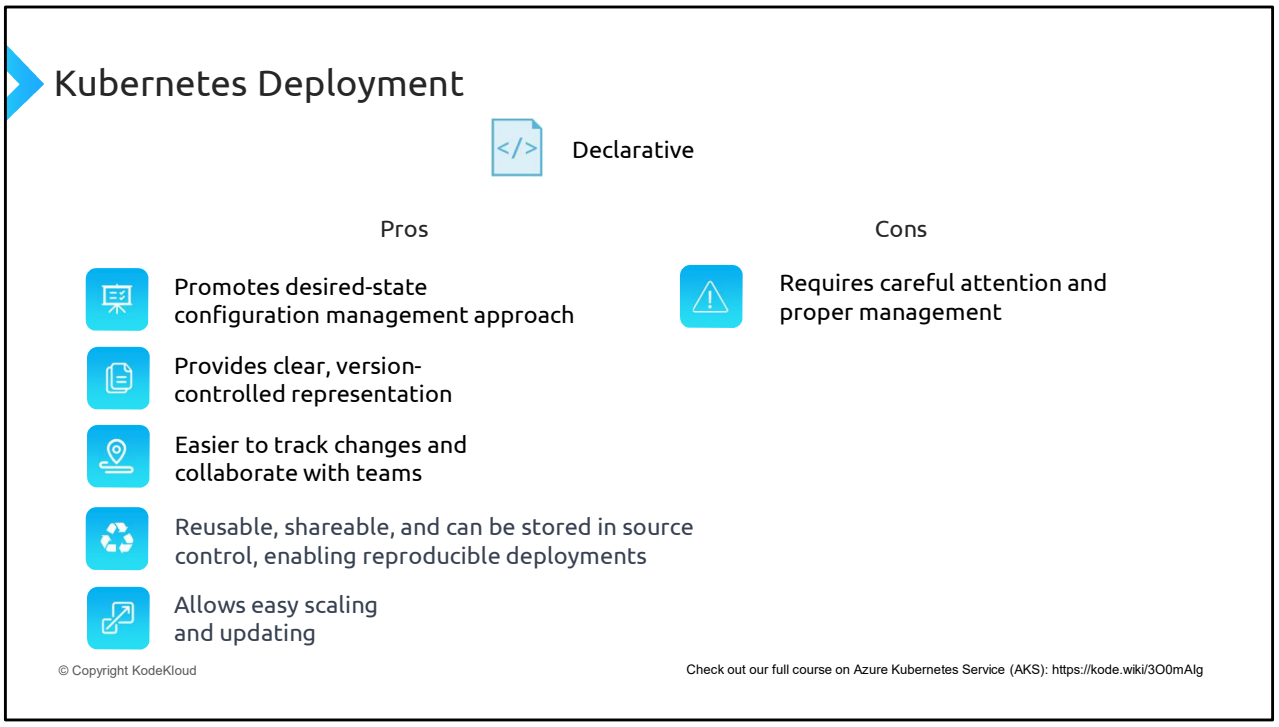
Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg



In Kubernetes, there are two primary ways to deploy applications: Imperative and Declarative.
Imperative deployment involves issuing specific commands to create and manage Kubernetes objects. With imperative commands, you define each step and action required to deploy and configure your application. For example, in module 4 we used imperative commands via kubectl run to create a deployment, then use kubectl expose to create a service, and so on.

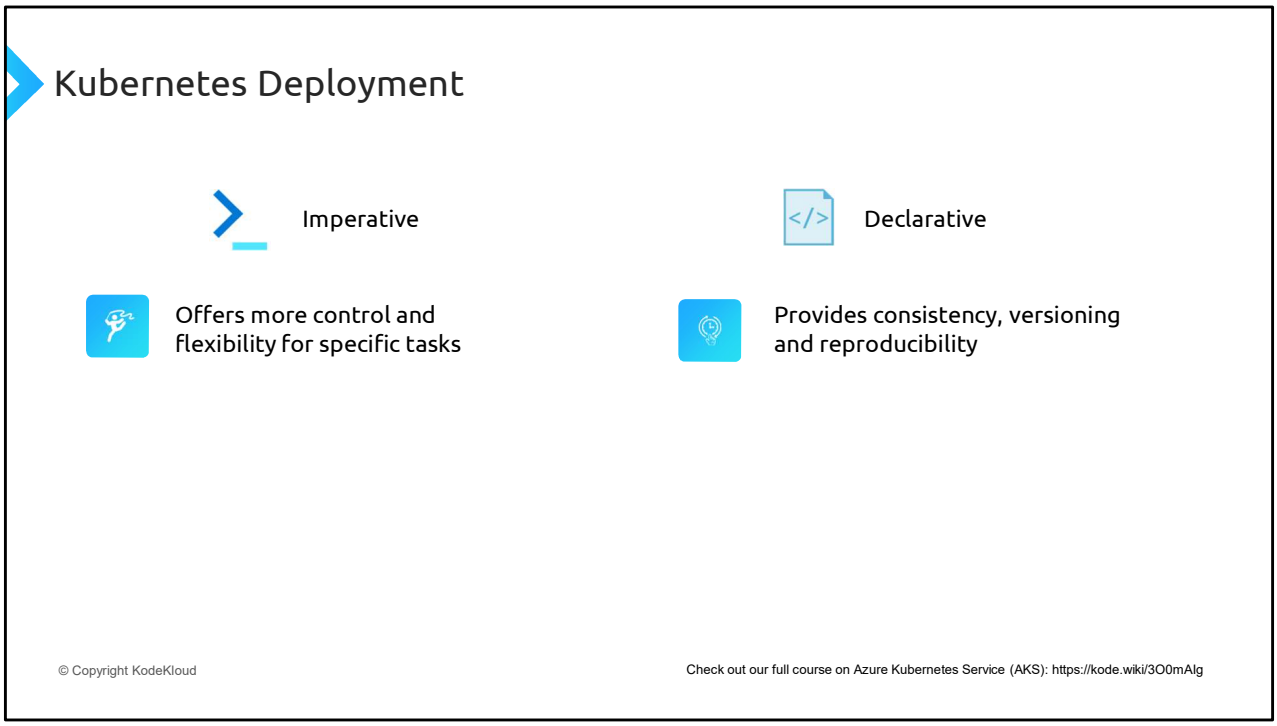


Now this is great for learning AKS and to get started with Kubernetes. In most environments, you will get exposed to Declarative way of deploying your applications on AKS.
Declarative deployment involves defining the desired state of your application in a YAML or JSON file called a manifest. You specify the desired configuration and characteristics of your application, and Kubernetes takes care of managing the resources to achieve that desired state. This approach treats the manifest as the source of truth, allowing Kubernetes to reconcile the actual state with the desired state.



Each approach has its pros and cons.

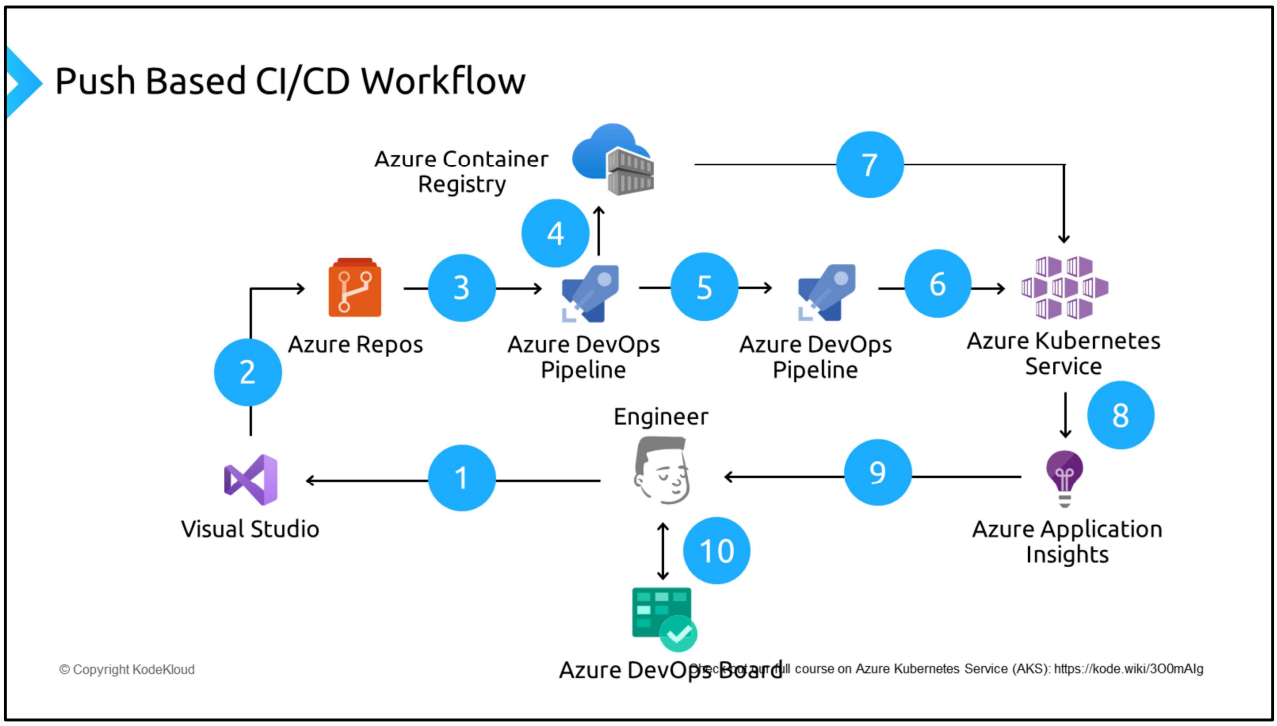
Imperative deployment offers a more fine-grained control over the deployment process. It allows for immediate execution of commands and provides more flexibility in managing individual resources. It is useful for ad-hoc or one-time operations, such as troubleshooting or quick prototyping. However, imperative commands can be more error-prone, especially in complex deployments, and they lack idempotency, meaning that re-running the same command may result in different outcomes. Declarative deployment, on the other hand, promotes a desired-state configuration management approach. It provides a clear, version-controlled representation of the application's configuration and makes it easier to track changes and collaborate with teams. Declarative manifests are reusable, shareable, and can be stored in source control, enabling reproducible deployments. It also allows for easy scaling and updating of deployments. However, declarative deployment requires careful attention to the manifest file and proper management of changes to avoid unintended consequences.



In summary, imperative deployment offers more control and flexibility for specific tasks, while declarative deployment provides consistency, versioning, and reproducibility benefits. The choice between them depends on the specific use case, the complexity of the deployment, and the desired level of control and automation. In practice, a combination of both approaches is often used, leveraging the strengths of each for different stages of the application lifecycle.

Push-based CI/CD Workflow

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg



While, we have taken shortcuts, a best practice deployment using Microsoft DevOps toolset wil look like this

1. Create the application source code, if you recall we created an ASP.net based application in Module 3.
2. We then Commit Application Code to azure repos
3. Continuous integration then triggers application build, container image is build and any unit tests that you might have configured will run. Because you are an awesome dev, this unit test will all succeed

4. In Step 4, Azure DevOps pipeline will push the Container image to Azure Container Registry.

5. Continuous deployment trigger orchestrates deployment of application artifacts with environment-specific parameters.

6. In Step 6, DevOps pipeline will Deployment to Azure Kubernetes Service

(AKS) cluster

7. Container is launched using Container Image from Azure Container Registry that we had saved in step 4

8. Optionally, Application Insights can collect and analyze health, performance, and usage data.
9. As an engineer, you can then Review health, performance, and usage information.

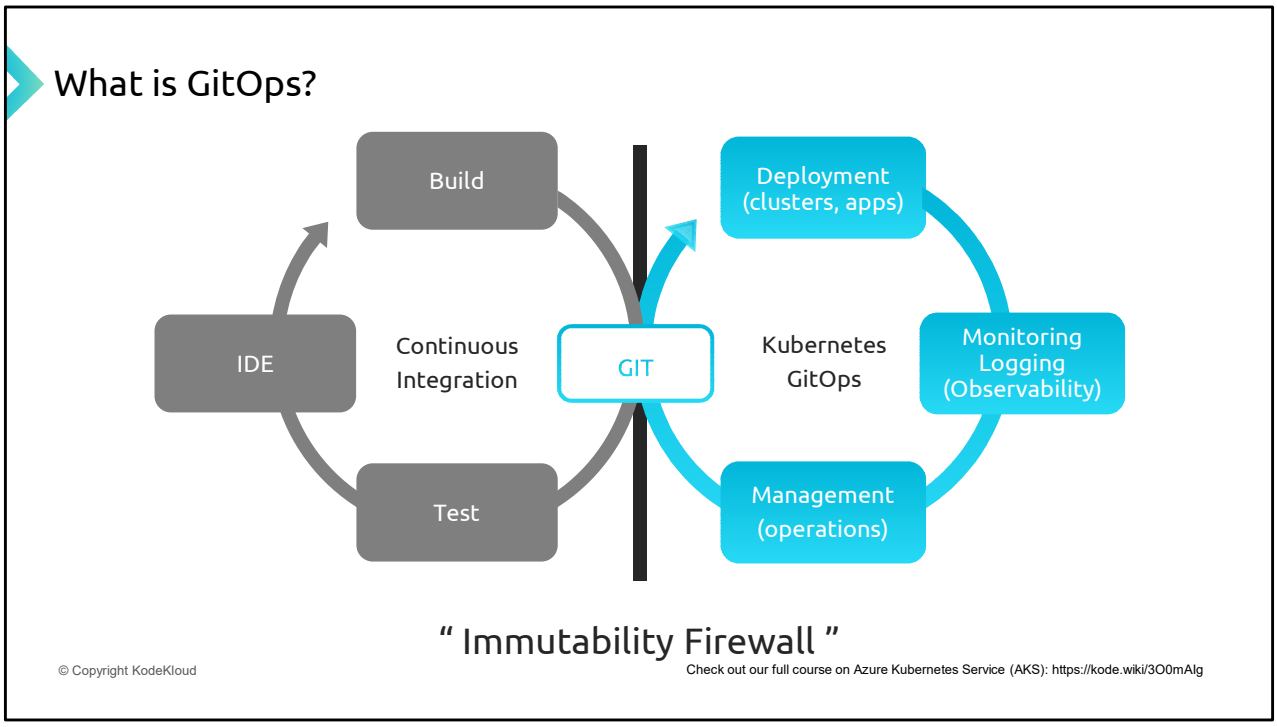
10. Finally, you can Update any backlog items.

You can offcourse, use any devops toolset, I have used Azure devops as an example. Now that you have seen a sample push based pipeline, lets see how the pull based pipeline is different.

Note Visio file for this image is : https://arch-center.azureedge.net/cicd-for-containers.vsdx

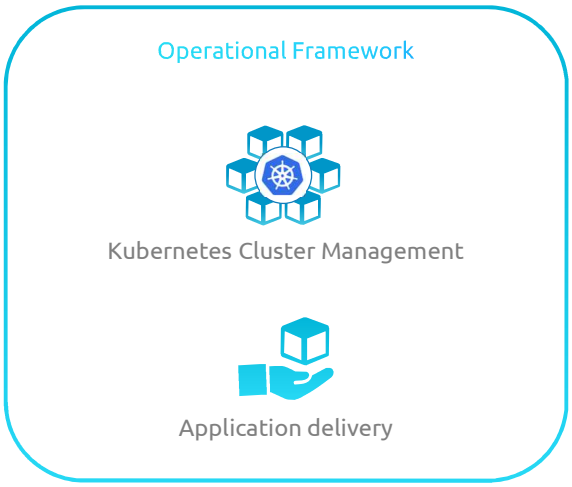
Pull-based CI/CD Workflow

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg



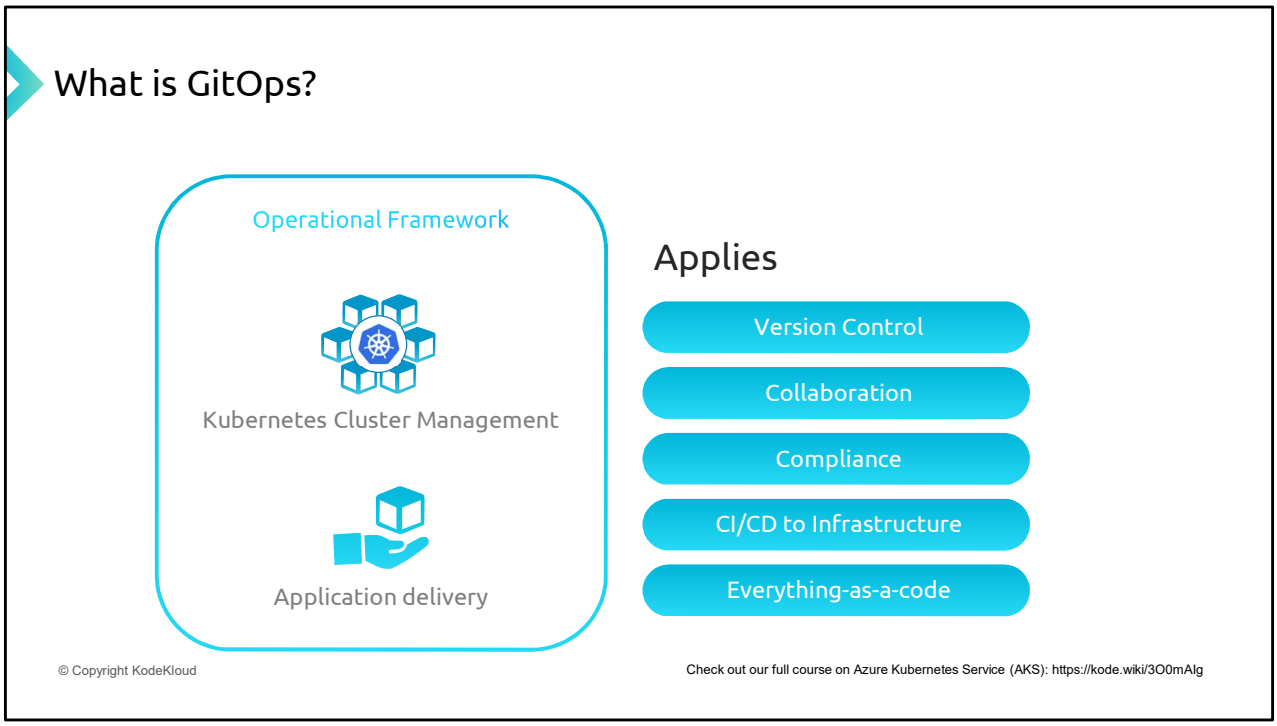
Before we discuss, pull based CICD, we need to understand what is GitOps. So here is the basic concept.

What is GitOps?



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

GitOps is a methodology for managing and automating application deployment. It involves storing application configurations and infrastructure as code in a Git repository.



Changes to the desired state are made through Git commits, as you would in Push based method as well but in this method the git commits triggers automated synchronization with the Kubernetes cluster using gitops tools such as Flux or Argo CD to ensure that the cluster state matches the desired state, enabling easy auditing and rollbacks.

What is GitOps?

Enabled by automation and
sync tools like FluxCD and ArgoCD

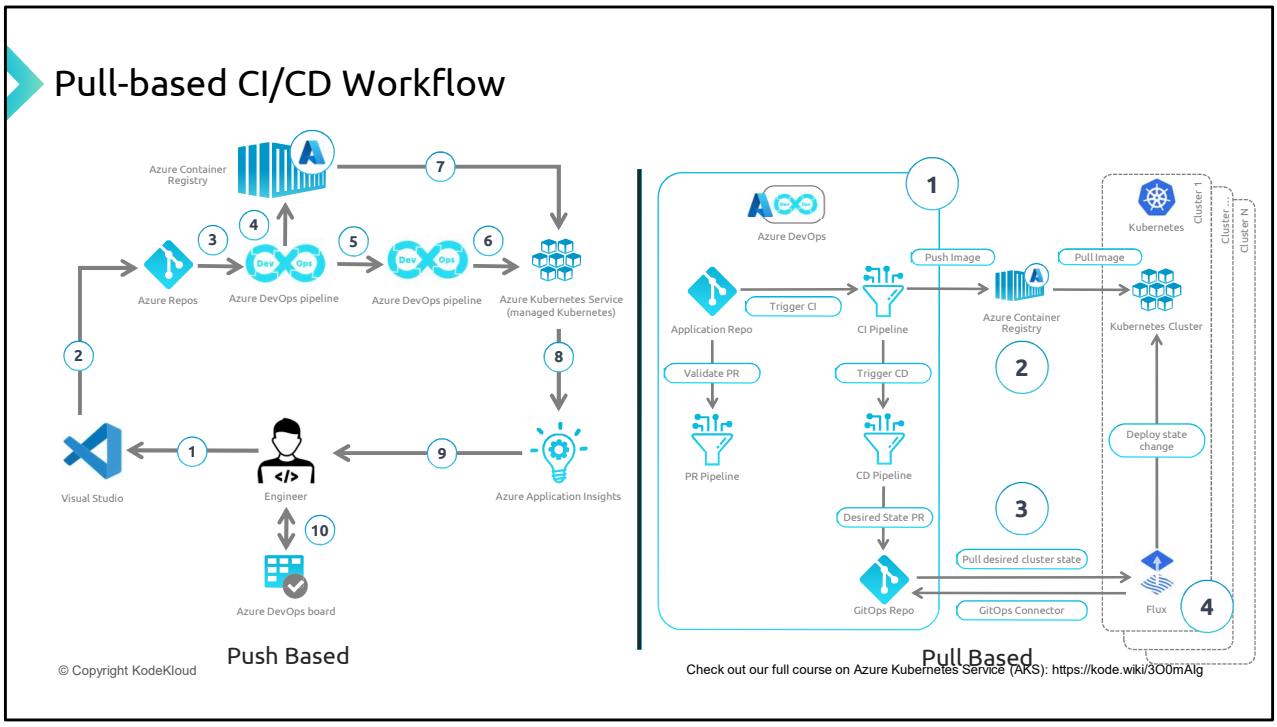
Operational Framework

Kubernetes Cluster Management

Application delivery

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

It promotes collaboration, continuous integration, and provides a reliable, version-controlled approach to managing Kubernetes applications



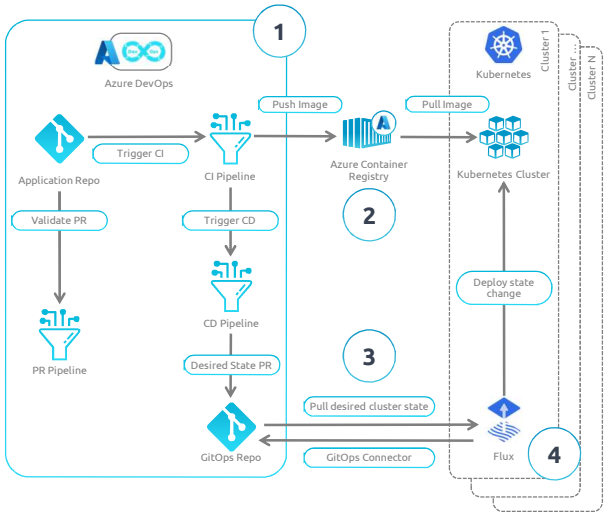
Lets now see who this differs to the Push based workflow we saw before. Two GitOps operators that you can use with AKS are Flux and Argo CD. They're both (CNCF) projects and are widely used.

While the outcome remains the same, there are couple of differences I want to highlight here. In the Pull based workflow, the GitOps operator – flux in this digram, takes care of monitoring and applying changes to the AKS cluster based on the Git repository's desired state. Developers simply make changes to the repository, and the operator ensures that the changes are propagated to the cluster. It provides a declarative and automated approach to managing and deploying applications on AKS, promoting consistency and traceability.

Overall, the push- based CI/CD workflow offers immediate feedback and deployment triggered by direct code pushes, while the pull- based (GitOps) workflow promotes automation and declarative management of AKS clusters based on changes made to a Git repository. The choice between these workflows depends on factors such as deployment speed requirements, desired level of automation, and team preferences.

Pull-based CI/CD Workflow

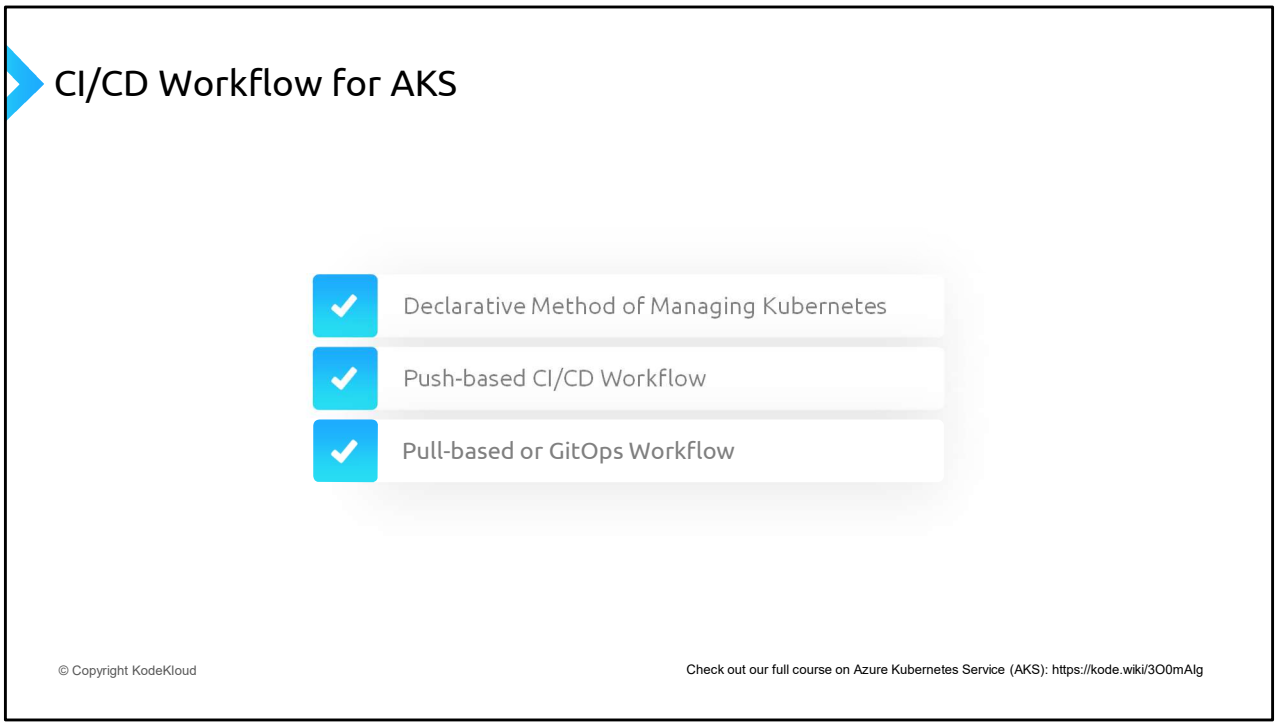
1. **Application Repo and Build Pipeline**
   This pipeline runs the basic quality gates on the application code repo
2. **CI Pipeline and Image Publication**
   The CI pipeline build the project, checks quality, builds and pushes images. It kicks of the CD pipeline automatically by publishing the templates.
3. **CD Pipeline and GitOps Repo**
   The CD pipeline is automatically triggered by successful CI builds. In this pipeline environment, environment-specific values are substituted into the previously published templates, and a new pull request is raised against the GitOps repository with these values.
4. **GitOps Operator and Application Updates**
   The GitOps repository represents the current desired state of all environments across clusters. Any change to this repository is picked up by the Flux service in each cluster and deployed.



Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

Lets talk a lttle bit more about the steps involved in pull based methos.

the Pull-based CI/CD workflow for AKS involves maintaining an application repository, executing build and CI pipelines to validate and publish images, triggering a CD pipeline to generate environment-specific configurations and create pull requests against the GitOps repository, and finally, leveraging the GitOps operator to apply changes to the target environments based on the desired state defined in the repository.



Ok, we have now reached the culmination of this video course on Azure Kubernetes Service (AKS) . Our journey began by delving into the fundamental aspects of Azure, encompassing essential concepts such as Compute, Storage, and Networking. With a some foundation in place, we transitioned towards the creation and containerization of a basic C# web application. Throughout that module, we focused on leveraging Docker desktop and crafting Dockerfiles to facilitate containerization.

Subsequently, we navigated to the Azure portal to establish an AKS cluster to deploy our world class application onto that cluster. Harnessing the power of kubectl, we effectively executed various imperative actions, including scaling of deployments. we explored the deployment of container images within the Azure Container Registry (ACR) , unearthing its potential for application enhancement and version management. By harnessing the image hosted on ACR , we facilitated a seamless application upgrade, only to effortlessly revert to a previous version when needed. Concluding this module, we bestowed upon you a overview of the Kubernetes Fleet, empowering you with a holistic understanding of this powerful toolset.

The subsequent module, entirely dedicated to AKS networking options and policies,

showcased the indispensable importance of robust network management within AKS deployments. Furthermore, we delved into security-centric topics, delving into the intricacies of Azure AD integration, defender for AKS, and Azure policy for AKS. Empowered with this knowledge, you are poised to fortify your AKS infrastructure with optimal security measures and governance practices.

As we progressed, we ventured into the realm of CI/CD patterns within AKS, looking at the art of managing AKS using declarative techniques. By adopting these practices, you can streamline the continuous integration and delivery pipelines within your AKS environment.

An indispensable component of any infrastructure, observability received some attention in module seven. We explored observability within AKS, equipping you with the necessary tools and techniques to monitor and gain valuable insights into your AKS deployments.

Finally, before we brought this course to a close, we presented you with an overview of alternative platforms that embrace the power of containers. By expanding your horizons and exploring these platforms, you can unlock a world of possibilities and further enhance your containerization endeavors.

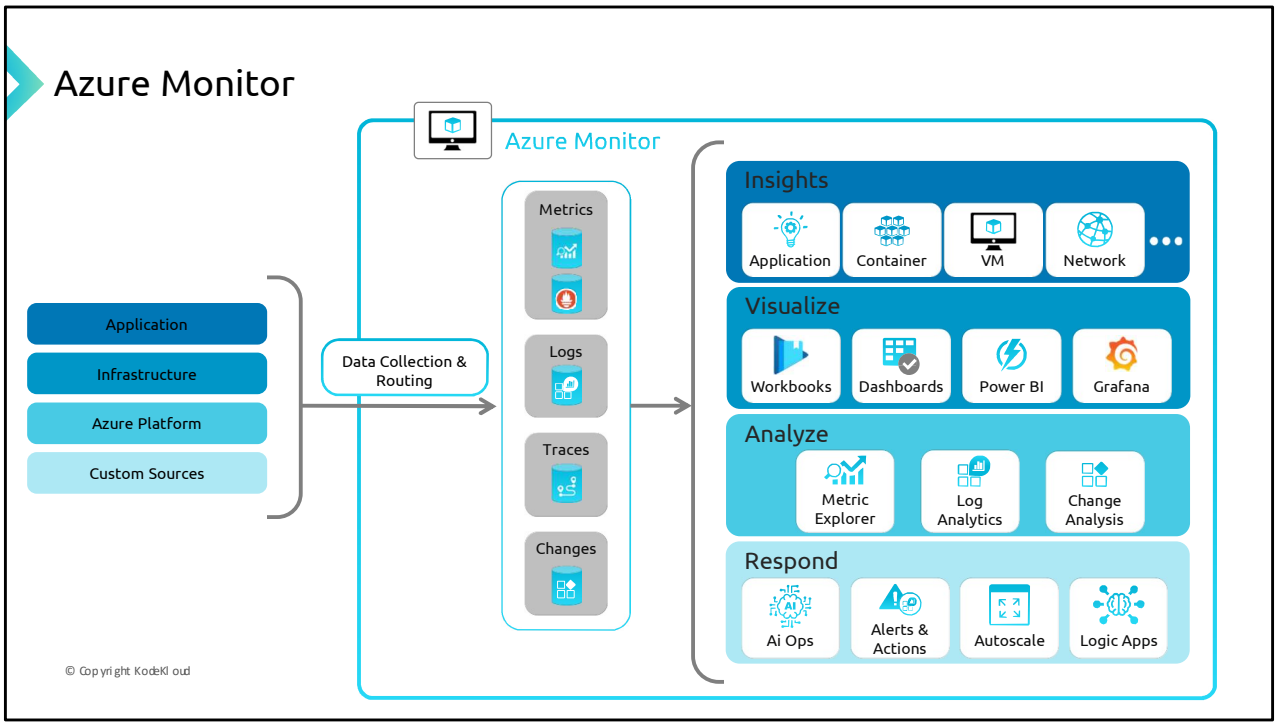
We sincerely hope that your journey throughout this course has been as fulfilling as our commitment to its production. It has been an absolute pleasure to guide you through the intricacies of AKS, empowering you with the knowledge and skills needed to excel in your containerization journey. This is your host Pranav, signing off for now.

KodeKloud

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg

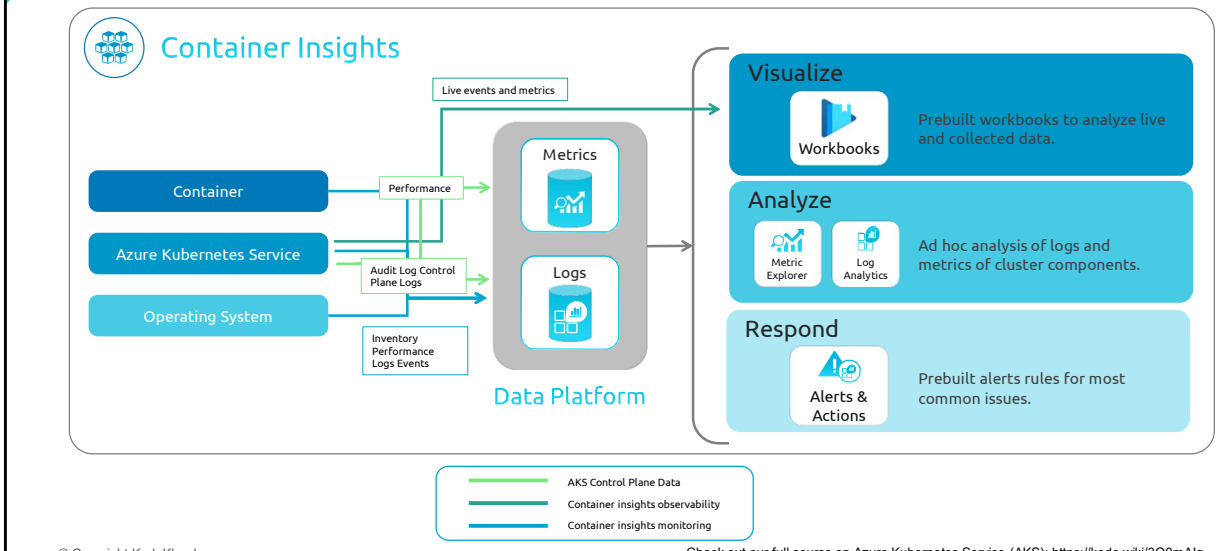
Container Insights for AKS

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg



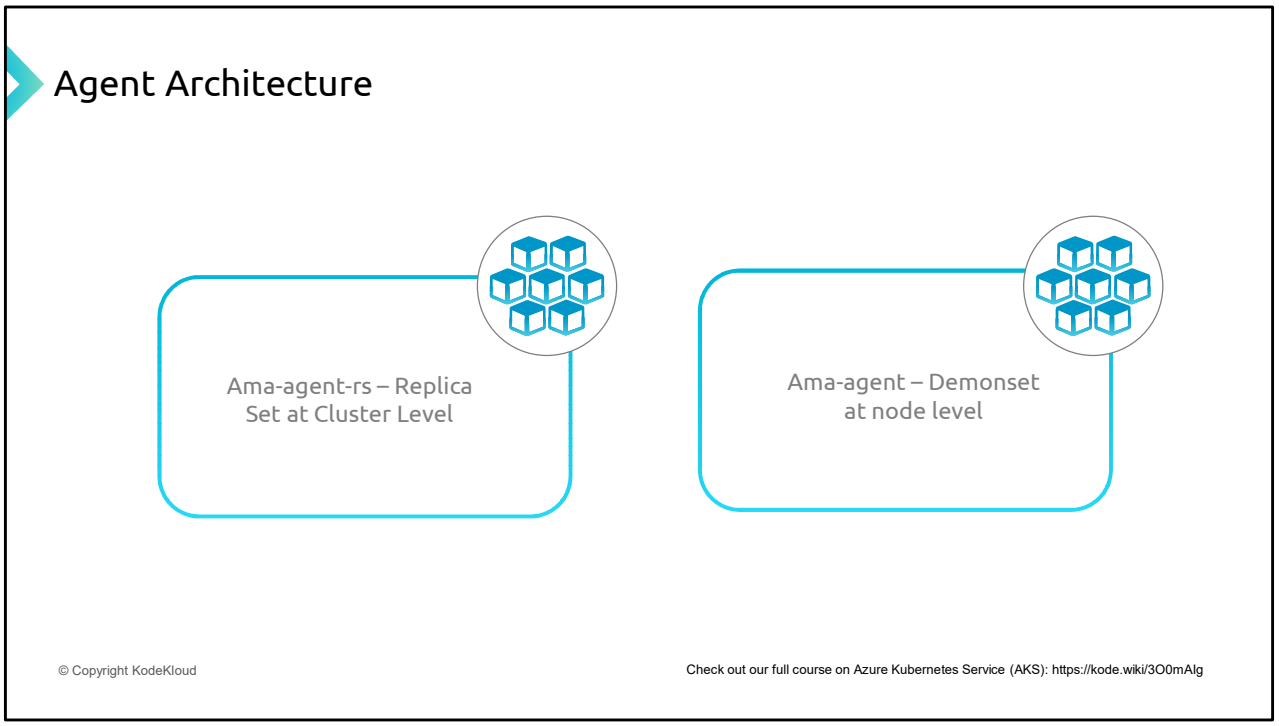
Azure Monitor is a monitoring solution provided by Azure that enables you to gain insights into the performance and health of your applications, infrastructure, and services running on the Azure platform. It offers a range of monitoring capabilities, including metrics, logs, alerts, and diagnostics, to help you effectively monitor and troubleshoot your Azure resources.

Container Insights



Container Insights, a feature within Azure Monitor, and is specifically designed for monitoring and managing containerized workloads deployed on Azure Kubernetes Service (AKS). It provides deep visibility into your AKS clusters and helps you gain insights into the performance, availability, and resource utilization of your containerized applications.

Azure Monitor collects and stores two primary types of data: metrics and logs.
1.Metrics: Metrics are numerical data that represent the performance and health of various resources. In the context of AKS and Container Insights, metrics can include information such as CPU usage, memory consumption, network traffic, and other resource-specific metrics. Azure Monitor aggregates and stores these metrics, allowing you to create charts, set up alerts, and perform analysis over time.
2.Logs: Logs contain structured or unstructured data that provide detailed information about the operations and activities within your AKS clusters. Container logs, application logs, and system logs are examples of log data that can be collected. Azure Monitor allows you to collect, store, and analyze these logs to gain insights into the behavior of your applications and diagnose issues.

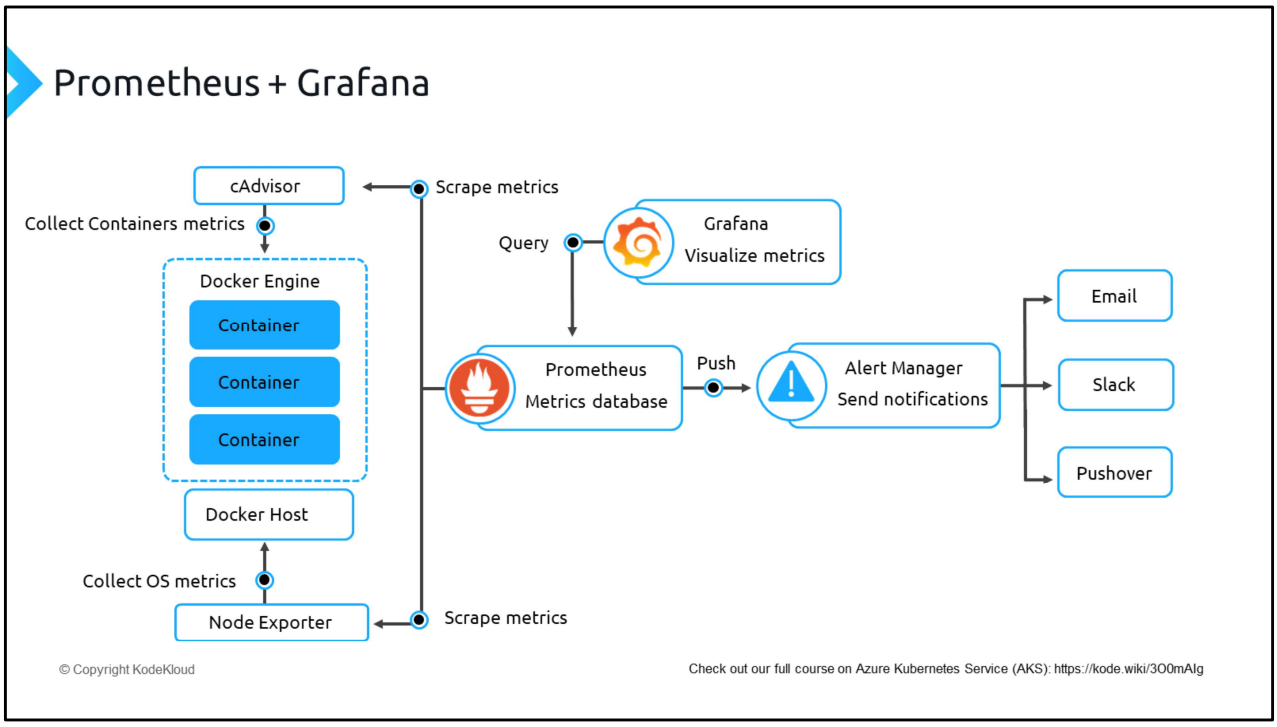


Let's take a moment to discuss the architecture of the AMA agent when enabling the monitoring add-on in AKS. There are two agents that are installed as part of this process.

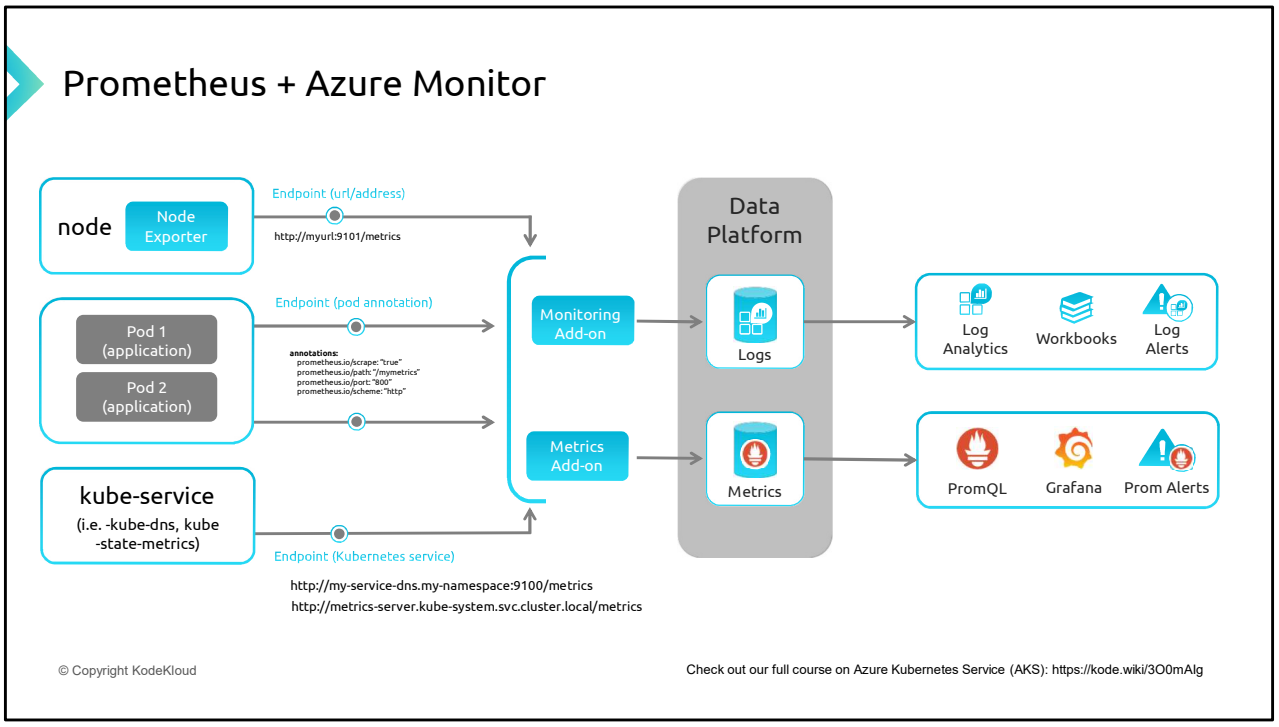
The first agent is the AMA agent replica set, which is deployed on one of the nodes in the cluster. This agent's primary role is to gather metrics at the cluster level. It ensures that even if the agent running on the node where it is deployed goes down, it still collects the necessary information.

The second agent is installed on every node in the cluster. This agent is responsible for collecting metrics at the node level, as well as monitoring all the pods running on that particular node.

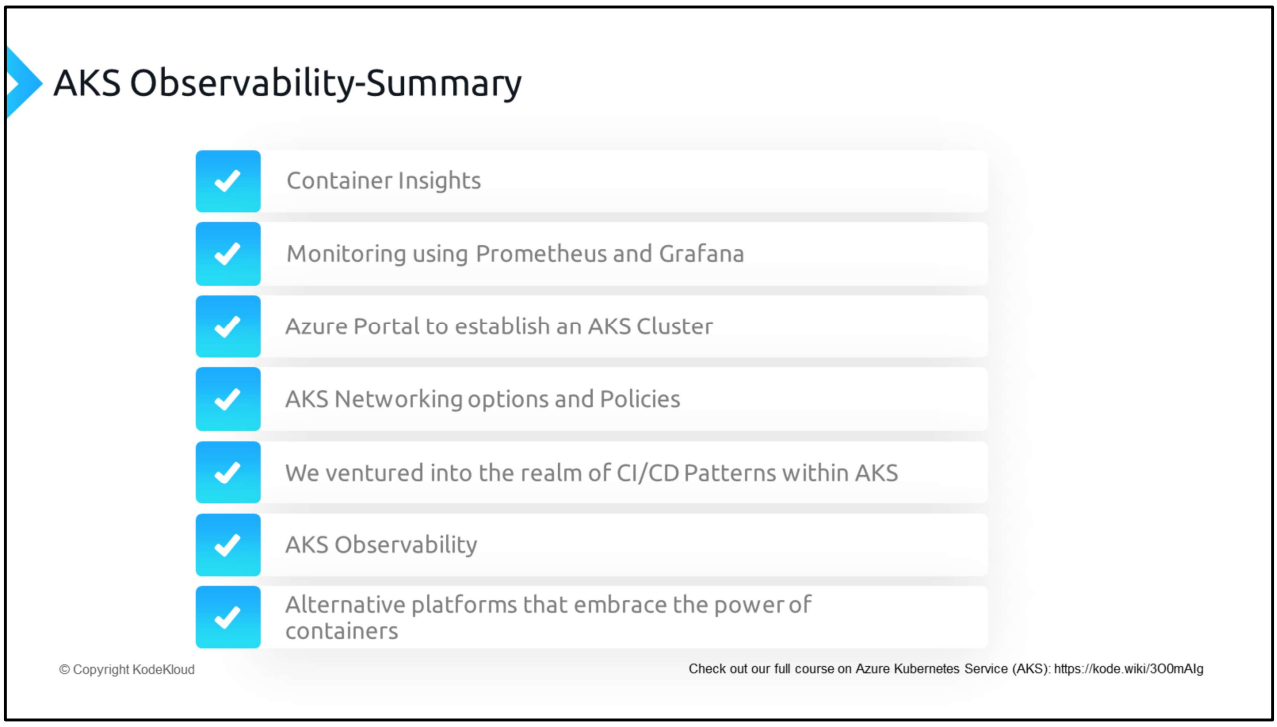
In summary, when enabling the monitoring add-on, two agents are installed: the AMA agent replica set for cluster-level metrics collection and fault tolerance, and the agent on each node for node-level metrics and pod monitoring.



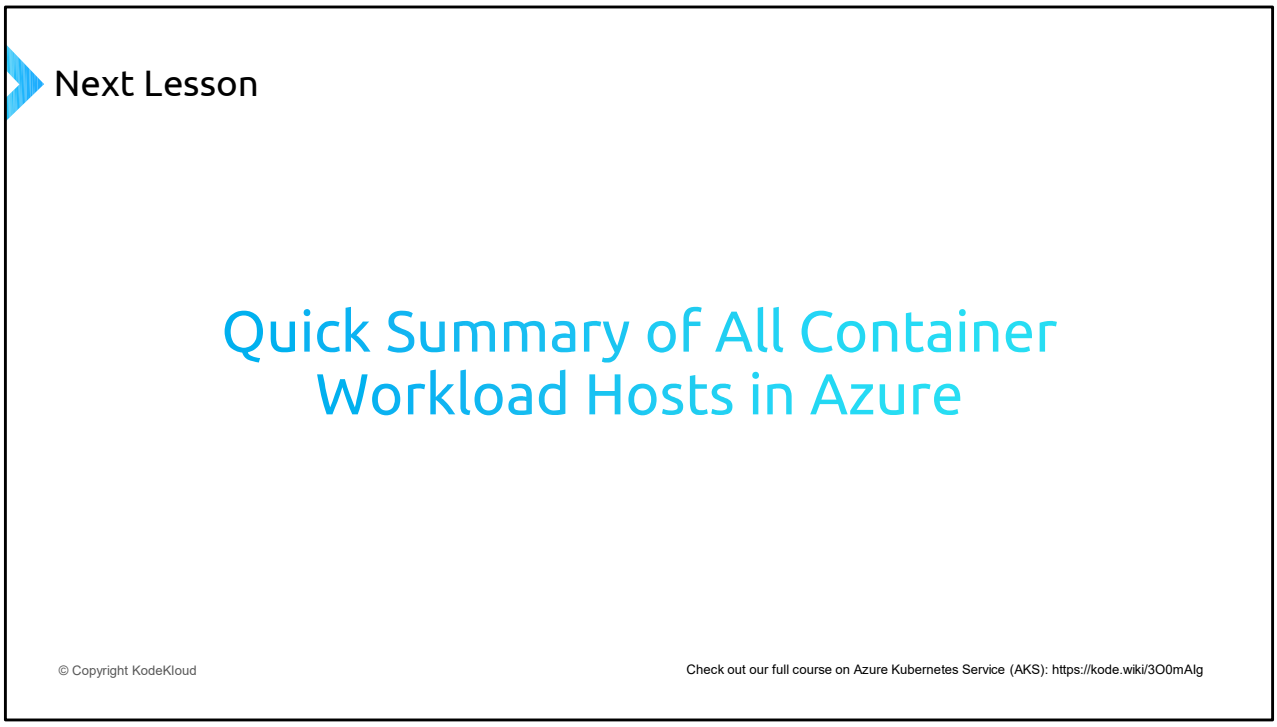
- Prometheus is an open-source systems monitoring and alerting toolkit.
- Grafana allows you to query, visualize, alert on and understand your metrics no matter where they are stored.



Prometheus is a popular open source metric monitoring solution and is a part of the Cloud Native Compute Foundation. Azure Monitor for containers provides a seamless onboarding experience to collect Prometheus metrics. Typically, to use Prometheus, you need to setup and manage a Prometheus server with a store. By integrating with Azure Monitor, a Prometheus server is not required. You just need to expose the Prometheus metrics endpoint through your exporters or pods (application), and the containerized agent for Azure Monitor for containers can scrape the metrics for you.



Prometheus is a popular open source metric monitoring solution and is a part of the Cloud Native Compute Foundation. Azure Monitor for containers provides a seamless onboarding experience to collect Prometheus metrics. Typically, to use Prometheus, you need to setup and manage a Prometheus server with a store. By integrating with Azure Monitor, a Prometheus server is not required. You just need to expose the Prometheus metrics endpoint through your exporters or pods (application), and the containerized agent for Azure Monitor for containers can scrape the metrics for you.



Ok, we have now reached the culmination of this video course on Azure Kubernetes Service (AKS). Our journey began by delving into the fundamental aspects of Azure, encompassing essential concepts such as Compute, Storage, and Networking. With a some foundation in place, we transitioned towards the creation and containerization of a basic C# web application. Throughout that module, we focused on leveraging Docker desktop and crafting Dockerfiles to facilitate containerization.

Subsequently, we navigated to the Azure portal to establish an AKS cluster to deploy our world class application onto that cluster. Harnessing the power of kubectl, we effectively executed various imperative actions, including scaling of deployments. we explored the deployment of container images within the Azure Container Registry (ACR), unearthing its potential for application enhancement and version management. By harnessing the image hosted on ACR, we facilitated a seamless application upgrade, only to effortlessly revert to a previous version when needed. Concluding this module, we bestowed upon you a overview of the Kubernetes Fleet, empowering you with a holistic understanding of this powerful toolset.

The subsequent module, entirely dedicated to AKS networking options and policies,

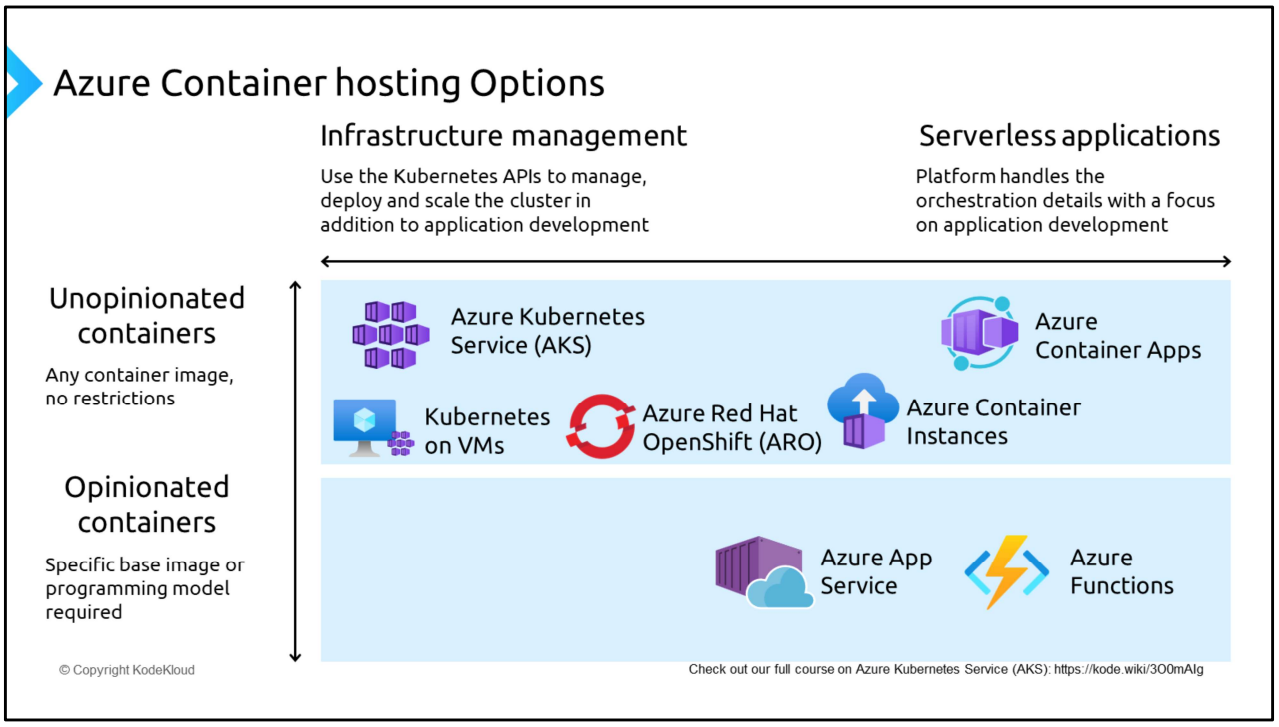
showcased the indispensable importance of robust network management within AKS deployments. Furthermore, we delved into security-centric topics, delving into the intricacies of Azure AD integration, defender for AKS, and Azure policy for AKS. Empowered with this knowledge, you are poised to fortify your AKS infrastructure with optimal security measures and governance practices.

As we progressed, we ventured into the realm of CI/CD patterns within AKS, looking at the art of managing AKS using declarative techniques. By adopting these practices, you can streamline the continuous integration and delivery pipelines within your AKS environment.

An indispensable component of any infrastructure, observability received some attention in module seven. We explored observability within AKS, equipping you with the necessary tools and techniques to monitor and gain valuable insights into your AKS deployments.

Finally, before we brought this course to a close, we presented you with an overview of alternative platforms that embrace the power of containers. By expanding your horizons and exploring these platforms, you can unlock a world of possibilities and further enhance your containerization endeavors.

We sincerely hope that your journey throughout this course has been as fulfilling as our commitment to its production. It has been an absolute pleasure to guide you through the intricacies of AKS, empowering you with the knowledge and skills needed to excel in your containerization journey. This is your host Pranav, signing off for now.



We are almost at the end of this course and AKS and I am glad you have made it this far. As you start to use AKS in your own environments, one common question you will get asked is if AKS is the only way to host containers on Azure and if it's the right platform to host your containers.

On the screen, you can see many native Azure services that can host containers. Depending on your use case, you can choose one of the option. For example, if you need full control of container orchestration and have a custom image to run, you will need to pick a service from top left segment i.e AKS, Kubernetes on a VM or Redhat open shift.

Lets quickly talk about various available options:

AKS and Azure Red Hat OpenShift provide access to a full Kubernetes or Red Hat OpenShift cluster, and you are in full control of all aspects of the cluster including **ingress, networking, maintenance, and scale**. If you want full control, you can also go old school and deploy k8s on VMs.

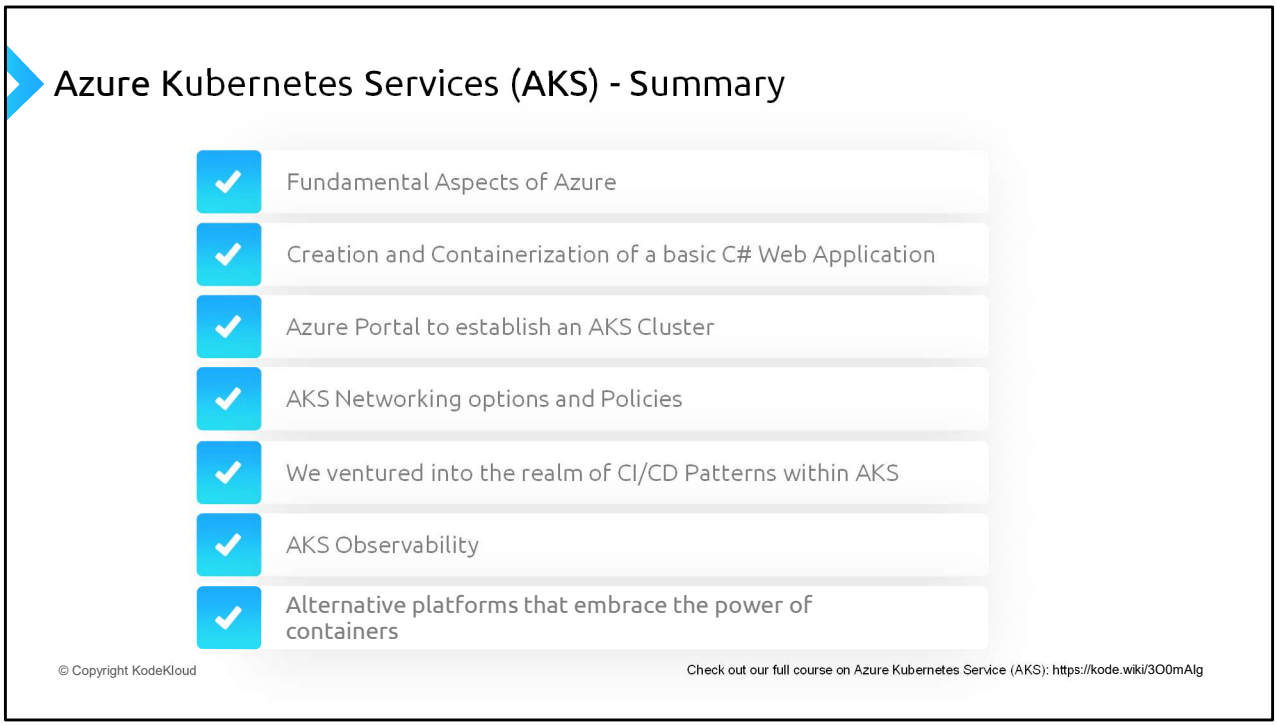
Azure Container Apps operates at the level above the cluster, where the developer only has to worry about the app itself, and the cluster management is handled by the service even though it uses k8s.. You don't have access to the apis, no Custom Resource Definition etc. It is also worth noting that only Linux based containers are supported on ACA.

Azure Container Instances provides a way to launch containers with hypervisor isolation. It provides a container as a unit in Azure. It's a low-level compute option, akin to provisioning a VM with a user experience designed to match a local container runtime. It's used to power experiences in a more generic fashion including Virtual Nodes as part of AKS. It is worth noting that (ACI) is primarily designed for running individual containers or microservices rather than supporting full-fledged container orchestrations like Kubernetes.

Azure App Service provides a purpose-built experience for hosting web applications and web APIs, including the ability to publish code directly without managing a container, and features like deployment slots and "test in production".
Azure Container Apps is optimized for microservices that need to communicate to each other, which is difficult to do with Azure App Service.

Finally, **Azure Functions** is about executing event-driven serverless code with an end-to-end development experience. Code deployed to Azure Functions must either use the Functions SDK or use the Functions base container image and adhere to the Functions predefined programming model.

I encourage you to read through some of these options and decide which one works for your application better.



Ok, we have now reached the culmination of this video course on Azure Kubernetes Service (AKS). Our journey began by delving into the fundamental aspects of Azure, encompassing essential concepts such as Compute, Storage, and Networking. With a some foundation in place, we transitioned towards the creation and containerization of a basic C# web application. Throughout that module, we focused on leveraging Docker desktop and crafting Dockerfiles to facilitate containerization.

Subsequently, we navigated to the Azure portal to establish an AKS cluster to deploy our world class application onto that cluster. Harnessing the power of kubectl, we effectively executed various imperative actions, including scaling of deployments. we explored the deployment of container images within the Azure Container Registry (ACR), unearthing its potential for application enhancement and version management. By harnessing the image hosted on ACR, we facilitated a seamless application upgrade, only to effortlessly revert to a previous version when needed. Concluding this module, we bestowed upon you a overview of the Kubernetes Fleet, empowering you with a holistic understanding of this powerful toolset.

The subsequent module, entirely dedicated to AKS networking options and policies,

showcased the indispensable importance of robust network management within AKS deployments. Furthermore, we delved into security-centric topics, delving into the intricacies of Azure AD integration, defender for AKS, and Azure policy for AKS. Empowered with this knowledge, you are poised to fortify your AKS infrastructure with optimal security measures and governance practices.

As we progressed, we ventured into the realm of CI/CD patterns within AKS, looking at the art of managing AKS using declarative techniques. By adopting these practices, you can streamline the continuous integration and delivery pipelines within your AKS environment.

An indispensable component of any infrastructure, observability received some attention in module seven. We explored observability within AKS, equipping you with the necessary tools and techniques to monitor and gain valuable insights into your AKS deployments.

Finally, before we brought this course to a close, we presented you with an overview of alternative platforms that embrace the power of containers. By expanding your horizons and exploring these platforms, you can unlock a world of possibilities and further enhance your containerization endeavors.

We sincerely hope that your journey throughout this course has been as fulfilling as our commitment to its production. It has been an absolute pleasure to guide you through the intricacies of AKS, empowering you with the knowledge and skills needed to excel in your containerization journey. This is your host Pranav, signing off for now.

KodeKloud

Check out our full course on Azure Kubernetes Service (AKS): https://kode.wiki/3O0mAIg