

Azure Kubernetes Service Landing Zone Accelerator / PCF V2

LLD

Contents

[Version control 1](#_Toc1643735233)

[Distribution list 2](#_Toc773693203)

[1. Introduction 3](#_Toc54726010)

[2. Objectives 3](#_Toc1333646027)

[3. Audience 3](#_Toc1269399344)

[4. Scope 3](#_Toc927494391)

[5. Disclaimer and Important notes 3](#_Toc1212267535)

[6. AZURE Kubernetes Service Landing Zone Accelerator– Enterprise-scale 3](#_Toc2138805767)

[7. AZURE Kubernetes Service Landing Zone Accelerator– Design principles: 4](#_Toc7347829)

[8. AKS Architecture for Enterprise Scale 4](#_Toc134259700)

[8.1 AKS Cluster 4](#_Toc73762966)

[8.1.1 AKS Public Cluster 4](#_Toc241602937)

[8.1.2 AKS Private Cluster 4](#_Toc313965069)

[8.1.2.1 AKS Private Cluster Architecture 4](#_Toc1350945981)

[8.1.2.2 AKS Cluster deployed with Private Endpoints 4](#_Toc939877291)

[8.1.2.3 Public vs Private AKS Cluster 4](#_Toc549046247)

[8.2 Central Azure Container Registry Design 5](#_Toc1166459487)

[8.2.1 Design: Central Azure Container Registry Architecture 5](#_Toc1276042134)

[8.2.2 Multiple AKS Clusters registered to a Centralized Azure Private DNS Zone 5](#_Toc298782266)

[8.3 Network Topology and Connectivity 5](#_Toc728468565)

[8.3.1 Understanding of Azure CNI (static) networking 5](#_Toc1284607280)

[8.3.2 Azure sample CNI Architecture 5](#_Toc2078705737)

[8.3.3 Azure CNI Dynamic IP Allocation 5](#_Toc1858947709)

[8.3.4 Sample POD Connectivity Azure CNI Dynamic IP 6](#_Toc1158417493)

[8.3.5 Design Decision for the Networking Model 8](#_Toc462453399)

[8.3.6 AKS Network Policies 9](#_Toc1411256186)

[8.3.7 AKS Network Policy Design decision: 9](#_Toc1560417827)

[8.4 AKS Ingress Controller Architecture: 10](#_Toc1060951702)

[8.4.1 AKS Ingress Controller Simple Architecture 10](#_Toc1670637212)

[8.4.2 Nginx Ingress Controller 10](#_Toc2096410564)

[8.4.3 AKS NGINX Ingress Controller sample Architecture: 10](#_Toc1598604490)

[8.4.4 Application Gateway Ingress Controller: 10](#_Toc966517487)

[8.4.5 Sample Design: Application Gateway Ingress Controller sample architecture: 10](#_Toc48625943)

[8.4.6 Ingress Controller Design decision 10](#_Toc200534859)

[8.5 AKS Egress Traffic Control 11](#_Toc355151473)

[8.5.1 Design: AKS Cluster deployed in PCFv2 Environment routing egress traffic to Apollo NVA setup in UEHUB01 11](#_Toc605912596)

[8.5.2 User Defined Route – Outbound communication to the internet 11](#_Toc397217702)

[8.5.3 Outbound communication to another service in the same cluster 11](#_Toc443163097)

[8.5.4 Outbound communication to a service outside the cluster but within the Uniper network 11](#_Toc170219159)

[8.5.5 Guidelines to be followed to setup networking requiruiremnts for PCFv2 Clusters 11](#_Toc1387889900)

[9. Application Team Owned vs HACT Team Central AKS Clusters: 12](#_Toc561014909)

[10. Security and Identity and access management recommendations: 13](#_Toc905507988)

[10.1 Enable threat protection for AKS Clusters 13](#_Toc1437998281)

[10.1.1 Defender Flow data flow to dedicated Central Workspace Design 13](#_Toc885155232)

[10.2 Secure access to the API server and cluster nodes via RBAC 13](#_Toc89774394)

[10.3 Access Azure container registry using Azure Private Link: 13](#_Toc2109377385)

[10.4 Azure AD workload identity for Azure Kubernetes Service (AKS) 13](#_Toc308306404)

[10.5 Azure Key Vault Provider for Secrets Store CSI Driver: 13](#_Toc579155931)

[10.6 Azure Policy Definitions for AKS Clusters 13](#_Toc549153161)

[10.7 AKS - API Server IP Whitelisting for Clusters deployed with public endpoint 13](#_Toc126219210)

[10.8 Customer managed keys for Azure Container registry 13](#_Toc1067279753)

[11. AKS Cluster High Availability and BCDR 13](#_Toc734855741)

[11.1 Use of availability zones for all node pools: 13](#_Toc2054367331)

[11.1.1 AKS Cluster High Availability Architecture Design: 13](#_Toc1238747155)

[11.2 Monthly Uptime Calculation and Service Levels for AKS Clusters that use Availability Zones 13](#_Toc1708288947)

[11.3 Enable redundant ingress controller configuration 13](#_Toc148097221)

[11.4 Zone redundant default storage account for persistent volumes 13](#_Toc1026524857)

[11.5 Enable Zone redundancy in Azure Container Registry for resiliency and high availability 13](#_Toc170017291)

[11.6 Enable Geo-Replication of the Uniper Central Container Registry 13](#_Toc441601331)

[12. Auto-scaling for AKS nodepools 13](#_Toc734072648)

[12.1 Cluster autoscaler 14](#_Toc65959160)

[12.2 Horizontal pod autoscaler 17](#_Toc1466473520)

[13. Persistent Volume Storage Recommendations for AKS PCFv2 18](#_Toc584706612)

[13.1 Persistent Volumes: 18](#_Toc896290448)

[13.2 Dynamically provision volumes 18](#_Toc761288995)

[13.3 Dynamically create Azure Files PVs by using the built-in storage classes 18](#_Toc148817413)

[13.4 Dynamically create Azure Disks PVs by using the built-in storage classes 18](#_Toc231686913)

[14. AKS Monitoring 18](#_Toc1623667141)

[14.1 Container Insights 18](#_Toc1737122324)

[14.1.1 Container Insights workflow 18](#_Toc499445093)

[14.1.2 Features of Container insights 18](#_Toc1819071517)

[14.2 Alerts 18](#_Toc2071269697)

[14.3 Metrics 18](#_Toc178885264)

[14.4 Resource Logs 18](#_Toc1777324744)

[14.5 Cost governance with Kubecost 18](#_Toc1665718156)

[15. Naming Conventions to be followed 18](#_Toc412587311)

[16. Patch Management and Version Upgrade for AKS Clusters in PCFv2 Environment 19](#_Toc1774386912)

[16.1 Local roles and responsibilities incl. RACI 19](#_Toc1283630551)

[16.2 Upgrade timelines 19](#_Toc738008698)

[16.3 Cluster Upgrade: 19](#_Toc1479448513)

[17. Certificate rotation in Azure Kubernetes Service (AKS) 19](#_Toc1655649178)

[17.1 Certificate Auto Rotation 19](#_Toc1577732791)

[17.2 Manually rotate cluster certificates 19](#_Toc1448301359)

# Version control

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Template Version History | | | | | |
| Version | Author | Reviewer | Release Date | Review Date | Remarks/Changes |
| 1.1 | Karthik Narayanan, Shobana Jayabalan | Dragos Schebesch |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# Distribution list

|  |  |  |
| --- | --- | --- |
| Team/Members |  | Remarks/Changes |
| Shobana Jayabalan |  |  |
| CS, Pradeep |  |  |
| K, Nagaraju |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# **Introduction**

This document describes the Azure Kubernetes Service Landing Zone approach and design principles which will be applied within Uniper’s Enterprise-scale model adoption project.

# **Objectives**

The Objective of this document is to provide the AKS Landing Zone Accelerator Guidelines for the new Microsoft Landing Zone Project. The document adheres to the architecture and best practices of the Cloud Adoption Framework's Azure landing zones with a focus on the design principles of enterprise-scale

# **Audience**

The audience for this document is primarily technical personnel, AKS, network, security and engineers, cloud engineers and operators and any other person interested in the subject.

# **Scope**

The scope of the document is related to Azure Kubernetes Service. Azure Container Registry, and the infrastructure services to effectively construct and operationalize a landing zone.

Out of the scope are other Azure services which are not part of the AKS architecture.

# **Disclaimer and Important notes**

Following Uniper’s IT Governance it is important to highlight that some AKS dependent Azure services are not managed by the AKS Team, therefore, those dedicated services should be covered on separate documents elaborated by the corresponding owners.

* Firewalls (Owner: T-System ([FCR-Uniper@telekom.de](mailto:FCR-Uniper@telekom.de)) and Security Team)
* IAM (Owner: Security Team)
* Vnet, Private Endpoints and Private DNS Zones (Owner: Network Team)
* Azure Policy: (Governance Team)
* Monitoring : (Monitoring Team)

An important consideration is that the following document does not constitute a green field approach. The AKS landing zone accelerator requires a platform foundation with shared services (network, security, identity, and governance), which will have to be kept in place to effectively construct and operationalize a landing zone.

# **AZURE Kubernetes Service Landing Zone Accelerator– Enterprise-scale**

The AKS landing zone accelerator represents the strategic design path and target technical state for an Azure Kubernetes Service (AKS) deployment. This solution provides an architectural approach and reference implementation to prepare landing zone subscriptions for a scalable Azure Kubernetes Service (AKS) cluster.

The implementation adheres to the architecture and best practices of the Cloud Adoption Framework's Azure landing zones with a focus on the design principles of enterprise-scale.

The landing zone accelerator approach provides the below assets.

* A modular approach that allows us to customize the environment variables.
* Design guidelines for evaluating critical decisions.
* The landing zone architecture.
* A Microsoft-recommended AKS reference implementation based on the AKS baseline.

AKS Baseline reference:

|  |  |
| --- | --- |
| Networking configuration: - | Network topology  Plan the IP address  Deploy Ingress resources |
| Cluster compute: - | Compute for the base cluster  Container image reference  Policy management |
| Identity management: - | Integrate Azure AD for the cluster  Integrate Azure AD for the workload |
| Secure data flow: - | Secure the network flow  Add secret management |
| Business continuity: - | Scalability  Cluster and node availability  Availability and multi-region support |
| Operations: - | Cluster and workload CI/CD pipelines  Cluster health and metrics  Cost management and reporting |

# **AZURE Kubernetes Service Landing Zone Accelerator– Design principles:**

Below are the design principles which will be applied as guidelines while creating the Landing Zone for AKS

* Identity and access management
* Network topology and connectivity
* Resource organization
* Security
* Management and BCDR
* Platform automation and DevOps
* Storage

The design principles are covered in detail in the below individual sections:

# **AKS Architecture for Enterprise Scale**

The below sections cover the Design considerations and decisions for the AKS Cluster deployment in the Landing Zone Subscriptions.

## **AKS Cluster**

In the PCFv1 Architecture, the Central AKS clusters are deployed with public endpoints, which means the cluster endpoints are exposed with Public IP Address which doesn’t comply with the Enterprise Scale governance, hence for PCVF v2 ESLZ, the AKS clusters are required to be deployed with the private endpoint architecture where the Private clusters expose the Kubernetes API over a private IP address and not over a public one, A comparison between the public and the private cluster deployment models is done in the following sections to understand the pros and cons.

**Overview**:

AKS Cluster can be deployed in both public and private endpoints.

AKS Cluster will have Managed Master Node, Worker Nodes, Advanced networking, Azure Active Directory (Azure AD) integration, monitoring, and other features.

Managed Kubernetes cluster in Azure offloads the operational overhead to Azure. As a hosted Kubernetes service, Azure handles critical tasks, like health monitoring and maintenance of the Master Node. Since Kubernetes masters are managed by Azure, Application team and HACT Team is only expected to manage and maintain the cluster nodes.

**Cluster Deployment Models:**

* Standard Cluster – A standard AKS cluster provides a public IP address for the API server.
* Private Cluster - A private AKS cluster, on the other hand, will use a private IP address for its API server endpoint. This enables traffic between Node pools and API server to be only on private network. Private Clusters have the additional requirement of creating Private Endpoint which happens automatically when private cluster model is chosen.

## **8.1.1 AKS Public Cluster**

Public or Standard clusters expose the Kubernetes API over Public IPs, how it is easier for the application teams to get started with compared to the Private cluster as the planning related to cluster connectivity needs to be planned in detail.

In Public Cluster Kubernetes CLI connects using the public endpoint and also the worker nodes connect to the control plane over the same public endpoint however the connectivity and traffic stays within the Microsoft backbone.

The Kubernetes API server is the core of the Kubernetes control plane and is the central way to interact with and manage the clusters. To improve the security of the AKS clusters and minimize the risk of attacks it is recommendation is to limit the IP address ranges which can access the API server

List of IPs to be whitelisted covered under the section ‘**9.7 AKS - API Server IP Whitelisting’**

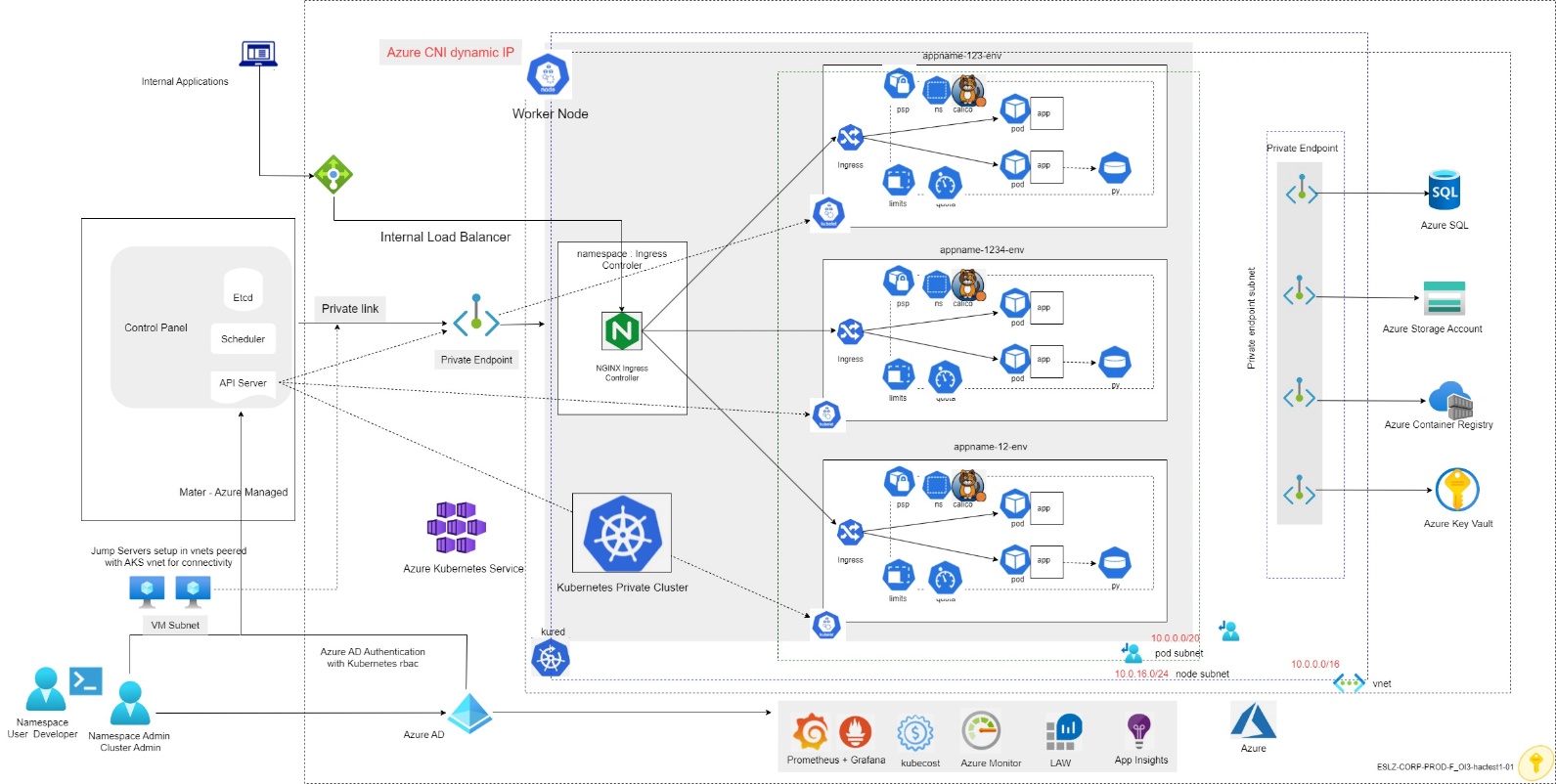
In PCF V1 the AKS clusters are deployed in the public cluster architecture which will not be suitable for an enterprise customer.

## **8.1.2 AKS Private Cluster**

Private clusters expose the Kubernetes API securely over a private IP address. The private IP address is represented in the AKS virtual network through a private endpoint, in this approach the Kubernetes API will not be accessed through its IP address, but instead through its fully qualified domain name (FQDN), the name resolution from the Kubernetes API FQDN to its IP address will typically be performed by an Azure Private DNS zone where an A host record is created for the Cluster FQDN which is managed centrally in the connectivity subscription of PCF v2 as per the ESLZ.

The Vnet of the AKS cluster created within the Landing Zone Subscription will be peered with the HUB vnet which is created within the connectivity subscription and is linked to the Private DNS Resolver service which in turn forwards the request to centrally managed Private DNS Zone for name resolution. AKS networking model for ES@Uniper is covered in detail under the section ‘**8.3 Network Topology and Connectivity’**.

## **8.1.2.1 AKS Private Cluster Architecture**



Visio reference url: [AKSPrivateCluster Architecture.vsdx](https://uniper.sharepoint.com/:u:/r/sites/CloudWorksTeam/HaCT/Project%20Documentation/PCFv2/Visio%20Diagrams%20-%20ES@Uniper/AKS/AKSPrivateCluster%20Architecture.vsdx?d=w05df297eb00746aa8fc6a05d86647a32&csf=1&web=1&e=7KvOy1)

## **8.1.2.2 AKS Cluster deployed with Private Endpoints**

In AKS, the connection between the master and the worker nodes is maintained through a pod called “tunnelfront” or “akslink” (depending on the cluster version) which is running on the worker nodes. The connection between them is created from akslink to the master — meaning from the inside to the outside. Earlier it was not possible to create this connection without a public IP on the masters. However, this is now different with private link. Using private link, we are projecting the private IP address of the AKS master, which is running in a virtual network into an IP address that is part of the AKS subnet. Furthermore, there is no need for the worker nodes to have a public IP assigned to a standard loadbalancer for egress the traffic as we are able to redirect the egress path through a Network Virtual Appliance or Azure Firewall, AKS Cluster Egress traffic for ES@Uniper is covered in detail under ‘**8.5 AKS Egress Traffic Control’.**

When an AKS Private Cluster is deployed, a Private Endpoint is automatically deployed and is a process which is part of the Cluster creation. Kubernetes Cluster connects with the PaaS services like Azure SQL Database, Storage account, Container Registry etc. using the Private Endpoint Connectivity.

**Note**: HaCT AKS Team, application teams and developers can access the aks cluster only from Jump Servers which have connectivity to the AKS Vnet setup in ES@Uniper because the access from public network is disabled, as per the update from HACT Networking Team currently communication outside ES@Uniper is not possible which will be available at later stages.

## **8.1.2.3 Public vs Private AKS Cluster**

The below section covers the pros and cons of the AKS Cluster Public and Private architecture:

**AKS Public Cluster**

**Pros:**

* Public Clusters are quite easy to setup, as the network connectivity doesn’t require detailed planning of setting up private endpoints and private DNS zones.
* Application Teams, developers or AKS Admins can connect easily through the public endpoint without having to use jump servers or bastion hosts.
* Azure DevOps Microsoft-hosted Agents are supported for AKS public clusters.

**Cons:**

* AKS public Cluster uses public endpoint exposure to connect to Control Pane and to connect with PaaS like Azure Container Registry, Zero Trust Network is not established.
* Worker nodes connect to control plane over public endpoint (within Azure backbone).
* Doesn’t comply with the Microsoft’s defined security guardrails and networking best practices.

**AKS Private Cluster**

**Pros**:

* No public endpoint exposed on internet (which helps implement Zero Trust Network).
* Worker nodes connect to control plane using private endpoint.
* AKS cluster can connect to PaaS services like azure container registry, azure SQL database, etc using private endpoint.
* Complies with the Microsoft’s security guardrails and networking best practices.

**Cons**:

* There's no support for Azure DevOps Microsoft-hosted Agents with private clusters. Consider using Self-hosted Agents.
* For Azure Container Registry to work with a private AKS cluster, set up a private link for the container registry in the cluster virtual network or set up peering between the Container Registry virtual network and the private cluster's virtual network.
* There's no support for converting existing AKS clusters into private clusters.
* Deleting or modifying the private endpoint in the customer subnet will cause the cluster to stop functioning.
* Supports IPv4 traffic only
* Supports TCP and UDP traffic only
* AKS admins, application team and developers can access the aks cluster only from Jump Servers / VDIs which have connectivity to the AKS Vnet.
* Private Link Service has an idle timeout of ~5 minutes (300 seconds). To avoid hitting this limit, applications connecting through Private Link Service must use TCP Keepalives lower than that time.

|  |
| --- |
| **Note**: The above point about idle timeout maybe a blocker for certain applications where the expected idle timeout is faultlessly more than 300 seconds, application teams should review this limitation before deciding their cluster architecture. |

**AKS Cluster design decision:**

From the Enterprise security and compliance requirement respective for the ES@Uniper landing zone subscription, Uniper’s architecture board’s recommendation is to build an AKS Private Cluster for HaCT and application teams.

However, If the Application Teams decide to opt for public cluster deployment model due to the networking and connectivity complexity involved with the private cluster architecture, it will be considered as a security risk, and they won’t be allowed to go ahead and they need to take the discussion to the Enterprise Landing Zone community.

**Note**: When a landing zone subscription is ordered through snow catalogue, Application or the HaCT Team will get the below option only during subscriptions ordering process.

* prod, UAT and Dev
* Sandbox
* POC

**For example:** When a prod subscription is ordered- through the catalogue, automatically 3 subscriptions are deployed (prod, UAT and Dev) and the application owner doesn't have control over which one they want to select / deploy.

Hence the HaCT AKS Team and the Application Teams when they order the subscriptions, they need to plan their AKS Cluster deployments respectively in prod/UAT/Dev Subscriptions accordingly.

## **8.2 Central Azure Container Registry Design**

Azure Container Registry is a managed registry service based on the open-source Docker Registry 2.0. Application team create and manage the container images and related artifacts in a central Azure container registry and as per the ESLZ ES@Uniper environment the central registry will be setup in the connectivity subscription and the AKS Clusters created in the Landing Zone subscriptions should leverage this central container registry.

In PCF V2, AKS Clusters deployed in Landing Zone Subscriptions need to establish a connectivity to the Azure Container Registry service for accessing the container images and these images need to be kept in a centralized repository, hence the approach is to create a centralized container registry in the management Subscription.

As per the below design, A centralized Azure Container Registry will be deployed in the management subscription and a private endpoint for the container registry is created.

As per the Enterprise scale Landing Zone Design, A Vnet peering will be set up between NVA Vnet and the Spoke vnets, which enables the AKS clusters deployed in the landing zone vnets to connect with the vnet in the management subscription where the Azure Container with Registry private endpoint is setup.

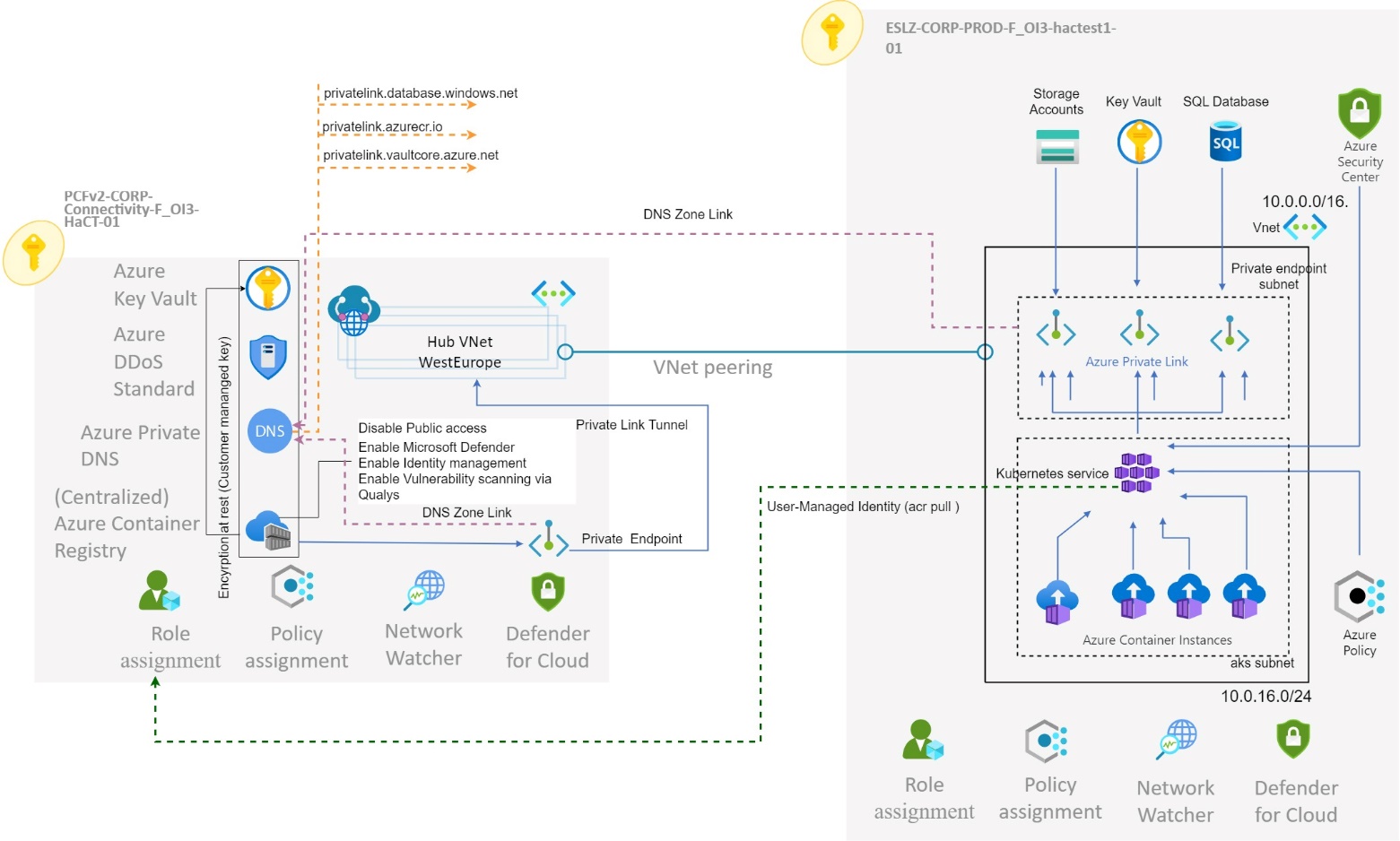
For ES@Uniper, User-assigned identity should be used for Azure resources like AKS Clusters to get authenticated and get authorized to access the central Azure container registry, without needing to provide or manage registry credentials, AKS Cluster to get acr pull rbac permissions for it to access the registry.

Azure Kubernetes Service cluster to use User managed Identity to pull container images from Azure Container Registry for pod deployments.

For devops CI/CD pipelines using service principal ID will be assigned with acr pull and push permissions to manage the images in the central container registry.

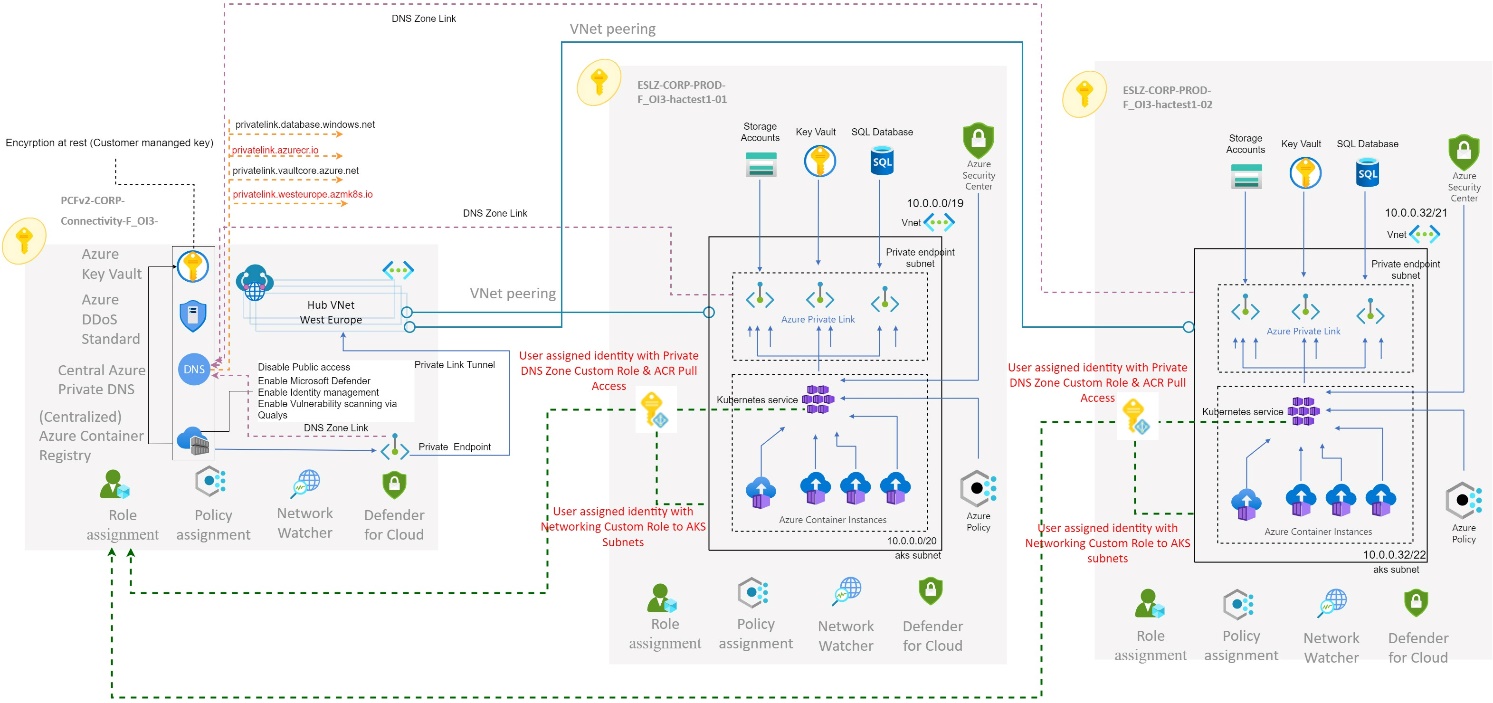
As per the ES@Uniper rbac recommendations, Users should be given only reader permissions to the container registry to view the images from the portal when the registry is created using the private endpoint AKS admins, application team and developers can access it only from Jump Servers which have connectivity to the private endpoint vnet.

## **8.2.1 Design: Central Azure Container Registry Architecture**



**Visio reference url:** [8.2.1 CENTRAL AZURE CONTAINER REGISTRY ARCHITECTURE.vsdx](https://uniper.sharepoint.com/:u:/r/sites/CloudWorksTeam/HaCT/Project%20Documentation/PCFv2/Visio%20Diagrams%20-%20ES@Uniper/AKS/8.2.1%20%20%20%20%20CENTRAL%20AZURE%20CONTAINER%20REGISTRY%20ARCHITECTURE.vsdx?d=wc4a067b4b3fa438dbf971879f9f281e0&csf=1&web=1&e=LzgxDy)

## **8.2.2 Multiple AKS Clusters registered to a Centralized Azure Private DNS Zone**



Visio reference url: [8.2.2 MULTIPLE AKS CLUSTERS REGISTERED TO A CENTRALIZED AZURE PRIVATE DNS ZONE.vsdx](https://uniper.sharepoint.com/:u:/r/sites/CloudWorksTeam/HaCT/Project%20Documentation/PCFv2/Visio%20Diagrams%20-%20ES@Uniper/AKS/8.2.2%20%20%20%20%20MULTIPLE%20AKS%20CLUSTERS%20REGISTERED%20TO%20A%20CENTRALIZED%20AZURE%20PRIVATE%20DNS%20ZONE.vsdx?d=w1c8c2dc3af354190ab8b4a3a1adc7225&csf=1&web=1&e=bda38T)

As shown in the above design, for ES@Uniper, the recommend architecture is to setup a shared dns private zone in the management subscription which can be accessed by the multiple AKS clusters from the landing zone subscriptions however it will require some preparation in advance. Since the hub vnet typically belongs to a connectivity subscription as per ESLZ, the private dns zone will be created in the connectivity subscription. In order for any AKS cluster to reference the private dns zone for the private endpoint setup.

AKS Vnet should be configured to use the private DNS Resolver inbound endpoint for name resolution. However, the private dns resolver vnet should be linked to the AKS private DNS Zone, which is created in the connectivity subscription per region, else the AKS cluster cannot be resolved with the FQDN.

The AKS Vnet (spoke) created in the landing zone subscription should be able to reach the DNS Resolver Vnet in the connectivity subscription through NVA Peering which is a pre-requisite for the solution to work.

**Permissions Required**: For any AKS cluster to reference the private dns zone in the connectivity subscription, an user assigned identity need to be setup for the cluster and provide Custom permissions on the Private dns zone before the AKS cluster can be created. This is not only required during creation but also for the lifecycle of the cluster since the private endpoint IP of an AKS cluster can change and the AKS service needs to have the ability to adjust the DNS entry accordingly to keep the cluster running.

The domain name of the dns zone needs to follow the naming convention of privatelink.region.azmk8s.io

DNS A-Record of the AKS Cluster need to be created in the Central AKS Private DNS Zone 'privatelink.region.azmk8s.io' at the time of the cluster creation for which a Custom role with minimal required permissions will be created and applied to the AKS user assigned identity on the private dns zone Resource group as a pre-requisite at the time of the AKS Cluster creation.

This custom role will be created at the Landing Zone Management Group hence it can be applied to the Landing Zone Subscriptions

**Note**: AKS private dns zone works differently from regular PaaS, hence the DINE policy cannot be applied here and Enable auto-regisration is not required to be selected.  
**minimal permissions required for the custom role:**

|  |
| --- |
| "actions": [  "Microsoft.Network/privateDnsZones/read",  "Microsoft.Network/privateDnsZones/write",  "Microsoft.Network/privateDnsZones/A/read",  "Microsoft.Network/privateDnsZones/A/write",  "Microsoft.Network/privateDnsZones/virtualNetworkLinks/read",  "Microsoft.Network/privateDnsZones/virtualNetworkLinks/write"  ],  "notActions": [  "Microsoft.Network/privateDnsZones/delete",  "Microsoft.Network/privateDnsZones/A/delete"  ] |

AKS vnet need to be linked to the private dns zone 'privatelink.region.azmk8s.io' as a pre-requisite, which will be done by the Automation Team at the time of AKS Vnet creation as the service principal ID

RBAC Roles are covered in detail under the section: **10.2 Secure access to the API server and cluster nodes via RBAC**

**Note**: As per the above Private DNS design for time being the solution is expected to work for the services deployed within the PCFv2 environment only based on the update from the HACT Networking Team as the Private DNS Zone and Custom DNS solutions are owned by them and in the upcoming releases, they will extend the solution to on-premise and PCFv1 environment as well.

## **8.3 Network Topology and Connectivity**

AKS Clusters supports two networking models:

* **kubenet** : Nodes and pods are placed on different IP subnets, User Defined Routing (UDR) and NAT is configured
* **Azure Container Networking Interface (CNI):** Azure CNI is a vendor-neutral protocol which lets the container runtime make requests to a network provider. It assigns IP addresses to pods and nodes and supplies IP address management (IPAM) features as you connect to existing Azure virtual networks.

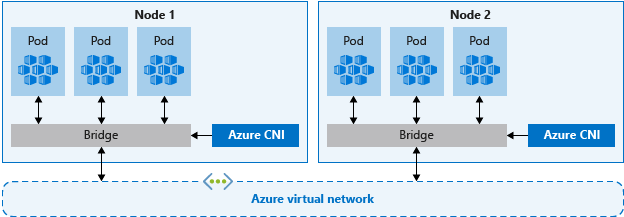
**Note**: Design decision in ES@Uniper for AKS Landing Zone Networking Model is ‘Azure CNI Dynamic IP Allocation’ covered under section ‘**8.3.3 Azure CNI Dynamic IP Allocation’**

## **8.3.1 Understanding of Azure CNI (static) networking**

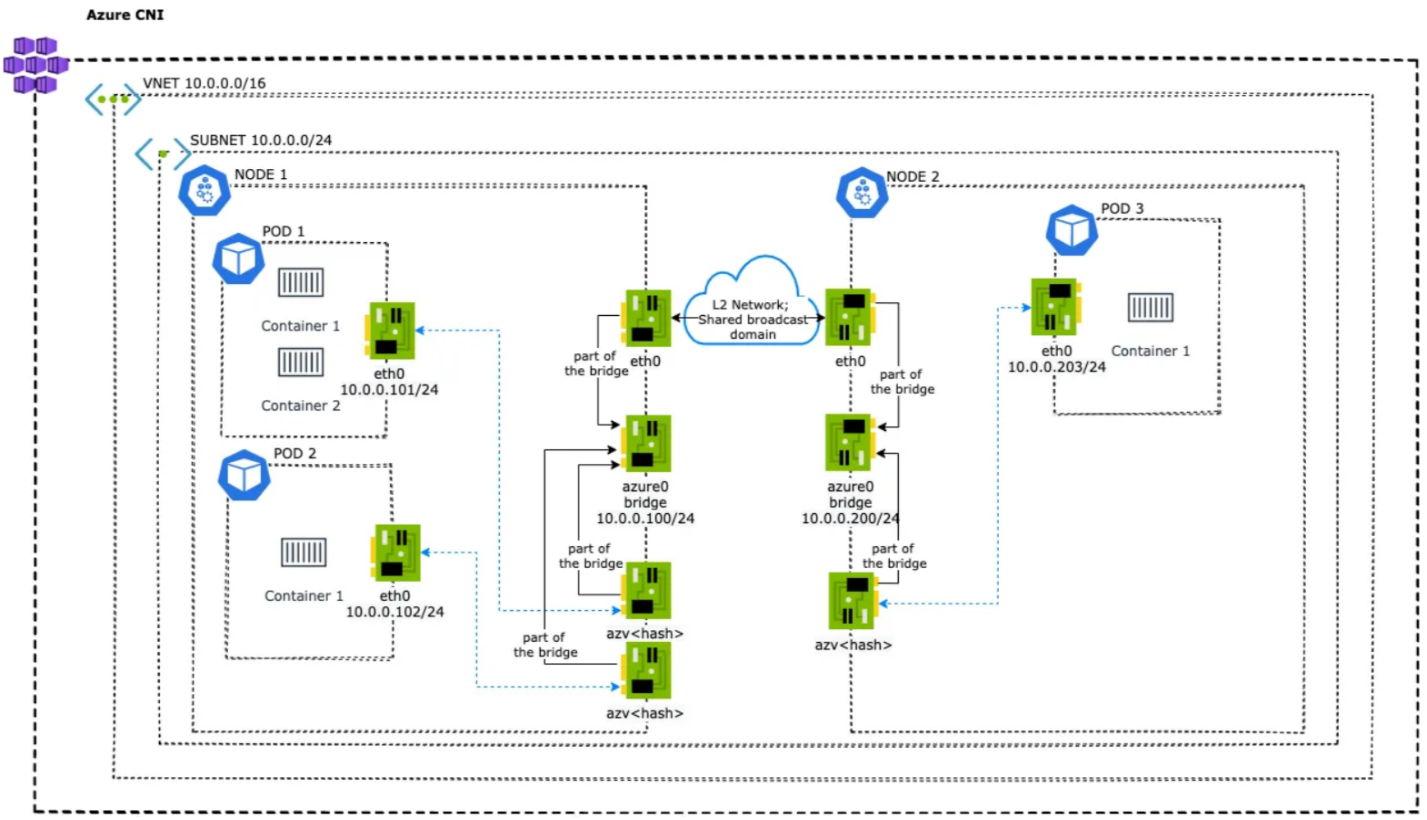
**Overview**: With Azure CNI, every pod gets an IP address from the subnet and can be accessed directly. These IP addresses must be planned and unique across the network space. Each node has a configuration parameter for the maximum number of pods it supports. The equivalent number of IP addresses per node are then reserved up front.

**Note**: This approach can lead to IP address exhaustion or the need to rebuild clusters in a larger subnet as the application demands grow however the limitation has been overcome with dynamic IP allocation approach CNI Model.

Unlike kubenet, traffic to endpoints in the same virtual network isn't NAT'd to the node's primary IP. The source address for traffic inside the virtual network is the pod IP. Traffic that's external to the virtual network still NATs to the node's primary IP.



## **8.3.2 Azure sample CNI Architecture**



* IP addresses for the pods and the cluster nodes are assigned from the specified subnet within the virtual network.
* Each node is configured with a primary IP address. By default, 30 additional IP addresses are pre-configured by Azure CNI that are assigned to pods scheduled on the node.
* The clusters can be as large as the IP address range we specify.
* Clusters configured with Azure CNI networking require additional planning. The size of the virtual network and its subnet must accommodate the number of pods planned to run and the number of nodes for the cluster.
* For intercommunications with other Azure services, e.g. VM, the source address of the packets arrived from AKS backed by Azure CNI is the pod IP address. So, it adds transparency to a network.
* User defined routes for pod connectivity is managed can be controlled by Uniper networking Team.

## **8.3.3** **Azure CNI Dynamic IP Allocation**

Azure Container Networking Interface (CNI) support for dynamic IP allocation and defining a different subnet for agent nodes and pods in Azure Kubernetes Service (AKS).

This enables IPs to be allocated dynamically when a pod needs it, instead of pre-allocating a subset of IPs to each agent node. This also reduces IP usage as compared to a static allocation

The new dynamic IP allocation capability in Azure CNI solves the IP exhaustion problem by allocating pod IPs from a subnet separate from the subnet hosting the AKS cluster.

The following benefits are achieved by adopting this approach:

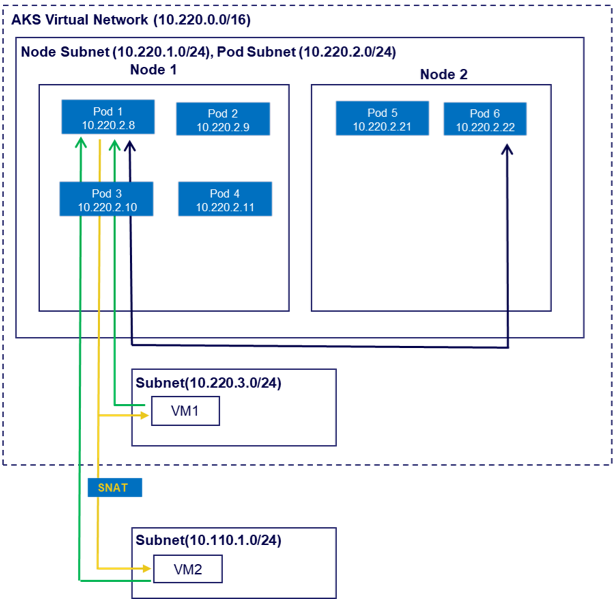
* **Better IP utilization**: IPs are dynamically allocated to cluster Pods from the Pod subnet. This leads to better utilization of IPs in the cluster compared to the traditional CNI solution, which does static allocation of IPs for every node.
* **Scalable and flexible**: Node and pod subnets can be scaled independently. A single pod subnet can be shared across multiple node pools of a cluster or across multiple AKS clusters deployed in the same VNet. We can also configure a separate pod subnet for a node pool.
* **High performance**: Since pods are assigned VNet IPs, they have direct connectivity to other cluster pods and resources in the VNet. The solution supports very large clusters without any degradation in performance.
* **Separate VNet policies for pods**: Since pods have a separate subnet, Separate VNet policies are configured for them that are different from node policies. This enables many useful scenarios such as allowing internet connectivity only for pods and not for nodes, fixing the source IP for pod in a node pool using a VNet Network NAT, and using NSGs to filter traffic between node pools.
* **Kubernetes network policies**: Both the Azure Network Policies and Calico work with this new solution.

**For ES@Uniper AKS Clusters, below are the Subnet CIDR Recommendations based on CNI Dynamic IP allocation model:**

|  |
| --- |
| **HaCT AKS Clusters Requirements:** |
| **Node Subnet CIDR**/24 = (251 usable IPs + 5 (Azure Reserved IPs)  **POD Subnet CIDR**/20 = (4091 usable IPs + 5 (Azure Reserved IPs)  There are 15 HaCT AKS Clusters from PCFv1 which will be deployed in the ES@Uniper landing Zones and will be split across the 3 environments respectively (Prod, UAT and Dev) and the vnet CIDR is planned accordingly by the Network Team via Infoblox.  Note: The recommendation is to create dedicated vnets for each AKS Cluster in ES@Uniper with the above-mentioned subnet CIDR |
|  |
| **Application Team Clusters Requirements:** |
| **Node Subnet CIDR**/24 (251 usable IPs + 5 (Azure Reserved IPs)  **POD Subnet CIDR**/22 = (1019 usable IPs + 5 (Azure Reserved IPs)  In PCF V2 the application teams will be deploying their AKS clusters in the respective landing zone subscriptions (Prod, UAT and Dev), and they will be required to use the above subnet CIDR configuration for their AKS Cluster deployments.  The recommendation for application Team is also to create dedicated vnets for each AKS Cluster in ES@Uniper with the above-mentioned subnet CIDR |

Note: The above table has complete dependency with the HACT networking and the automation team.

## **8.3.4 Sample POD Connectivity Azure CNI Dynamic IP**



|  |  |  |  |
| --- | --- | --- | --- |
| **Pod Outbound** | | | |
| Traffic Type | Source | Destination | Description |
| Pod-Pod  (Same VM) | Pod 1 | Pod 2 | The packets reaches destination directly with no UDR as the pod resides in the same node |
| Pod-Pod (Other VM) | Pod 1 | Pod 5 | The packets reaches destination directly as the pod resides in the same aks subnet. |
| Pod-VM (Same Cluster Network) | Pod 1 | VM1 | Pod1 can access VM1, VM1 sees pod IP as the source IP |
| Pod-VM (peered or connected Network) | Pod 1 | VM2 | Pod1 can access VM2, VM2 sees Node IP of Pod1 as source IP (SNAT) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Pod Inbound** | | | |
| Traffic Type | Source | Destination | Description |
| VM1-Pod | VM1 | Pod 1 | Traffic can reach Pod 1 as the Pod1 has routable IP address. |
| VM2-Pod | VM2 | Pod 1 | Traffic can reach Pod 1 as the Pod 1 has routable IP |

## **8.3.5 Design Decision for the Networking Model**

The below table gives an overall summary on why the design decision has been made to go with the Azure CNI Dynamic IP Allocation.

|  |  |
| --- | --- |
| Capability | Azure CNI with Dynamic IP Allocation |
| Multiple clusters in same subnet. | Supported |
| Performance | Performance is better since the UDR is not used for pod communication |
| UDR Management | CNI will not use any UDR where Uniper can have full control. |
| IP Address for Pod | The nodes and pods scale independently, their address spaces can also be planned separately |
| Supported Platforms | Windows and Linux |
| Pod to VM / PaaS connectivity (VM / PaaS Can be in same or peered of connected Network) | Works both ways |
| Network Policy | Supports Azure Network Policy and Calico |
| Maximum Nodes per Cluster | 1000 |
| Default Maximum Pods per node | 30 (Maximum Pods grow up to 250) |
| Expose AKS service using a LB, App Gateway or Ingress Controller. | Supported |
| Vnet Deployment Model | The recommendation is to deploy a dedicated Vnet for each AKS Cluster deployed in ES@Uniper, which will be followed for the HaCT and Application Team Clusters. |
| Suitable use Case | •Suitable for complex Application architecture and where end to end security compliance needs to be achieved. |
| •Better network performance is required |
| •When Resources outside the cluster need to reach pods directly |
|

## **8.3.6 AKS Network Policies**

**Overview**:

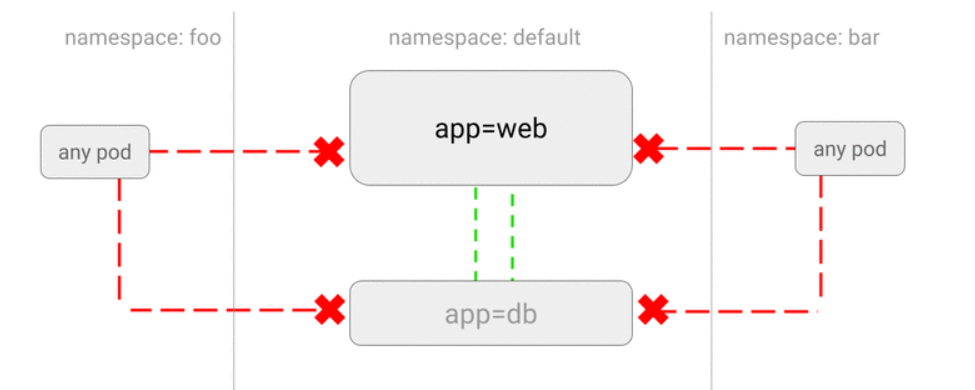
All pods in an AKS cluster can send and receive traffic without limitations, by default. To improve security, application teams need to define network rules which control the flow of traffic.

Back-end applications are often only exposed to required front-end services, for example. database components are only accessible to the application tiers that connect to them.

Network Policies is a Kubernetes feature to configure how groups of pods are allowed to communicate with each other and other network endpoints. In other words, it creates firewalls between pods running on a Kubernetes cluster.

By default, Kubernetes does not restrict traffic between pods running inside the cluster. This means any pod can connect to any other pod as there are no firewalls controlling the intra-cluster traffic. Network Policies gives us a way to declaratively configure which pods are allowed to connect to each other.

These Network Policy rules are defined as YAML manifests. Network policies can be included as part of a wider manifest which also creates a deployment or service.



**Network Policy options available in AKS:**

A Network Policy option can be chosen only when the AKS cluster is created. The policy option can't be changed afterwards.

Azure provides two ways to implement Network Policy.

* Azure's own implementation, called Azure Network Policy Manager.
* Calico Network Policies, an open-source network and network security solution founded by Tigera.

For ES@Uniper Calico Network policy is recommended for both the Central and the Application Team managed AKS Clusters

## **8.3.7 AKS Network Policy Design decision:**

**Design Decision**: The table below gives an overall summary of why the design decision has been made to go with **Calico** for the Network Policy, the Central and the application team AKS Clusters are required to adapt to Calico for the AKS Clusters.

|  |  |
| --- | --- |
| **Capability** | **Calico Network Policy** |
| Supported platforms | Linux, Windows Server 2019 and 2022 |
| Supported networking options | Azure CNI (Linux, Windows Server 2019 and 2022) and kubenet (Linux) |
| Compliance with Kubernetes specification | All policy types supported |
| Additional features | Extended policy model consisting of Global Network Policy, Global Network Set, and Host Endpoint. For more information on using the calicoctl CLI to manage these extended features, see calicoctl user reference. |
| Support | Calico community support. For more information on additional paid support, see Project Calico support options. |
| Logging | All the logs are logged in to /var/log/calico. All logging is done using svlogd. |

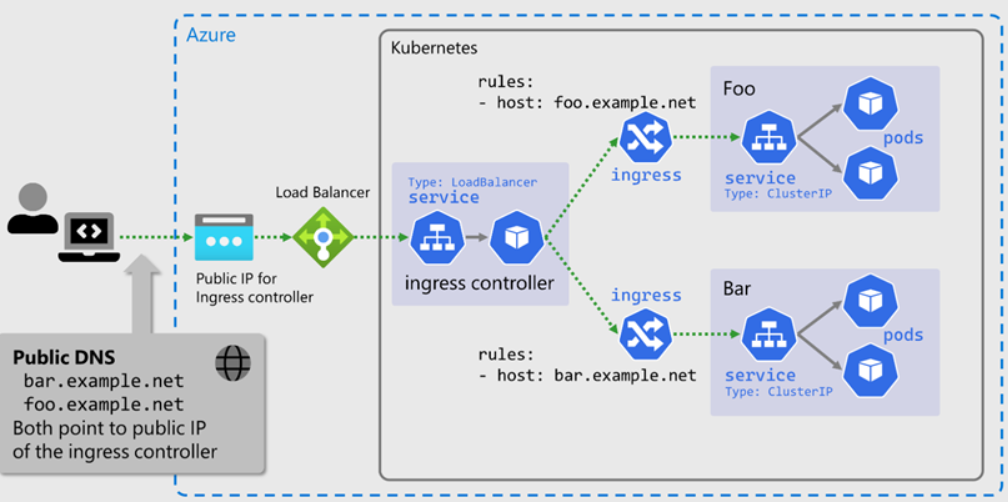
## **8.4 AKS Ingress Controller Architecture:**

**Overview**: An ingress controller is a piece of software which provides reverse proxy, configurable traffic routing, and TLS termination for Kubernetes services. Kubernetes ingress resources are used to configure the ingress rules and routes for individual Kubernetes services. An ingress controller and ingress rules use a single IP address which can be used to route traffic to multiple services in a Kubernetes cluster.

Ingress Controller key points:

* Accept traffic from outside the Kubernetes platform, and load balance it to pods (containers) running inside the platform
* Can manage egress traffic within a cluster for services which need to communicate with other services outside of a cluster
* Are configured using the Kubernetes API to deploy objects called “Ingress Resources”
* Monitor the pods running in Kubernetes and automatically update the load‑balancing rules when pods are added or removed from a service

## **8.4.1 AKS Ingress Controller Simple Architecture**



In The ES@Uniper landing zone, the HACT and the application teams are expected to adhere to the below Ingress controller architecture based on the requirements by following the guidance in the design decision table under the section **8.4.6 Ingress Controller Design Decision**

* Nginx Ingress Controller
* Application gateway Ingress controller

The ingress controller should be used by the applications for ingress traffic. Application teams must define a suitable ingress definition to route traffic from the ingress controller to their service inside their namespace.

In this case the application team is responsible for its maintenance and support. The private IP address of the internal loadbalancer for the ingress controller is fixed and must be used for DNS entries to route traffic to the AKS cluster based on DNS names.

Note: An ingress controller is always required for secure communication (https)

**Sample ingress file:**

<https://wiki.intranet.uniper.energy/sales/download/attachments/44830107/sample-ingress.yml?api=v2>

## **8.4.2 Nginx Ingress Controller**

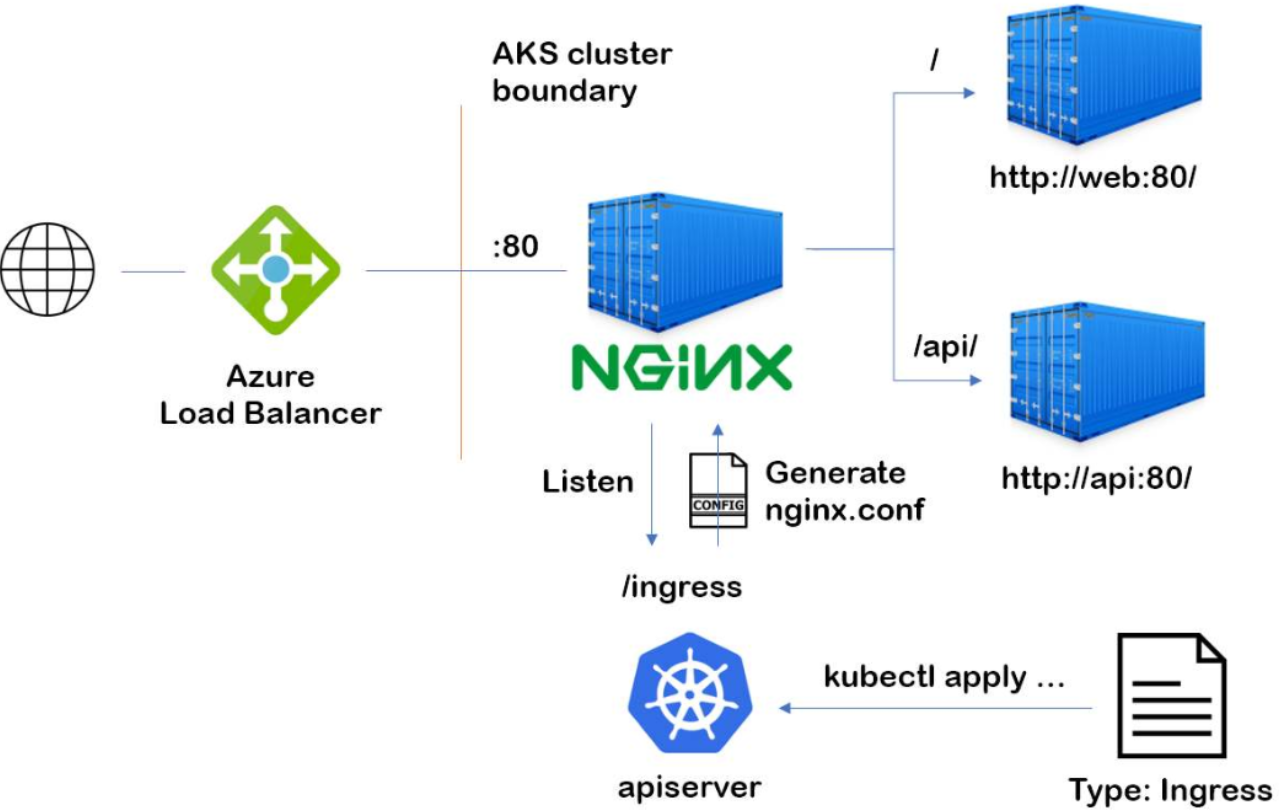
In a well‑architected AKS Cluster like in PCFV2 environment, the Ingress controller is the only point of entry for data plane traffic flowing to services running within the cluster.

It is a universal Kubernetes-native tool for implementing API gateways, load balancers, and Ingress controllers at the edge of a Kubernetes cluster

NGINX Ingress controllers are being deployed and used in the PCFv1 environment by the HACT and the application Teams hence the same architecture and the deployment guide can apply for PCFv2 clusters as well

Reference url: <https://wiki.intranet.uniper.energy/sales/display/CF/AKS+-+Ingress+Controller>

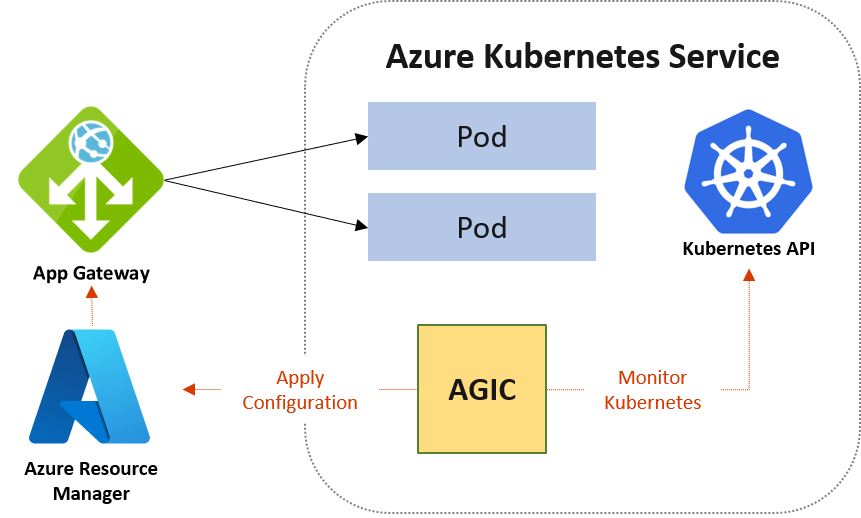
## **8.4.3 AKS NGINX Ingress Controller sample Architecture:**



## **8.4.4 Application Gateway Ingress Controller:**

**Overview**: The Application Gateway Ingress Controller (AGIC) is a Kubernetes application, which makes it possible for Azure Kubernetes Service (AKS) customers to leverage Azure's native Application Gateway L7 load-balancer to expose applications to the Internet securely. AGIC monitors the Kubernetes cluster it is hosted on and continuously updates an Application Gateway, so that selected services are exposed to the Internet.

## **8.4.5 Sample Design: Application Gateway Ingress Controller sample architecture:**



AGIC helps eliminate the need to have another load balancer/public IP in front of the AKS cluster and avoids multiple hops in the data path before requests reach the AKS cluster. Application Gateway talks to pods using their private IP directly and doesn't require NodePort or KubeProxy services. This also brings better performance to the deployments.

Ingress Controller is supported exclusively by Standard\_v2 and WAF\_v2 SKUs, which also brings the autoscaling benefits. Application Gateway can react in response to an increase or decrease in traffic load and scale accordingly, without consuming any resources from your AKS cluster.

Using Application Gateway in addition to AGIC also helps protect the AKS cluster by providing TLS policy and Web Application Firewall (WAF) functionality

Reference link for setting up AGIC: <https://learn.microsoft.com/en-us/azure/application-gateway/ingress-controller-install-new>

## **8.4.6 Ingress Controller Design decision**

Design Decision – For ES@Uniper the below table summarizes the guidance to the Application and the HACT Teams on how to make the decision approach for the Ingress controller architecture:

|  |  |  |
| --- | --- | --- |
| **Condition** | **AGIC** | **Nginx** |
|
| Definition | Use Azure Application Gateway as the Load Balancing tool | Use Azure Load balancer as the Endpoint for exposing applications deployed on AKS. |
|  |  |
| This uses Azure Application gateway ingress controller | This uses Nginx as the Ingress Controller |
|  |  |
| Support | Microsoft Support Available | Microsoft Support Not Available. commercial version can be purchased from F5. |
|
| Layer 7 & Layer 4 Support | Only Layer 7. Need to include one more Azure LB for Layer 4 | Without Nginx its layer 4 and with Nginx its Layer 7 |
|
| Popularity & long in market | Microsoft Native Solution but new to this Area | Nginx is the most widely used Kubernetes Ingress Controller |
|
| Cost | AAG is more expensive compared to Nginx | Open source is less need to pay for F5 like support if required |
|
| Performance | Less hops. All Layer 7 functionalities moved out of AKS | All Layer 7 functionalities like TLS, URL rewriting, traffic management needs to be configured in AKS only |
|  |
| Theoretically it should be better but most cases the performance is the same. |
|  |
| Other Features | Autoscaling, Zone redundancy, HA is built in and WAF inspection are supported.  Better for requests emerging from outside the Uniper network. | Consider adding redundancy by deploying 2 replicas of the ingress controller for HA. |
|
| Provisioning and management overhead | HA is inbuilt. Provisioning is simple but configuration might get complex. | Complex setup and configuration compared to AGIC. HA setup adds additional work and complexity. |
|

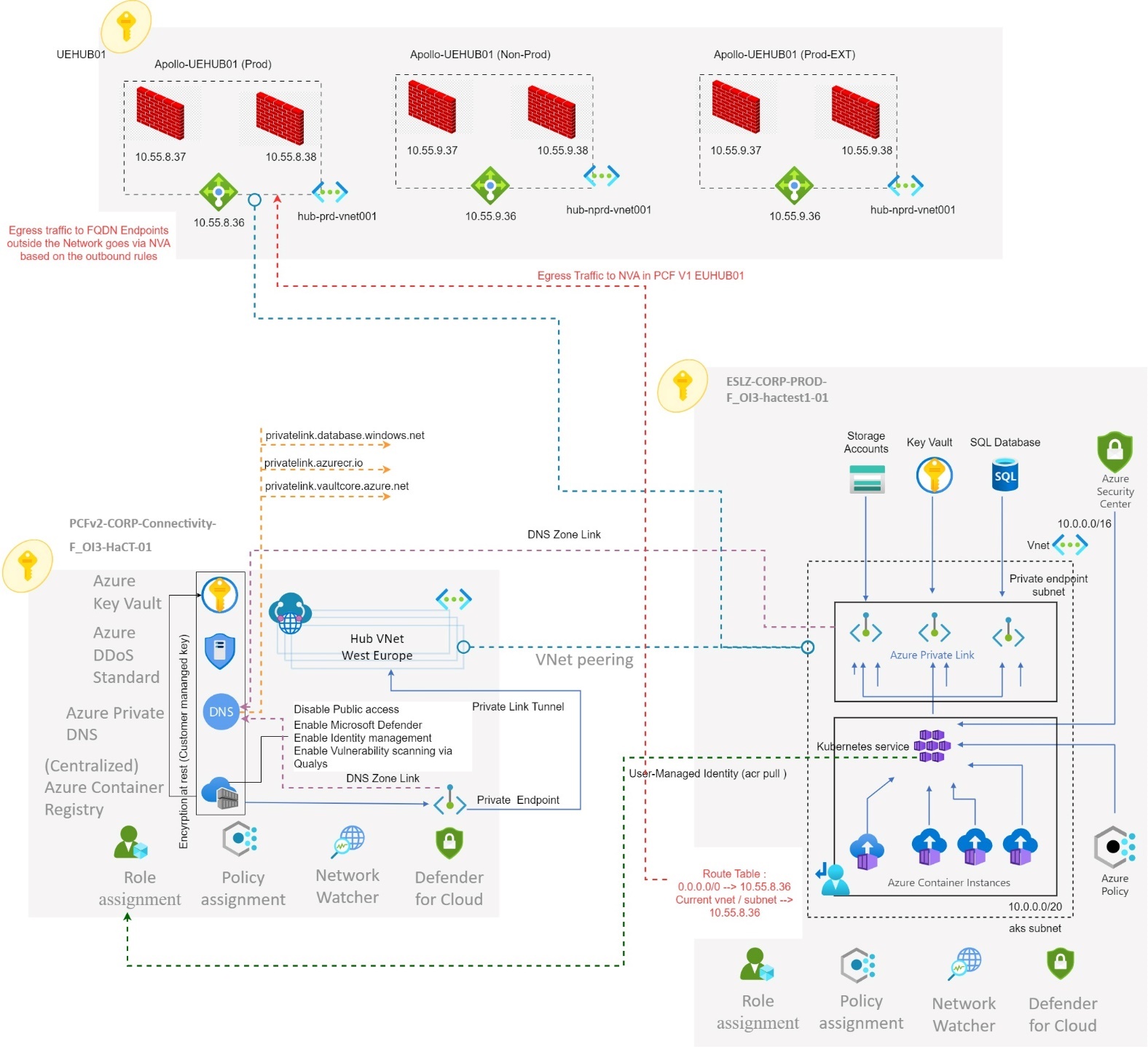
## **8.5 AKS Egress Traffic Control**

**Overview**: AKS clusters have the outbound, or egress, dependencies on services outside of the virtual network for management and operational purposes, Nodes in an AKS cluster need to access certain ports and fully qualified domain names (FQDNs). These endpoints are required for the nodes to communicate with the API server or to download and install core Kubernetes cluster components and node security updates. For example, the cluster needs to pull base system container images from Microsoft Container Registry (MCR), These outbound dependencies are almost entirely defined with FQDNs, which don't have static addresses behind them, hence we can't use network security groups (NSGs) to lock down the outbound traffic from an AKS cluster.

By default, AKS clusters have unrestricted outbound internet access which allows nodes and services to access external resources as needed.

As per the Microsoft security recommendations, in ES@Uniper only authorized ports and FQDNs must be accessible to maintain healthy cluster maintenance tasks. The recommended solution to securing outbound addresses is using a Firewall appliance which can control outbound traffic based on domain names.

## **8.5.1 Design: AKS Cluster deployed in ES@Uniper Environment routing egress traffic to Apollo NVA setup in UEHUB01**



Visiofile reference url: [8.2.1 CENTRAL AZURE CONTAINER REGISTRY ARCHITECTURE.vsdx](https://uniper.sharepoint.com/:u:/r/sites/CloudWorksTeam/HaCT/Project%20Documentation/PCFv2/Visio%20Diagrams%20-%20ES@Uniper/AKS/8.2.1%20%20%20%20%20CENTRAL%20AZURE%20CONTAINER%20REGISTRY%20ARCHITECTURE.vsdx?d=wc4a067b4b3fa438dbf971879f9f281e0&csf=1&web=1&e=LzgxDy)

## **8.5.2 User Defined Route – Outbound communication to the internet**

AKS cluster is deployed into an existing virtual network with a subnet and the application running on Pod requires explicitly to connect to FQDN Endpoints outside the AKS Cluster then the architecture requires sending egress traffic to an appliance like a firewall, Outbound type of User defined Route requires a route for 0.0.0.0/0 and a next hop destination of NVA in the route table.

In ES@Uniper we are leveraging the apollo firewall in the ‘UEHUB01’ subscription as the next hope for the AKS cluster egress traffic as per the above architecture.

PCF V2 landing zone subscription vnet will be peered with the PCVFv1 UEHUB01 NVA Vnet, The Node and the Pod Subnet from the AKS vnet will be linked to the route table with Outbound type of

UDR. a route for 0.0.0.0/0 and a next hop destination of NVA in the route table.

The firewall rules to allow the egress traffic to the authorized FQDN Endpoints, the outbound firewall rules are created and managed by the security team.

Currently for the PCFv1 Vnets/subnets the next hop is the apollo NVAs setup in UEHUB01, however as per the HaCT networking team update the same NVAs for next hop will be also applied to ES@Uniper Vnets/subnets. Hence the same outbound rules already configured for PCFv1 can be used for ES@Uniper environment by updating the source IP for communication in the NVA’s outbound firewall rules ie. CIDR of the AKS subnets from ESLZ.

## **8.5.3 Outbound communication to another service in the same cluster**

To achieve outbound communication from a pod to another service in the same cluster we can use the cluster internal DNS services. Use the DNS entry of the cluster for internal communication: <https://kubernetes.io/docs/tasks/administer-cluster/namespaces/#understanding-namespaces-and-dns> . In addition, we need to allow this communication in the network policy settings for both namespaces (source and target).

## **8.5.4 Outbound communication to a service outside the cluster but within the Uniper network**

Outbound communication to a destination inside the Uniper network requires firewall settings i.e east to west traffic. The source IP for this communication can be any of the private IPs of the subnet the cluster is assigned to.

## **8.5.5** **Guidelines to be followed to setup networking requiruiremnts for ES@Uniper Clusters**

The expectation is that the HaCT networking team in conjunction with the HaCT AKS Team replicate / migrate the UDRs and the NSGs mapped to the AKS Subnets from PCFv1 environment to the AKS vnets/subnets created in PCF v2 environment.

HaCT Team’s Central Clusters will be migrated / setup in the ES@Uniper vnet/subnets where the outbound firewall rules, NSGs and UDRs will be replicated from the existing PCFv1 environment to the ES LZ VNETs/subnets. HaCT AKS and Networking Team will work closely together for this migration requirement, so the application teams do not have to go through the vnet requirements setup in ESLZ.

The main purpose behind this approach is to save the application teams from setting up everything from scratch and it will save them many man hours which will be benefit the business and customers.

|  |
| --- |
| **Note**: There is HaCT Networking Team dependency for the above requirement to define the strategy inorder to replicate or migrate the NSGs and UDRs applied to the AKS subnets from PCFv1 to the ES LZ AKS Vnets / subnets and to route the traffic from PCF v2 clusters to the NVA Firewall in UEHUB01 where the outbound rules are explicitly setup and already used by PCFv1 Clusters, HaCT Networking Team is expected to check for the feasibility and define the approach. |

# **Application Team Owned vs HACT Team Central AKS Clusters:**

**Overview**: The below table provides extensive details on what will be different in application teams clusters vs centrally managed HaCT Team’s clusters for ESLZ as there could be possibilities for different levels of completeness for each.

|  |  |  |
| --- | --- | --- |
| **Topic** | **HACT Central Clusters** | **Application Team managed Clusters** |
| Cluster Architecture | • HACT AKS Team manages Central Clusters and will be expected to follow the end-to-end defined security guardrails, • HACT Team central clusters should be deployed in a private cluster architecture as per the section ‘8.1.2 AKS PRIVATE CLUSTER’ inorder to ensure there is no public endpoint exposed on internet and HACT Team’s central clusters are expected to fall within the Zero Trust Network, Central AKS clusters deployed in the Landing Zone Subscriptions are expected to connect to PaaS services like azure container registry, azure SQL database, etc using private endpoint only and there required to use only Self-Hosted agent for their DevOps Pipelines. | • Application Teams are also expected to follow the Private cluster architecture but however due to the complexity in networking and connectivity requirements application owners may decide to go with the public cluster architecture, reference section ‘8.1.1 AKS PUBLIC CLUSTER’. Which will have challenges complying with Zero Trust Network in such scenario application teams are expected to follow the security recommendations for whitelisting IPs to access the API Server. covered under the section ‘9.7 AKS - API SERVER IP WHITELISTING', and there is possibility for them to use Microsoft-hosted agents for DevOps Pipelines. |
|  |  |  |
| Network Topology | The HACT Team is expected to follow the networking model covered under the section ‘8.3.3 AZURE CNI DYNAMIC IP ALLOCATION’, however there is difference in CIDR requirements when compared to the application team owned cluster subnets. **Central Cluster requirements:** • Node Subnet CIDR/24 = (251 usable IPs + 5 (Azure Reserved IPs) • POD Subnet CIDR/20 = (4091 usable IPs + 5 (Azure Reserved IPs) | The Application Team is expected to follow the same networking model covered under the section ‘8.3.3 AZURE CNI DYNAMIC IP ALLOCATION’ however there is a difference in the CIDR requirements.  **Application Team owned clusters:** • Node Subnet CIDR/24 (251 usable IPs + 5 (Azure Reserved IPs) • POD Subnet CIDR/22 = (1019 usable IPs + 5 (Azure Reserved IPs) |
|  |  |  |
| Network policy | The HACT Team is expected to apply the same network policy for their clusters covered in detail under the section ‘8.3.7 AKS NETWORK POLICY DESIGN DECISION | The Application Team is expected to apply the same network policy for their clusters covered in detail under the section ‘8.3.7 AKS NETWORK POLICY DESIGN DECISION |
|  |  |  |
| Ingress Controller | Central Clusters managed by HACT Team are expected to be setup with NGINX Ingress controller for the inbound traffic as the traffic is expected to emerge only from within the Uniper network. | The Application Team clusters may host applications which could expect traffic outside Uniper network or from 3rd party sources hence they might use application gateway ingress controller with application gateway covered in detail under the section ‘8.4.6 INGRESS CONTROLLER DESIGN DECISION’ |
|  |  |  |
| Cluster Management | HACT Team is expected to follow the recommended approach for patching and version upgrades without deviation | Application Team is also expected to follow the recommended approach for patching and version upgrade, however there are chances for deviation. |
|  |  |  |
| Azure Monitor | Prometheus and Grafana will not be implemented for the central clusters managed by the HACT Team | Prometheus and Grafana may or maynot be implemented for the application owned clusters. |

# **Security and Identity and access management recommendations:**

Managing clusters in Azure Kubernetes Service, workload and data security is a key consideration.

The main AKS security components provided include:

* Active Directory
* Azure Policy
* Microsoft Defender for Containers
* Azure Key Vault
* Network security groups
* Orchestrated cluster upgrades

The below security best practices need to be included in the AKS Cluster deployment as a part of the ES@Uniper Environment

## **10.1 Enable threat protection for AKS Clusters**

In PCF V2 [Defender for Containers](https://learn.microsoft.com/en-us/azure/defender-for-cloud/defender-for-containers-introduction) will be enabled for AKS Clusters via Azure policy scoped at the ‘ES@Uniper Enterprise Scale’ Management Group level, as the defender profile need to be mapped to a central and dedicated log analytics workspace, this can be achieved by deploying the policy ‘Configure Azure Kubernetes Service clusters to enable Defender profile’ to help secure the containers.

Reference url to enable [Defender for Containers:](https://learn.microsoft.com/en-us/azure/defender-for-cloud/defender-for-containers-introduction)

[How to enable Microsoft Defender for Containers in Microsoft Defender for Cloud | Microsoft Learn](https://learn.microsoft.com/en-us/azure/defender-for-cloud/defender-for-containers-enable?pivots=defender-for-container-aks&tabs=aks-deploy-portal%2Ck8s-deploy-asc%2Ck8s-verify-asc%2Ck8s-remove-arc%2Caks-removeprofile-api)

**Solution approach:**

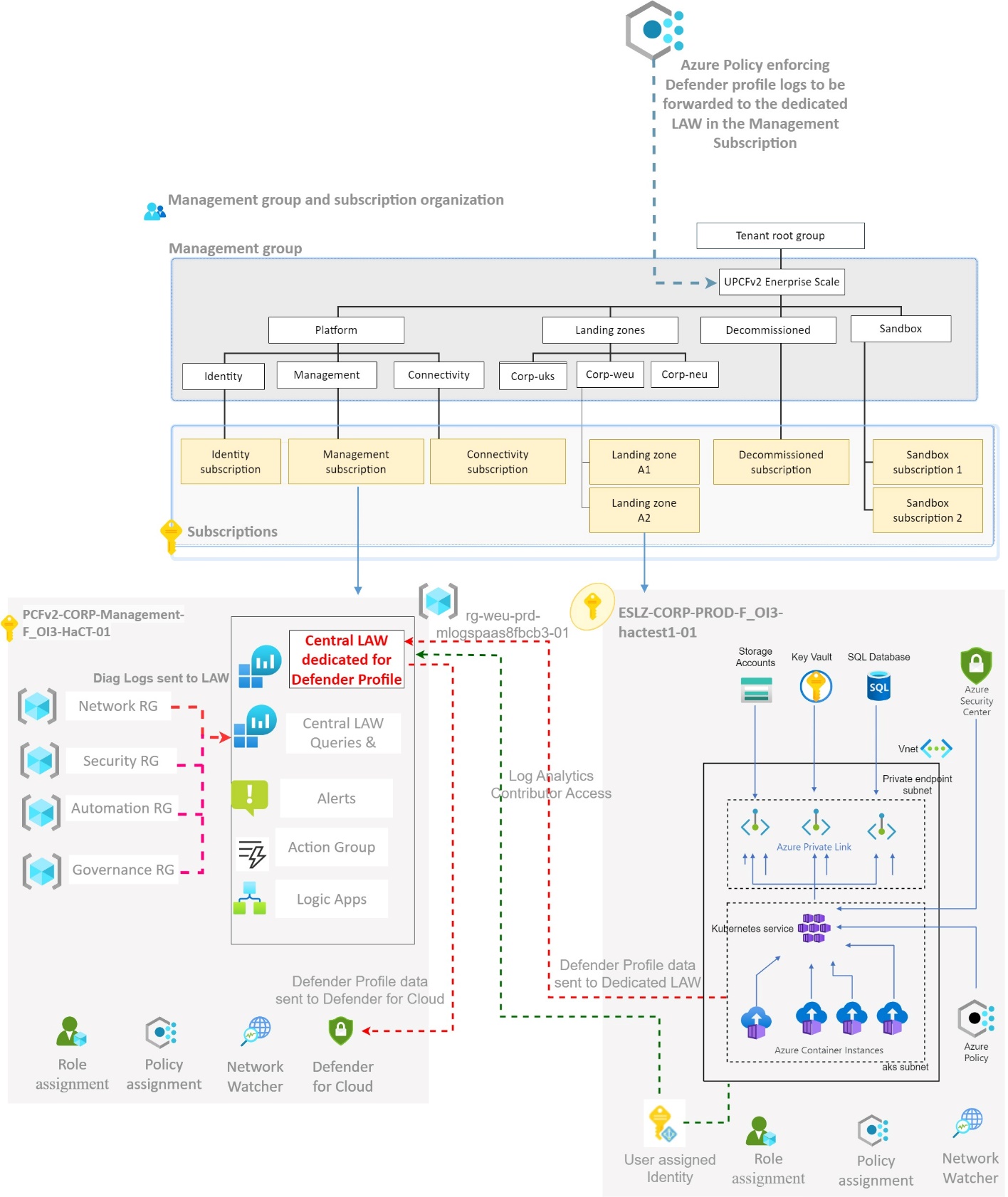
A Log Analytics workspace is used by the Defender profile as a data pipeline to send data from the cluster to Defender for Cloud without retaining any data in the Log Analytics workspace itself. As a result, there won't be additional cost involved in this case.

Our recommendation is to create a dedicated and central Log Analytics workspace in the Management Subscription ‘PCFv2-CORP-Management-F\_OI3-HaCT-01' within the Monitoring RG ‘rg-weu-prd-mlogspaas8fbcb3-01' which will be used by the Defender profile only as a place holder so the data can be forwarded to 'Defender for Cloud' as a result there is no additional cost involved, by this way the current central LAW won't be impacted and since the 'defender profile' placeholder LAW is going to be empty there is no need to for enabling any diag settings or alerts (Note: In PCFv1 environment the same approach with the dedicated and central workspace has been applied 'clusdfndrlogs-uni-oms-001')

The Azure policy will be scoped at the ‘UPCFv2 Enterprise Scale’ Management Group level, so it applies to all the PCFv2 Subscriptions, and the Log analytics Contributor access is required for the AKS Cluster's Azure AD workload identity to push the logs to the workspace.

This is required to be setup as a security requirement so Defender for Containers can assess cluster configurations and provide security recommendations, run vulnerability scans, and provide real-time protection and alerting for Kubernetes nodes and clusters

## **10.1.1 Defender Flow data flow to dedicated Central Workspace Design**



Defender for Containers manages three core aspects of container security:

* **Environment hardening** - Defender for Containers protects the AKS clusters and it continuously assesses clusters to provide visibility into misconfigurations and guidelines to help mitigate identified threats.
* **Vulnerability assessment via Qualys**- Vulnerability assessment and management tools for images stored in Azure Container Registry. For ES@Uniper, Qualys scans should be enabled on the central azure container registry which will be deployed in the PCFv2 environment to conduct regular automated scans and to capture the Common Vulnerabilities and Exposures (CVEs).

Reference url: <https://learn.microsoft.com/en-us/azure/defender-for-cloud/defender-for-containers-vulnerability-assessment-azure>

* **Run-time threat protection for nodes and clusters** - Threat protection for clusters and nodes generates security alerts for suspicious activities.

Refer to the link below for complete documentation on defender for containers.

<https://learn.microsoft.com/en-us/azure/defender-for-cloud/defender-for-containers-introduction>

## **10.2 Secure access to the API server and cluster nodes via RBAC**

The Kubernetes API server provides a single connection point for requests to perform actions within a cluster. To secure and audit access to the API server, limit access and provide the least permission levels

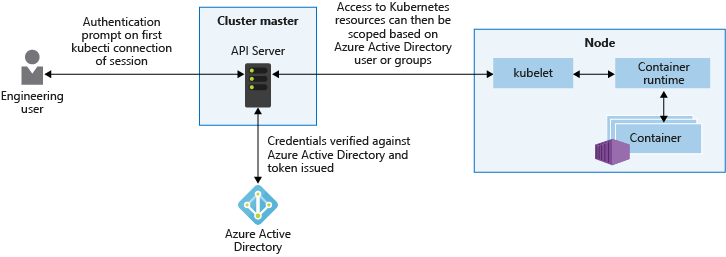
**Recommended RBAC model for ES@Uniper:**

* Azure AD Authentication with Kubernetes RBAC

Using Kubernetes RBAC and Azure AD-integration, we can secure the API server and provide the minimum permissions required to a scoped resource set, like a single namespace.

Azure RBAC for Kubernetes enables unified management and access control across Azure resources, AKS, and Kubernetes resources. When Azure RBAC for Kubernetes is enabled User identities and credentials for Kubernetes don’t have to be managed separately. Azure AD principals will be exclusively validated by Azure RBAC, and regular Kubernetes users and service accounts will be exclusively validated by Kubernetes RBAC.

The recommendation is to use groups to provide access to the resources instead of individual identities. Use Azure AD group membership to bind users to Kubernetes roles rather than individual users and local accounts to be disabled



Reference Documentation link for ES@Uniper RBAC

[ES@uniper - AKS- Role based Access.docx](https://uniper.sharepoint.com/:w:/r/sites/CloudWorksTeam/HaCT/Project%20Documentation/PCFv2/AKS%20Landing%20Zone%20Accelerator%20LLD/ES@uniper%20-%20AKS-%20Role%20based%20Access.docx?d=we40e300a74e84a9788342b8280c6dbd8&csf=1&web=1&e=lUXQc9)

## **10.3 Access Azure container registry using Azure Private Link:**

For ES@Uniper, the recommendation for azure container registry architecture follows the private endpoint model, to limit access to a registry by assigning virtual network private IP addresses to the registry endpoints and using Azure Private Link.

Network traffic between the clients on the virtual network and the registry's private endpoints traverses the virtual network and a private link on the Microsoft backbone network, eliminating exposure from the public internet.

**Refer to the Section:** 8.1.2 AKS Private Cluster

**Note:** Some functionality may be unavailable or require more configuration in a container registry that restricts access to private endpoints, selected subnets, or IP addresses.

* When public network access to a registry is disabled, registry access by certain trusted services including Azure Security Center requires enabling a network setting to bypass the network rules.
* Once the public network access is disabled, Instances of certain Azure services including Azure DevOps Services are currently unable to access the container registry.
* Private endpoints are not currently supported with Azure DevOps managed agents. Hence a self-hosted agent with network line of sight to the private endpoint need to be created
* If the registry has an approved private endpoint and public network access is disabled, repositories and tags can't be listed outside the virtual network using the Azure portal, Azure CLI, or other tools.

## **10.4 Azure AD workload identity for Azure Kubernetes Service (AKS)**

A workload identity is an identity which can be assigned to a workload (such as an application, service, script, or container) to authenticate and access other services and resources.

Workload identities are applications, service principles, and managed identities.

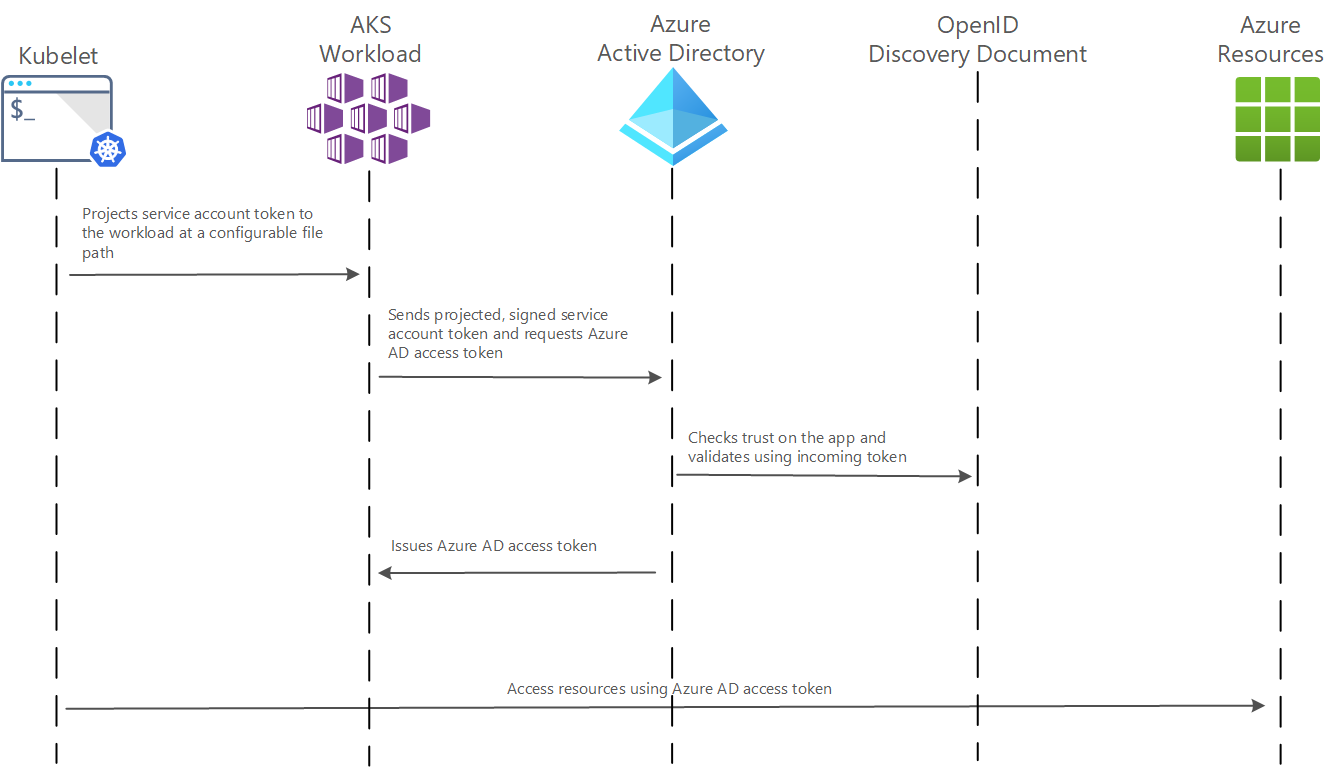
For ES@Uniper as a part of the AKS Cluster Architecture It is recommended to use ‘**User assigned Identity**’ which is secured and eliminates the need for developers to manage credentials.

Reference URL:

<https://learn.microsoft.com/en-us/azure/aks/learn/tutorial-kubernetes-workload-identity>

**Token workflow:**

Azure Active Directory uses OpenID Connect to discover public signing keys and verify the authenticity of the service account token before exchanging it for an Azure AD token. The workload can exchange a service account token projected to its volume for an Azure AD token using the Azure Identity client library or the Microsoft Authentication Library.



Five attack surfaces will be considered when creating a security strategy for Kubernetes clusters:

* Build.
* Registry.
* Cluster.
* Node
* Application.

Below reference url for more information:

<https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/scenarios/app-platform/aks/security>

## **10.5 Azure Key Vault Provider for Secrets Store CSI Driver:**

The CSI-driver provides a way to connect secrets, keys and certificates in Key Vault to deployments in Kubernetes and it allows the applications teams in ES@Uniper to mount secrets stored in a vault to the pods in the PCFV2 Environments.

In order to access the secrets stored in key vault, an user assigned identity needed to be configured into the cluster.

**Out of the box features:**

* Mounts secrets, keys, and certificates to a pod by using a CSI volume
* Supports CSI inline volumes
* Supports mounting multiple secrets store objects as a single volume
* Supports pod portability with the SecretProviderClass CRD
* Supports Windows containers
* Syncs with Kubernetes secrets
* Supports autorotation of mounted contents and synced Kubernetes secrets

**Limitations:** A container using subPath volume mount won't receive secret updates when it's rotated.

**Note**: HACT and Application Teams can adopt the existing solution and implementation approach for their clusters from the PCFv1 environment, reference url: <https://wiki.intranet.uniper.energy/sales/display/CF/Azure+Key+Vault+Provider+for+Secrets+Store+CSI+Driver>

## **10.6 Azure Policy Definitions for AKS Clusters**

To improve the security of the Azure Kubernetes Service (AKS) cluster, we need to apply and enforce security policies on the cluster using Azure Policy. Azure Policy helps to enforce organizational standards and to assess compliance at-scale. After installing the Azure Policy Add-on for AKS, we can apply individual policy definitions or groups of policy definitions called initiatives to the clusters.

Azure Policy Service makes it possible to manage and report on the compliance state of the Kubernetes clusters from a central location. Azure Policy add-on enacts the following functions:

* Checks with Azure Policy service for policy assignments to the cluster.
* Deploys policy definitions into the cluster as constraint template and constraint custom resources.
* Reports auditing and compliance details back to Azure Policy service.

The below url references the list of built-in Azure Policies for AKS which should be applied in the PCFv2 clusters.

**Note**: The Policies are inherited from the PCFv1 clusters to ensure the applications migrated to central and application owned clusters in PCFv2 meet the functional and non-functional requirements.

[Azure Enterprise LLD AKS Policies.docx](https://uniper.sharepoint.com/:w:/r/sites/CloudWorksTeam/HaCT/Project%20Documentation/PCFv2/AKS%20Landing%20Zone%20Accelerator%20LLD/Azure%20Enterprise%20LLD%20AKS%20Policies.docx?d=w1f08623fa3934910a142f33175979fe6&csf=1&web=1&e=xof2sk)

## **10.7 AKS - API Server IP Whitelisting for Clusters deployed with public endpoint**

The Kubernetes API server is the core of the Kubernetes control plane and is the central way to interact with and manage the clusters. To improve the security of the AKS clusters and minimize the risk of attacks, If the Application Teams are deploying AKS Clusters will public endpoints the recommendation is to limit the IP address ranges which can access the API server

Recommended IP Address Ranges to be White-listed:

* Azure DevOps Public ip list - 13.107.6.0/24,13.107.9.0/24,13.107.42.0/24,13.107.43.0/24
* Cloud shell - https://www.microsoft.com/en-us/download/details.aspx?id=56519
* ZScaler Ips - Frankfurt IV 165.225.72.0/22 & Amsterdam 185.46.212.0/23
* West Europe list IP
* UK South list IP
* Microsoft-Hosted Agent hosted on Azure IP range
* Azure DevOps ExpressRoute connections

## **10.8 Customer managed keys for Azure Container registry**

**Overview**: A customer-managed key gives ownership to the HACT Team in PCF v2 to bring their own key in Azure Key Vault. When the customer-managed key is enabled in Central Container Registry.

HACT Team will be responsible for managing its rotations, controlling access and permissions to use it as it is setup in the Central Container Registry.

**Key features include:**

**Regulatory compliance**: Azure automatically encrypts registry content at rest with service-managed keys.

**Integration with Azure Key Vault**: Customer-managed keys support server-side encryption through integration with Azure Key Vault. With customer-managed keys, HACT Team can create their own encryption keys and store them in a key vault.

**Key lifecycle management**: Integrating customer-managed keys with Azure Key Vault gives the HACT Team full control and responsibility for the key lifecycle, including rotation and management.

**Points to be noted by the HACT Team:**

* This feature is available in the Premium service tier for a container registry.
* The HACT Team will be allowed to enable a customer-managed key only while creating a registry.
* The HACT Team can't disable the encryption after they enable a customer-managed key on a registry.
* The HACT Team have to configure a user-assigned managed identity to access the key vault.
* Azure Container Registry supports only RSA or RSA-HSM keys. Elliptic-curve keys aren't currently supported.
* In a registry that's encrypted with a customer-managed key, it can retain logs for Azure Container Registry tasks for only 24 hours. To retain logs for a longer period, refer to <https://learn.microsoft.com/en-us/azure/container-registry/container-registry-tasks-logs#alternative-log-storage>
* Content trust is currently not supported in a registry that's encrypted with a customer-managed key.

**Content Trust:**

We will be enabling content trust policy for Azure container registry, so it enables push only for trusted images to the registry.

# **11. AKS Cluster High Availability and BCDR**

In PCF v2 Environment, AKS clusters will be configured with high resilience and the application teams need to plan to run their applications in a highly distributed AKS Cluster The following scenarios have been identified and should be considered by application teams when they plan for their application high availability:

* Application failure/node failure
* Zone outage
* Cluster outage

**Note**: In ES@Uniper, as per the architecture board decision disaster recovery is expected to provide resilience from the datacenter / zone outages with the same region e.g., West Europe

AKS clusters deployed in PCFv2 Landing Zone Subscriptions are configured for high-availability and advanced resilience to support BCDR (Business Continuity and Disaster Recovery) using the the availability Zone feature for all Zones.

## **11.1 Use of availability zones for all node pools:**

An Azure Kubernetes Service (AKS) cluster distributes resources such as nodes and storage across logical sections of underlying Azure infrastructure. Using availability zones physically separates nodes from other nodes deployed to different availability zones.

AKS clusters deployed with multiple availability zones configured across a cluster provide a higher level of availability to protect against a hardware failure or a planned maintenance event.

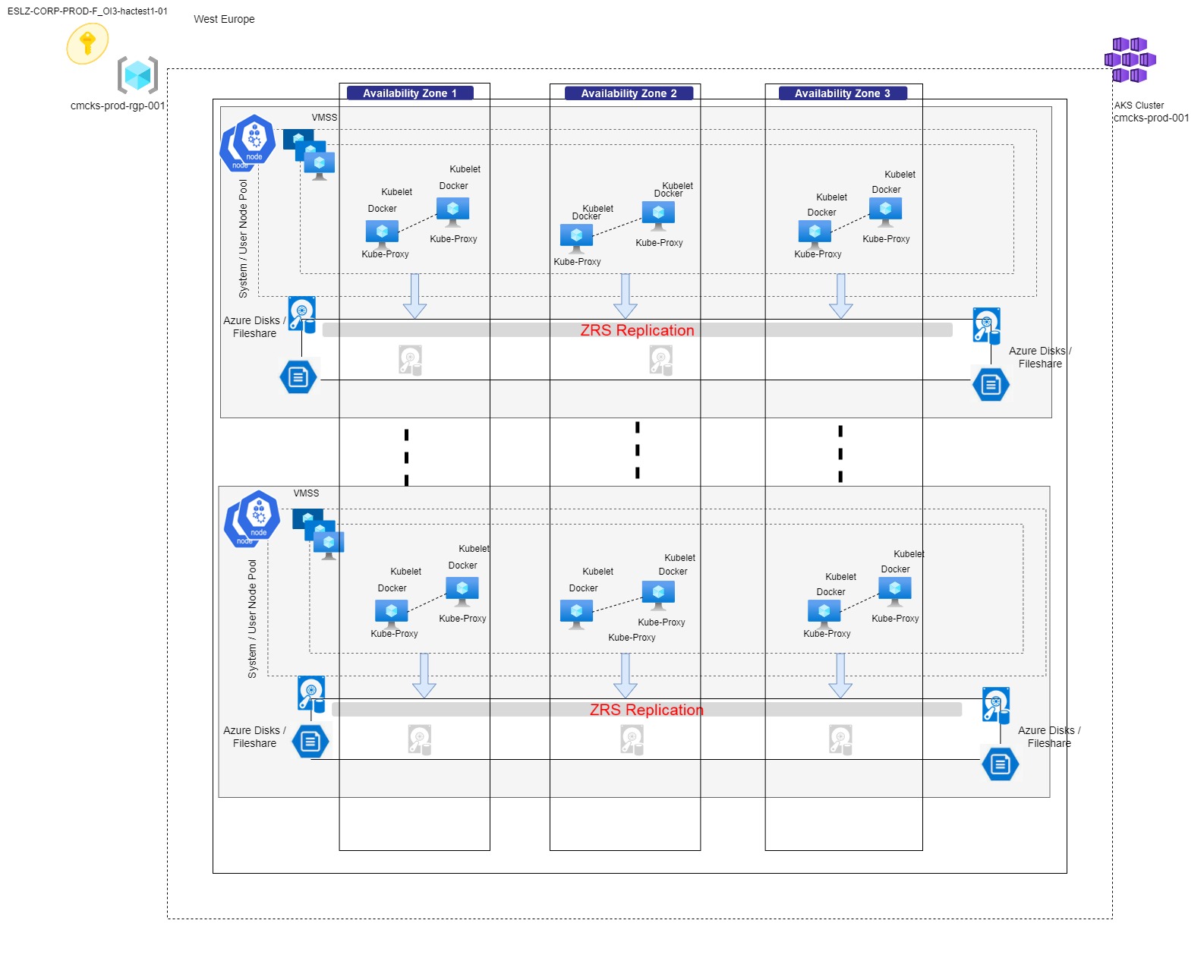
By defining node pools in a cluster to span multiple zones, nodes in each node pool are able to continue operating even if a single zone has gone down. The applications can continue to be available even if there's a physical failure in a single datacenter if orchestrated to tolerate failure of a subset of nodes.

West Europe and UK South supports 3 Availability Zones

**Recommendation:** To deploy Minimum of 3 Nodes with 3 Availability Zone for the AKS workload in HaCT Central AKS Clusters and application team owned clusters.

**Note**: Availability Zone should be enabled at the time of cluster deployment, hence the feature is enabled via Terraform Template and is expected to be used by the Application Team.

## **AKS Cluster High Availability Architecture Design:**



**Visiofile reference url:** [AKS HA.vsdx](https://uniper.sharepoint.com/:u:/r/sites/CloudWorksTeam/HaCT/Project%20Documentation/PCFv2/Visio%20Diagrams%20-%20ES@Uniper/AKS/AKS%20HA.vsdx?d=w810028db70714dd28f1cc22c8a981f52&csf=1&web=1&e=Jpo5YJ)

**Limitations:**

* Availability zones can be defined only when creating the cluster or node pool.
* Availability zone settings can’t be updated after the cluster is created.
* Node size (SKU) selected must be available across all availability zones selected.
* Azure Standard Load Balancers is required for clusters with availability zones enabled.
* Not all the Regions are supported to create Cluster using availability zones.

## **11.2 Monthly Uptime Calculation and Service Levels for AKS Clusters that use Availability Zones**

**Maximum Available Minutes** is the total accumulated minutes of an Availability Zone enabled AKS Cluster to the time Customer has initiated an action to stop or delete the AKS Cluster during a billing month.

**Downtime** is the total accumulated minutes that are part of Maximum Available Minutes with no connectivity to Kubernetes API Server from provisioned Availability Zone enabled AKS Cluster in the region.

**Monthly Uptime Percentage** for Availability Zone enabled AKS Clusters is calculated as Maximum Available Minutes Less Downtime divided by Maximum Available Minutes in a billing month for a given Microsoft Azure subscription. Monthly Uptime Percentage is represented by the following

Formula:

**"Maximum Available Minutes-Downtime" / "Maximum Available Minutes" x 100**

Uptime SLA Guaranteed by Microsoft is **99.95%** availability of the Kubernetes API server endpoint for clusters which use Availability Zones.

**Note**: The above SLA is covered only for the AKS Service, composite SLA for the complete application stack need to be calculated separately.

## **11.3 Enable redundant ingress controller configuration**

For added redundancy, for ES@Uniper it is recommended to deploy two replicas of the NGINX ingress controllers which are deployed with the ‘--set controller.replicaCount’ parameter. To fully benefit from running replicas of the ingress controller and follow the recommendations to run Minimum of 3 nodes within the Cluster.

## **11.4 Zone redundant default storage account for persistent volumes**

Azure Storage always stores multiple copies of the data so that it's protected from planned and unplanned events, including transient hardware failures, network or power outages, and massive natural disasters. Redundancy ensures that the storage account meets its availability and durability targets even in the face of failures.

When deciding which redundancy option is best for your scenario, consider the tradeoffs between lower costs and higher availability. The factors that help determine which redundancy option you should choose include:

For ES@Uniper, the storage accounts used for persistent volumes are Azure Fileshare and Azure Disks (Refer:) and they should use **Zone-redundant storage (ZRS)** setup, which copies the data synchronously across three Azure availability zones in the primary region. For applications requiring high availability, Microsoft recommends using ZRS in the primary region, and also replicating to a secondary region.

## **11.5 Enable Zone redundancy in Azure Container Registry for resiliency and high availability**

Azure Container Registry supports zone redundancy in West Europe and UK South Regions consumed by Uniper with Premium Tier, Zone redundancy provides resiliency and high availability to a registry or replication resource (replica) in a specific region for instance UK South.

For ES@Uniper Landing Zone, The Central ACR should be configured with Azure availability zones to create a resilient and high availability Azure container registry.

Application Teams should also consider setting up zone-redundant Azure container registry with other supported Azure resources to meet data residency or other compliance requirements, while providing high availability within a region.

## **11.6 Enable Geo-Replication of the Uniper Central Container Registry**

Azure Container Registry with Premium Tier supports Geo-replication to function as a single registry, serving multiple regions with multi-primary regional registries.

A geo-replicated registry provides the following benefits:

* Single registry, image, and tag names can be used across multiple regions
* Improve performance and reliability of regional deployments with network-close registry access
* Reduce data transfer costs by pulling image layers from a local, replicated registry in the same or nearby region as the container host.
* Single management of a registry across multiple regions
* Registry resilience if a regional outage occurs
* Push to a single registry, while ACR automatically manages the geo-replication. ACR only replicates unique layers, reducing data transfer across regions

For ES@Uniper Landing Zone, The Central ACR should be configured with geo-replication to provide a highly available registry that is resilient to regional outages for the application teams and also to manage a central repository across regions consumed by Uniper.

Reference url for AKS service quotas and limits of Azure Regions for

<https://learn.microsoft.com/en-us/azure/aks/quotas-skus-regions>

# **12. Auto-scaling for AKS nodepools**

To keep up with application demands in Azure Kubernetes Service (AKS), the Application Team needs to adjust the number of nodes which run their workloads.

The cluster autoscaler component watches for pods in the cluster which can't be scheduled because of resource constraints. When such issues are detected, the number of nodes in a node pool increases to meet the application demand.

To adjust to changing application demands, such as between the workday and evening or on a weekend, clusters often need a way to automatically scale. AKS clusters can scale in one of two ways:

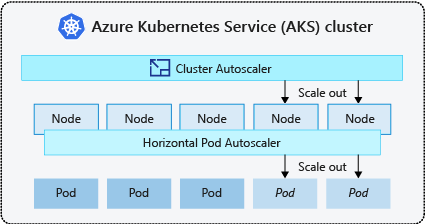
* The **cluster autoscale**r watches for pods that can't be scheduled on nodes because of resource constraints. The cluster then automatically increases the number of nodes.
* The **horizontal pod autoscaler** uses the Metrics Server in a Kubernetes cluster to monitor the resource demand of pods. If an application needs more resources, the number of pods is automatically increased to meet the demand.

|  |
| --- |
| **Note**: In ES@Uniper, AKS Autoscaling is recommended to be achieved by applying cluster autoscaler alongside the horizontal pod autoscaler, the required parameter values for auto-scaling is expected to be recommended by the application teams based on the workload expected to run on the cluster and incorporate into the application code. |

## **12.1 Cluster autoscaler**

To respond to changing pod demands, the Kubernetes cluster autoscaler adjusts the number of nodes based on the requested compute resources in the node pool. By default, the cluster autoscaler checks the Metrics API server every 10 seconds for any required changes in node count. If the cluster autoscaler determines that a change is required, the number of nodes in the AKS cluster is increased or decreased accordingly.

**Note**: The cluster autoscaler works with Kubernetes RBAC-enabled AKS clusters that run Kubernetes 1.10.x or higher.



The cluster autoscaler is typically used alongside the horizontal pod autoscaler. When combined, the horizontal pod autoscaler increases or decreases the number of pods based on application demand, and the cluster autoscaler adjusts the number of nodes to run additional pods.

**Scale out events:** If a node doesn't have sufficient compute resources to run a requested pod, that pod can't progress through the scheduling process. The pod can't start unless additional compute resources are available within the node pool.

When the cluster autoscaler notices pods that can't be scheduled because of node pool resource constraints, the number of nodes within the node pool is increased to provide the additional compute resources.

**Scale in events:** This event indicates the node pool has more compute resources than required, and the number of nodes can be decreased. By default, nodes that pass a threshold for no longer being needed for 10 minutes is scheduled for deletion. When this situation occurs, pods are scheduled to run on other nodes within the node pool, and the cluster autoscaler decreases the number of nodes.

**Use the cluster autoscaler profile:** Application Teams deploying AKS Clusters in ES@Uniper can configure more granular details of the cluster autoscaler by changing the default values in the cluster-wide autoscaler profile, the values for settings will be defined by the application teams based on the workload running on the clusters.

For example, a scale down event happens after nodes are under-utilized after 10 minutes. If there are workloads which run every 15 minutes, Application teams can change the autoscaler profile to scale down under-utilized nodes after 15 or 20 minutes

The below table defines the cluster-wide autoscaler profile default value which can be used for reference by the HaCT and the application teams.

|  |  |  |
| --- | --- | --- |
| **Setting** | **Description** | **Default value** |
| scan-interval | How often cluster is reevaluated for scale up or down | 10 seconds |
| scale-down-delay-after-add | How long after scale up that scale down evaluation resumes | 10 minutes |
| scale-down-delay-after-delete | How long after node deletion that scale down evaluation resumes | scan-interval |
| scale-down-delay-after-failure | How long after scale down failure that scale down evaluation resumes | 3 minutes |
| scale-down-unneeded-time | How long a node should be unneeded before it is eligible for scale down | 10 minutes |
| scale-down-unready-time | How long an unready node should be unneeded before it is eligible for scale down | 20 minutes |
| scale-down-utilization-threshold | Node utilization level, defined as sum of requested resources divided by capacity, below which a node can be considered for scale down | 0.5 |
| max-graceful-termination-sec | Maximum number of seconds the cluster autoscaler waits for pod termination when trying to scale down a node | 600 seconds |
| balance-similar-node-groups | Detects similar node pools and balances the number of nodes between them | false |
| expander | Type of node pool [expander](https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-are-expanders) to be used in scale up. Possible values: most-pods, random, least-waste, priority | random |
| skip-nodes-with-local-storage | If true cluster autoscaler will never delete nodes with pods with local storage, for example, EmptyDir or HostPath | false |
| skip-nodes-with-system-pods | If true cluster autoscaler will never delete nodes with pods from kube-system (except for DaemonSet or mirror pods) | true |
| max-empty-bulk-delete | Maximum number of empty nodes that can be deleted at the same time | 10 nodes |
| new-pod-scale-up-delay | For scenarios like burst/batch scale where you don't want CA to act before the kubernetes scheduler could schedule all the pods, you can tell CA to ignore unscheduled pods before they're a certain age. | 0 seconds |
| max-total-unready-percentage | Maximum percentage of unready nodes in the cluster. After this percentage is exceeded, CA halts operations | 45% |
| max-node-provision-time | Maximum time the autoscaler waits for a node to be provisioned | 15 minutes |
| ok-total-unready-count | Number of allowed unready nodes, irrespective of max-total-unready-percentage | 3 nodes |

**Sample code to set the cluster autoscaler profile on a new cluster**

|  |
| --- |
| az aks create \  --resource-group myResourceGroup \  --name myAKSCluster \  --node-count 3 \  --enable-cluster-autoscaler \  --min-count 3 \  --max-count 6 \  --cluster-autoscaler-profile scan-interval=30s |

Note to application teams when using the autoscaler profile:

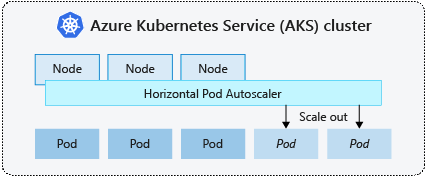
|  |
| --- |
| * The cluster autoscaler profile affects all node pools which use the cluster autoscaler. An autoscaler profile can’t be set per node pool. When the profile is set, any existing node pools with the cluster autoscaler enabled immediately start using the profile. * The cluster autoscaler profile requires version 2.11.1 or greater of the Azure CLI. |

## **12.2 Horizontal pod autoscaler**

AKS uses the horizontal pod autoscaler (HPA) to monitor the resource demand and automatically scale the number of pods. By default, the HPA checks the Metrics API every 15 seconds for any required changes in replica count, and the Metrics API retrieves data from the Kubelet every 60 seconds.

When changes are required, the number of replicas is increased or decreased accordingly.

**Note**: The HPA works with AKS clusters that have deployed the Metrics Server for Kubernetes 1.8+.



For ES@Uniper when Application team configures Cluster deployment, they need to define the minimum and maximum number of replicas that can run along with it define the metric to monitor such as CPU, memory, etc. usage limits.

**Note**: Metric parameter value for setting up HPA need to be identified and defined by the Application Team depending on the workload they run on the AKS cluster

**Cooldown of scaling events**: To minimize race events, a delay value is set. This value defines how long the HPA must wait after a scale event before another scale event can be triggered.

**Note**: There's [no delay for scale-up events as of Kubernetes 1.12](https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/#support-for-cooldown-delay), however, the default delay on scale down events is 5 minutes.

**Sample code to set the HPA**

|  |
| --- |
| containers:  - name: azure-vote-front  image: mcr.microsoft.com/azuredocs/azure-vote-front:v1  ports:  - containerPort: 80  resources:  requests:  cpu: 250m  limits:  cpu: 500m |

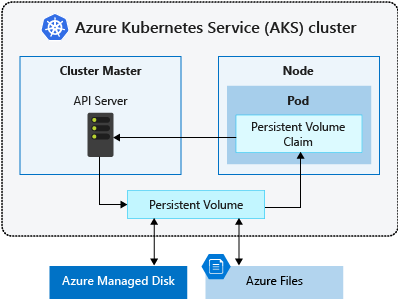
Reference url: <https://learn.microsoft.com/en-us/azure/aks/cluster-autoscaler>

# **13. Persistent Volume Storage Recommendations for AKS PCFv2**

Applications running in Azure Kubernetes Service (AKS) may need to store and retrieve data. While some application workloads can use local, fast storage on unneeded, emptied nodes, others require storage that persists on more regular data volumes within the Azure platform.

Multiple pods may need to:

* Share the same data volumes.
* Reattach data volumes if the pod is rescheduled on a different node.



In AKS data storage capability is provided by Azure storage. For the Central and for the Application Team Clusters in ES@Uniper the application teams will be required to attach Azure Disk or Azure Files. This storage is mounted on deployments/pods with the help of persistent volumes.

## **13.1 Persistent Volumes:**

**Azure Disk or Azure Files** are used to provide the PersistentVolume. The choice of Disks or Files is often determined by the need for concurrent access to the data or the performance tier.

A Persistent Volume is expected to be created dynamically by the Application Teams with the help of Kubernetes API server, Dynamic provisioning uses a StorageClass to identify what type of Azure storage needs to be created.

We have different tiers of storage.

* Premium
* Standard

Application Team can also create a StorageClass on their own. It defines the reclaimPolicy.  
reclaim policies are - "Retain", "Recycle", and "Delete"

This reclaimPolicy controls the behaviour of the underlying Azure storage resource when the pod is deleted, and the persistent volume may no longer be required. The underlying storage resource can be deleted or retained for use with a future pod.

Application Teams need to use Container Storage Interface (CSI) drivers in reference to the StorageClasses

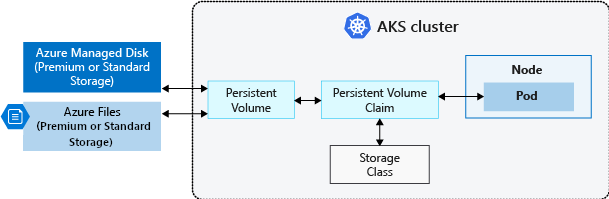
CSI Storage class reference url: <https://learn.microsoft.com/en-us/azure/aks/concepts-storage>

when no storage class is specified for a persistent volume, the default storage class will be used, so the application teams need to ensure the appropriate storage for persistent volumes- is provided.

## **13.2 Dynamically provision volumes**

To reduce management overhead and enable scaling, avoid statically create and assign persistent volumes. Application Teams should use dynamic provisioning and for the storage classes they need to define the appropriate reclaim policy to minimize unneeded storage costs once pods are deleted.

Creating persistent volumes manually adds management overhead and limits the ability to scale. Hence for PCF V2 Environment persistent volume will be provisioned dynamically to simplify storage management and allow the applications to grow and scale as needed.

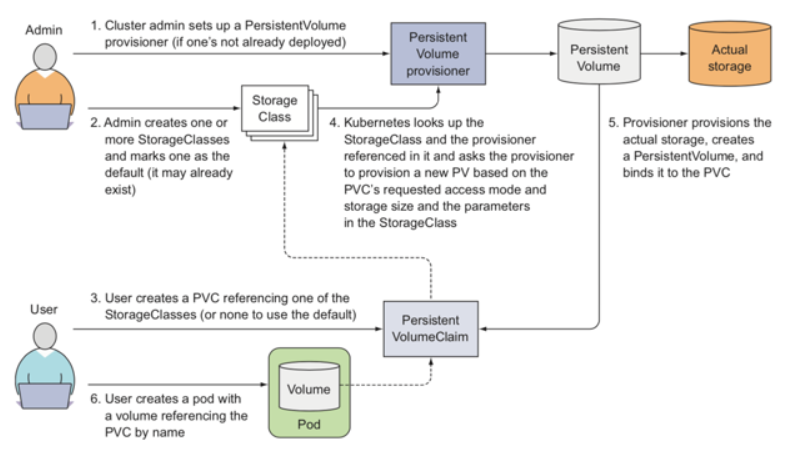


A persistent volume claim (PVC) lets the application teams dynamically create storage as needed. In the pod definition, request a volume to be created and attached to a designated mount path.

Below are the two approved Storage types in ES@Uniper and how they can be created dynamically:

**Type1 : Azure Fileshare** : This approach is followed when multiple pods need concurrent access to the same storage volume, Application Team and the HACT Team are expected to go with Azure Fileshares for all their requirements unless there are some technical limitations and in such cases they can opt for managed disks The application teams can use Azure Files to connect using the Server Message Block (SMB) version 3.1.1 share or Network File System (NFS) version 4.1. For central and for the application team clusters in ES@Uniper, we will be required to use ZRS based storage accounts to meet the high availability requirements. Reference section for ZRS Storage ‘**10 AKS Cluster High Availability Architecture Design’**

**Sample process for assigning PVC:**



If the Filehare doesn’t exist, it is dynamically created when the PersistentVolumeClaim is created. The storage account will be managed by the HaCT or Application teams depending upon which Landing Zone Subscription.

* A default ZRS storage account is provided to store file shares of the application teams.
* The application team defines a PersistentVolumeClaim together with its size and a specific access mode and deploys it.

## **13.3 Dynamically create Azure Files PVs by using the built-in storage classes**

Storage CSI drivers on AKS provides two built-in StorageClasses that uses the Azure Files CSI storage drivers.

* azurefile-csi: Uses Azure Standard Storage to create an Azure file share.
* azurefile-csi-premium: Uses Azure Premium Storage to create an Azure file share.

The reclaim policy on both storage classes ensures that the underlying Azure files share is deleted when the respective PV is deleted

Reference url: <https://learn.microsoft.com/en-us/azure/aks/azure-files-csi>

**Type2: Azure Disks:**

An Azure disk can only be mounted with Access mode type ReadWriteOne, which makes it available to only a single pod in AKS, hence application teams are expected to choose azure managed disks based on technical requirements as recommended by Microsoft.

If they need to share a persistent volume across multiple pods, then they need to use Azure Files.

## **13.4 Dynamically create Azure Disks PVs by using the built-in storage classes**

A storage class is used to define how a unit of storage is dynamically created with a persistent volume.

When the application teams use the Azure Disk CSI driver on AKS, there are more built-in StorageClasses which use the Azure Disk CSI storage driver.

* managed-csi: Uses Azure Standard SSD locally redundant storage (LRS) to create a managed disk.
* managed-csi-premium: Uses Azure Premium LRS to create a managed disk.

The reclaim policy in both storage classes ensures that the underlying Azure Disks are deleted when the respective PV is deleted.

Reference url: <https://learn.microsoft.com/en-us/azure/aks/azure-disk-csi>

# **14. AKS Monitoring**

Azure’s native monitoring tool is Azure Monitor which is used to monitor the health and performance of the azure services. It helps with effectively monitoring the AKS cluster, which includes features like container insights, alerts, metrics, logs, diagnostic settings, advisor recommendations, and workbooks.

AKS generates platform metrics and resource logs which the application teams can use to monitor basic health and performance.

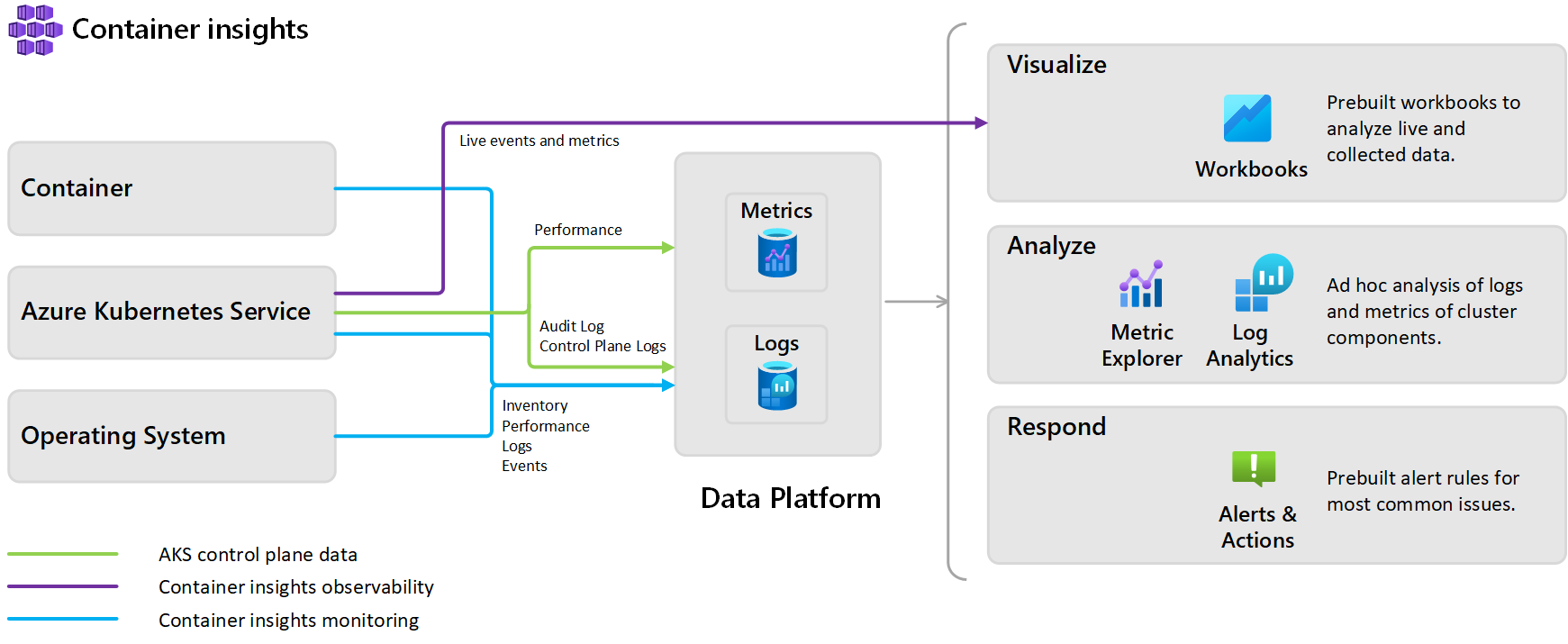
## **14.1 Container Insights**

In PCF V2 environment, Container Insights should be enabled for the Central and the Application Teams Clusters to expand on the monitoring capabilities.

Container Insights is a feature in Azure Monitor which monitors the health and performance of managed Kubernetes clusters hosted on AKS, it gives performance visibility by collecting memory and processor metrics from controllers, nodes, and containers which are available in AKS through the Metrics API.

Container Insights data is sent to a Log analytics workspace, for ES@Uniper the insights data should be sent to the De-central Log Analytics workspace which is dedicated for the specific Landing Zone Subscription.

## **Container Insights workflow**



## **Features of Container insights**

Container insights deliver a comprehensive monitoring experience to understand the performance and health of the AKS Cluster and container workloads, In ES@Uniper Application Teams and the AKS Central Team should be able to follow the below guidelines for AKS Monitoring using the native solution.

* Identify resource bottlenecks by identifying AKS containers running on the node and their processor and memory utilization.
* View the controller's or pod's overall performance by identifying where the container resides in a controller or a pod.
* Review the resource utilization of workloads running on the host which are unrelated to the standard processes that support the pod.
* Identify capacity needs and determine the maximum load that the cluster can sustain by understanding the behavior of the cluster under average and heaviest loads.

Configure alerts to proactively notify the respective teams or record when CPU and memory utilization on nodes or containers exceed the thresholds which are defined, or when a health state change occurs in the cluster at the infrastructure or nodes health rollup.

Reference url: <https://wiki.intranet.uniper.energy/sales/display/CF/Azure+Log+Analytics+and+Azure+Monitor+-+Enabling+AKS+logging>

## **Alerts**

AKS service has native azure monitor alert feature which allows us to configure notifications for certain events which occur in the cluster and based on the severity of the alerts they can be sent as email notifications to team’s DL or create service now tickets or send voice calls during non-business hours.

For example, Alerts can be configured to notify the application teams when the cluster resources consumption, like CPU or memory consumption, reaches the 95 percent mark which can be classified as critical alert.

Alert notifications for AKS will be captured in the monitoring LLD

url: <https://uniper.sharepoint.com.mcas.ms/:w:/r/sites/CloudWorksTeam/_layouts/15/Doc.aspx?sourcedoc=%7BE8AC7F3E-440F-401D-AD95-5386A7D2EBF3%7D&file=Azure%20Enterprise-Scale%20PCFv2%20-%20Monitoring%20-%20Low%20Level%20Design%201.2.docx&action=default&mobileredirect=true>

Section: **7.2.1 Dedicated LAW and several Azure Policies**

Sub Section: **2) Configuration of the basic alerts and their severity classifications for de-centralized resources:**

## **Metrics**

The Metrics page displays a time series reporting of different components and resources in the cluster, which the application teams should be able to leverage out of the box to visually correlate trends, and investigate spikes and dips in your metrics values, as per the ES@Uniper Landing Zone Architecture ‘All Metrics’ logs for AKS will be sent to the De-Centralized log Analytics workspace.

For example, the application teams can view a time series of the percentage usage of the CPU resources. Or can view the ready state of your cluster pods across a period.

## **Resource Logs**

Azure Monitor Logs collects data from the Kubernetes components and resources. These logs, generated from the nodes, pods, containerized applications, and Kubernetes control plane, can be viewed and analyzed on the Logs queries page.

This page has a sophisticated query analyzer which the application teams can use to query the data using Kusto Query Language.

As per the ES@Uniper Landing Zone design Diagnostics settings is enabled for AKS to collect resource logs. The logs are sent to a single destination which is the de-centralized Log Analytics workspace dedicated for the specific landing zone subscription using Azure Policy.

Collecting AKS resource logs and sending them to the decentralized LAW has been captured in the monitoring LLD

url: <https://uniper.sharepoint.com.mcas.ms/:w:/r/sites/CloudWorksTeam/_layouts/15/Doc.aspx?sourcedoc=%7BE8AC7F3E-440F-401D-AD95-5386A7D2EBF3%7D&file=Azure%20Enterprise-Scale%20PCFv2%20-%20Monitoring%20-%20Low%20Level%20Design%201.2.docx&action=default&mobileredirect=true>

Section: **7.2.1 Dedicated LAW and several Azure Policies**

Sub Section: **1) Diagnostic logs from de-centralized resources**

**Note**: The Log analytics workspace approach is de-centralized meaning each Landing Zone in ES will be deployed with dedicated or decentralized workspace and the respective Application / HaCT team will have access to their workspaces where the cluster insight logs are ingested.

Permission details are covered under the rbac section **‘10.2 Secure access to the API server and cluster nodes via RBAC’**

Activity and audit logs are sent to central workspace in the management Subscription which will be accessed by the HaCT security team.

## **Cost governance with Kubecost**

Cost governance is the continuous process of implementing policies to control costs. In AKS, there are several ways application teams can control and optimize costs. In ES@Uniper, Teams can utilize native Kubernetes tooling to manage and govern resource usage and consumption and proactively monitor and optimize the underlying infrastructure.

Kubecost is already used by the central and the application teams in PCFv1, they can follow the below documentation to setup Kubecost for ES@Uniper Clusters.

<https://wiki.intranet.uniper.energy/sales/display/CF/AKS+V2+-+Resource+Utlization+Tool+-+KubeCost>

**Note**: Application Teams can explore and implement monitoring solutions like Prometheus and Grafana for their AKS Clusters based on their requirement,

The below reference url can be used for guidance:

<https://wiki.intranet.uniper.energy/sales/display/CF/Prometheus+and+Grafana+-+Setting+up+in+AKS>

# **15. Naming Conventions to be followed**

**Overview**: The below table lists out various naming conventions which can be adopted while hosting applications on Application Team owned and HaCT Team’s centrally managed clusters / containers namespace.

| **Resources** | **Resources Names** | **Example** | **Convention guideline** |
| --- | --- | --- | --- |
| Resource Group name | Rg-location-env-resource type-version | rg-weu-prd-aks-01  Length: 1-64 |  |
|  |  |  |  |
| Cluster | ApplicationTeam/HACT Team-environment-version | prd-hact-aks-01, |  |
| Namespaces | application\_name-iteraplanid-env | helloworld-1234-dev | Use a namespace for all Kubernetes cluster objects. Request a namespace from [HaCT](https://wiki.intranet.uniper.energy/sales/pages/viewpage.action?pageId=45089074)   to distinguish applications Uniper-wide. |
| Deployments | application\_name-deployment\_name-env | helloworld-nginx-dev |  |
| pods | application\_name | helloworld |  |
| Container images and tags | application\_name-image-version | helloworld:20180731.1  helloworld:V2.1 | Use the official version of an image and use V as a prefix. Do not use the ‚latest‘ tag in the container registry.  For developed applications use an additional tag to determine the source version the image was created with as secondary tag.\* This additional tag should have the format YYYYMMDD.<numbe>. |
| Services | application\_name-service | helloworld-service |  |
| Secrets | application\_name-secret | application1-secret |  |
| Ingress | application\_name-ingress | helloworld-ingress |  |
| ConfigMaps | application\_name-configmap | helloworld-configmap |  |
| persistentvolumeclaims | application\_name-pvc | helloworld-pvc |  |
| persistentvolumes | application\_name-pv | helloworld-pv |  |
| resourcequotas | namespacename-quota | helloworld-quota |  |
| daemonsets | application\_name-daemonset | helloworld-deamonset |  |
| limitranges | application\_name-limits | helloworld-limits |  |
| serviceaccounts | application\_name-sa | helloworld-sa |  |
| cronjobs | application\_name-cronjob | helloworld-cronjob |  |
| jobs | application\_name-job | helloworld-job |  |
| networkpolicies at namespacelevel | namespacename-netpol | helloworld-netpol |  |
| networkpolicies at app level | application\_name-netpol | helloworld-netpol |  |
| csidrivers | application\_name-csidriver | helloworld-csidriver |  |

# **16. Patch Management and Version Upgrade for AKS Clusters in PCFv2 Environment**

**Overview**: The following Assets are covered by the patch process defined in the PCFv2 environment for the Central Clusters which are managed by the HaCT AKS Team and the application team owned clusters:

* The Kubernetes version that is running on the cluster and the cluster nodepools.
* Containers run by HaCT and the Application Teams to manage their cluster environment (Currently these are: Kured, Nginx Ingress Controller, Qualys vulnerability scanner, Kubecost, Open Service Mesh, Prometheus, Github runner, self-hosted windows agent, Flux).
* HaCT and the application teams are responsible for patching their respective managed container images.
* The Controlplane of the AKS clusters including Kubernetes version and the Kubernetes version of the underlying nodepools.

The virtual machines are patched by Microsoft on a regular basis and may in certain cases require a restart of the corresponding virtual machine (node). HaCT is running a restart daemon (kured) on each HaCT Managed AKS cluster that will automatically trigger a restart of a virtual machine when required.

In addition, the Application team upgrades all AKS clusters and the underlying nodepools to the latest Kubernetes version on a quarterly basis. The upgrade will happen on non-production environments first and afterwards (usually 4 weeks later) on the corresponding production environment.

Note: It is the responsibility of Application team to keep their container images up to date. It is recommended to provide an updated container image at least quarterly so that not only potential application issues are fixed but also to ensure that the application images are using base images with the latest security updates.

**Regular Patch Process for cluster upgrade**

* Prepare the Patch Request
* Identify the target cluster version
* Check for API changes in the new cluster version
* Test the patch in POC cluster
* Compile mail to required stakeholders and inform them about the planned date and API changes if necessary for non-prod cluster
* Compile mail to required stakeholders and inform them about the planned date and API changes if necessary for prod cluster

## **Local roles and responsibilities incl. RACI**

* Application teams are responsible for patching their own application images running on the application team owned clusters in the ESLZ.
* HaCT Team is responsible for patching the images running on the central clusters.
* HaCT Team identifies the vulnerabilities in the images from the Central ACR and share the report quarterly.
* HaCT Team fixes the vulnerabilities identified in the Central ACR.

## **Upgrade timelines**

* Major upgrade – To be done based on Microsoft decelerations.
* Minor upgrade – To be done on quarterly basis
* Patch update – Automated and scheduled on Monday between 6 AM – 8 AM CET.

Reference links for Patch update from PCFv1, which will be adopted for PCFv2 based on the RACI mentioned in the section 16.1 LOCAL ROLES AND RESPONSIBILITIES INCL. RACI

[Patching AKS Cluster and nodepools - Hosting Framework - Uniper Wiki](https://wiki.intranet.uniper.energy/sales/pages/resumedraft.action?draftId=145490760&draftShareId=6c9a6478-24d7-4ca6-9377-4c6a4f2df3f1&)  
[Patching HaCT managed container images - Hosting Framework - Uniper Wiki](https://wiki.intranet.uniper.energy/sales/pages/resumedraft.action?draftId=145490762&draftShareId=df811a32-621a-474c-9109-19c814515f16&)

[Patching application images - Hosting Framework - Uniper Wiki](https://wiki.intranet.uniper.energy/sales/pages/resumedraft.action?draftId=145490763&draftShareId=0185ead8-d096-417c-ad0e-5543fe065074&)

## **Cluster Upgrade:**

As part of the life cycle of an AKS cluster, HaCT and the Application Teams need to upgrade their clusters to the latest Kubernetes version.

It is important the teams apply the latest Kubernetes security releases, or upgrade to get the latest features.

Cluster upgrade process from PCFv1 will be adopted for PCFv2 based on the RACI mentioned in the section 16.1 LOCAL ROLES AND RESPONSIBILITIES INCL. RACI

Reference link for AKS cluster upgrades .

[Upgrade an Azure Kubernetes Service (AKS) cluster - Hosting Framework - Uniper Wiki](https://wiki.intranet.uniper.energy/sales/display/CF/Upgrade+an+Azure+Kubernetes+Service+%28AKS%29+cluster)

# **17. Certificate rotation in Azure Kubernetes Service (AKS)**

**Overview**: Azure Kubernetes Service (AKS) uses certificates for authentication with many of its components.

For RBAC-enabled cluster built after March 2022 Microsoft has enabled certificate auto-rotation.

AKS will automatically rotate non-CA certificates on both the control plane and agent nodes within 80% of the client certificate valid time, before they expire with no downtime for the cluster.

However, for PCF v2 Clusters in ES@Uniper, HaCT and the application teams can rotate the certificates by performing an AKS version upgrade periodically.

## **Certificate Auto Rotation**

For AKS to automatically rotate non-CA certificates, the cluster must have TLS Bootstrapping which has been enabled by default in all Azure regions.

**Note**: If the application team have an existing cluster that where they have to upgrade a cluster to enable Certificate Auto-Rotation. It is recommended not to not disable bootstrap to keep the certificate auto-rotation enabled.

To verify if TLS Bootstrapping is enabled on the cluster browse to the following paths:

**Linux node**: /var/lib/kubelet/bootstrap-kubeconfig or /host/var/lib/kubelet/bootstrap-kubeconfig

**Windows node**: C:\k\bootstrap-config

Section ‘16.3 CLUSTER UPGRADE’ can be referenced for cluster upgrade or Azure CLI can be used

‘az aks upgrade -g $RESOURCE\_GROUP\_NAME -n $CLUSTER\_NAME’

## **Manually rotate cluster certificates**

Using the below az command cluster certificates can be rotated periodically example periodically every 90 days.

**az aks rotate-certs -g $RESOURCE\_GROUP\_NAME -n $CLUSTER\_NAME**

refersh the certificate used by kubectl

**Beware**: Rotating the certificates manually using ‘az aks rotate-certs’ will recreate all of the cluster nodes, VM scale set and their Disks and can cause up to 30 minutes of downtime for your AKS cluster

**Recommendation for AKS certificate rotation:**

Certificate auto-rotation is recommended for the Clusters deployed in PCFv2.

Reference url: <https://learn.microsoft.com/en-us/azure/aks/certificate-rotation>

.