Design Considerations for Azure Kubernetes Services

Created By: Atul Raizada, Dated: 08-18-19

Topics

- Green Blue AKS Cluster Strategy
- Cluster Isolation
  - Physical
  - Logical
  - Hybrid
  - Network
- Resource Quota
- AKS Scaling
  - Manual Scaling
  - Horizontal Pod Autoscaler (HPA)
  - Cooldown of Scaling Events
- Cluster Autoscaler
  - Scale up Events
  - Scale down Events
- Burst to Azure Container Instances
- AKS Security
  - Cluster Level Security
  - API Server
  - Authentication and Authorization (AAD + RBAC)
    - Cluster Level - Identity and Access Management through AAD & RBAC
    - Azure Level – Identity & Access Management through AAD and RBAC
    - Least privileged access for common tasks
  - Admission Controllers
  - ETCD Security
  - Container Level Security – Images
  - Pod Level Security
    - Pod Level Security Context
    - Pod Level Policies (Preview)
    - Always Pull Images
    - Pod Identity
    - Pod Network Isolation using Network Policies
  - Managing Secrets
  - Azure Container Registry Security
- Implementing Trusted Software Supply Chain (DevOps)
  - Source Control
  - Imaging Scanning
  - Run as Root User
  - Image Integrity Controls
  - Enforce use of Trusted Images
- Securing Applications and APIs
- AKS Cluster Update
- AKS Node Upgrade and Patching
- AKS Audit and Monitoring
- AKS RACI

AKS Green Blue Cluster Strategy

- Blue-green deployment is a technique that reduces downtime and risk by running two identical production environments called Blue and Green. At any time, only one of the environments is live, with the live environment serving all production traffic.
- This strategy brings required resiliency, but at the same time adds cost, operational overheads, failover design complexity and strategy.

Cluster Isolation

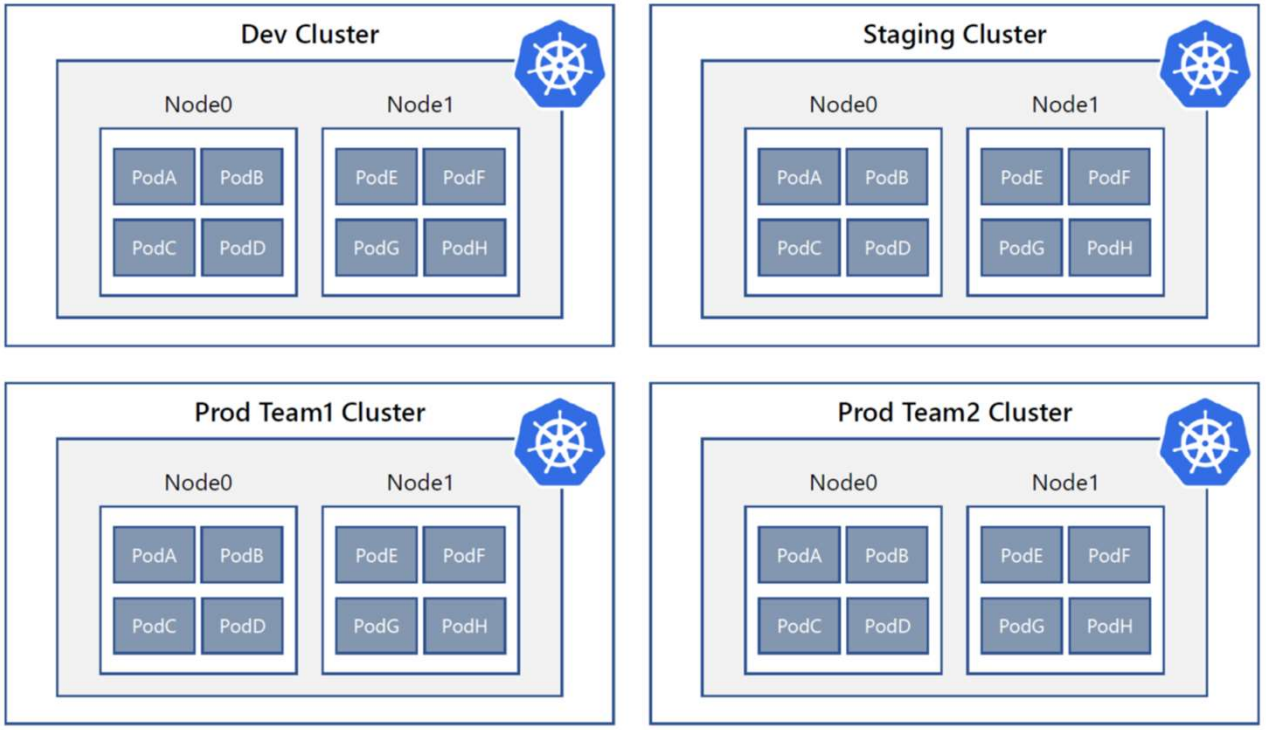
Cluster Isolation can be achieved through,

- Physical Isolation
- Logical Isolation
- Hybrid Approach
- Network Isolation

Pros and Cons of Physical vs Logical Isolation Approach

	Physical	Logical
Pod Density	Low to Medium	Medium to High
Cost	$$	$
Kubernetes Experience	Low to Medium	Medium to High
Security	High (Surface is small)	High*
Blast Radius of Changes	Small	Big
Management and Operations	Owner Team	Single or Cross Functional Team

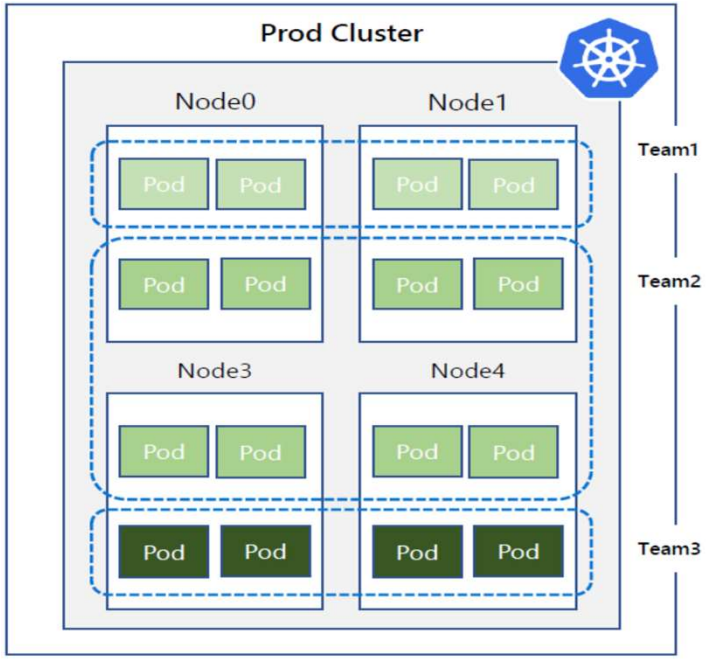
Physical Isolation



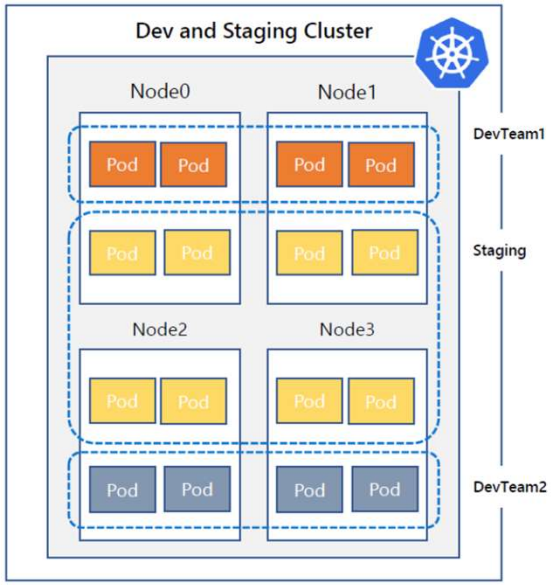
Logical Isolation

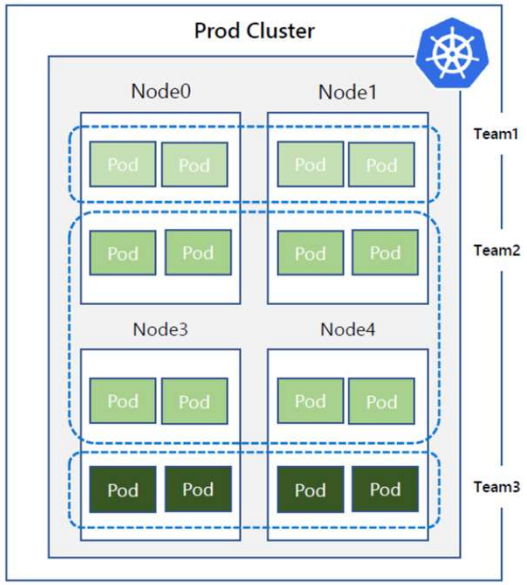
Logical Isolation is achieved by,

- Namespaces
- Network Policies
- Authentication and Authorization (Kubernetes RBAC)
- Container Level Isolation (Hard) for advanced scenario



Hybrid Isolation Approach





AKS Resource Quota

AKS allows to implement Resource Quotas to deal with Noisy Neighbor situation. Noisy Neighbor in this scenario refers to a POD consuming excessive compute resources such as CPU and Memory depriving other PODS. AKS Resource Quotas can also be configured to limit number of objects such as PODS, Services etc.)

- Constraints that limit aggregate resource consumption per namespace
- You can limit Compute Resources (CPU, Memory, Storage,…) and/or limit the number of Objects (Pods, Services, etc..) and
- When enabled, users must specify requests or limits, otherwise the quota system will fail the request.
- Kubernetes will not overcommit

Create a namespace:

```
$ kubectl create namespace ignite
```

Apply a resource quota to the namespace:

admin/resource/ignite.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
 name: mem-cpu-demo
spec:
hard:
   requests.cpu: "1"
   requests.memory: 1Gi
   limits.cpu: "2"
   limits.memory: 2Gi
```

Kube-advisor will return pods that are missing resource and request limits, please refer to,
https://github.com/Azure/kube-advisor

AKS Scaling

As the number of applications instances may change, the number of underlying Kubernetes nodes may also need to change.

AKS Scaling can be done via,

- Manually scale
- Horizontal pod autoscaler (HPA)
- Cluster autoscaler
- Azure Container Instance (ACI) integration with AKS

Manual Scaling

Manual Scaling is tedious and ineffective

For configuring AKS Manual Scaling, please refer to, https://docs.microsoft.com/en-us/azure/aks/scale-cluster

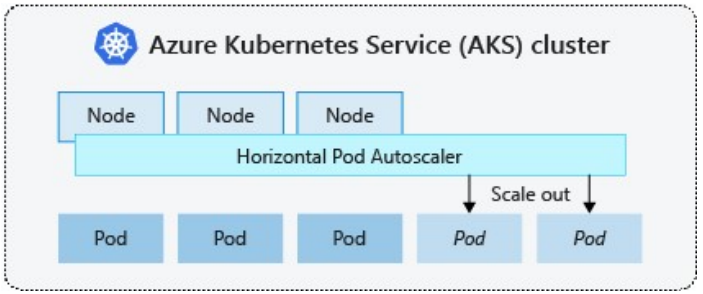
https://docs.microsoft.com/en-us/cli/azure/aks?view=azure-cli-latest#az-aks-scale

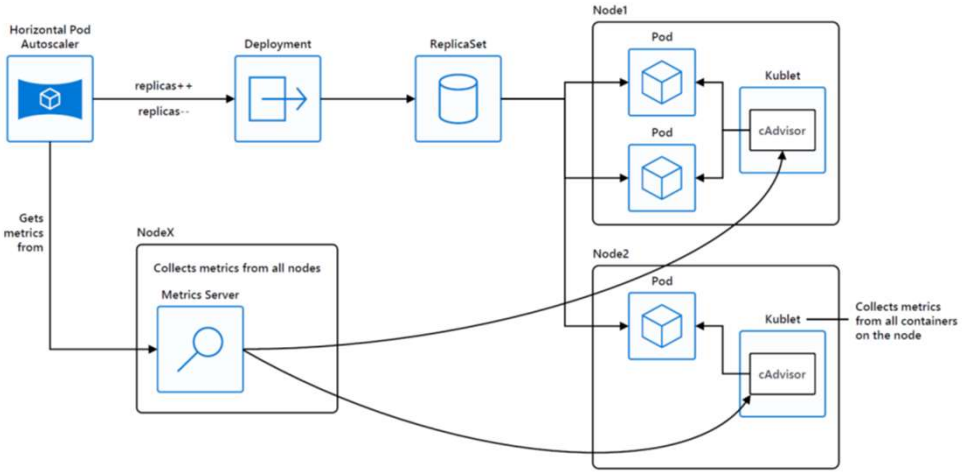
Manually scale Nodes & Pods,

https://docs.microsoft.com/bs-cyrl-ba/azure/aks/tutorial-kubernetes-scale

Horizontal Pod Autoscaler (HPA)

Kubernetes uses the horizontal pod autoscaler (HPA) to monitor the resource demand and automatically scale the number of replicas. By default, the horizontal pod autoscaler checks the Metrics API every 30 seconds for any required changes in replica count. When changes are required, the number of replicas is increased or decreased accordingly. Horizontal pod autoscaler works with AKS clusters that have deployed the Metrics Server for Kubernetes 1.8+.



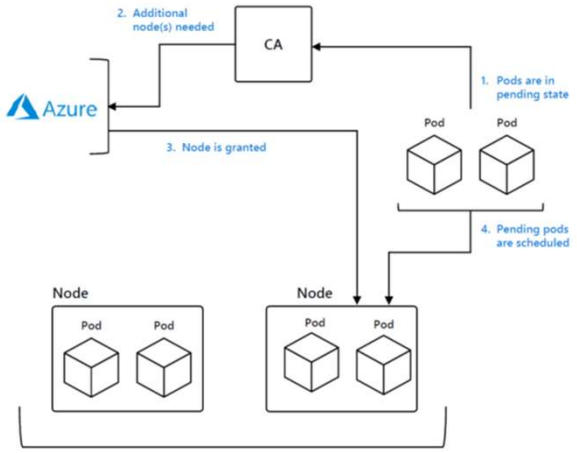


Please refer to https://docs.microsoft.com/bs-cyrl-ba/azure/aks/concepts-scale#horizontal-pod-autoscaler

Cluster Autoscaling (Preview)

To respond to changing pod demands, Kubernetes has a cluster autoscaler (currently in preview in AKS) that adjusts the number of nodes based on the requested compute resources in the node pool. By default, the cluster autoscaler checks the 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 your AKS cluster is increased or decreased accordingly. The cluster autoscaler works with RBAC-enabled AKS clusters that run Kubernetes 1.10.x or higher.

- Scales nodes based on pending pods
- Scale up and scale down
- Reduces dependency on monitoring
- Removes need for users to manage nodes and monitor service usage manually

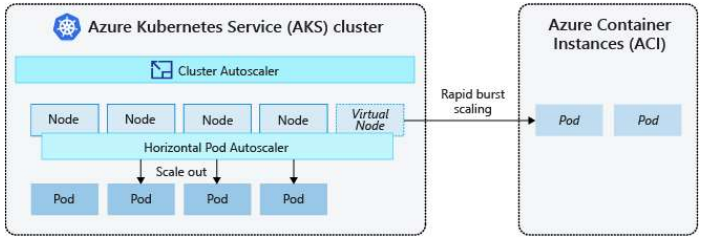


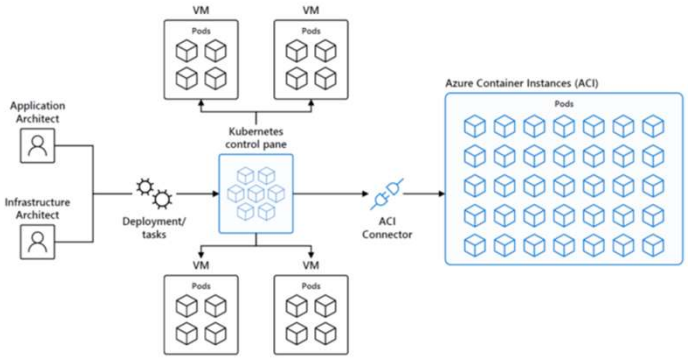
https://docs.microsoft.com/bs-cyrl-ba/azure/aks/concepts-scale#cluster-autoscaler

Burst to Azure Container Instances

To rapidly scale AKS cluster, Azure Container Instances (ACI) can be integrated with AKS. Kubernetes has built-in components to scale the replica and node count. However, if application needs to rapidly scale, the horizontal pod autoscaler may schedule more pods than can be provided by the existing compute resources in the node pool. If configured, this scenario would then trigger the cluster autoscaler to deploy additional nodes in the node pool, but it may take a few minutes for those nodes to successfully provision and allow the Kubernetes scheduler to run pods on them.

https://docs.microsoft.com/en-us/azure/aks/concepts-scale#burst-to-azure-container-instances

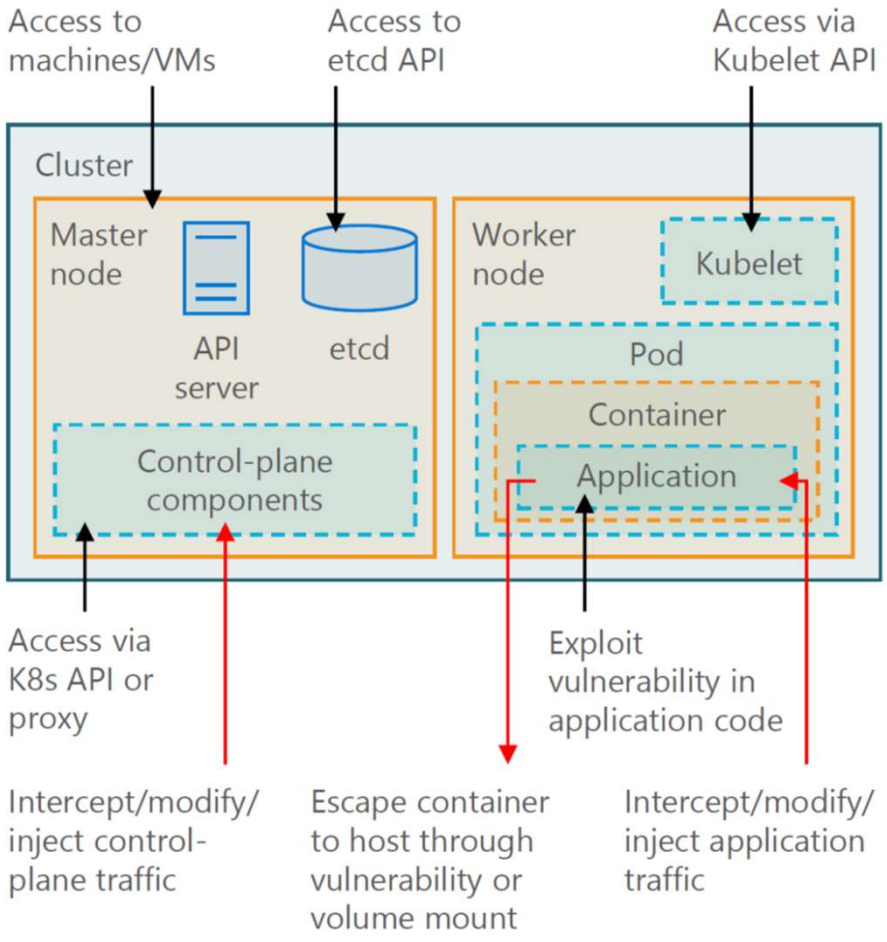




AKS Security

Cluster Level Security

The Master Node controls the configuration and operation of the entire cluster and is therefore a key area to secure.



API Server

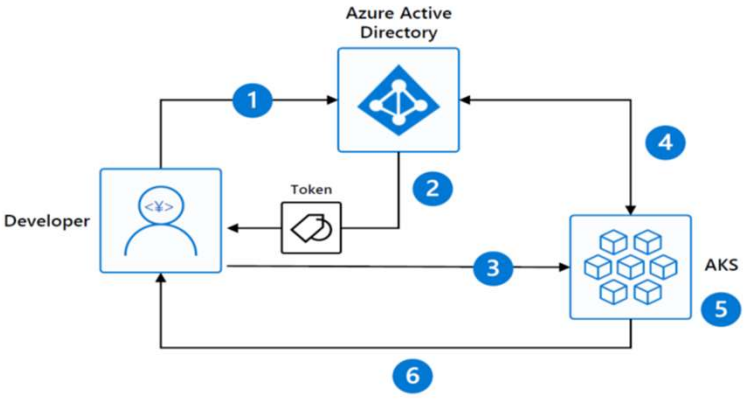
The API server offers REST API access to control the cluster. From v1.10 onwards, the kubeadminstaller disables the API server's insecure port so that API access is restricted to encrypted TLS connections made over a secured port by default **but is not limited by default to authenticated users**. To further limit access to the API server, Secure endpoints for API server and cluster nodes through,

- Ensuring authentication and authorization (AAD + RBAC)
- Setting up & keeping least privileged access for common tasks
- Admission Controllers

Authentication and Authorization

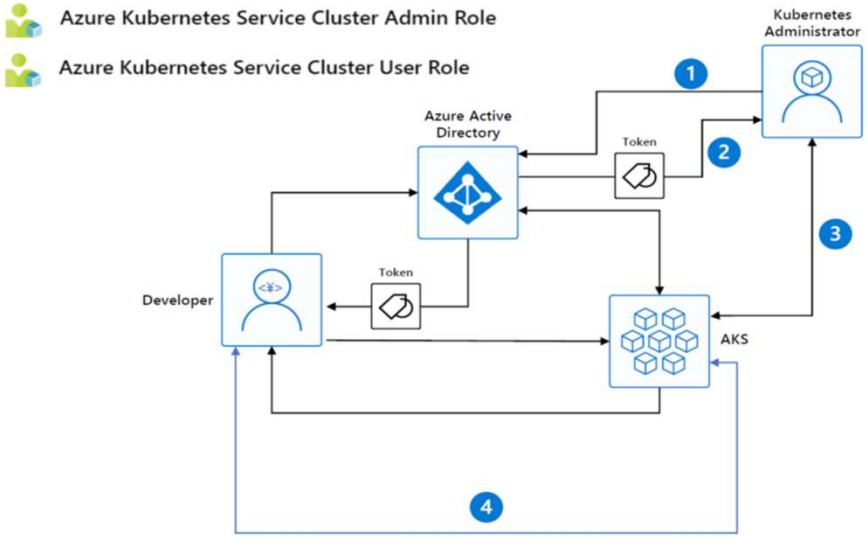
Cluster Level – Identity and Access Management through AAD and Kubernetes RBAC

- Kubernetes Developer authenticates with AAD
- The AAD token issuance endpoint issues the access token
- Developer performs action w/ AAD token.Eg.*kubectlcreate pod*
- Kubernetes validates token with AAD and fetches the Developer's AAD Groups for example: Dev Team A, App Group B
- Kubernetes RBAC and cluster policies are applied
- Request is successful or not based on the previous validation



Azure Level – Identity and Access Management through AAD and RBAC

- Kubernetes Administrator authenticates with AAD
- The AAD token issuance endpoint issues the access token
- Administrator fetches the admin kubeconfigand configures RBAC roles and bindings
- Kubernetes Developer fetches the user kubeconfig



AKS AD Integration and Kubernetes RBAC

Please refer for AKD AD Integration to,
https://docs.microsoft.com/en-us/azure/aks/azure-ad-integration

Kubernetes RBAC example,

Setting up a Cluster Role

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  labels:
   kubernetes.io/cluster-service: "true"
  name: cluster-admin
rules:
- apiGroups:
  - extensions
  - apps
  resources:
  - deployments
  verbs:
  - get
  - list
  - watch
  - update
  - patch
- apiGroups:
  - ""
  resources:
  - events
  - namespaces
  - nodes
  - pods
  verbs:
  - get
  - list
  - watch
```

Bind the Cluster Role to a user

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
name: contoso-cluster-admins
roleRef:
apiGroup: rbac.authorization.k8s.io
kind: ClusterRole
name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
kind: User
name: "user@contoso.com"
```

Bind the Cluster Role to a group

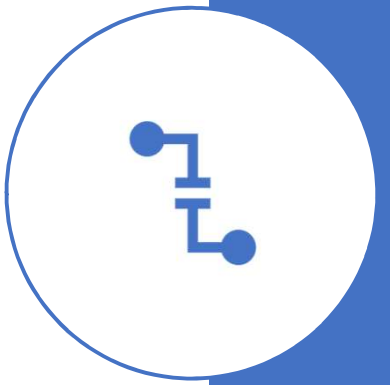
```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
name: contoso-cluster-admins
roleRef:
apiGroup: rbac.authorization.k8s.io
kind: ClusterRole
name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
kind: Group
name: "894656e1-39f8-4bfe-b16a-510f61af6f41"
```

Admission Controllers

An admission controller is a piece of code that intercepts requests to the Kubernetes API server prior to persistence of the object, but after the request is authenticated and authorized.

AKS supports the following admission controllers (as of now it is not modifiable)

- NamespaceLifecycle
- LimitRanger
- ServiceAccount
- DefaultStorageClass
- DefaultTolerationSeconds
- MutatingAdmissionWebhook
- ValidatingAdmissionWebhook
- ResourceQuota
- DenyEscalatingExec
- AlwaysPullImages



ETCD Security

Kubernetes stores cluster configuration and state information in a distributed key-value store named etcd. Unauthorized access to etcd may jeopardize the entire cluster, which is why access to it should be strictly limited.

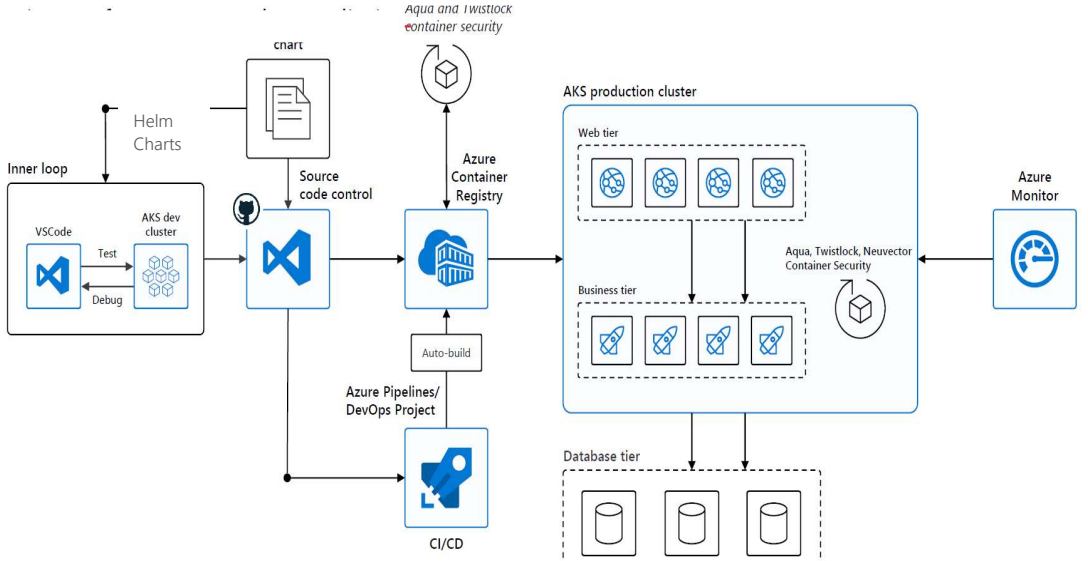
Container Level Security - Images

Container Level Security revolves around,

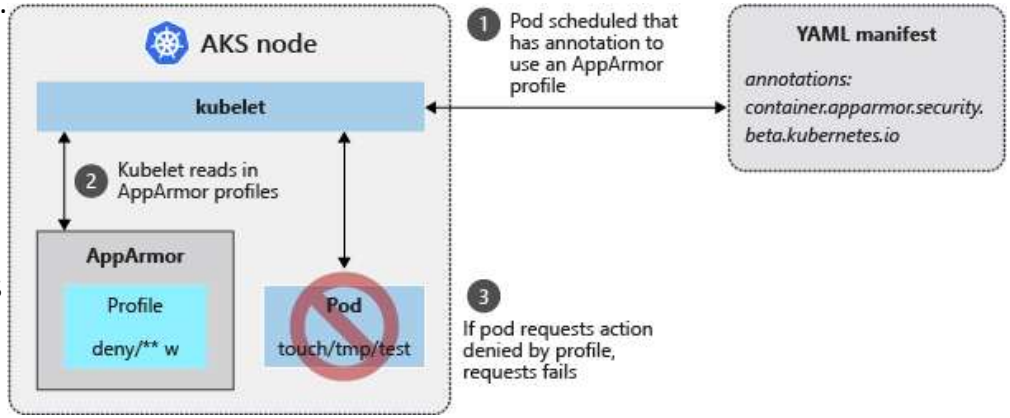
- Trusted Registry
- Regularly apply security updates to the container images

This shall be accomplished through

- Scan images and containers using Aqua or Twistlock tools
- Runtime enforcement and remediation using AppArmor Tool



To limit the actions that containers can perform, AKS admins can use the AppArmor Linux kernel security module. AppArmor is available as part of the underlying AKS node OS and is enabled by default. AKS admins create AppArmor profiles that restrict actions such as read, write, or execute, or system functions such as mounting filesystems. Default AppArmor profiles restrict access to various /proc and /sys locations and provide a means to logically isolate containers from the underlying node. AppArmor works for any application that runs on Linux, not just Kubernetes pods.



Pod Level Security

- Pod Security Context
- Pod Security Policies (Preview)
- AlwaysPullImages
- Pod Identity
- Pod Network Isolation using Network Policies

Pod Security Context Example

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    fsGroup: 2000
  volumes:
  - name: sec-ctx-vol
    emptyDir: {}
  containers:
  - name: sec-ctx-demo
    image: ignite.azurecr.io/nginx-demo
    volumeMounts:
    - name: sec-ctx-vol
      mountPath: /data/demo
    securityContext:
      runAsUser: 2000
      allowPrivilegeEscalation: false
      capabilities:
        add: ["NET_ADMIN", "SYS_TIME"]
      seLinuxOptions:
        level: "s0:c123,c456"
```

Pod Security Policies

To improve the security of AKS clusters, Pod scheduling can be limited and controlled. Pods that request resources you don't allow can't run in the AKS cluster. You define this access using pod security policies.

Please refer to, https://docs.microsoft.com/en-us/azure/aks/use-pod-security-policies

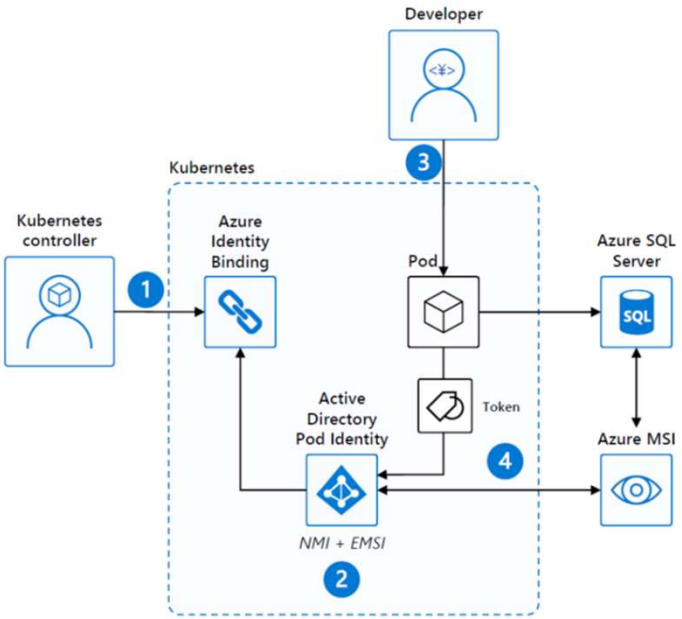
Always Pull Signed (Trusted) Images

AKS Cluster shall always be pulling only signed images from Azure Container Registry.

Please refer to https://docs.microsoft.com/en-us/azure/container-registry/container-registry-content-trust

Pod Identity

1. Kubernetes operator defines an identity map for K8s service accounts.
2. Node Managed Identity (NMI) watches for mapping reaction and syncs to Managed Service Identify (MSI)
3. Developer creates a pod with a service account. Pod uses standard Azure SDK to fetch a token bound to MSI
4. Pod uses access token to consume other Azure services; services validate token



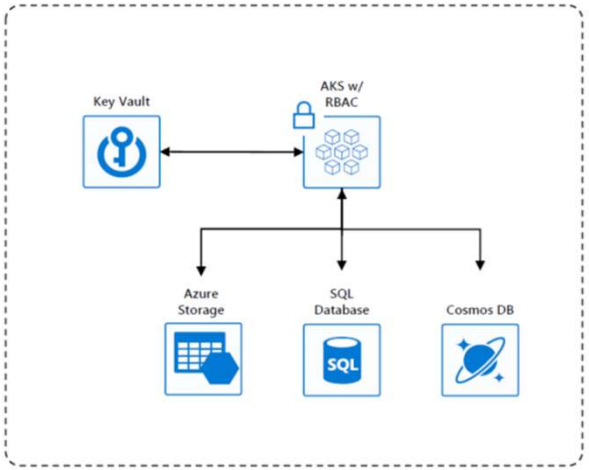
Please refer to,
https://blog.jcorioland.io/archives/2018/09/05/azure-aks-active-directory-managed-identities.html
https://github.com/Azure/aad-pod-identity

AKS Secrets Management

A Kubernetes *Secret* is used to inject sensitive data into pods, such as access credentials or keys. Secret are created using the Kubernetes API. Secrets can be requested during Pod Deployment. Secrets are only provided to nodes that have a scheduled pod that requires it, and the Secret is stored in *tmpfs*, not written to disk. When the last pod on a node that requires a Secret is deleted, the Secret is deleted from the node's tmpfs. Secrets are stored within a given namespace and can only be accessed by pods within the same namespace.

The use of Secrets reduces the sensitive information that is defined in the pod or service YAML manifest. Instead, you request the Secret stored in Kubernetes API Server as part of your YAML manifest. This approach only provides the specific pod access to the Secret. Please note: the raw secret manifest files contains the secret data in base64 format (see the official documentation for more details). Therefore, this file should be treated as sensitive information, and never committed to source control.

Secrets shall be stored and called from Azure Key Vault.



Azure Container Registry Security

Following security principles shall be followed,

- Image Scanning to be performed by Twistlock
- Azure RBAC to be used for Azure Container Registry, with least privileges approach
- Unsigned images can’t be pushed to Azure Container registry
- Authentication will leverage Azure AD, please refer to https://docs.microsoft.com/en-us/azure/container-registry/container-registry-authentication
- Follow best practices for Azure Container Registry, please refer to https://docs.microsoft.com/en-us/azure/container-registry/container-registry-best-practices

K8s is ultimately used to run software in the form of containers. Where those containers came from and what they contain has a direct impact on the security of K8s applications. This means implementing controls across the pipeline to ensure that what goes in is validated, and that code integrity is maintained throughout.

Implementing Trusted Software Supply Chain (DevOps)

Implementing Trusted Software Supply Chain

Source Control

- Ensure that container images are created using an approved set of base images, and do not allow developers to access unvetted open source components and use them in images. This can be achieved through firewall rules.

Image Scanning

- DevOps team will regularly scan images, both during CI/CD builds as well as in the registry, to detect known vulnerabilities as well as other issues such as embedded secrets and image configuration issues.
- Known vulnerabilities present a potentially high risk that's easy to detect and manage, and eliminating or mitigating them (especially high severity ones) should be step one of any security

Run as Root User

- Running a container as root is **not allowed as** running a container as root will make it easier for an attacker to use a compromised container to control the host. It is preferred to use different IDs for different images, as it will make auditing and forensics more accurate.

Image Integrity Controls

- Images are constantly being updated and pushed through the pipeline. It's important to ensure that the containers that end up running in cluster are instantiated from the correct images, with no tampering or drift, especially in applications that are mission-critical or that handle sensitive data.
- Aqua or Twistlock will be used to ensure Image deployment lifecycle through CI/CD pipelines.

Enforce use of Trusted Images

- Only approved and signed images will be deployed from CI\CD pipeline.

Azure Networking

Capability	Kubenet	Azure CNI
Deploy cluster in existing or new virtual network	Supported - UDRs manually applied	Supported
Pod-pod connectivity	Supported	Supported
Pod-VM connectivity; VM in the same virtual network	Works when initiated by pod	Works both ways
Pod-VM connectivity; VM in peered virtual network	Works when initiated by pod	Works both ways
On-premises access using VPN or Express Route	Works when initiated by pod	Works both ways
Access to resources secured by service endpoints	Supported	Supported
Expose Kubernetes services using a load balancer service, App Gateway, or ingress controller	Supported	Supported
Default Azure DNS and Private Zones	Supported	Supported
Network Policies	N/A	Supported

AKS clusters can be deployed with one of the following two network models:

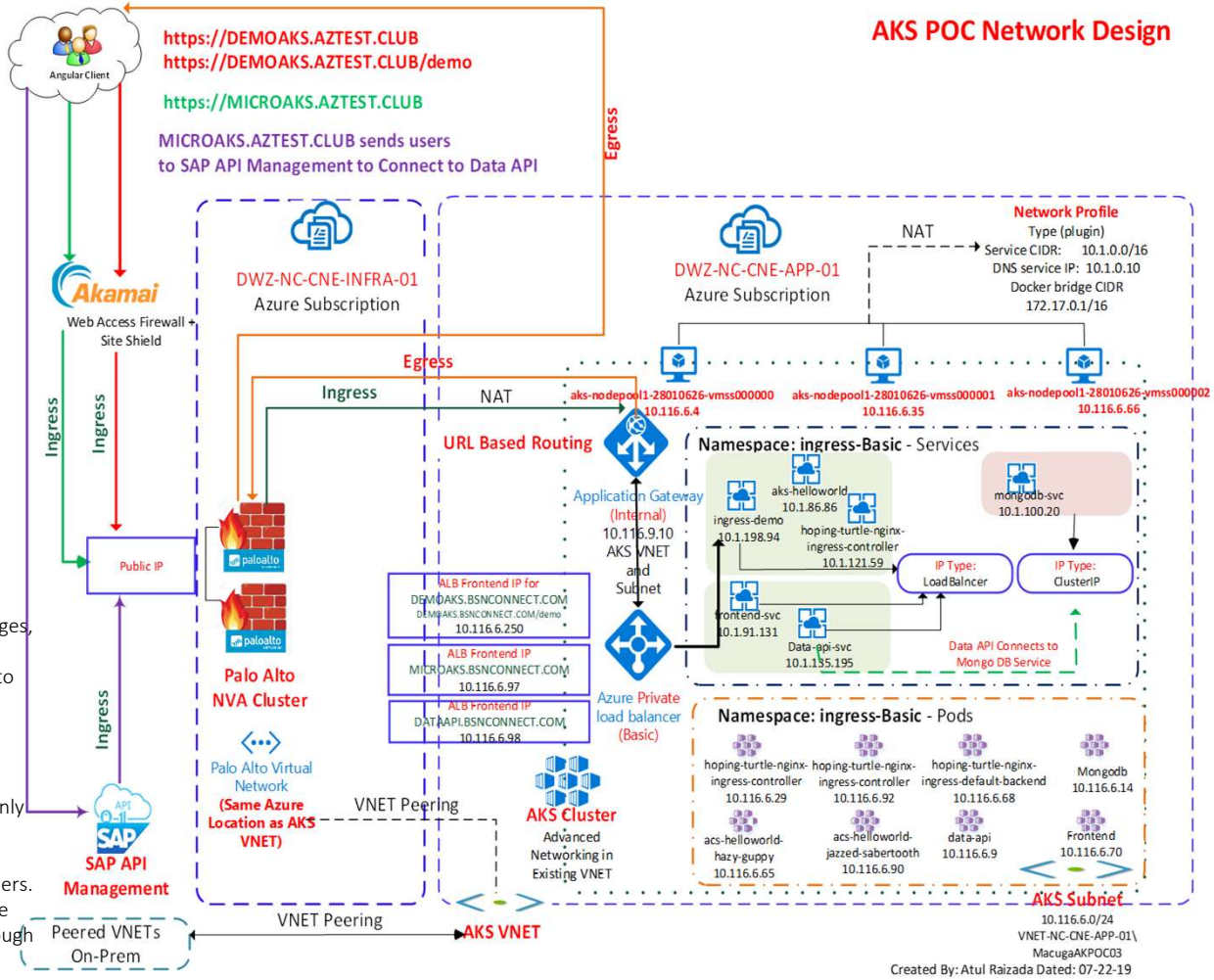
- *Kubenet* networking (Basic) - The network resources are typically created and configured as the AKS cluster is deployed.
- *Azure Container Networking Interface (CNI)* networking (Advanced) -The AKS cluster is connected to existing virtual network resources and configurations

Network Design to Secure Applications and APIs

- Define Requirements for example:
  - Advanced/Basic Networking
  - Do not expose any Azure Public IPs and force all Internet Ingress and Egress through Azure Firewall or NVAs such as Palo Alto
  - Azure Application Gateway Web Access Firewall or third party WAF such as Akamai for all External Web Apps hosted at AKS
  - Azure API Management Gateway or other Third-Party API Management Gateway such as SAP API Management Gateway for Externally accessible APIs hosted at AKS
  - NGINX Ingress Controllers
  - High Availability
  - Azure Firewall or Palo Alto NVA for North South Traffic Management
  - AKS Network Policies for Intra AKS Cluster east west traffic
  - NSG to control AKS VNET traffic
  - Ideally NVAs and AKS in different subscriptions, resource groups and VNETs as per Azure Architecture
  - Do not expose any Azure Public IPs
  - Internal applications shall only be accessible internally without exposing them to public

- Azure Limitations:
  - AKS by default creates Azure Load Balancer for Ingress Controllers or any Public Facing AKS Services with Basic SKU
  - Azure has a limitation that Basic SKU Private ALB Frontend IP is not reachable by other peered VNETs
  - As a result Azure Firewall or Palo Alto NVAs running in other VNETs and Subscription can't reach to Basic SKU Private ALB Frontend IP
  - Azure AKS Preview mode allows to create ALB with Standard SKU, but by default create Public ALB defeating the purpose of using private IP Addresses

Applications and API Secure Design Example



Salient Features of the solution:

- Using NGINX Controller through **Private** Azure Load Balancer (meets requirements of not exposing public IPs), please note that Ingress Controller by default create a Public ALB.
- Use Azure Application Gateway (Internal) for URL Based routing to meet following challenges,
  - Meets the challenges of Basic Private ALB Front End IP
  - It also minimizes need of multiple Public IPs on Palo Alto NVAs and multiple one to one NAT requirements.
  - Azure Application Gateway being a layer 7 device also offers lot of other customization and security enhancements
- NSG deployed at AKS Subnet will allow further security enhancements
- Design uses Akamai as web Access Firewall and NSGs will be configured to accept traffic only from Akamai Site Shield IPs and SAP API Management Gateways
- Design also incorporates SAP API Management Gateway for APIs hosted on AKS Clusters
- Web Routing can be accomplished at both Application Gateway as well as Ingress Controllers.
- It is to be noted that AKS POD will use IPs from VNET and are routable, but liable to change during recreation, therefore as per AKS best practice generally Web App are exposed through AKS Service (IP does not change). Service IPs Type can be created as ClusterIP or LoadBalancer. LoadBalancer IP will add additional Private IP from VNET to ALB, and thus can be configured through Application Gateway URL Based routing.
- On the contrary Services created with ClusterIP type will use Service CIDR and are meant for Intra AKS networking not reachable from outside AKS, this is where Ingress Controller Ingress Routes can be used for Web Routing.

AKS Cluster Update

AKS Cluster Green Blue Strategy will define cluster upgrade process.

- If Green Blue Strategy is used than Standby Cluster shall be upgraded and once successful, Prod Cluster shall be destroyed and recreated
- If Green Blue Strategy is not used, then deploy a new cluster using DevOps CI\CD Pipeline, upgrade the newly deployed cluster and if successful, make it Prod Cluster and destroy existing Prod Cluster.

AKS Auditing and Monitoring

AKS Monitoring through Azure Monitor workspace and develop a notification and alert process based on performance counters and other desired parameters.

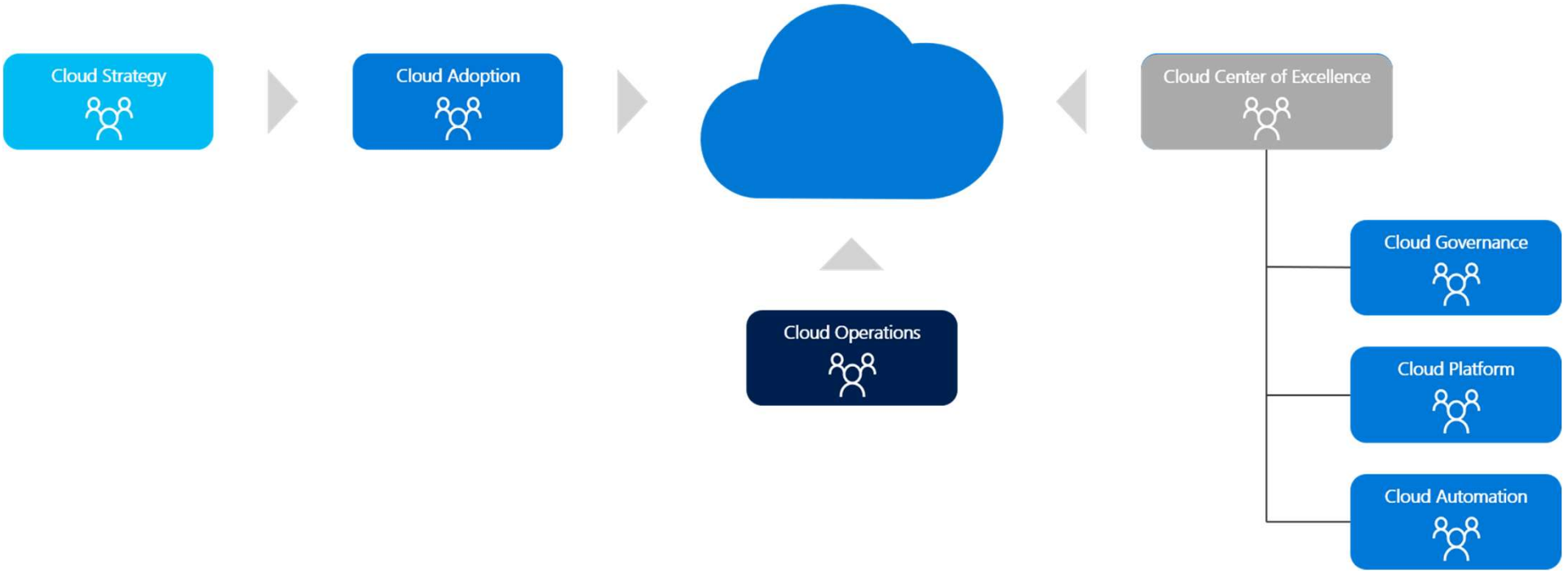
- **Key Metrics:**
- Node metrics (CPU Usage, Memory Usage, Disk Usage, Network Usage)
- Kube_node_status_condition
- Pod memory usage / limit; memory_failures_total
  - container_memory_working_set_bytes
- Pod CPU usage average / limit
- Filesystem Usage / limit
- Network receive / transmit errors
-
- **Azure Monitor Metrics available today**
- Kube-controller-manager
- Kube-api-server
- Kube-scheduler
- Audit logs on the roadmap

AZURE RACI

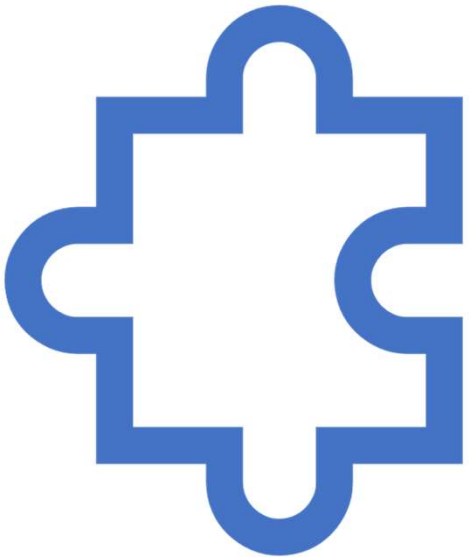
RACI Chart for the Fully-staffed Best Practice

Additional information available online

	Solution delivery	Business alignment	Change management	Solution operations	Governance	Platform maturity	Platform operations	Platform automation
Cloud strategy team	Consulted	Accountable	Accountable	Consulted	Consulted	Informed	Informed	Informed
Cloud adoption team	Accountable	Consulted	Responsible	Consulted	Informed	Informed	Informed	Informed
Cloud operations team	Consulted	Consulted	Responsible	Accountable	Consulted	Informed	Accountable	Consulted
Cloud governance team	Consulted	Informed	Informed	Consulted	Accountable	Consulted	Responsible	Informed
Cloud platform team	Consulted	Informed	Informed	Consulted	Consulted	Accountable	Responsible	Responsible
Cloud automation team	Consulted	Informed	Informed	Informed	Consulted	Responsible	Responsible	Accountable
Aligned cloud capability	cloud adoption	cloud strategy	cloud strategy	cloud operations	CCoE-cloud governance	CCoE-cloud platform	CCoE-cloud platform	CCoE-cloud automation


Cloud Strategy
Cloud Adoption
Cloud Operations
Cloud Center of Excellence
Cloud Governance
Cloud Platform
Cloud Automation

Azure RACI



AKS RACI

Workstream	Activity	Cloud strategy team	Cloud adoption team	Cloud operations team	Cloud governance team	Cloud platform team	Cloud automation team	Cloud Network Team	Cloud DevOps	Cloud Security Team
AKS Strategy	Overall AKS Strategy, that includes, • Identify existing applications as a candidate for AKS • Applications Refactoring to take advantage of AKS • New Applications (Microservices) Strategy • DevSecOps and Modern Software Engineering • AKS Cluster Architecture • Phased Implementation Plan • Success Criteria	R, A	C	C	C	C	C	C	C	C
AKS Cluster Creation	Create AKS Cluster as per AKS Cluster Architecture (will include all key decision points enumerated in this presentation)	I	R,A	I	I	I	I	I	I	I
AKS Cluster Operations	• Manage Nodes (add, remove, scaling, upgrades) • Managing Quotas • Cluster Upgrade • Namespaces • DR and HA • Storage • Azure Container Registry • Pods Management (Periodic Recycle)	I	C	R, A	I	I	I	I	I	I
AKS Security	Manage AKS Security as defined in this presentation	C	C	C	I	I	I	I	I	R, A
AKS Networking	NVAs/Firewalls, Application Gateways, Azure Load Balancers, Ingress Controllers, AKS Network Policies	C	C	C	I	I	I	R, A	I	C
AKS Application Deployment	Images, Codes, CI/CD, Pipelines, Image Vulnerability, Image Repos, Image Versioning and Stale Images	I	C	I	I	I			R,A	
AKS Automation	ARM, Terraform or CLI for AKS Clusters Creation	I	C	I	I	I	R	I	A	I
Data Integrity and Controls		I	I	I	R,A	I	I	I	C	C