Microsoft

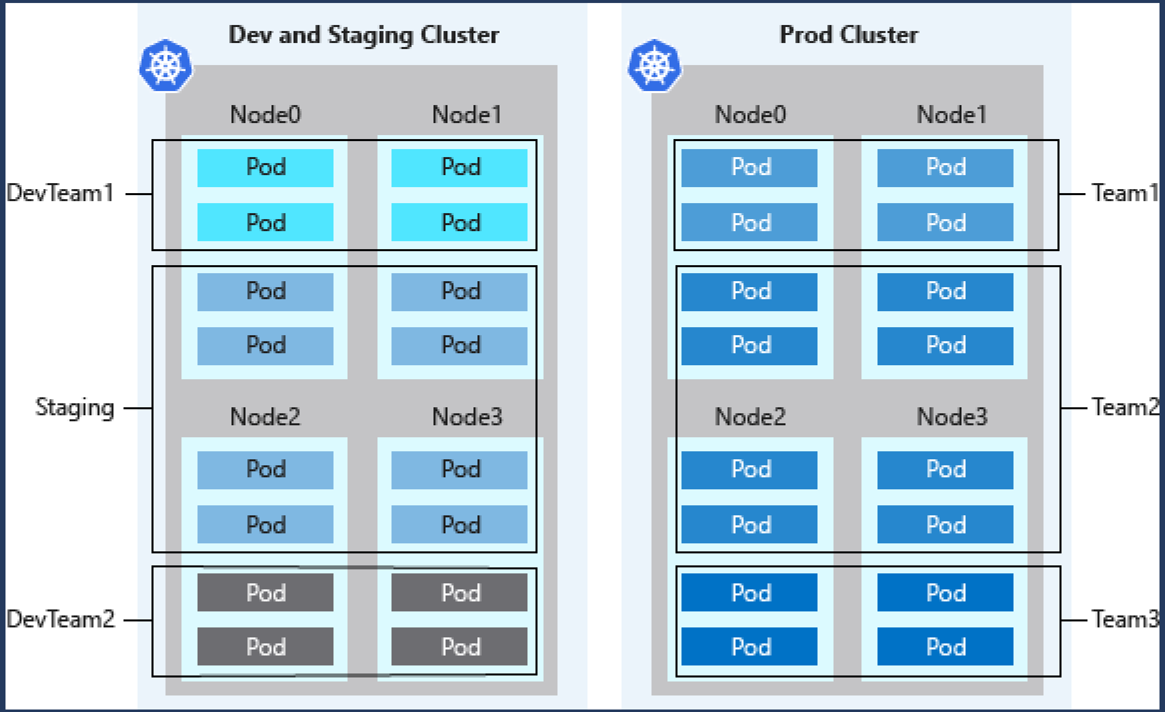
AKS – Best Practices

1

- Multi-Tenancy
  - Cluster isolation
  - Schedular Features
  - Authentication & Authorization
- Security
  - Cluster upgrades
  - Container image management
  - Pod security
- Network and Storage
  - Network Connectivity
  - Storage and Backups
- BCP DR

Cluster Isolation

- Logical Isolation
  - A single AKS cluster can be used for multiple workloads, teams, or environments
  - higher pod density than
  - **Namespace** creates a logical isolation boundary



Not completely safe for hostile multi tenant usage

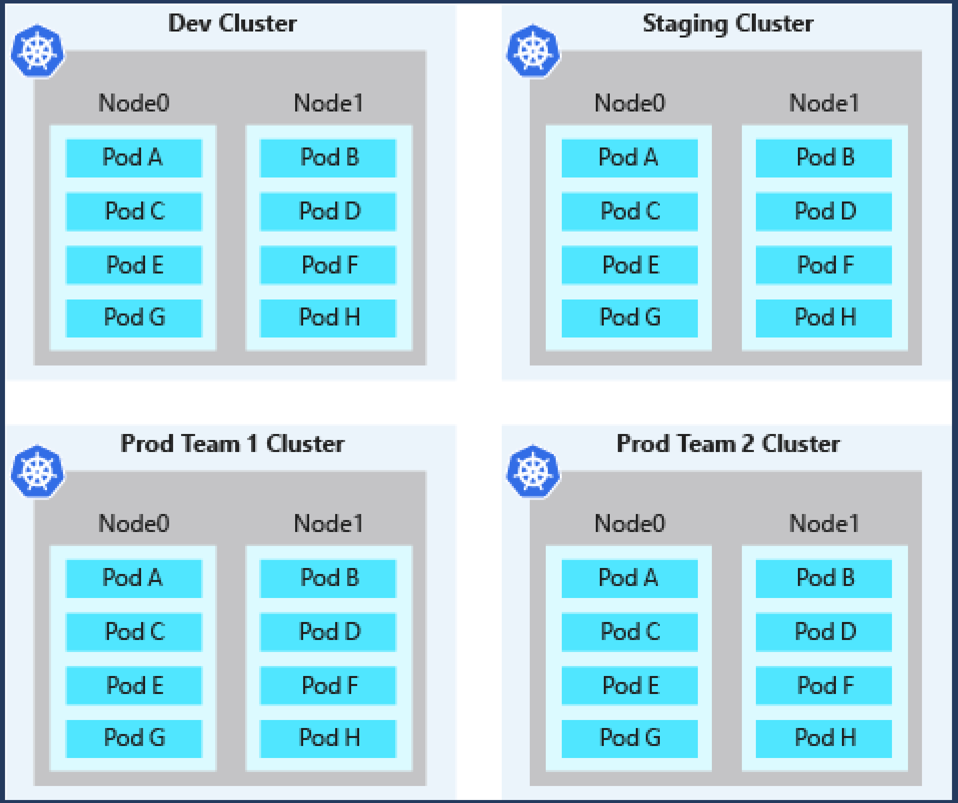
Cluster Isolation

Logical Isolation

Cluster Isolation

Physical Isolation

- Teams or workloads are assigned their own AKS cluster
- Adds management and financial overhead



Dev Cluster

Node0	Node1
Pod A	Pod B
Pod C	Pod D
Pod E	Pod F
Pod G	Pod H

Staging Cluster

Node0	Node1
Pod A	Pod B
Pod C	Pod D
Pod E	Pod F
Pod G	Pod H

Prod Team 1 Cluster

Node0	Node1
Pod A	Pod B
Pod C	Pod D
Pod E	Pod F
Pod G	Pod H

Prod Team 2 Cluster

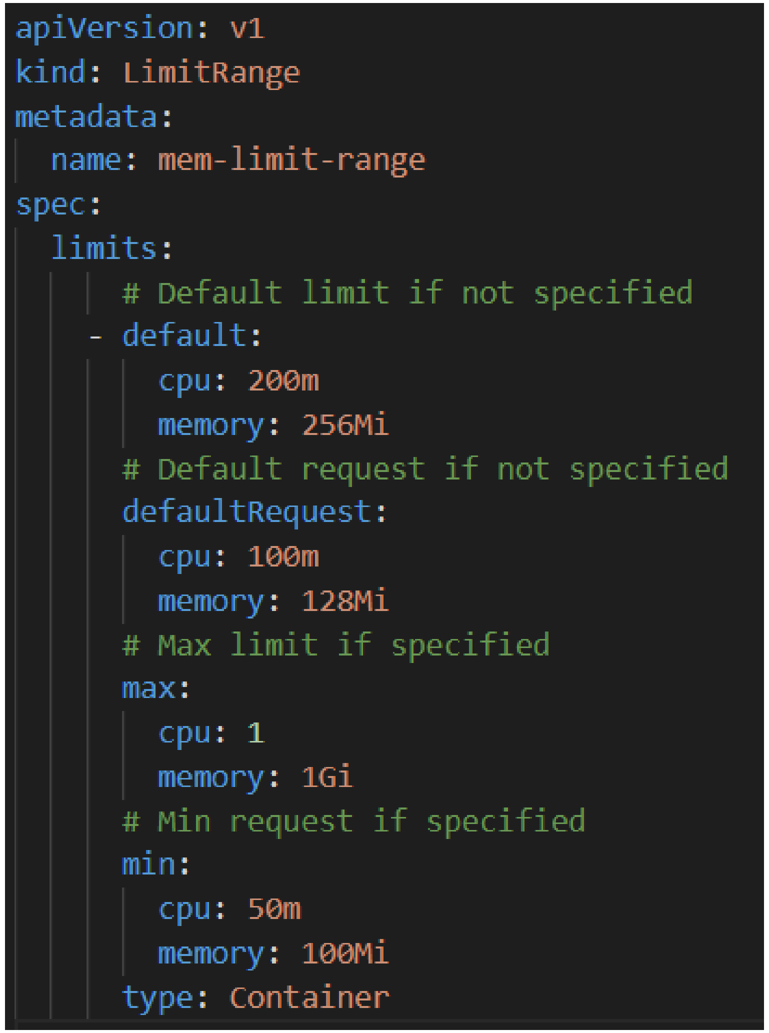
Node0	Node1
Pod A	Pod B
Pod C	Pod D
Pod E	Pod F
Pod G	Pod H

- Minimize the use of physical isolation for each separate team or application deployment

Scheduler Features

Default Requests and Limits

- In a large system, it may not be practical to define resource requests/limits for all the existing containers.
- The **LimitRange** object allows you to specify a default set of requests (***defaultRequests***) and limits (***defaults***) for any Container created in a *Namespace*.
- When requests/limits are not specified by the Container, the defaults in the **LimitRange** will be used.
- When requests/limits are specified, those values override the defaults in the **LimitRange**.
- The ***min*** (requests) and ***max*** (limits) can also be specified to ensure values specified for a Container stay within set bounds.



```
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-limit-range
spec:
  limits:
      # Default limit if not specified
    - default:
        cpu: 200m
        memory: 256Mi
      # Default request if not specified
      defaultRequest:
        cpu: 100m
        memory: 128Mi
      # Max limit if specified
      max:
        cpu: 1
        memory: 1Gi
      # Min request if specified
      min:
        cpu: 50m
        memory: 100Mi
      type: Container
```

Enforce resource quotas

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: dev-app-team
spec:
  hard:
    cpu: "10"
    memory: 20Gi
    pods: "10"
```

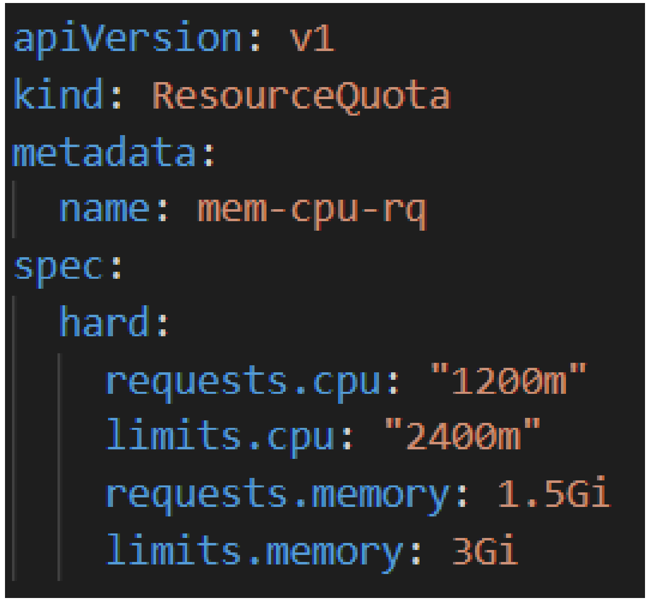
```
kubectl apply -f dev-app-team-quotas.yaml --namespace dev-apps
```

Plan and apply resource quotas at the namespace level. If pods don't define resource requests and limits, reject the deployment. Monitor resource usage and adjust quotas as needed.

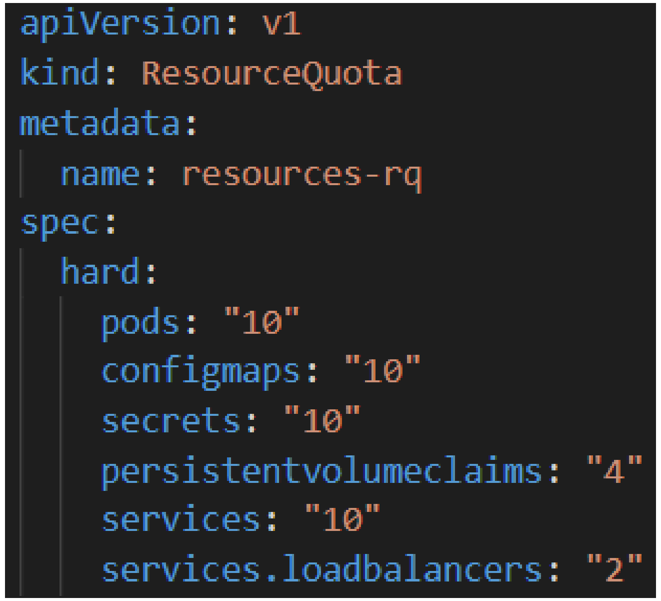
- **Compute resources**, such as CPU and memory, or GPUs.
- **Storage resources**, including the total number of volumes or amount of disk space for a given storage class.
- **Object count**, such as maximum number of secrets, services, or jobs can be created.

Namespace Resource Quota

- A **ResourceQuota** object, defines constraints that limit aggregate resource (total CPU and memory) consumption per namespace.
- It's **highly recommended** to create a **LimitRange** resource in the same namespace as the **ResourceQuota**.
- Limits should be large enough to accommodate upgrades.



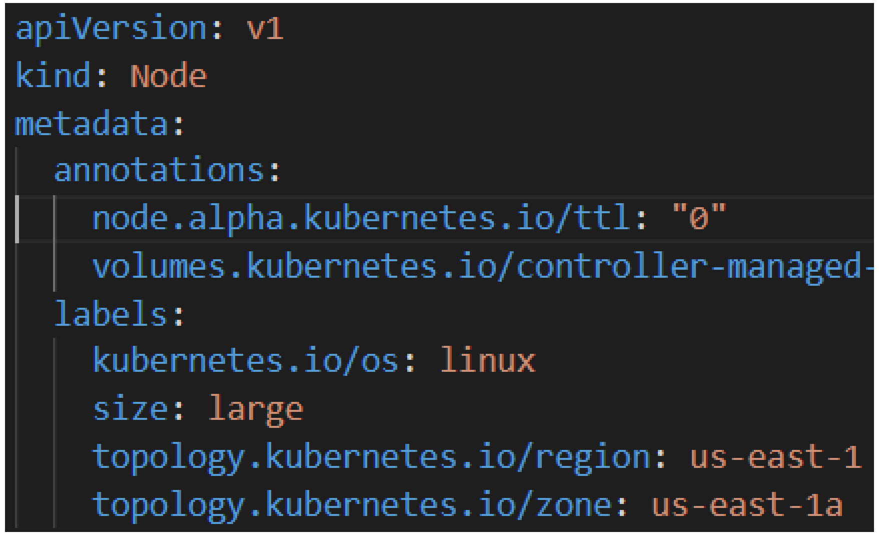
```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-rq
spec:
  hard:
    requests.cpu: "1200m"
    limits.cpu: "2400m"
    requests.memory: 1.5Gi
    limits.memory: 3Gi
```



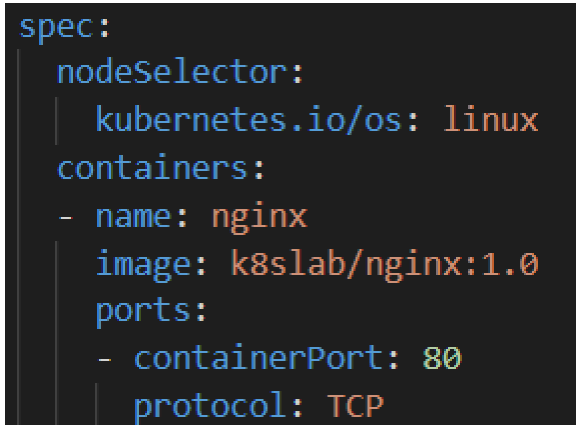
```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: resources-rq
spec:
  hard:
    pods: "10"
    configmaps: "10"
    secrets: "10"
    persistentvolumeclaims: "4"
    services: "10"
    services.loadbalancers: "2"
```

Node Selectors

- Pods can use Node labels to specify which Nodes to be scheduled on.
- Labels *must be* on the Nodes prior to deploying Pods selecting the Nodes.
- If selected Node label is not found on any node, Pods *will not get scheduled*.



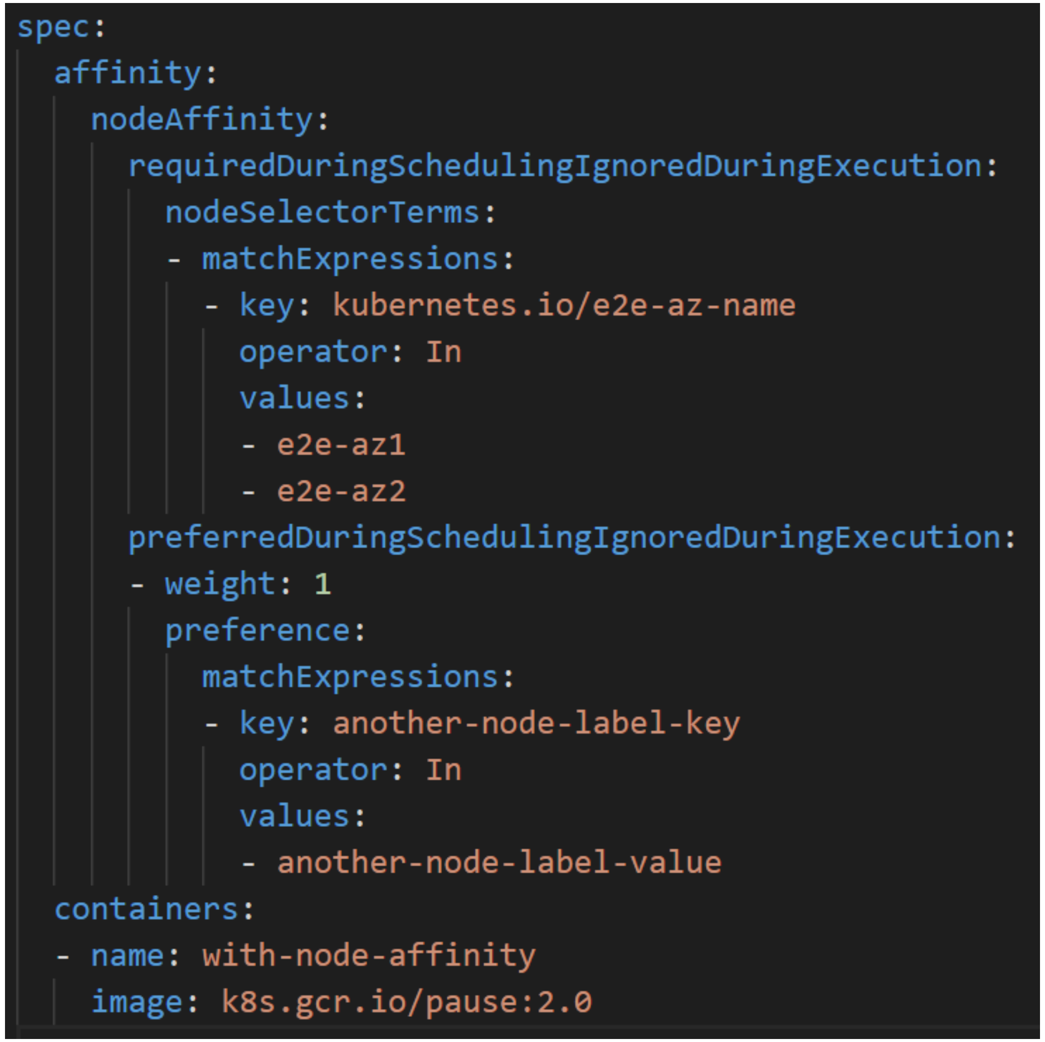
```yaml
apiVersion: v1
kind: Node
metadata:
  annotations:
    node.alpha.kubernetes.io/ttl: "0"
    volumes.kubernetes.io/controller-managed-
  labels:
    kubernetes.io/os: linux
    size: large
    topology.kubernetes.io/region: us-east-1
    topology.kubernetes.io/zone: us-east-1a
```



```yaml
spec:
  nodeSelector:
    kubernetes.io/os: linux
  containers:
  - name: nginx
    image: k8slab/nginx:1.0
    ports:
    - containerPort: 80
      protocol: TCP
```

Node Affinity

- The affinity/anti-affinity language is more expressive. The language offers more matching rules besides exact matches created with a logical AND operation
- Can indicate that the rule is soft "preference" rather than a hard requirement, so if the scheduler can't satisfy it, the Pod will still be scheduled.
- Use ***weight*** to set preference order.

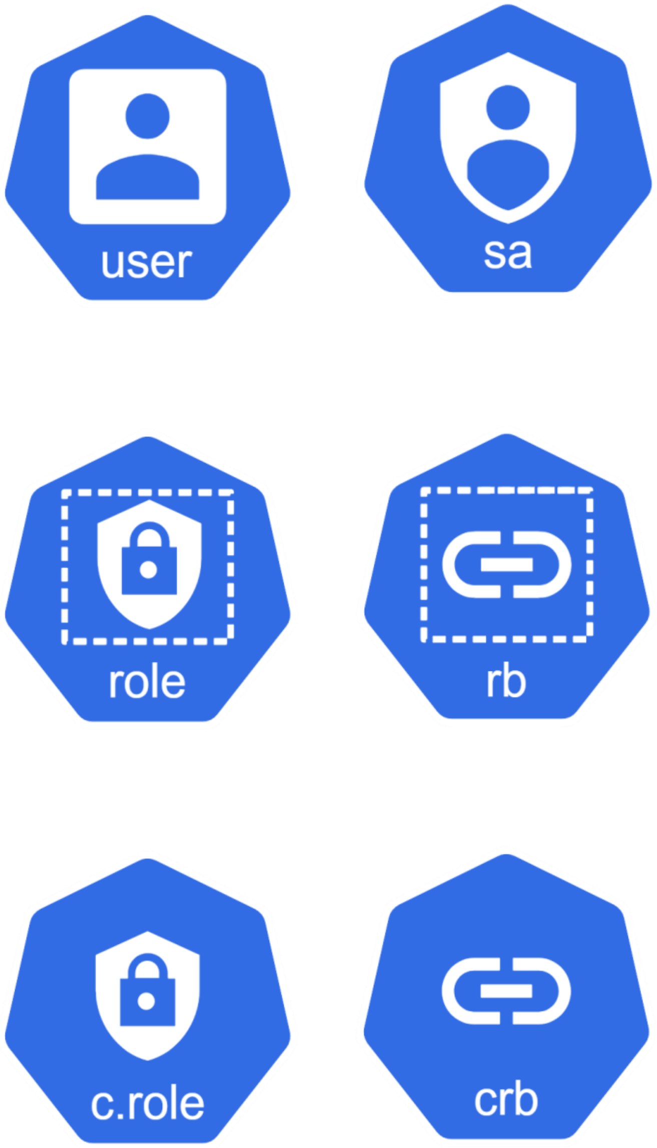


```yaml
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: kubernetes.io/e2e-az-name
            operator: In
            values:
            - e2e-az1
            - e2e-az2
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: another-node-label-key
            operator: In
            values:
            - another-node-label-value
  containers:
  - name: with-node-affinity
    image: k8s.gcr.io/pause:2.0
```

Taints and Tolerations

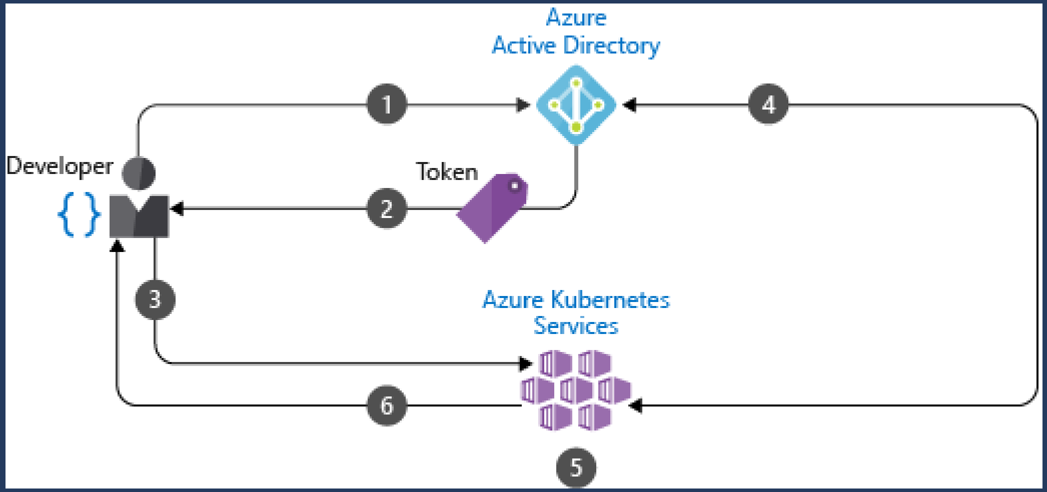
- Since a cluster can be a collection of heterogenous nodes, sometimes it makes sense to set a policy restricting which Pods are scheduled on which Nodes.
- Nodes can be marked as "*tainted*" with particular attributes
- Pods can be designated as being able to "*tolerate*" certain taints
- Depending on the taint-effect, the scheduler will decide if a Pod can be scheduled on a tainted Node.
- Node taint-effects can be:
  - **NoSchedule** – New Pods will NOT be scheduled on the tainted node unless they can tolerate the taint
  - **PreferNoSchedule** – Pods CAN be scheduled ONLY IF they won't fit on any other node.
  - **NoExecute** – New Pods will NOT be scheduled on the tainted node *and* existing Pods without toleration will be EVICTED from a tainted node.

Authentication and Authorization



Authentication and Authorization

Use Azure Active Directory (Azure AD)



Use Kubernetes role-based access control (Kubernetes RBAC)

Use Azure RBAC

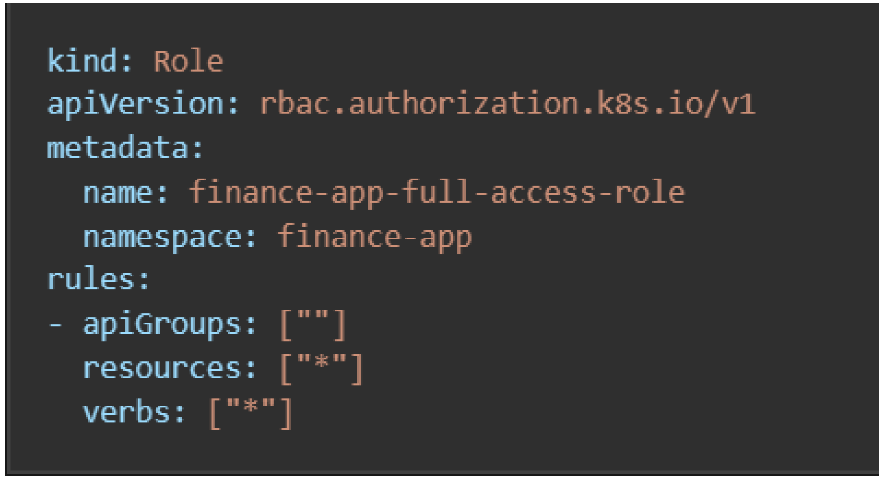
Kubernetes Role-based Access Control (RBAC)

RBAC is a method of regulating access to resources based on the roles of individual users or service accounts within your organization.

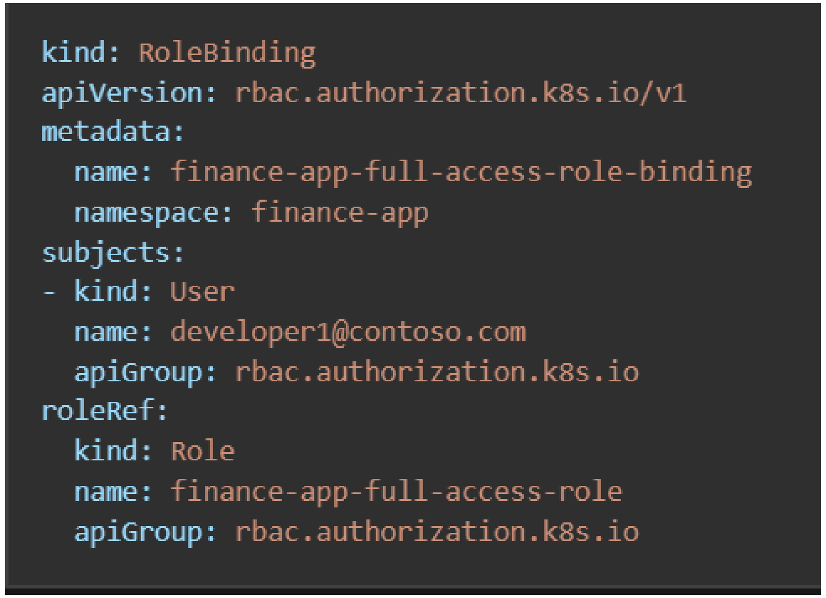
RBAC Components:

- **Service Account** - Provides an identity for processes that run in a Pod.
- **Role** – A list of rights (permissions) to specific resource types within a namespace.
- **RoleBinding** – Defines the *binding* of a user/service account to a Role within a namespace
- **ClusterRole** – A cluster-wide resource listing permissions for specific namespaces, all namespaces or cluster-scoped resources.
- **ClusterRoleBinding** – Defines the binding of user/service account to a **ClusterRole** throughout the cluster

Kubernetes Role-based Access Control (RBAC)



```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: finance-app-full-access-role
  namespace: finance-app
rules:
- apiGroups: [""]
  resources: ["*"]
  verbs: ["*"]
```



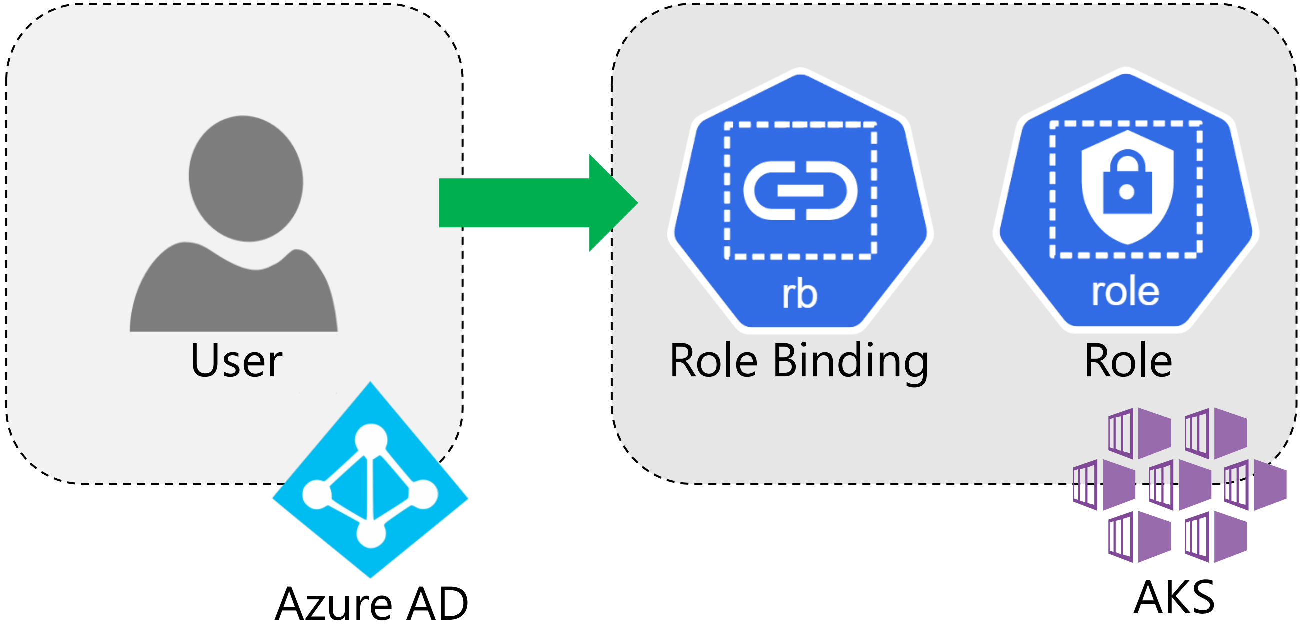
```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: finance-app-full-access-role-binding
  namespace: finance-app
subjects:
- kind: User
  name: developer1@contoso.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: finance-app-full-access-role
  apiGroup: rbac.authorization.k8s.io
```

Two levels of access needed to fully operate an AKS cluster:

- Access the AKS resource on your Azure subscription.
  - Control scaling or upgrading your cluster using the AKS APIs
  - Pull your kubeconfig.
- Access to the Kubernetes API.
  - Kubernetes RBAC (traditionally) or
  - By integrating Azure RBAC with AKS for kubernetes authorization

Kubernetes RBAC with Azure AD Integration

- Kubernetes RBAC is enabled by default in AKS. Disabling RBAC is **not recommended**.
- To enable Azure AD integration, use the `--enable-aad` option when creating or updating an AKS cluster.
- Kubernetes RBAC works with Azure AD as follows:
  1. A user is authenticated in Azure
  2. The user's token and email is passed to the cluster.
  3. A **RoleBinding** resource binds the user to a **Role**, which define which actions are allowed on which Kubernetes resources.



Cluster Security & Upgrades

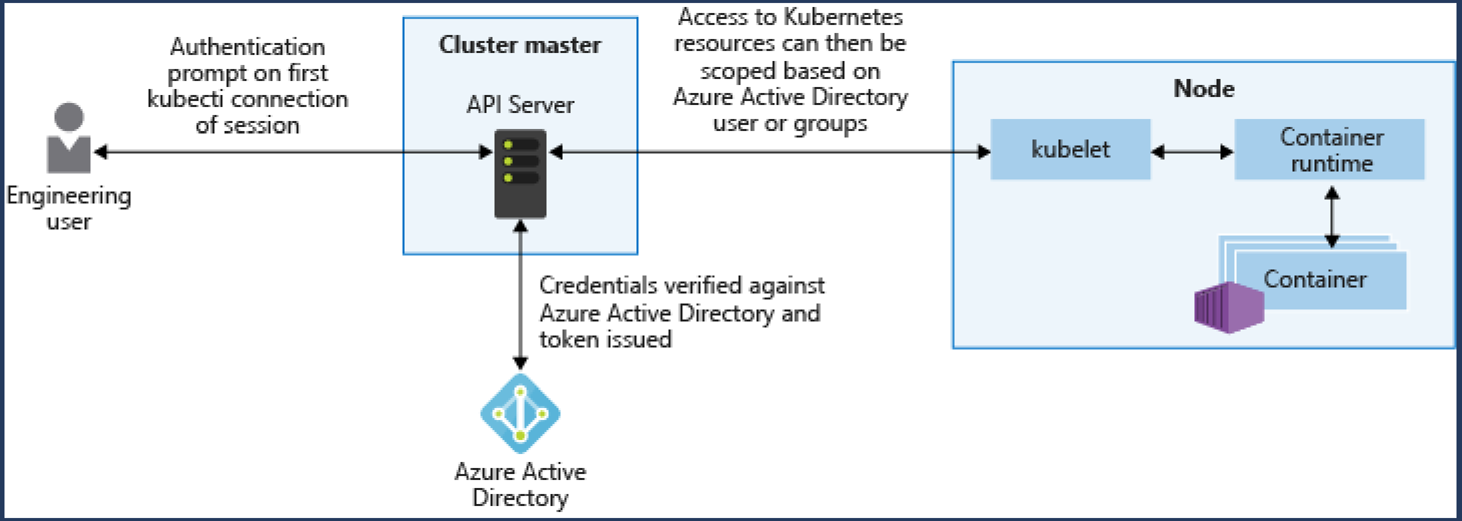
Threat protection

Defender for Containers

Assess cluster configurations and provide security recommendations, run vulnerability scans, and provide real-time protection and alerting for Kubernetes nodes and clusters.

Secure access to cluster and nodes

Integrate with K8s RBAC with Azure AD


Authentication prompt on first kubecti connection of session
Engineering user
Cluster master
API Server
Access to Kubernetes resources can then be scoped based on Azure Active Directory user or groups
Node
kubelet
Container runtime
Container
Credentials verified against Azure Active Directory and token issued
Azure Active Directory

Secure Container Access

Permissions to containers

Use minimum privileges , avoid root access or privileged escalations

App Armor

Seccomp

Upgrade Kubernetes versions

Regularly upgrade the Kubernetes version

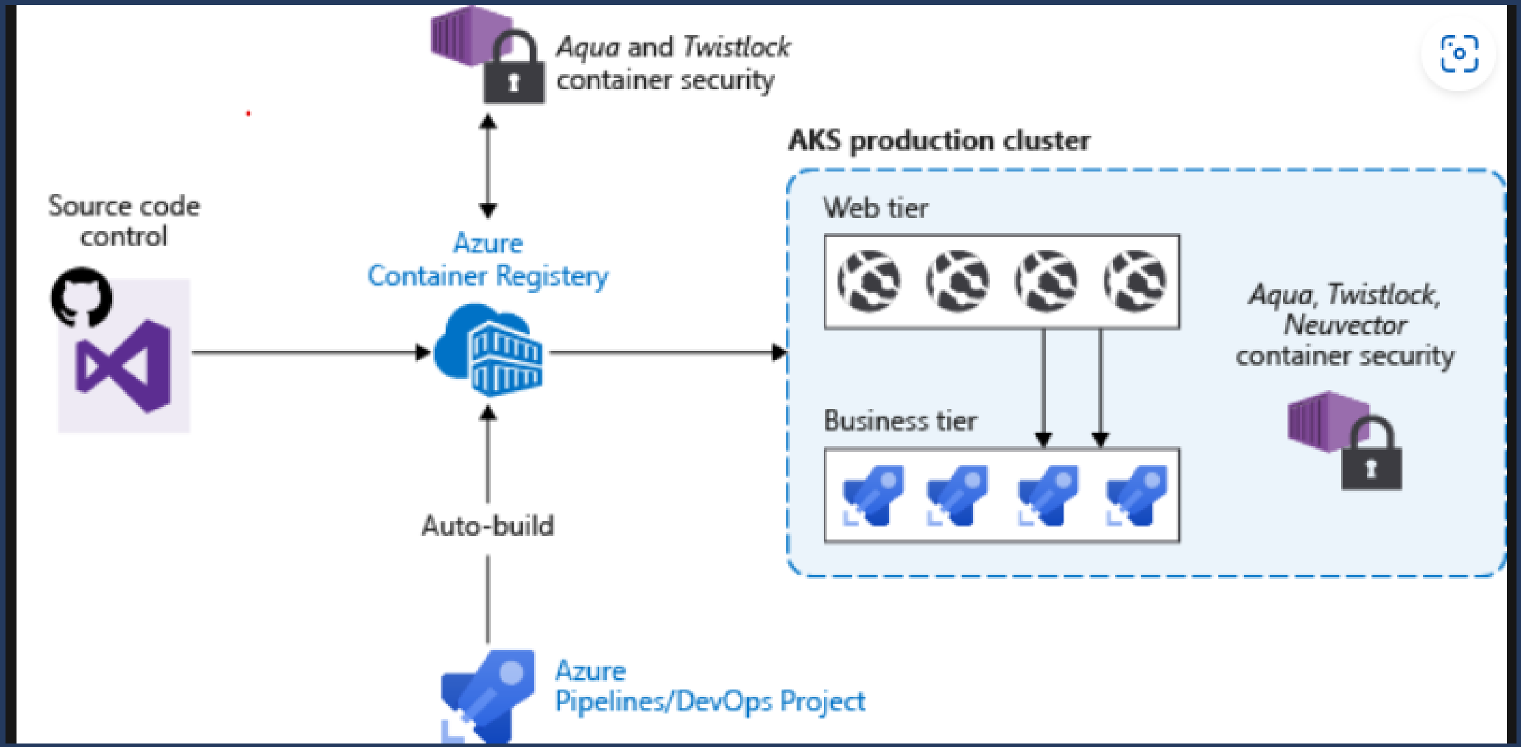
```
az aks get-upgrades --resource-group myResourceGroup --name myAKSCluster --output table
```

```
az aks upgrade --resource-group myResourceGroup --name myAKSCluster --kubernetes-version KUBERNETES_VERSION
```

Container Image management

Build and Runtime Security

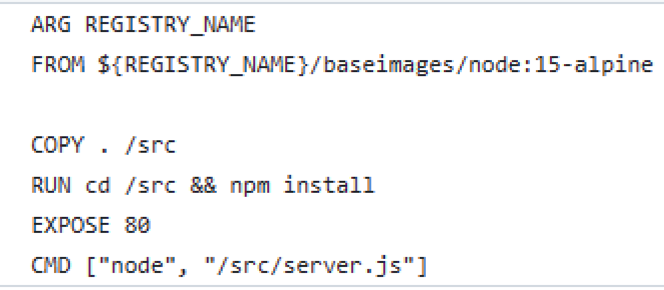
- Scan your container images for vulnerabilities. Only deploy validated images.
- Include in your deployment workflow a process to scan container images using tools such as Twistlock or Aqua.
-


Aqua and Twistlock container security
AKS production cluster
Source code control
Azure Container Registery
Web tier
Aqua, Twistlock, Neuvector container security
Business tier
Auto-build
Azure Pipelines/DevOps Project

Container Image management

- Automatically build new images on base image update
  - Use automation to build new images when the base image is updated.
  - ACR Tasks can also automatically update container images when the base image is updated

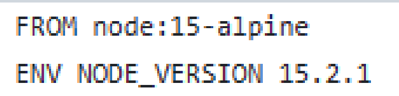
Dockerfile-app



```
ARG REGISTRY_NAME
FROM ${REGISTRY_NAME}/baseimages/node:15-alpine

COPY . /src
RUN cd /src && npm install
EXPOSE 80
CMD ["node", "/src/server.js"]
```

Dockerfile-base



```
FROM node:15-alpine
ENV NODE_VERSION 15.2.1
```

- Build base image

```
az acr build --registry $ACR_NAME --image baseimages/node:15-alpine --file
Dockerfile-base .
```

- Create ACR task

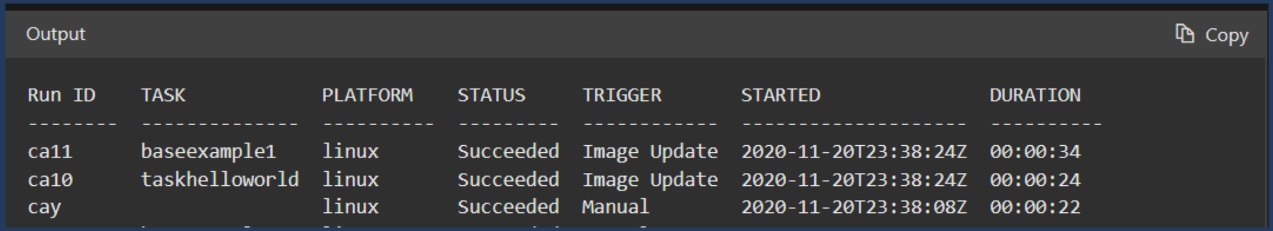
```
az acr task create \
  --registry $ACR_NAME \
  --name baseexample1 \
  --image helloworld:{{.Run.ID}} \
  --arg REGISTRY_NAME=$ACR_NAME.azurecr.io \
  --context https://github.com/$GIT_USER/acr-build-helloworld-node.git#main \
  --file Dockerfile-app \
  --git-access-token $GIT_PAT
```

- Manual Trigger

```
az acr task run --registry $ACR_NAME --name baseexample1
```

Update node version : ENV NODE_VERSION 15.2.1a

```
az acr build --registry $ACR_NAME --image baseimages/node:15-alpine --file Dockerfile-base .
az acr task list-runs --registry $ACR_NAME --output table
```

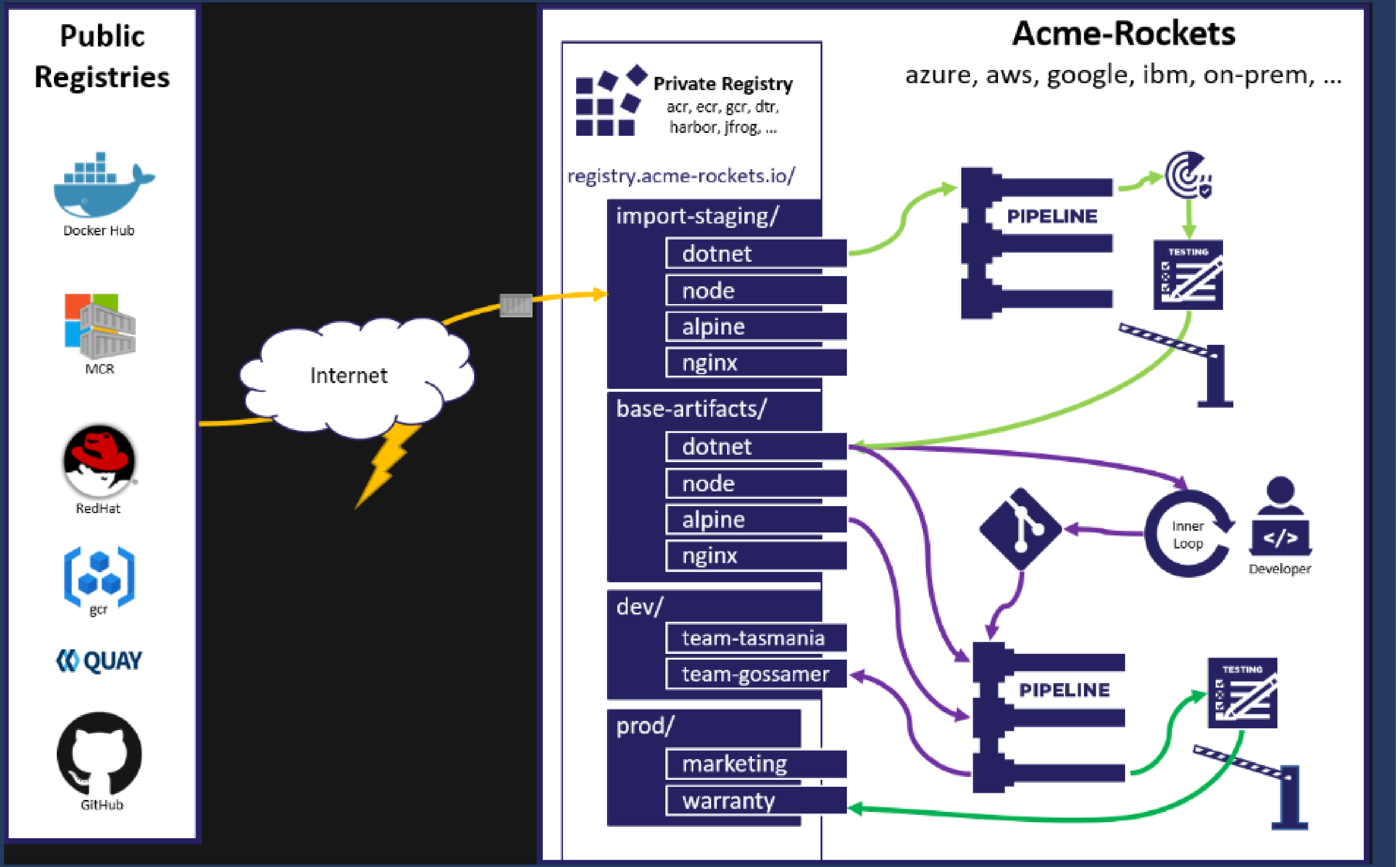


```
Run ID    TASK            PLATFORM   STATUS     TRIGGER       STARTED               DURATION
--------  --------------  ---------  ---------  ------------  --------------------  ----------
ca11      baseexample1    linux      Succeeded  Image Update  2020-11-20T23:38:24Z  00:00:34
ca10      taskhelloworld  linux      Succeeded  Image Update  2020-11-20T23:38:24Z  00:00:24
cay                       linux      Succeeded  Manual        2020-11-20T23:38:08Z  00:00:22
```

- Now set the scheduled trigger (--schedule "0 21 * * *" \ in acr task create command)

Container Image management

Builds the base image, pushes it to container registry if the build is successful.



Container Image management

Public content with ACR Tasks

Container Image Tagging

Stable Tags :

Tags that is reused, for example, major or minor version mycontainerimage:1.0.

Use stable tags to maintain **base images** for your container builds

These tags continue to receive updates.

Unique tags : *A different tag for each image pushed to a registry, mycontainerimage:abc123.*

Use unique tags for **deployments**, especially in an environment that could scale on multiple nodes.

Some patterns

- *date-time*
- *git commit*
- *manifest digest*
- *build id*

Lock deployed image tags

lock any deployed image tag, by setting its write-enabled attribute to false.

For Operators

- Multi-tenancy and cluster isolation
  - Logical Vs Physical
- Scheduling features
  - Resource Quotas
  - Taint & Toleration
  - Node affinity
- Networking
  - Azure CNI , Kubenet, Ingress traffic
  - Network Policies
- Storage
  - Dynamic provisioning
  - Secure and backup - Velero

Pod Security

- **allowPrivilegeEscalation** Design your applications so this setting is always set to *false*.

- **Linux capabilities** let the pod access underlying node processes.

- **SELinux labels** is a Linux kernel security module that lets you define access policies for services, processes, and filesystem access.

- Pod runs as user ID *1000* and part of group ID *2000*
- Can't escalate privileges to use root
- Allows Linux capabilities to access network interfaces and the host's real-time (hardware) clock

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    fsGroup: 2000
  containers:
    - name: security-context-demo
      image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
      securityContext:
        runAsUser: 1000
        allowPrivilegeEscalation: false
        capabilities:
          add: ["NET_ADMIN", "SYS_TIME"]
```

For Developers

Resource management

VS Code extension for Kubernetes

Debug with bridge to kubernetes

Pod Security

Networking and Storage

Networking

Choose the right network model (kubenet,Azure CNI)

Distribute ingress traffic – Ingress controllers

Secure traffic with a web application firewall

Control traffic flow with network policies

Securely connect to nodes through a bastion host

Network Policy

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: backend-policy
spec:
  podSelector:
    matchLabels:
      app: backend
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
```

Use network policies to allow or deny traffic to pods. By default, all traffic is allowed between pods within a cluster. For improved security, define rules that limit pod communication.

- You create a network policy as a Kubernetes resource using a YAML manifest.
- Policies are applied to defined pods, with ingress or egress rules defining traffic flow.
- As pods are dynamically created in an AKS cluster, required network policies can be automatically applied.

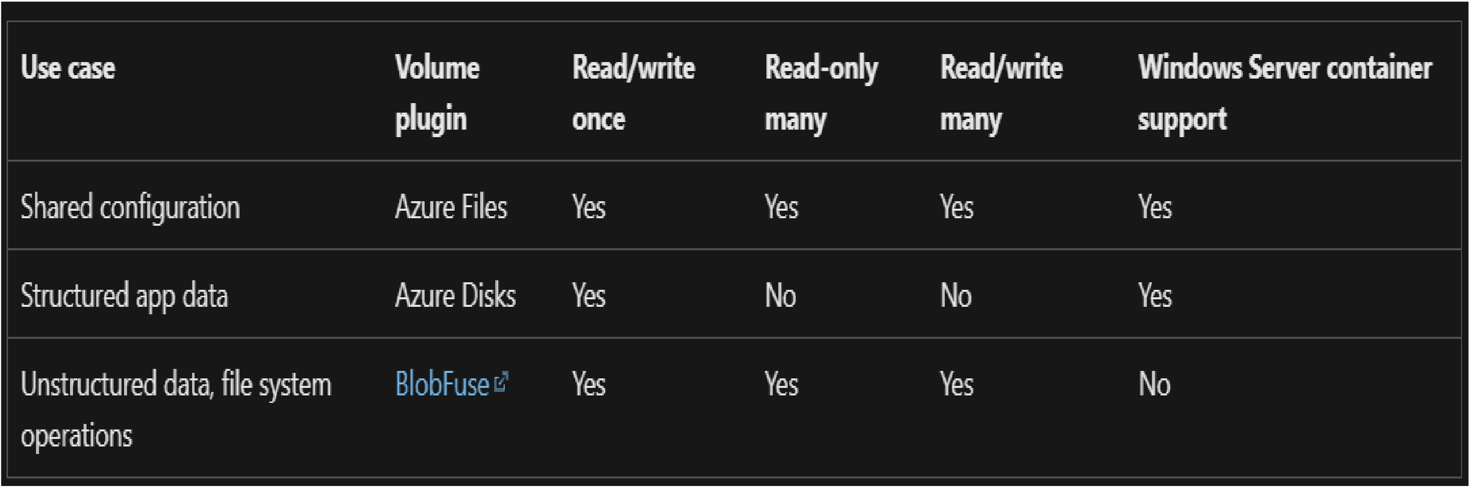
Reference:

[ahmetb/kubernetes-network-policy-recipes: Example recipes for Kubernetes Network Policies that you can just copy paste (github.com)]

Storage

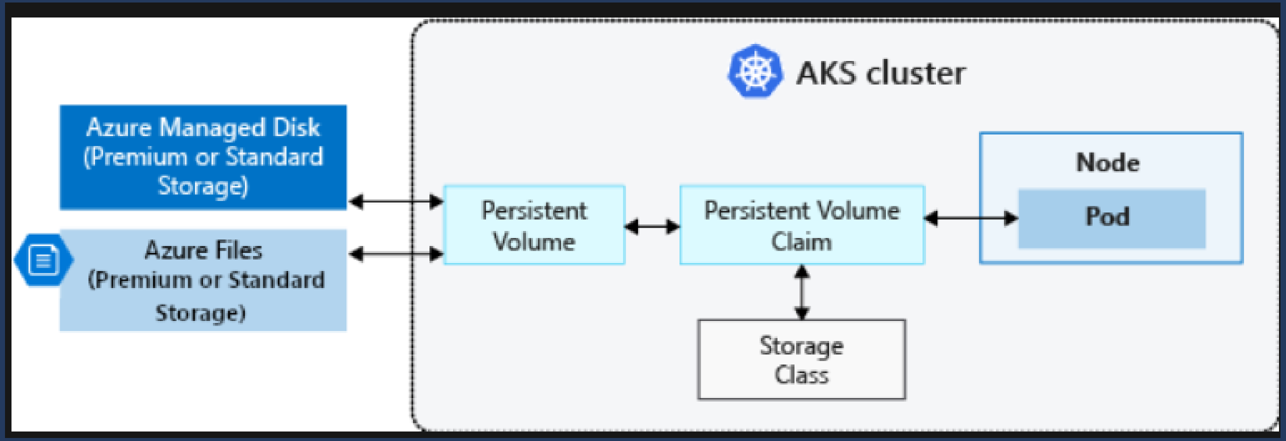
Backup data with Velero or Azure backup

Choose the right storage type



Use case	Volume plugin	Read/write once	Read-only many	Read/write many	Windows Server container support
Shared configuration	Azure Files	Yes	Yes	Yes	Yes
Structured app data	Azure Disks	Yes	No	No	Yes
Unstructured data, file system operations	BlobFuse	Yes	Yes	Yes	No

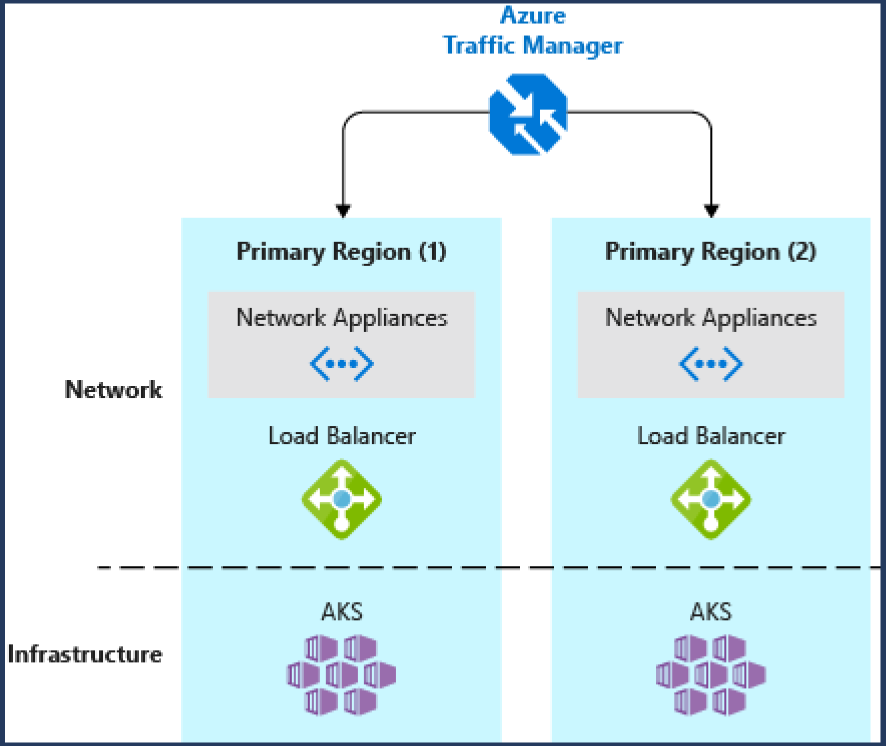
Dynamic volume provisioning


AKS cluster
Azure Managed Disk (Premium or Standard Storage)
Azure Files (Premium or Standard Storage)
Persistent Volume
Persistent Volume Claim
Node
Pod
Storage Class

Business Continuity and Disaster Recovery

BCP/DR

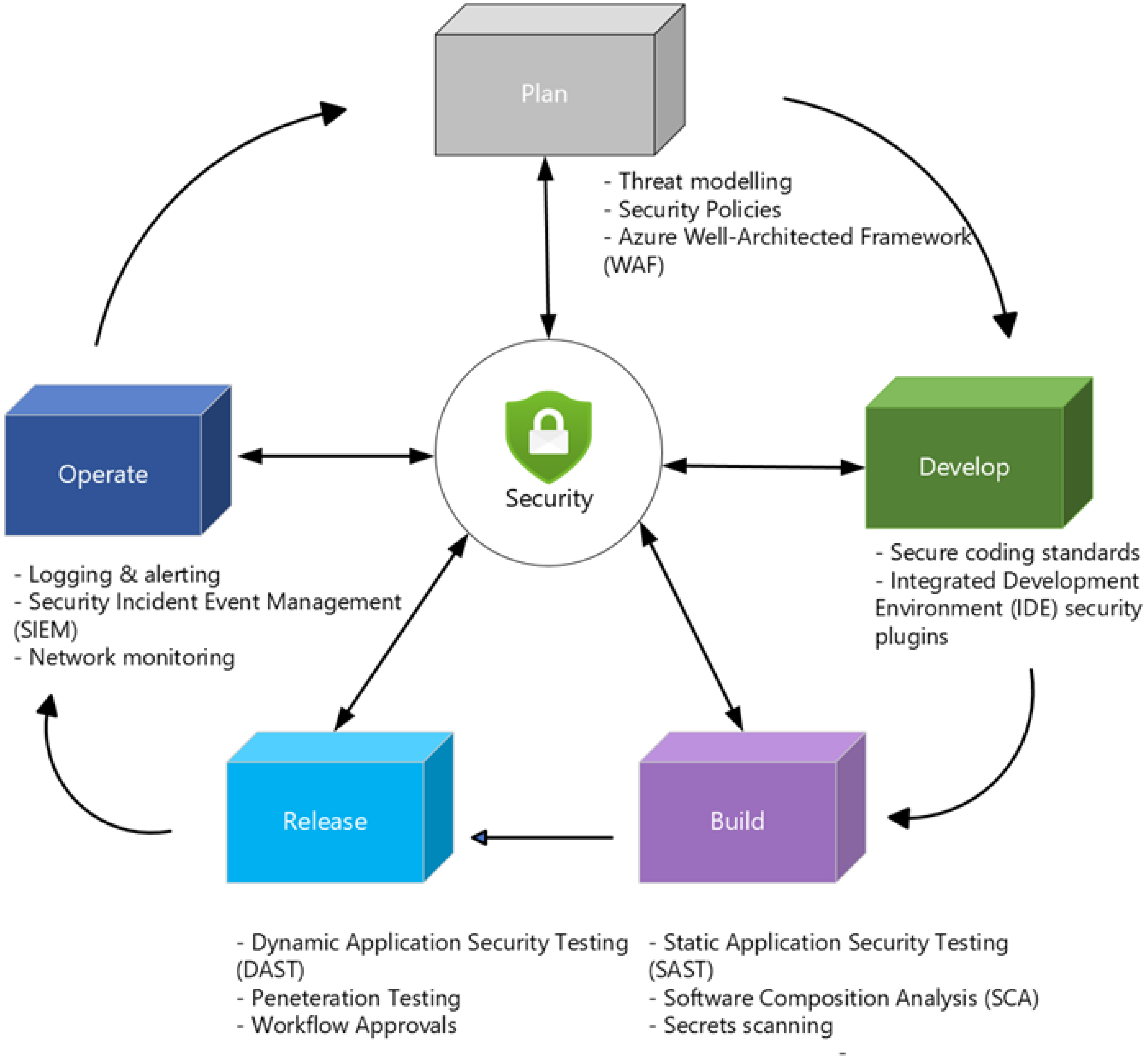
- Plan for multi region deployment
- Use Azure Traffic manager to route request



- Enable Geo-replication for container images

3

DevSecOps in AKS



- Threat modelling
- Security Policies
- Azure Well-Architected Framework (WAF)

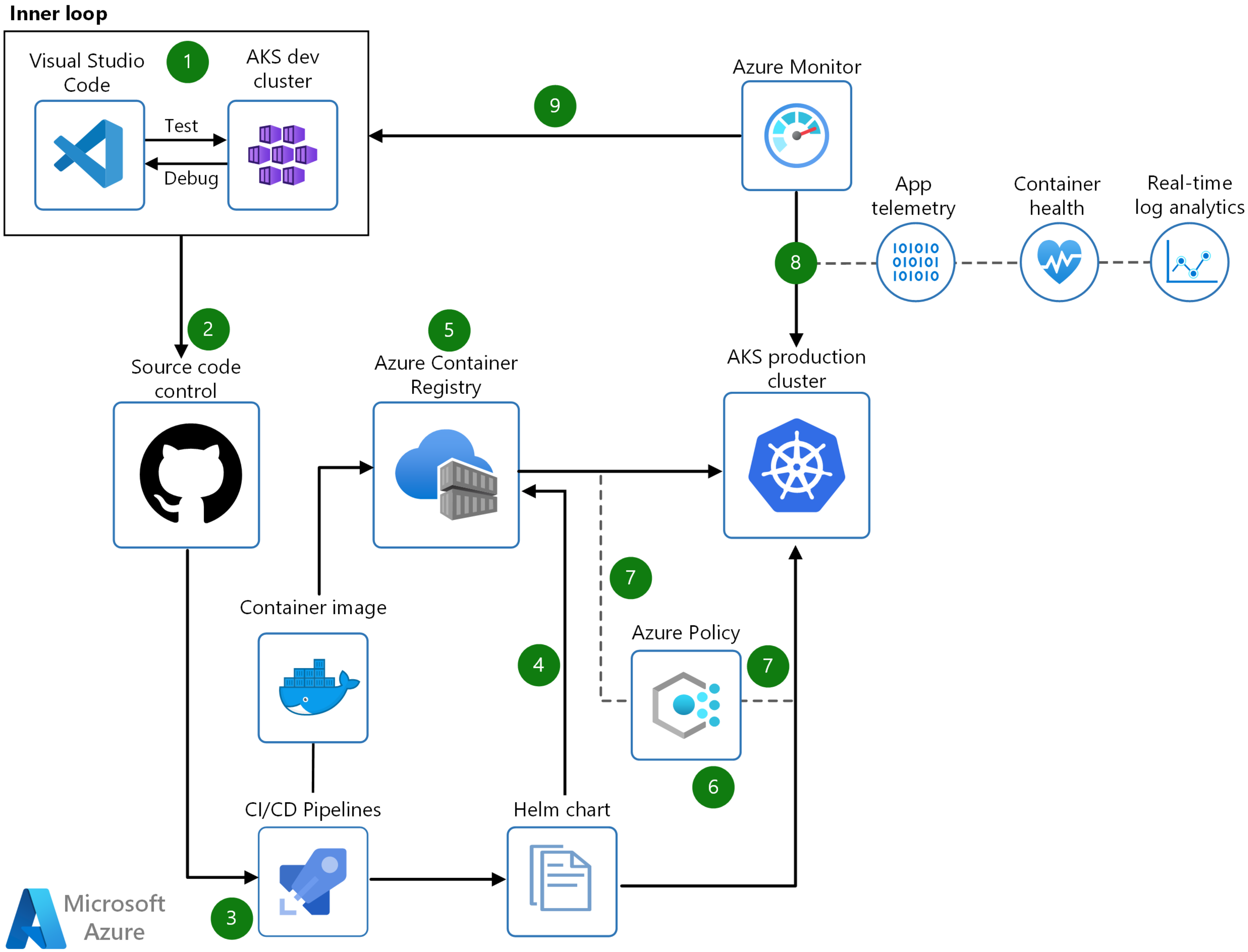
- Secure coding standards
- Integrated Development Environment (IDE) security plugins

- Static Application Security Testing (SAST)
- Software Composition Analysis (SCA)
- Secrets scanning

-

- Dynamic Application Security Testing (DAST)
- Peneteration Testing
- Workflow Approvals

- Logging & alerting
- Security Incident Event Management (SIEM)
- Network monitoring


Inner loop
Visual Studio Code
1
AKS dev cluster
Test
Debug
9
Azure Monitor
App telemetry
Container health
Real-time log analytics
8
2
Source code control
5
Azure Container Registry
AKS production cluster
7
Container image
4
Azure Policy
7
6
CI/CD Pipelines
Helm chart
Microsoft Azure
3

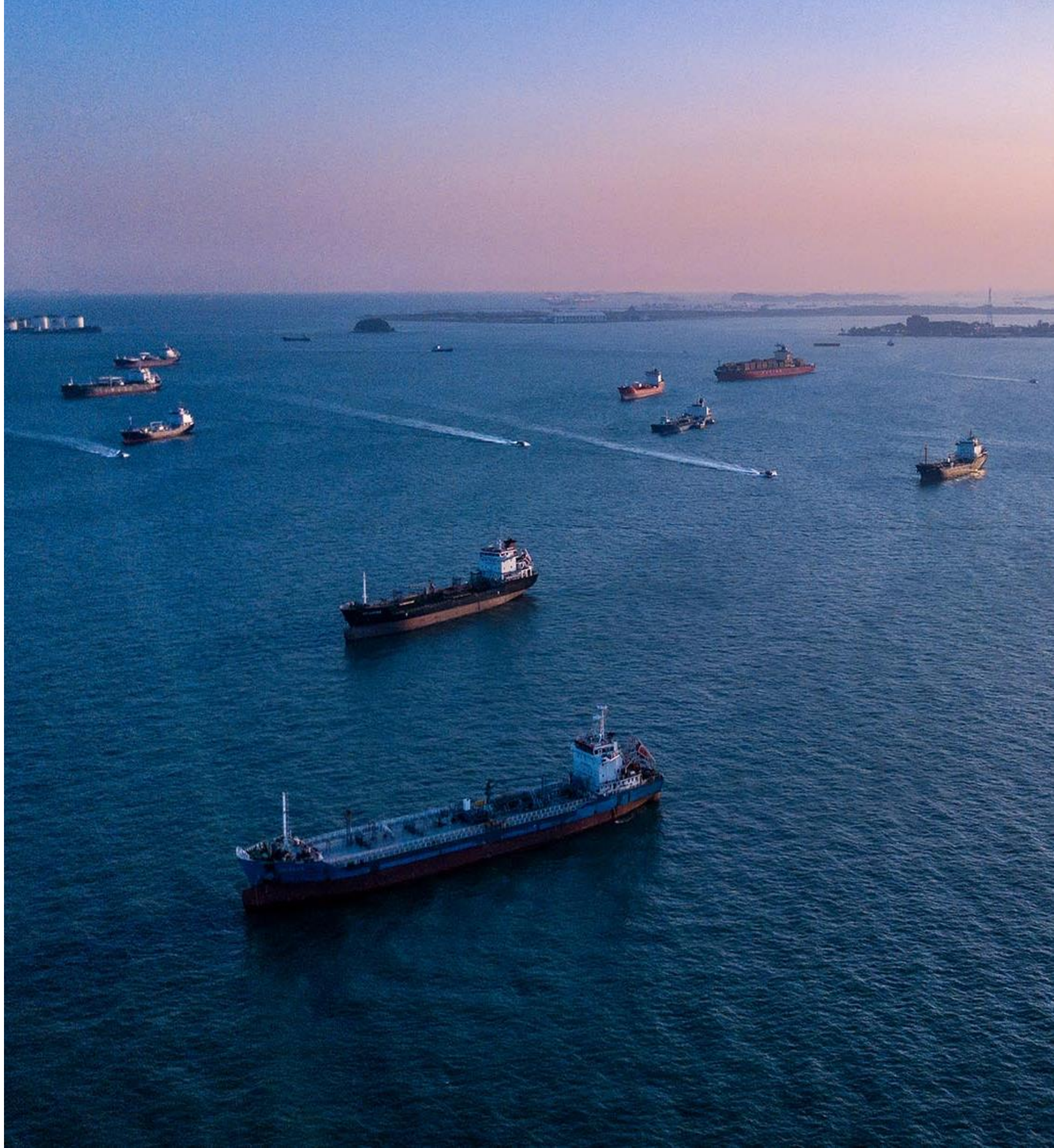
Microsoft

[Configure Azure CNI networking in Azure Kubernetes Service (AKS) - Azure Kubernetes Service | Microsoft Docs]

[Baseline architecture for an AKS cluster - Azure Architecture Center | Microsoft Docs]

[11 Ways (Not) to Get Hacked | Kubernetes]

[The Azure Kubernetes Service Checklist - ✨ Be ready for production ✨ (the-aks-checklist.com)]



Microsoft

Thank you