



kubernetes

Securing the Cluster

Kubernetes



Course Objectives

In this module, you will learn:

- Implementing Security in Kubernetes Cluster using
 - RBAC
 - Namespaces &
 - Network Policies

Use a Minimal OS

- Kubernetes nodes are VM instances
- The nodes should run a minimal OS to reduce any possible vulnerabilities
 - Container-Optimized OS (cos) from Google
 - Container-Optimized OS with containerd (cos_containerd)
 - Ubuntu
- Recommended for security to use the Container-Optimized OS
 - Optimized to enhance node security
 - cos_containerd is a variant of the cos image with containerd as the runtime
- Use a cloud provider's managed Kubernetes service

Role-Based Access Control (RBAC)

- RBAC is used to grant permissions to resources at the cluster or namespace level
 - RBAC allows you to define roles with rules containing a set of permissions
- When using RBAC, define roles with the desired permissions
 - The roles can then be bound to users
- RBAC permissions provide finer-grained control over access to resources within each cluster

RBAC Roles and ClusterRoles

- Permissions are defined within a Kubernetes Role or ClusterRole
 - A *Role* grants access to resources within a single namespace
 - A *ClusterRole* grants access to resources in the entire cluster
- A RoleBinding (or ClusterRoleBinding) grants the permissions in the Role (or ClusterRole) to a set of users
 - Contains a list of the users, and a reference to the Role (or ClusterRole) being granted to those users

Defining Roles and ClusterRoles

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: blue
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

Example Role in the “blue” namespace that can be used to grant read access to pods

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  # "namespace" omitted since ClusterRoles are not namespaced
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

Example ClusterRole to grant read access to secrets in any particular namespace, or across all namespaces, depending on how it's bound

Binding a Role or ClusterRole

```
kind: RoleBinding # must be RoleBinding or ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: blue
subjects:
- kind: User
  name: steve@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role # must be Role or ClusterRole
  name: pod-reader # must match a Role or ClusterRole name to bind to
  apiGroup: rbac.authorization.k8s.io
```

This role binding assigns "steve@example.com" the pod-reader role in the "blue" namespace


Use Namespaces

- Namespaces allow a single Kubernetes cluster to be divided into multiple “virtual clusters”
 - Can be used to divide cluster resources between multiple users
 - Multiple namespaces inside a single Kubernetes cluster are logically isolated from each other
 - Can help with organization, security, and even performance
- By default, Kubernetes clusters will have a namespace called **default**
 - Actually three namespaces (default, kube-system, kube-public)
 - **kube-public** could be used to share configmaps across namespaces
 - **kube-system** should be left alone

Creating Namespaces


```
kind: Namespace
apiVersion: v1
metadata:
  name: development
  labels:
    name: development
```

Configuration to create a namespace



```
$ kubectl apply -f pod.yaml --namespace=development
```

Use the namespace parameter to put resources in a particular namespace

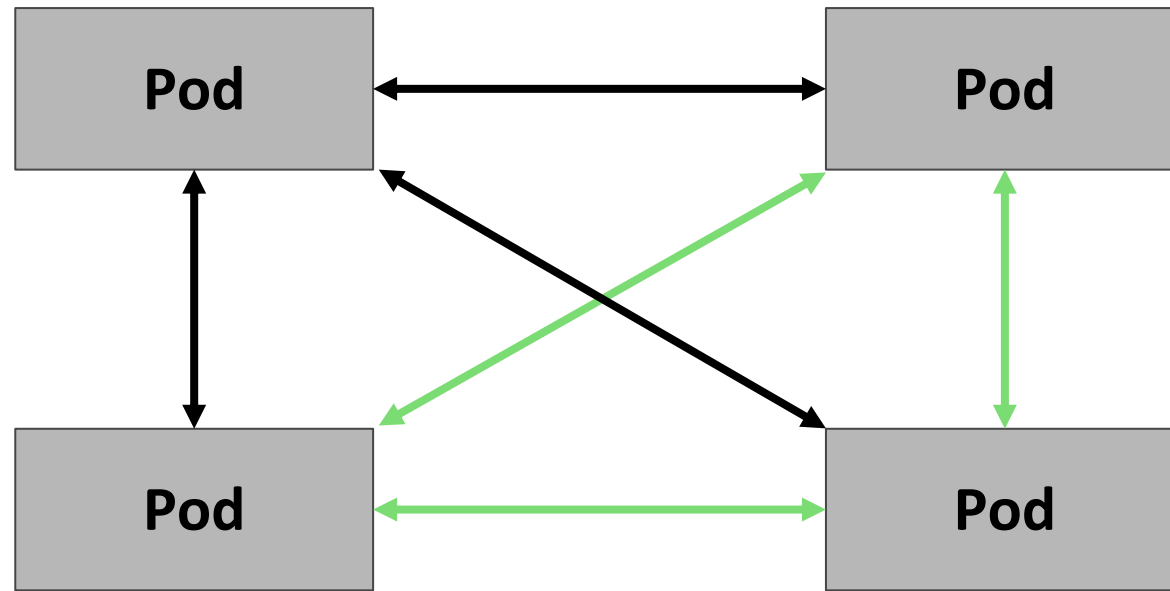


Isolating Resources in a Namespace

- Namespaces are separate from each other, but they are not fully isolated by default
 - A service in one namespace can talk to a service in another namespace
- Built-in DNS service discovery using a common DNS pattern
 - <Service Name>.<Namespace Name>
 - For example: myapp.development
- Network Policies can be used to isolate namespaces
- RBAC can also apply at the namespace level

Restrict Traffic Among Pods with a Network Policy

- By default, all Pods in a cluster can communicate with each other
 - Pod to Pod communication should be controlled as needed for your applications
 - Make it more difficult for attackers to move laterally within your cluster



Restrict Traffic Among Pods with a Network Policy (continued)

- Pods become isolated by having a network policy that selects them
 - Once there is a network policy in a namespace selecting a particular pod, that pod will reject any connections that are not allowed by any network policy
 - Other pods in the namespace that are not selected by any network policy will continue to accept all traffic
- For a network policy to be enforced, you must first enable the network policy flag on the cluster itself

```
gcloud container clusters create my-cluster --enable-network-policy ...
```

```
gcloud container clusters update my-cluster --enable-network-policy ...
```

Kubernetes Network Policy

- The following network policy will limit access to the myapp service so connections are only allowed from pods with the label access: true

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: access-myapp
spec:
  podSelector:
    matchLabels:
      run: myapp
  ingress:
  - from:
    - podSelector:
        matchLabels:
          access: "true"
```

myapp-policy.yaml

- Use kubectl to create a network policy from the myapp-policy.yaml file:

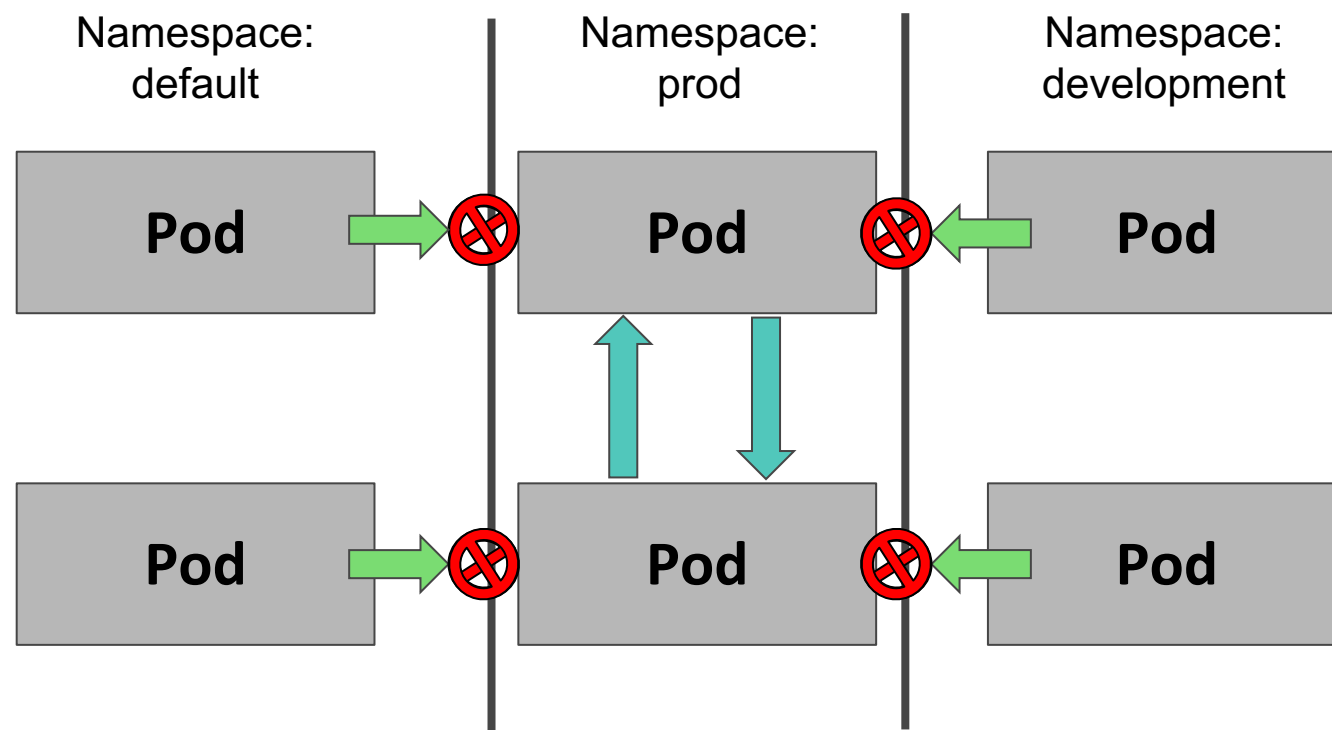
```
$ kubectl create -f myapp-policy.yaml
```

```
networkpolicy.networking.k8s.io/access-myapp created
```

Kubernetes Network Policy

- The following policy will deny all the traffic from other namespaces while allowing all the traffic coming from the same namespace (prod)

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: prod
  name: deny-from-other-namespaces
spec:
  podSelector:
    matchLabels:
  ingress:
    - from:
      - podSelector: {}
```



Other Network Policy Features

- Egress network policies
 - Restrict your workloads from establishing connections to resources outside specified IP ranges
- IP blocks
 - Specify IP ranges to allow/deny traffic in ingress or egress rules
- Cross-namespace policies
 - Enforce network policies for particular namespaces in the cluster
- Restrict traffic to port numbers
 - Specify port numbers for the policy to enforce
- Search the web for “k8s network policy recipes” for examples