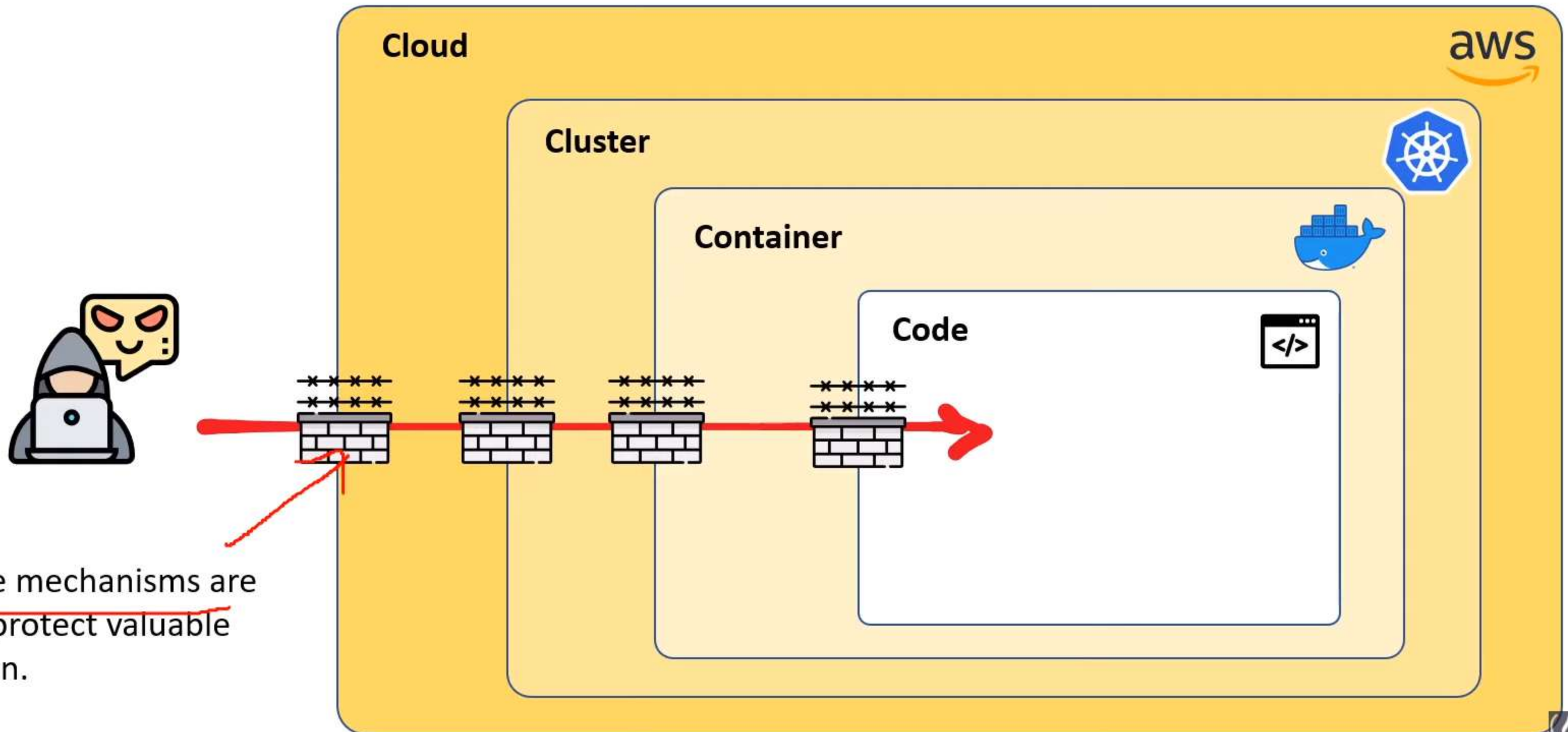


4Cs of Cloud Native security

Cheat sheets, Practice Exams and Flash cards 🖱️ www.examprompro.co/kcna

The 4C's of Cloud Native security are **Cloud, Clusters, Containers, and Code**
Each layer of the Cloud Native security model **builds upon the next outermost layer**



Depth in Defense

a series of defensive mechanisms are layered in order to protect valuable data and information.

4Cs of Cloud Native security – Cloud Layer

Cheat sheets, Practice Exams and Flash cards  www.examprompro.co/kcna

The Cloud Layer is also known as the **base layer**.

Security at this layer is going to vary based on **managed** or **self-managed infrastructure**

Managed Infrastructure (IaaS)

- Cloud Service Provider (CSP)

- AWS, Azure, GCP



- Cloud Platform

- CIVO



Security is going to greatly vary based on provider, and independent research will need be undertaken



Eg. AWS's K8s managed offering is called EKS.
Research EKS security would need to be performed

Self-Managed

- Co-Located Servers

- Equinix



- Corporate Datacenters

Security is going to be based on Infrastructure Security

- Network access to API Server (Control plane)
- Network access to Nodes (nodes)
- Kubernetes access to Cloud Provider API
- Access to etcd
- etcd Encryption

4Cs of Cloud Native security – Cluster Layer

Cheat sheets, Practice Exams and Flash cards 🖱️ www.examprompro.co/kcna

There are two parts to Cluster Layer security

1. Components of the cluster

Securing configurable cluster components

- Controlling access to the Kubernetes API
 - Use Transport Layer Security (TLS) for all API traffic
 - API Authentication
 - API Authorization
- Controlling access to the Kubelet
- Controlling the capabilities of a workload or user at runtime
 - Limiting resource usage on a cluster
 - Controlling what privileges containers run with
 - Preventing containers from loading unwanted kernel modules
 - Restricting network access
 - Restricting cloud metadata API access
 - Controlling which nodes pods may access
- Protecting cluster components from compromise
 - Restrict access to etcd
 - Enable audit logging
 - Restrict access to alpha or beta features
 - Rotate infrastructure credentials frequently
 - Review third party integrations before enabling them
 - Encrypt secrets at rest
 - Receiving alerts for security updates and reporting vulnerabilities

2. Components in the cluster

Securing the applications running within the cluster

- RBAC Authorization (Access to the Kubernetes API)
- Authentication
- Application secrets management (and encrypting them in etcd at rest)
- Ensuring that pods meet defined Pod Security Standards
- Quality of Service (and Cluster resource management)
- Network Policies
- TLS for Kubernetes Ingress

4Cs of Cloud Native security – Container and Code Layer

Cheat sheets, Practice Exams and Flash cards 🖱️ www.exampor.co/kcna

Container Layer

- Container Vulnerability Scanning and OS Dependency Security
- Image Signing and Enforcement
- Disallow privileged users
- Use container runtime with stronger isolation

Code Layer

Application code is one of the primary attack surfaces over which you have the most control

- Access over TLS only
- Limiting port ranges of communication
- 3rd Party Dependency Security
- Static Code Analysis
- Dynamic probing attacks

Infrastructure Security

Cheat sheets, Practice Exams and Flash cards 🖱️ www.examprompro.co/kcna

Suggestions for securing your infrastructure in a Kubernetes cluster

Network access to API Server (Control plane)

All access to the Kubernetes control plane is not allowed publicly on the internet and is controlled by network access control lists restricted to the set of IP addresses needed to administer the cluster.

Network access to Nodes (nodes)

Nodes should be configured to *only* accept connections from the control plane on the specified ports, and accept connections for services in Kubernetes of type NodePort and LoadBalancer.

Kubernetes access to Cloud Provider API

Each cloud provider grant a different set of permissions to the Kubernetes control plane and nodes. It is best to provide the cluster with cloud provider access that follows the principle of least privilege for the resources it needs to administer.

Access to etcd

Access to etcd (the datastore of Kubernetes) should be limited to the control plane only. Depending on your configuration, you should attempt to use etcd over TLS. More information can be found in the etcd documentation.

etcd Encryption

Wherever possible it's a good practice to encrypt all storage at rest, and since etcd holds the state of the entire cluster (including Secrets) its disk should especially be encrypted at rest



AAA

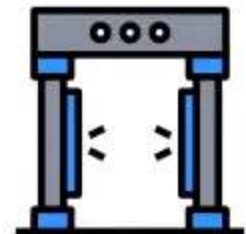
Cheat sheets, Practice Exams and Flash cards 🖱️ www.examprompro.co/kcna

Authentication, Authorization and Accounting (AAA) framework for Identity management systems.



Authentication — to identify

- Static passwords
- One-time password (OTP) — MFA/UFA
- Digital certificates (x.509)
- Basic Auth



Authorization — to get permission

- Role Based Access Controls (RBAC)



Accounting (auditing) —to log and audit trail

- Audit Policies
- Audit Backends (where the logs will be stored)

Role-based Access Controls

Cheat sheets, Practice Exams and Flash cards 🖱️ www.examprompro.co/kcna

Role-based Access Controls (RBAC) is a way of defining permissions for identities based on an organizational role.

RBAC authorization uses the **rbac.authorization.k8s.io** API group to drive authorization decisions, allowing you to dynamically configure policies through the Kubernetes API.

To enable RBAC, start the API server with the **--authorization-mode** flag

```
kube-apiserver --authorization-mode=RBAC
```



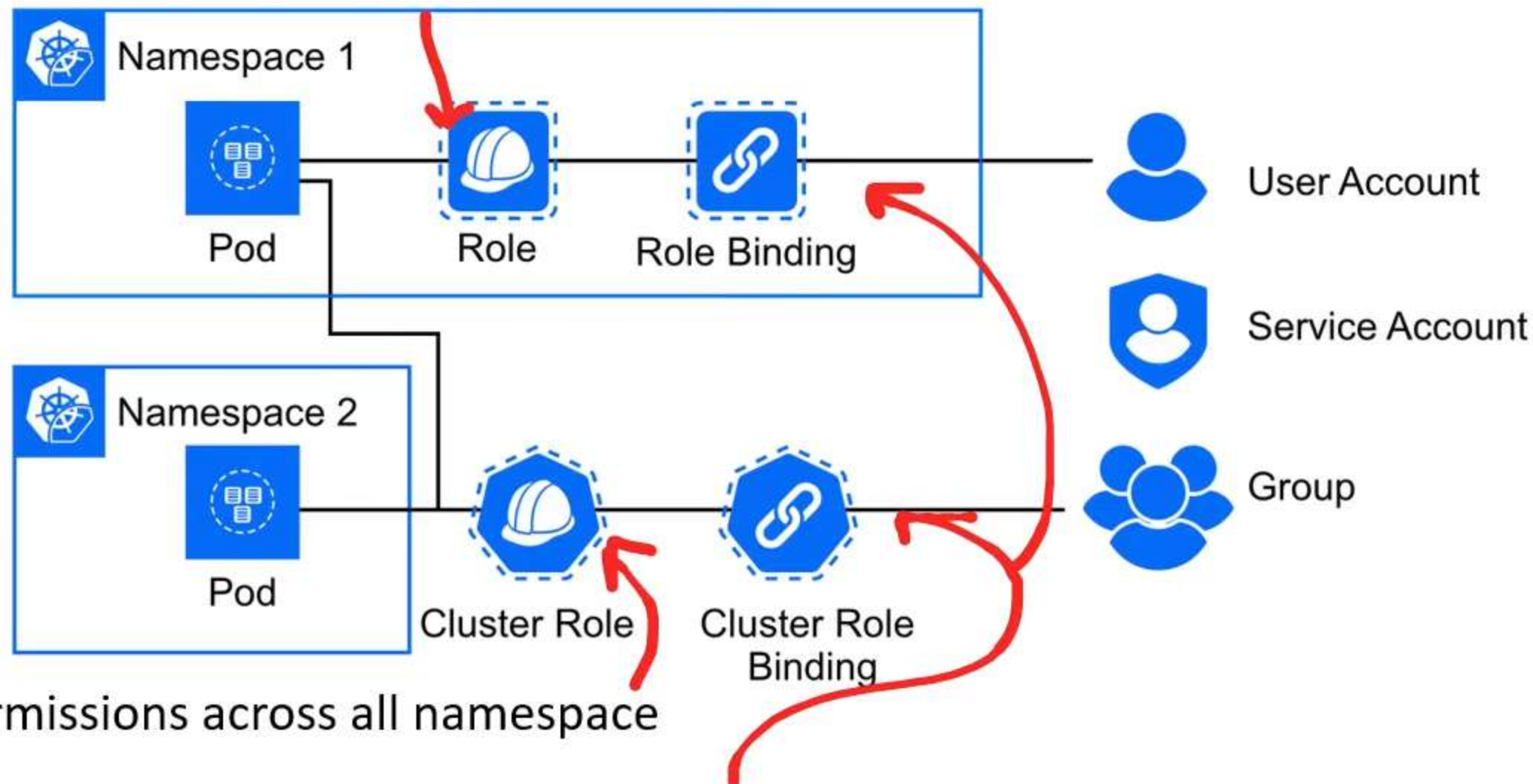
With Kubernetes RBAC there are only Allow Rules, Everything is Deny by default.

Role-based Access Controls Types

Cheat sheets, Practice Exams and Flash cards 🖱️ www.examprompro.co/kcna

The RBAC API declares four kinds of Kubernetes object: Role and RoleBinding, ClusterRole and ClusterRoleBinding.

A **Role** is a set of permissions for a particular namespace



A **ClusterRole** is a set of permissions across all namespace

Role Binding and **Cluster Role Binding**, link Permissions to to **Subjects** (an Identity).

- User Account – A single user
- Service Account – Represents a machine user, to be used by an application service
- Group – A group of User Accounts and/or Service Accounts

Role Configuration Example

Cheat sheets, Practice Exams and Flash cards 🖱️ www.examprompro.co/kcna

Example of a **Role configuration** in the default namespace

Kind:

- Role
- ClusterRole

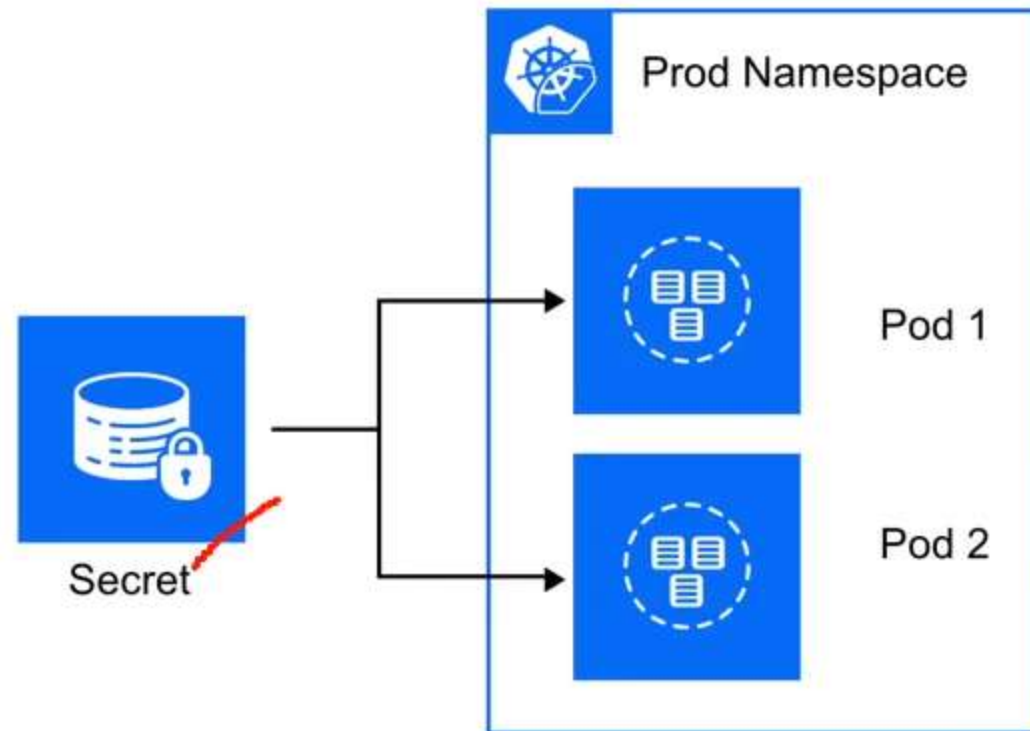
```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

The rules (permissions) of **actions** this role is allowed to perform

Secrets Management

Cheat sheets, Practice Exams and Flash cards 🖱️ www.examprompro.co/kcna

A Secret is similar to a ConfigMap with the exception that they can be encrypted



```
apiVersion: v1
kind: Secret
metadata:
  name: secret-basic-auth
type: kubernetes.io/basic-auth
stringData:
  username: admin
  password: t0p-Secret
```



By default Secrets are unencrypted in etcd store.

- Anyone with access to the etcd store has access to the secrets
- Anyone who has access to a Pod within Namespace will have access to the Secrets used by that pod



How to keep Secrets safe:

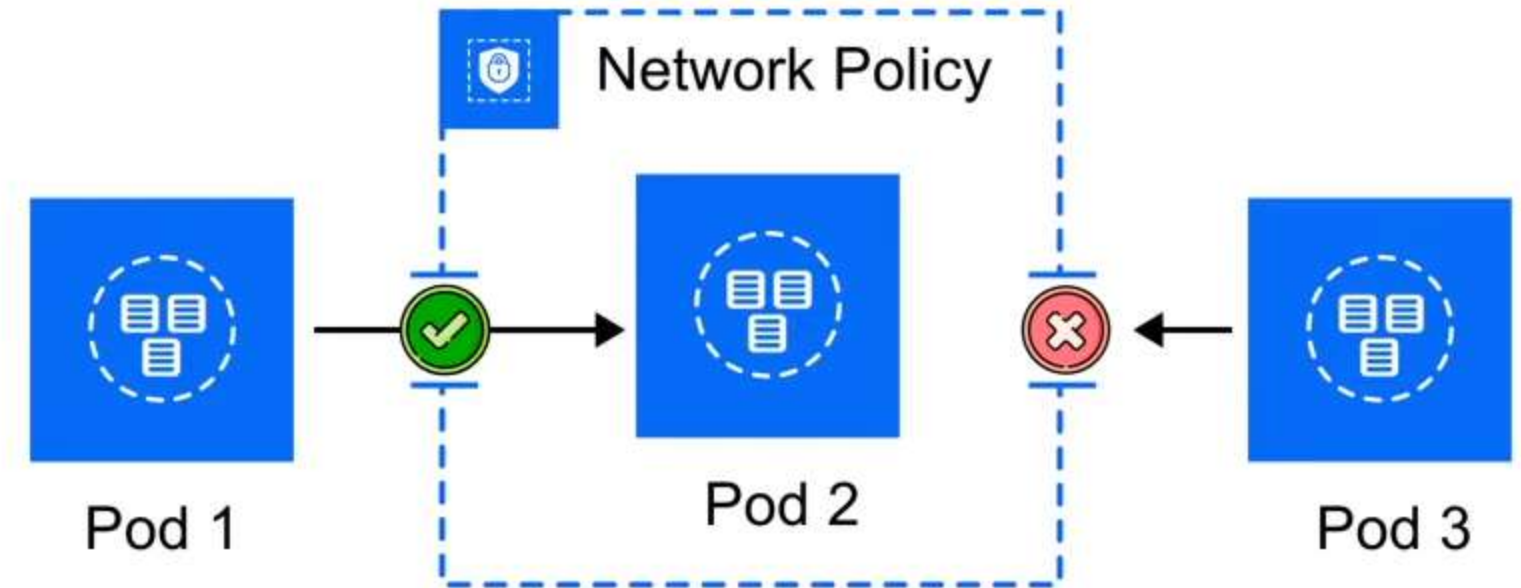
- Enable Encryption at Rest for Secrets
- Enable or configure RBAC rules that restrict reading data in Secrets
- Use mechanisms such as RBAC to limit which principals are allowed to create new Secrets or replace existing ones

Network Policy

Cheat sheets, Practice Exams and Flash cards 🖱️ www.exampopro.co/kcna

Network Policies act as virtual firewalls for pod communication.
Pod communication can be restricted based on the follow scopes:

- Pod-to-pod
- Namespaces
- Specific IPs



Selectors are used to determine to select resources with matching labels for the Network Policy to be applied to



The Network Plugin you are using must support Network Policies
Eg. **Calico**, **Weave Net**, Cilium, Kube-router, Romana.

Calico

Cheat sheets, Practice Exams and Flash cards 🖱️ www.examprompro.co/kcna



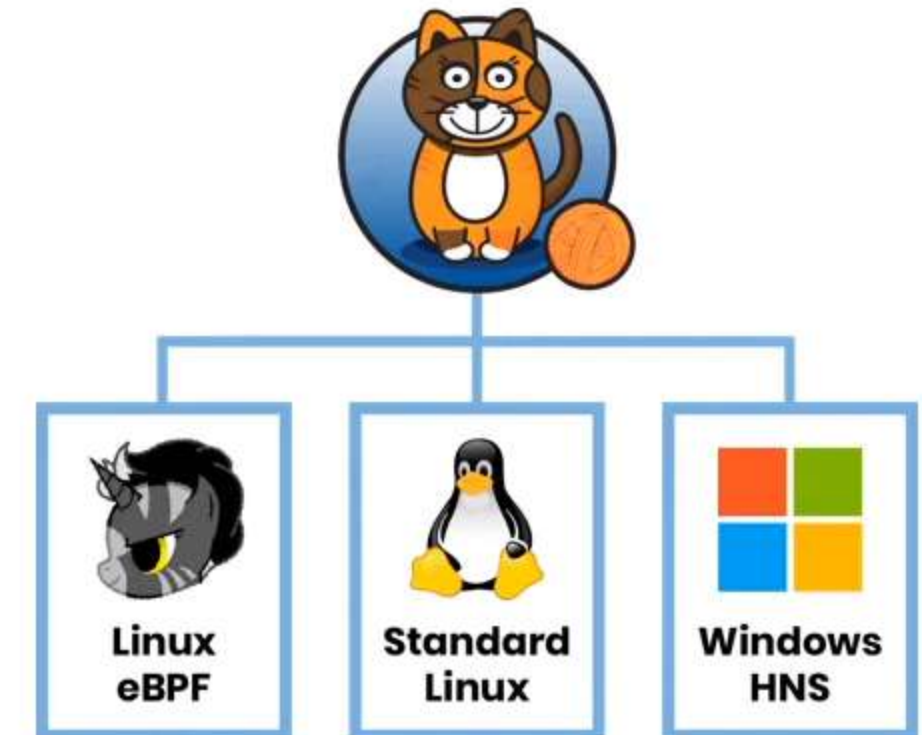
Calico is an **open-source network and network security solution** for **containers, VMs, native host-based workloads.**

Calico supports a broad range of platforms including:

- Kubernetes
- OpenShift
- Mirantis Kubernetes Engine (MKE)
- OpenStack
- Bare Metal Services

Calico gives you a choice of dataplanes:

- Linux eBPF
- Standard Linux
- Windows HNS



Calico Network Policies extends the base functionality of Network Policies:


- policies can be applied to any object
- ~~the~~ rules can contain the specific action
- you can use ports, port ranges, protocols, HTTP/ICMP attributes, IPs or subnets, any selectors as a source/target of the rules
- you can control traffic flows via DNAT settings and policies for traffic forwarding

Calico can perform better than alternative eg. Flannel, Cilium, WeaveNet

Anatomy of a Network Policy File

Cheat sheets, Practice Exams and Flash cards  www.examprompro.co/kcna

For a network policy you'll specify either a **Pod Selector** or **Namespace Selector** 

You can define **Ingress rules**
(permitted traffic entering a pod) 

You can define **Egress rules**
(permitted traffic to exit a pod) 

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
      - ipBlock:
          cidr: 172.17.0.0/16
          except:
            - 172.17.1.0/24
      - namespaceSelector:
          matchLabels:
            project: myproject
      - podSelector:
          matchLabels:
            role: frontend
      ports:
        - protocol: TCP
          port: 6379
  egress:
    - to:
      - ipBlock:
          cidr: 10.0.0.0/24
      ports:
        - protocol: TCP
          port: 5978
```

In-Transit vs At-Rest Encryption

Cheat sheets, Practice Exams and Flash cards 🖱️ www.examprompro.co/kcna

Encryption In-Transit

Data that is secure when moving between locations

Algorithms: **TLS, SSL**

Encryption At-Rest

Data that is secure when residing on storage or within a database

Algorithms: **AES, RSA**

Transport Layer Security (TLS)

An encryption protocol for data integrity between two or more communicating computer application.

*TLS 1.0, 1.1 are deprecated. **TLS 1.2** and **1.3** is the current best practice*

Secure Sockets Layers (SSL)

An encryption protocol for data integrity between two or more communicating computer application

SSL 1.0, 2.0 and 3.0 are deprecated

Certificates and TLS

Cheat sheets, Practice Exams and Flash cards 🖱️ www.examprompro.co/kcna

Kubernetes provides a **certificates.k8s.io** API, which lets you provision TLS certificates signed by a Certificate Authority (CA) that you control. These CA and certificates can be used by your workloads to establish trust.

What is Public key infrastructure (PKI)?

PKI is a set of roles, policies, hardware, software and procedures needed to create, manage, distribute, use, store and revoke digital certificates and manage public-key encryption.



What is a x.509 certificate?

A standard defined by the International Telecommunication Union (ITU) for **public key certifications**.

X.509 certificates are used in many Internet protocol:

- SSL/TLS and HTTPS
- Signed and encrypted email
- Code Signing and Document Signing

A certificate contains

- An identity — hostname, organization or individual
- A public key — RSA, DSA, ECDA etc...

Certificates and TLS

Cheat sheets, Practice Exams and Flash cards 🖱️ www.examprompro.co/kcna

Kubernetes requires PKI for the following operations:

- Client certificates for the kubelet to authenticate to the API server
- Server certificate for the API server endpoint
- Client certificates for administrators of the cluster to authenticate to the API server
- Client certificates for the API server to talk to the kubelets
- Client certificate for the API server to talk to etcd
- Client certificate/kubeconfig for the controller manager to talk to the API server
- Client certificate/kubeconfig for the scheduler to talk to the API server.
- Client and server certificates for the front-proxy

most certificates are stored in **/etc/kubernetes/pki**

- Lightweight distributions store in different locations

K8 Security Best Practices

Cheat sheets, Practice Exams and Flash cards 🖱️ www.examprompro.co/kcna

Best Practices of securing Kubernetes according to Aqua Security



1. Enable Kubernetes Role-Based Access Control (RBAC)
2. Use Third-Party Authentication for API Server
3. Protect etcd with TLS, Firewall and Encryption
4. Isolate Kubernetes Nodes
5. Monitor Network Traffic to Limit Communications
6. Use Process Whitelisting
7. Turn on Audit Logging
8. Keep Kubernetes Version Up to Date
9. Lock Down Kubelet