**Kubernetes Security**

**Securing Container vs Securing VM**

* **Surface of attack:** Minimalist host OS limits the surface of an attack
* **Resource Isolation:** Host resources are separated using namespaces and cgroups
* **Permissions:** Access controls are for app privileges and shared resources
* **Lifetime:** Containers have a shorter average lifetime

# Simplified Kubernetes architecture

If attacker controls a container, they might be able to escape and attack the node

**Kubernetes Cluster**



**Node**

**Master**

**etcd**

**Pod**

Container

If attacker controls the Nodes, they also control the pods running on them and can potentially attack the master or abuse compute

If attacker controls the Master, they control the cluster

If attacker controls etcd, they can access, modify or destroy cluster

# Security journey

Maturity

Setup a cluster

-

Restrict access to kubectl

-

Use RBAC

-

Use a Network Policy

-

Use Namespaces

-

Bootstrap TLS

Prevent known attacks

-

Disable Dashboard

-

Disable default service account token

-

Protect node metadata

-

Scan images for known vulnerabilities

Follow security hygiene

Prevent/limit impact of microservice

* Keep Kubernetes updated
* Use a minimal OS
* Use minimal IAM roles
* Use private IPs on your nodes
* Monitor access with Audit Logging
* Verify binaries that are deployed compromise
* Set a Pod Security Policy
* Protect secrets
* Consider sandboxing
* Limit the identity used by pods
* Use a Service Mesh for

Authentication & Encryption, etc

|  |
| --- |
| **Security journey**  Maturity  Setup a cluster  -  Restrict access to kubectl  -  Use RBAC  -  Use a Network Policy  -  Use Namespaces  -  Bootstrap TLS  Prevent known attacks  -  Disable Dashboard  -  Disable default service account token  -  Protect node metadata  -  Scan images for known vulnerabilities  Follow security hygiene  Prevent/limit impact of microservice |

* Keep Kubernetes updated
* Use a minimal OS
* Use minimal IAM roles
* Use private IPs on your nodes
* Monitor access with Audit Logging
* Verify binaries that are deployed compromise
* Set a Pod Security Policy
* Protect secrets
* Consider sandboxing
* Limit the identity used by pods
* Use a Service Mesh for

Authentication & Encryption, etc

# Getting Started

**1**

**. Restrict access to kubectl**

**2**

**. Use RBAC**

**3**

**. Use a Network Policy**

Prevent unauthorized users from accessing your cluster Use role-based access control to define roles with rules containing a set of permissions

Control pod to pod traffic

1. **Protect Kube Dashboard**

Either disable it or restrict access, as it uses highly privileged Kubernetes service account

1. **Disable account token**

Disable automatic mounting of service account token, as it can be abused by attacker

1. **Use Pod Security Policy**

Enable Docker seccomp and other security restrictions

**Getting Started1. Restrict access to kubectl**

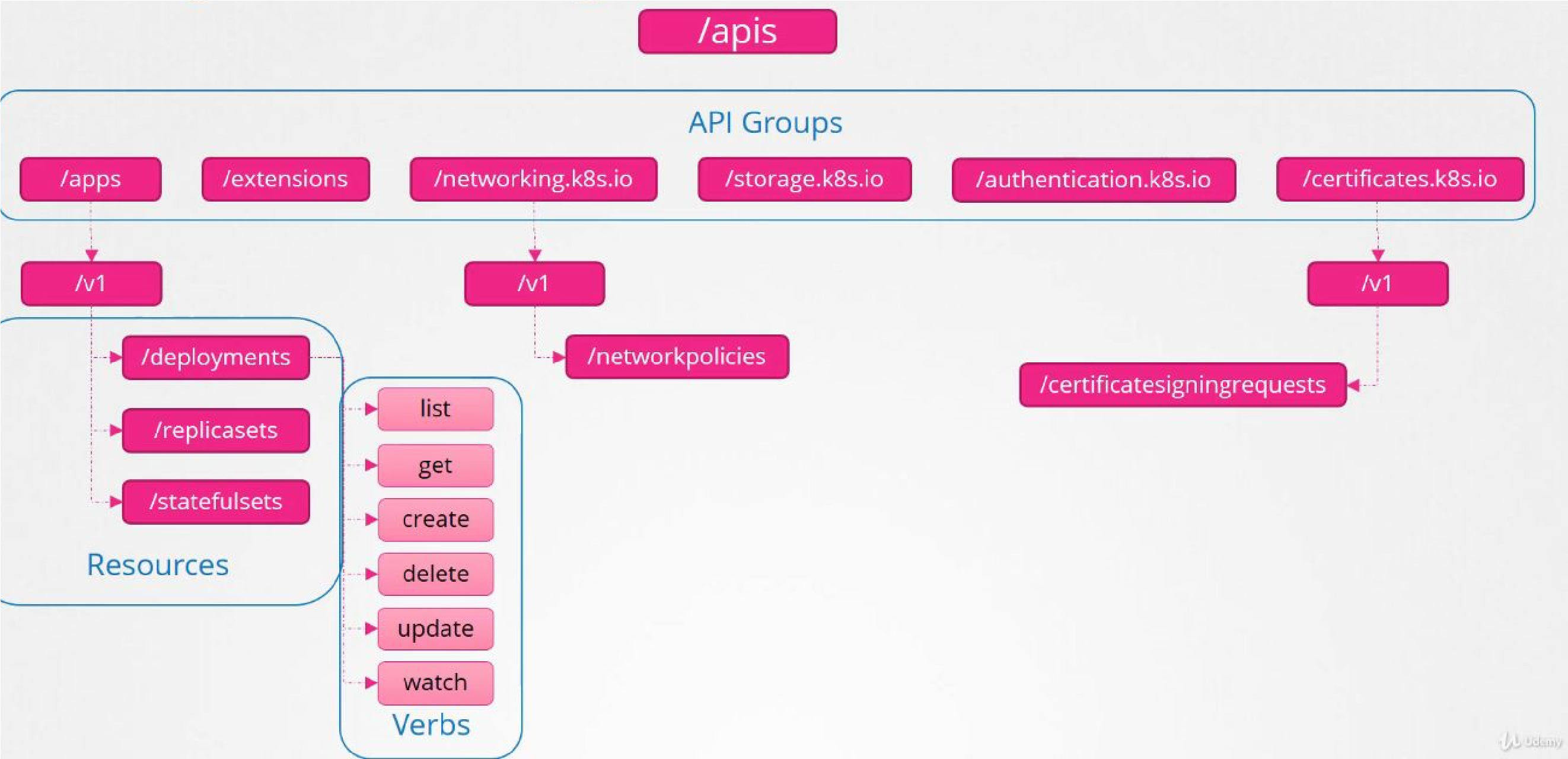
● If an attacker can run kubectl commands, they can effectively control your cluster ● Use admin.conf with certificates

## ● Set permissions to individual resources ● Define roles based on your use cases ● Was introduced in Kubernetes 1.6 and became default in 1.8

* 4 top-level types
* Role
* The rules are applicable to a single namespace
* ClusterRole
* Cluster-wide permissions representation
* RoleBinding
* Grants the (Cluster)Role to a set of Subjects inside

Namespace

* ClusterRoleBinding
* Grants the ClusterRole to a set of Subjects cluster-wide



## ● Enable it on creation time with flag --authorization-mode=RBAC ● Example Role and ClusterRole definitions

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

|  |  |  |
| --- | --- | --- |
| # This role binding allows  "rajendrait99@gmail.com" to read pods in the "default" namespace.  kind: RoleBinding apiVersion: rbac.authorization.k8s.io/v1 metadata:  name: read-pods namespace: default subjects:  - kind: User name: rajendrait99@gmail.com apiGroup: rbac.authorization.k8s.io  roleRef:  kind: Role name: pod-reader |  | # This role binding allows "drajendrait99@gmail.com to read secrets in the "development" namespace. kind: RoleBinding apiVersion: rbac.authorization.k8s.io/v1 metadata:  name: read-secrets  namespace: development # This only grants  permissions within the "development" namespace. subjects:  - kind: User name: rajendrait99@gmail.com apiGroup: rbac.authorization.k8s.io  roleRef:  kind: ClusterRole name: secret-reader apiGroup: rbac.authorization.k8s.io |

apiGroup: rbac.authorization.k8s.io

**3. Use a Network Policy**

## ● A specification of how groups of pods are allowed to communicate with each other and other network endpoints

* By default, pods are non-isolated; they accept traffic from any source
* Pods become isolated when there is a NetworkPolicy that selects them. (Others will still be non-isolated)
* As a policy types you can set *Ingress*, *Egress* or both. Defaults to *Ingress*.
* You will then define Whitelist rules to these policy types.
* Following selectors are available: *ipBlock, namespaceSelector, podSelector*

**3. Use a Network Policy**

## ● Policy examples

|  |  |  |
| --- | --- | --- |
| # Prevents all Ingress AND Egress traffic in namespace apiVersion: networking.k8s.io/v1 kind: NetworkPolicy  metadata: name: default-deny  spec:  podSelector: {} policyTypes:   * Ingress * Egress |  | # Allow access to Pods labeled "run=nginx"  # from Pods labeled"access=true"  kind: NetworkPolicy apiVersion: networking.k8s.io/v1 metadata:  name: access-nginx  spec:  podSelector:  matchLabels:  run: nginx  ingress:   * from: * podSelector:   matchLabels:  access: "true" |

**3. Use a Network Policy**

● Create a cluster with NetworkPolicy enabled using flag

--enable-network-policy

## ● Currently supports only Project Calico

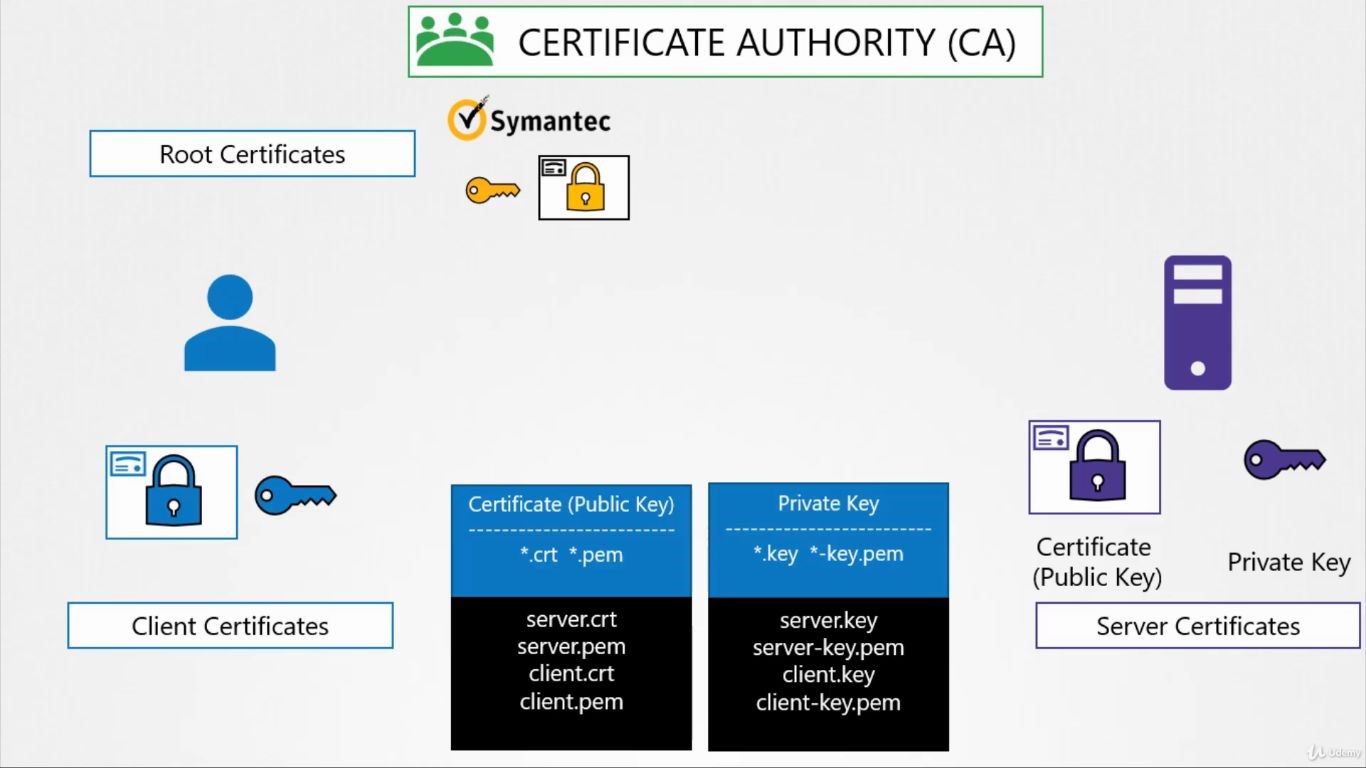
**5. Disable account token**

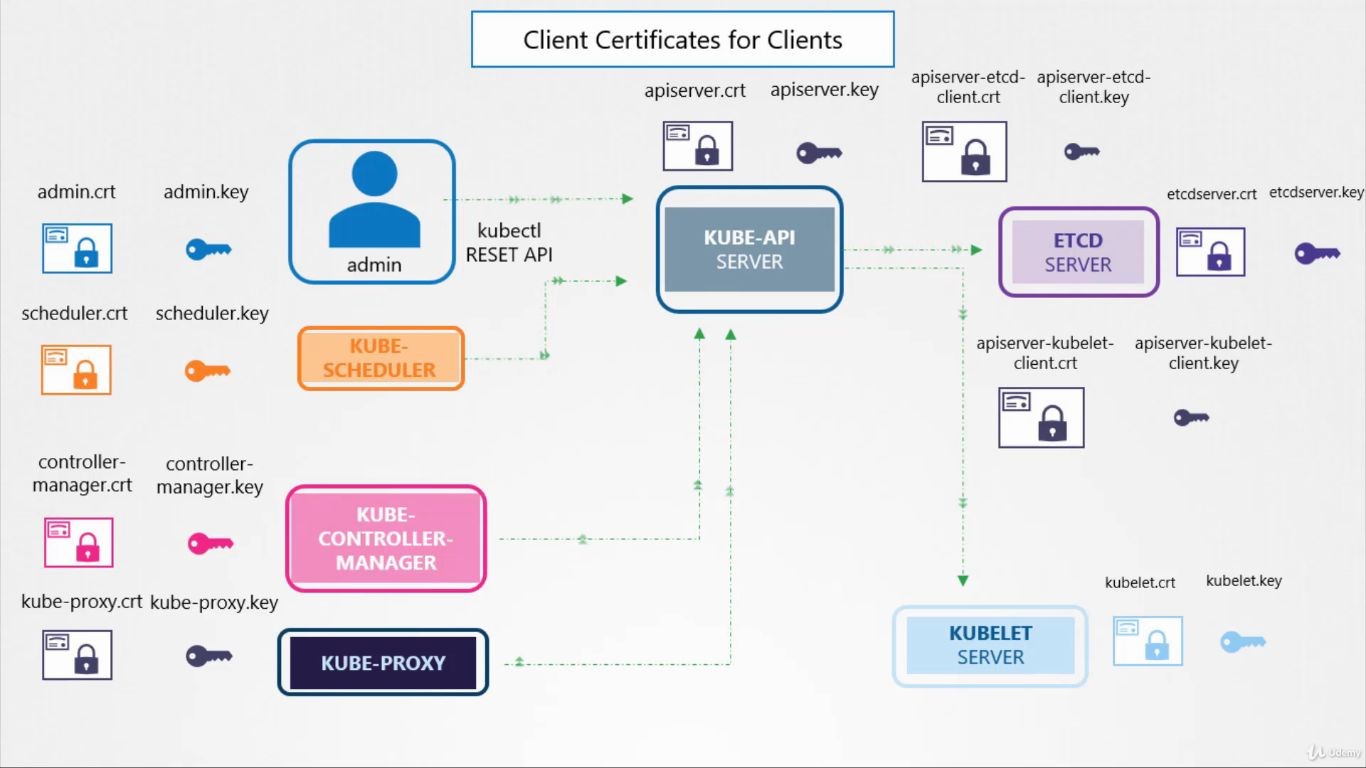
● Every namespace has a default (Kubernetes) service account

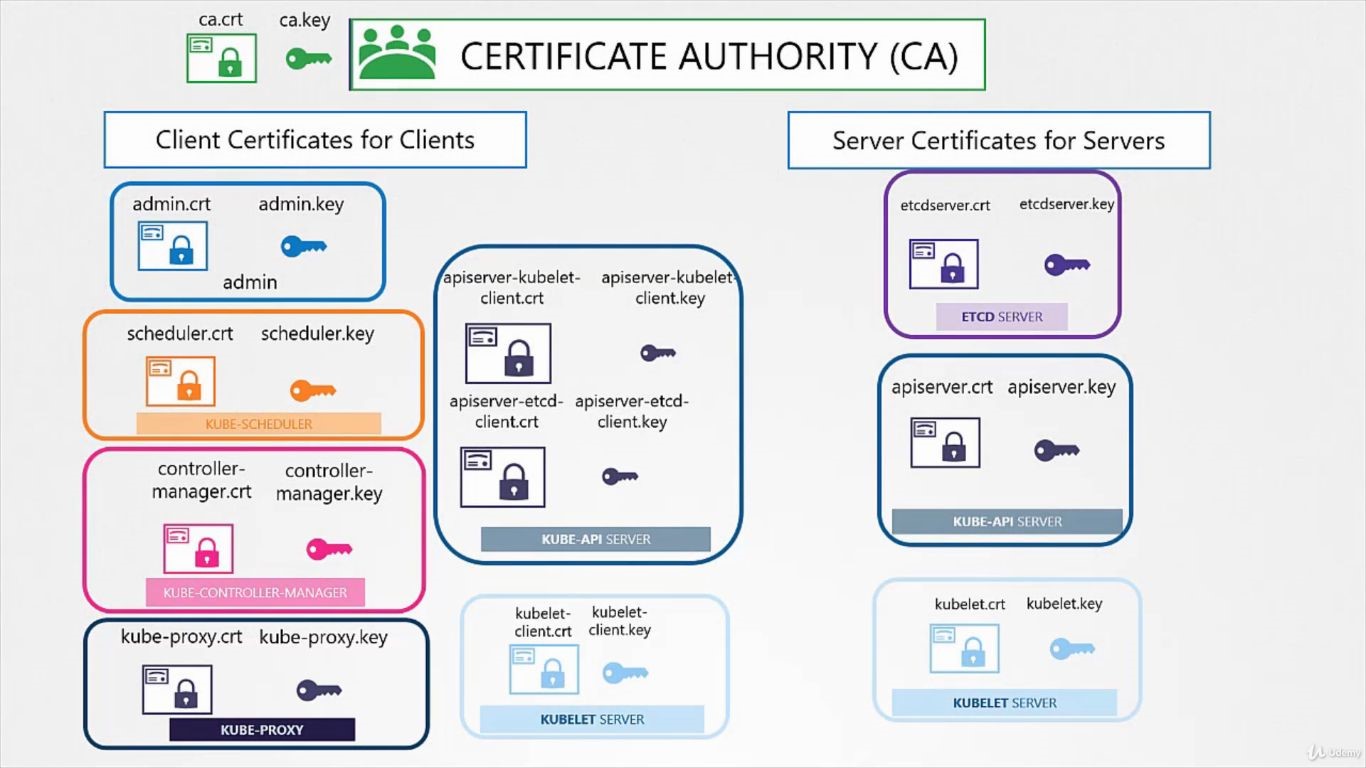
## ● Pods use service accounts to assert their identity to other workloads, including to the API server

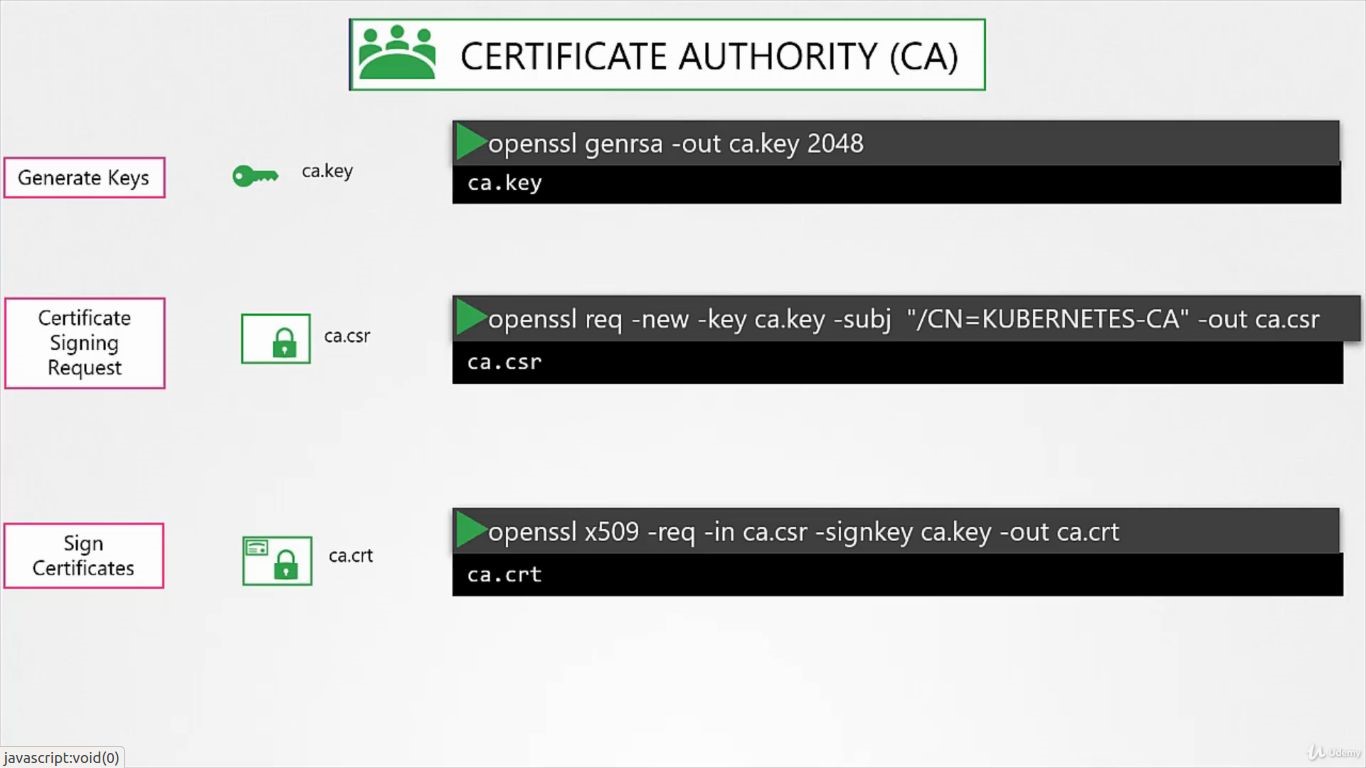
* When you create a pod, if you do not specify a service account, the pod will assign to default service account for that NS.
* The pod mounts these service account credentials, which are authorized to talk to the API server, if pod is compromised, these credentials can be used to perform arbitrary operations.

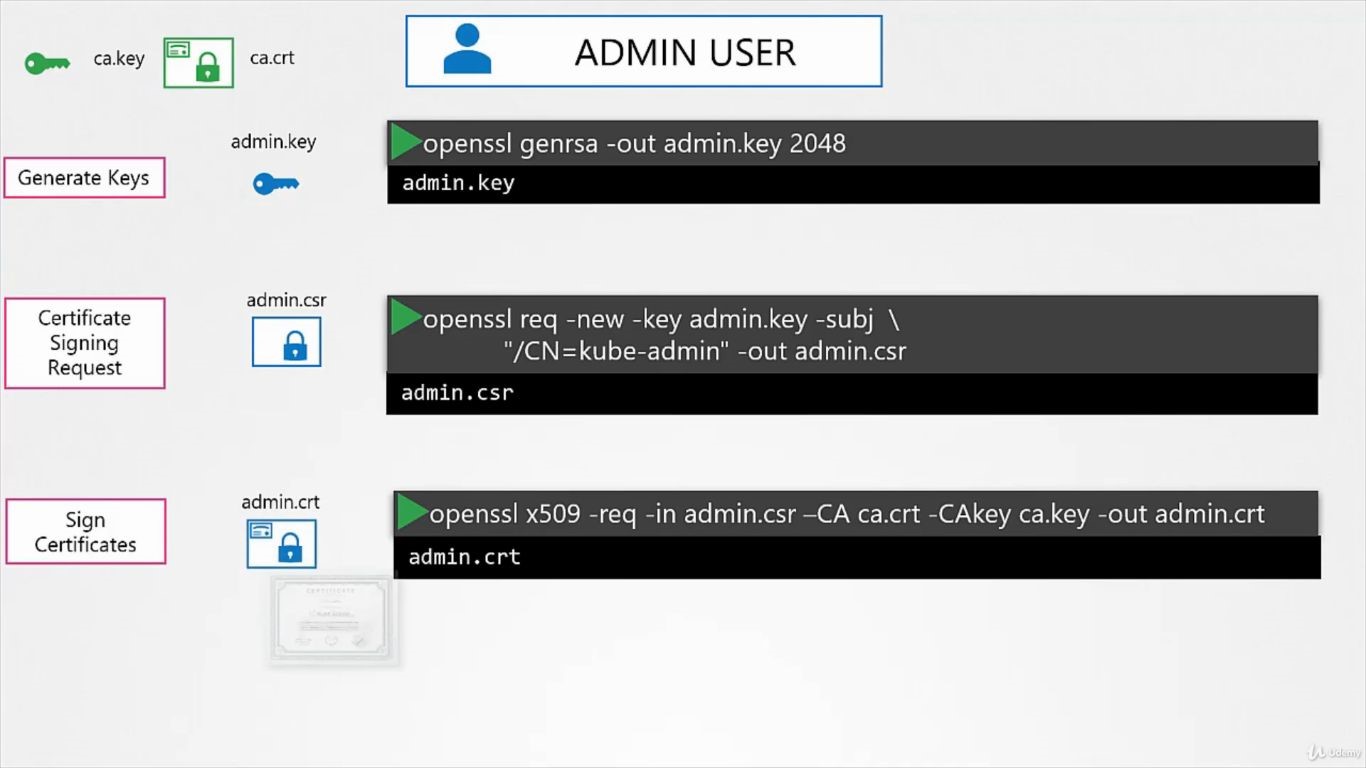
# Certificates

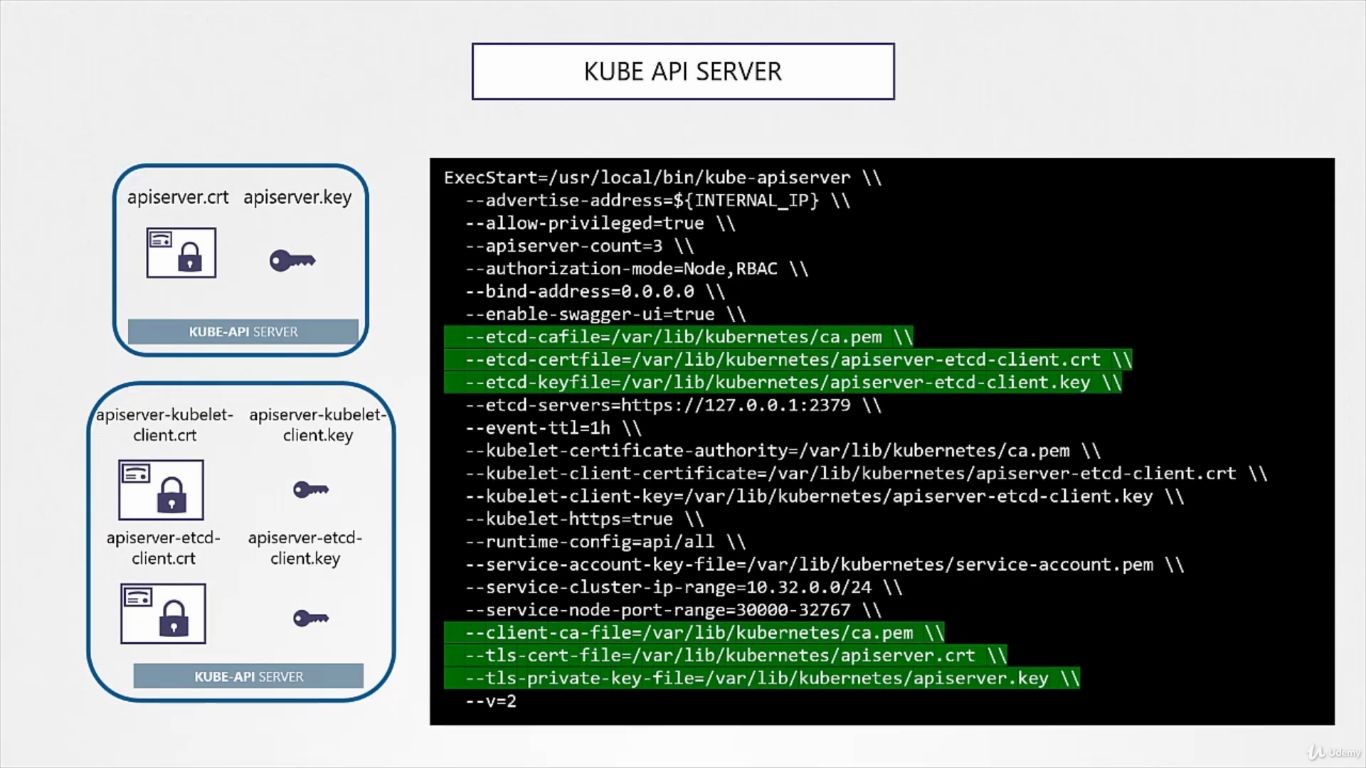




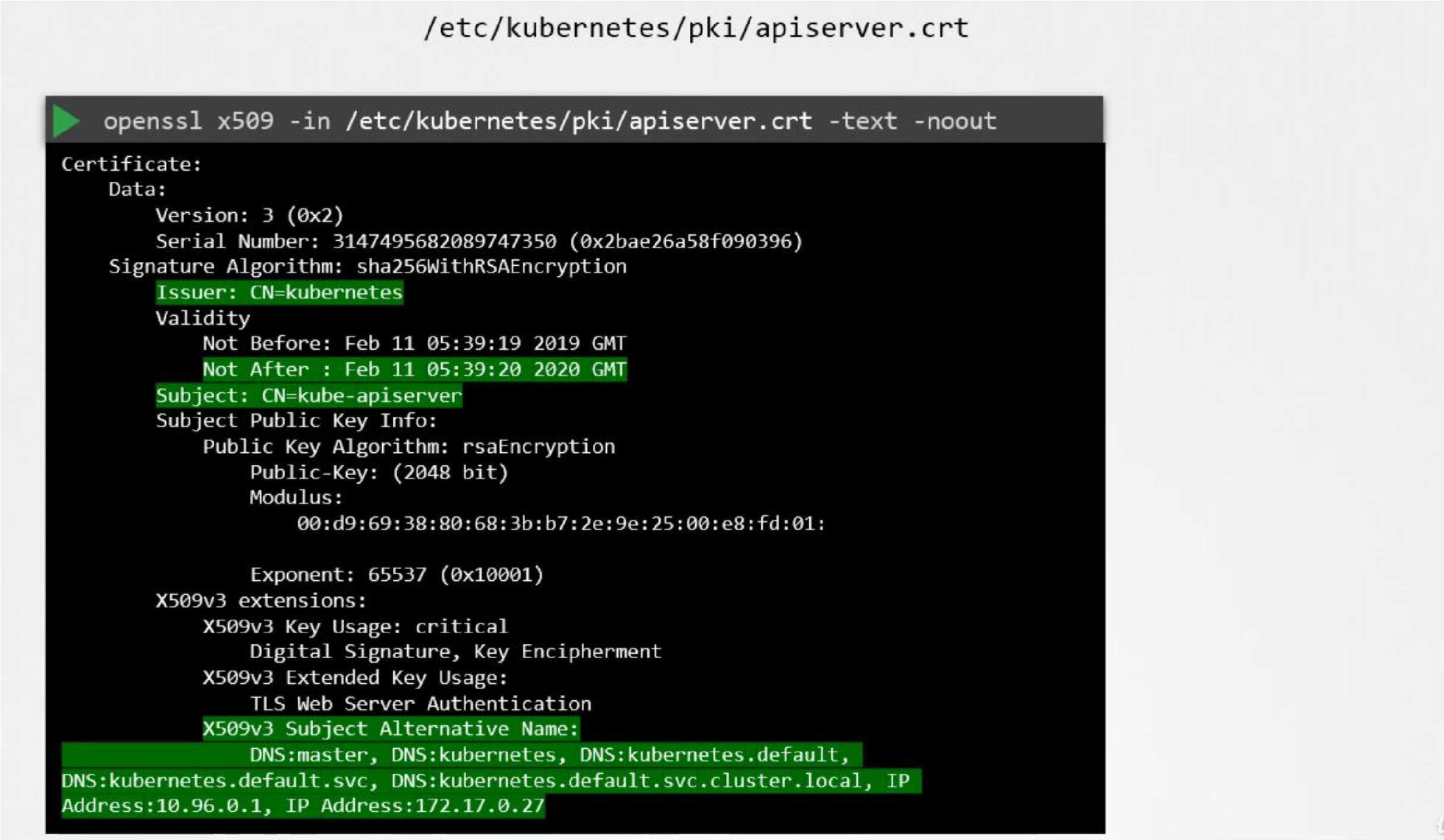








**View Kubernetes Certificates**



**OR**

**use**

**docker/kubectl**

**logs**

# KubeConfig

