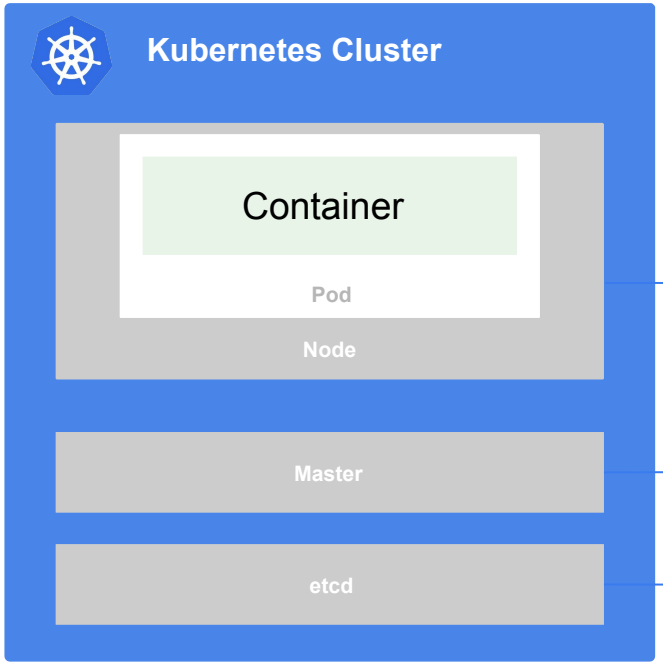
Kubernetes Security

Securing Container vs Securing VM

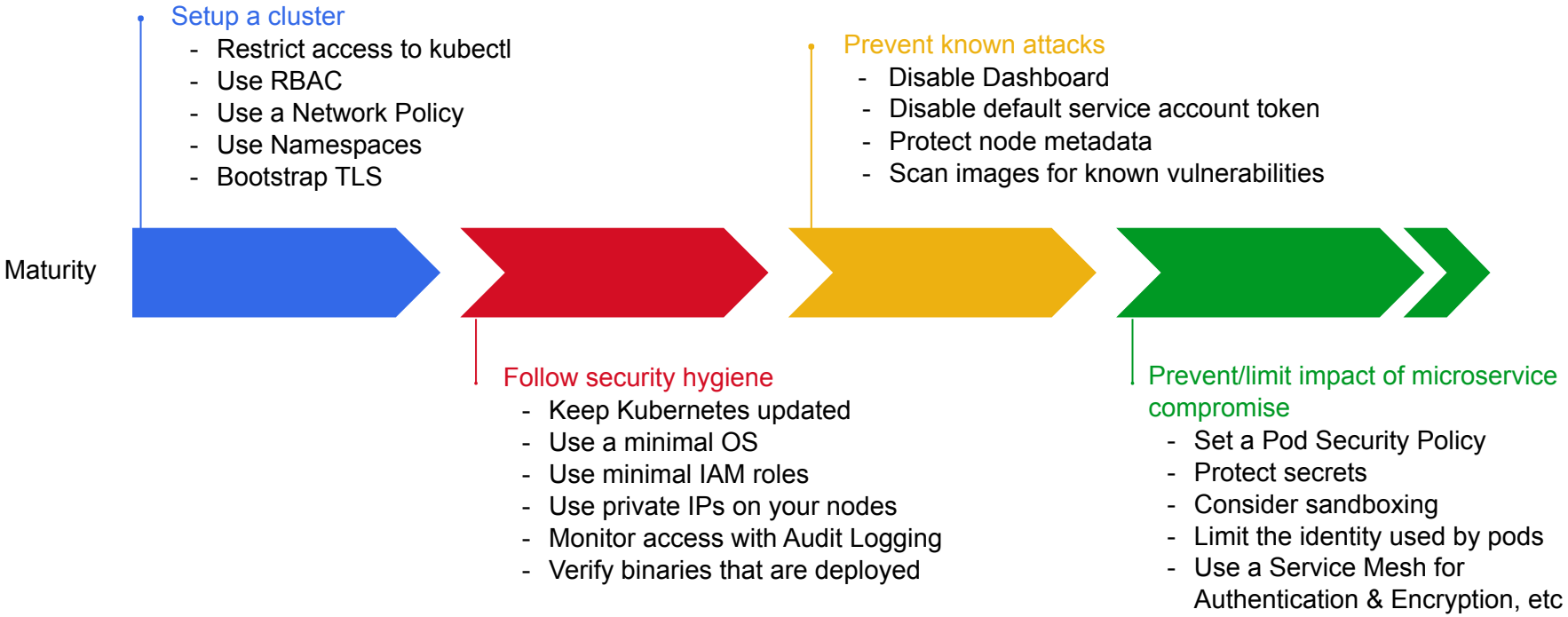
- **Surface of attack:** Minimalist host OS limits the surface of an attack
- **Resource Isolation:** Host resources are separated using namespaces and cgroups
- **Permissions:** Access controls are for app privileges and shared resources
- **Lifetime:** Containers have a shorter average lifetime

Simplified Kubernetes architecture

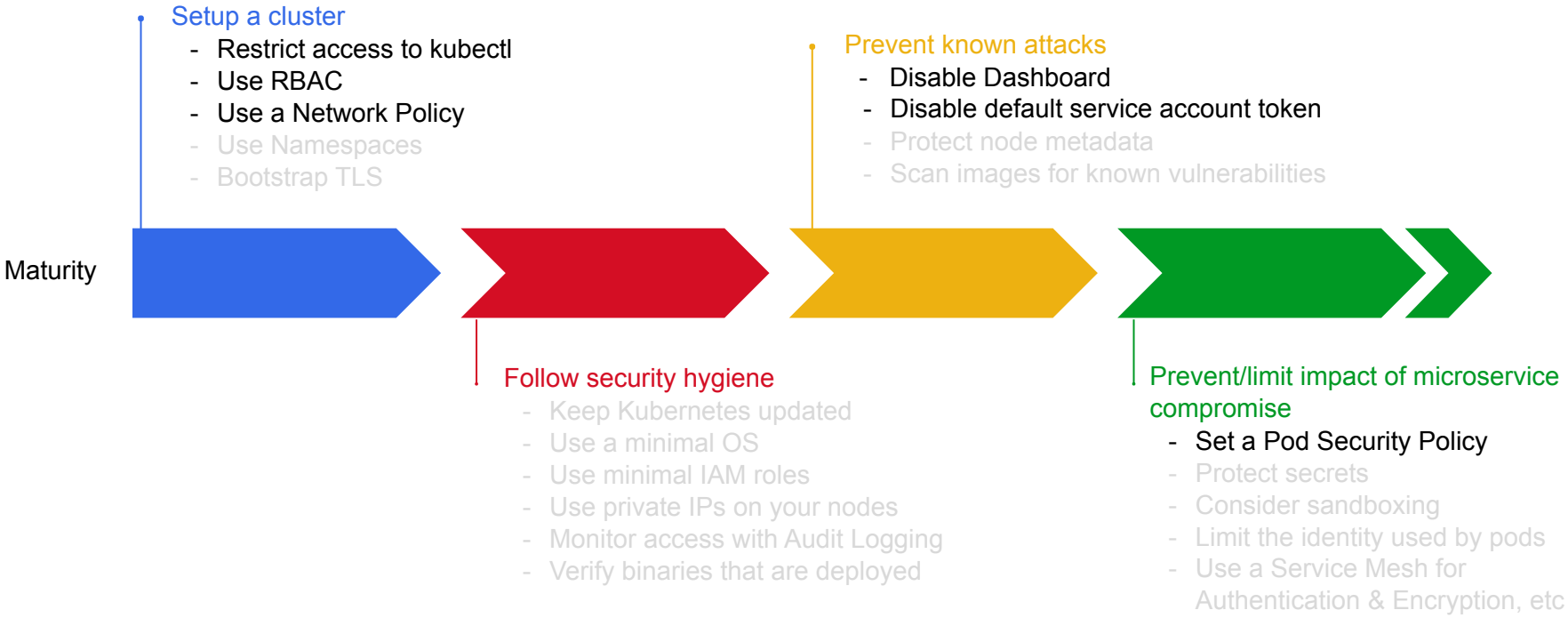


- If attacker controls a container, they might be able to escape and attack the node
- If attacker controls the Nodes, they also control the pods running on them and can potentially attack the master or abuse compute
- If attacker controls the Master, they control the cluster
- If attacker controls etcd, they can access, modify or destroy cluster

Security journey



Security journey



Getting Started

1. Restrict access to kubectl

Prevent unauthorized users from accessing your cluster

2. Use RBAC

Use role-based access control to define roles with rules containing a set of permissions

3. Use a Network Policy

Control pod to pod traffic

4. Protect Kube Dashboard

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

5. Disable account token

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

6. Use Pod Security Policy

Enable Docker seccomp and other security restrictions

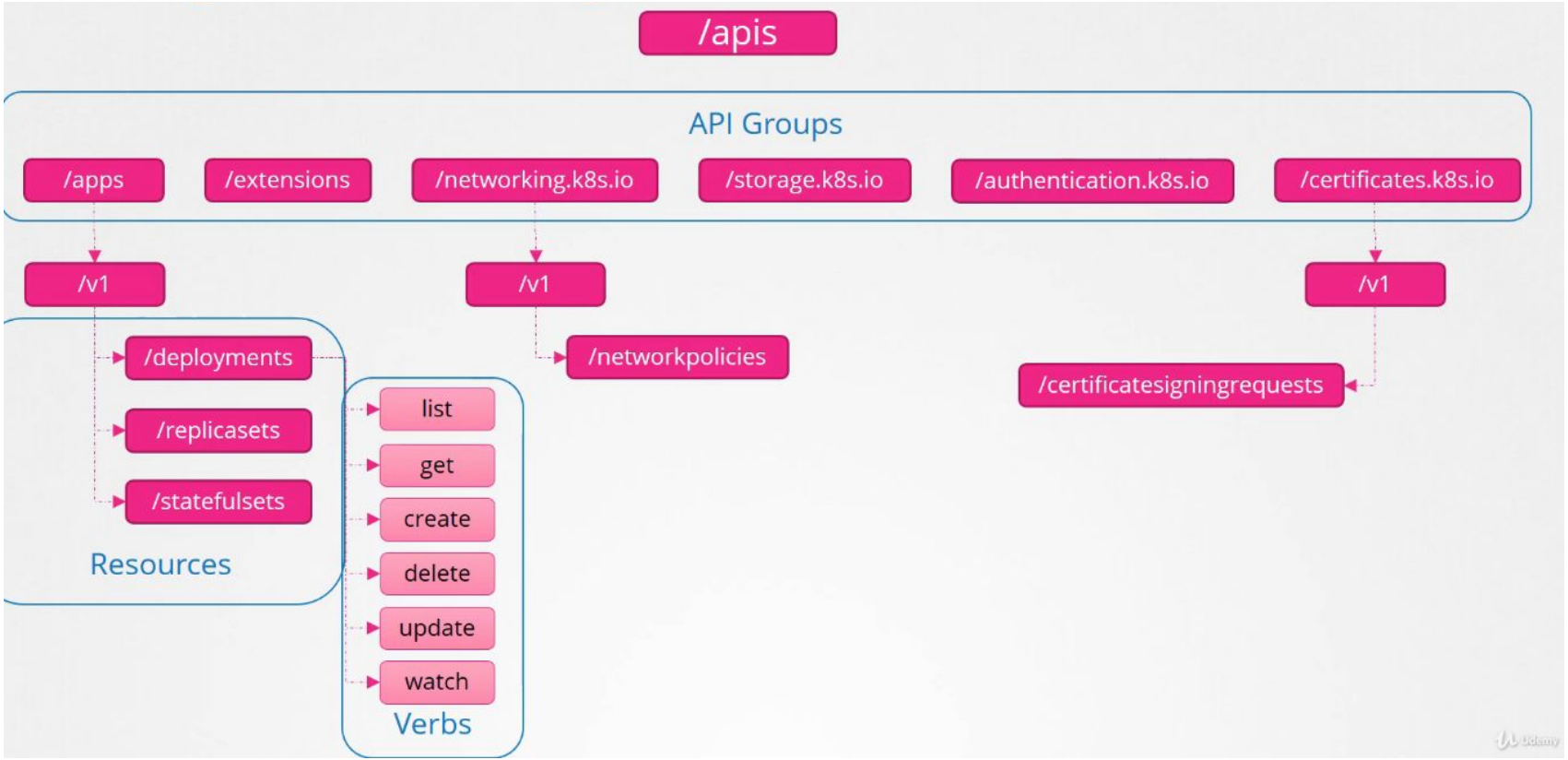
Getting Started

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

2. Use RBAC

- 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

2. Use RBAC


/apis
API Groups
/apps
/extensions
/networking.k8s.io
/storage.k8s.io
/authentication.k8s.io
/certificates.k8s.io
/v1
/v1
/v1
/deployments
/replicasets
/statefulsets
Resources
/networkpolicies
/certificatesigningrequests
list
get
create
delete
update
watch
Verbs

2. Use RBAC

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

```yaml
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"]
```

```yaml
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"]
```

2. Use RBAC

- Example RoleBinding and ClusterRoleBinding definitions

```yaml
# 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
  apiGroup: rbac.authorization.k8s.io
```

```yaml
# 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
```

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

```yaml
# Prevents all Ingress AND Egress traffic in
namespace
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy

metadata:
  name: default-deny
spec:
  podSelector: {}
  policyTypes:

  - Ingress
  - Egress
```

```yaml
# 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"
```

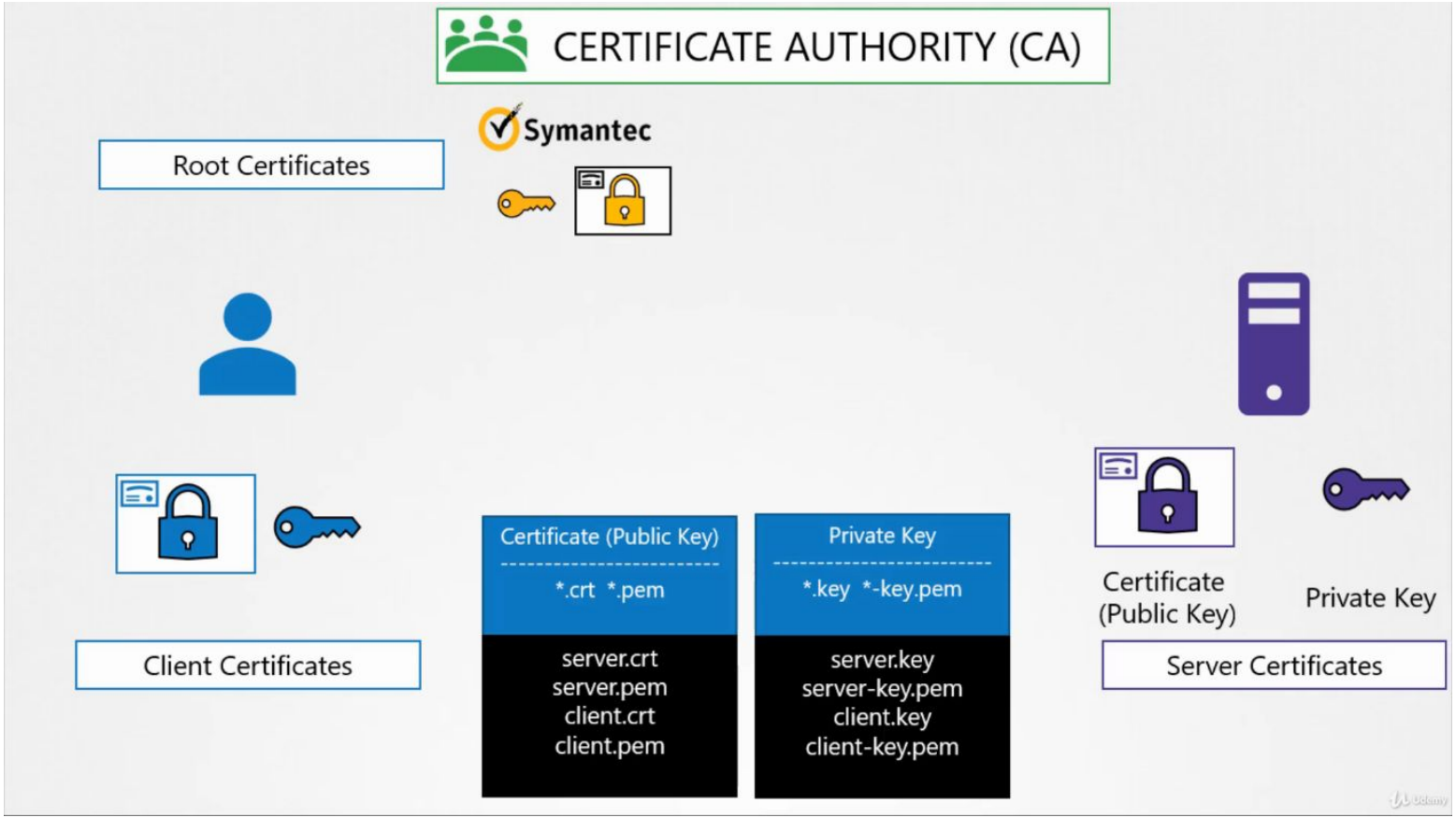
3. Use a Network Policy

- Create a cluster with NetworkPolicy enabled using flag --enable-network-policy
- Currently supports only Project Calico

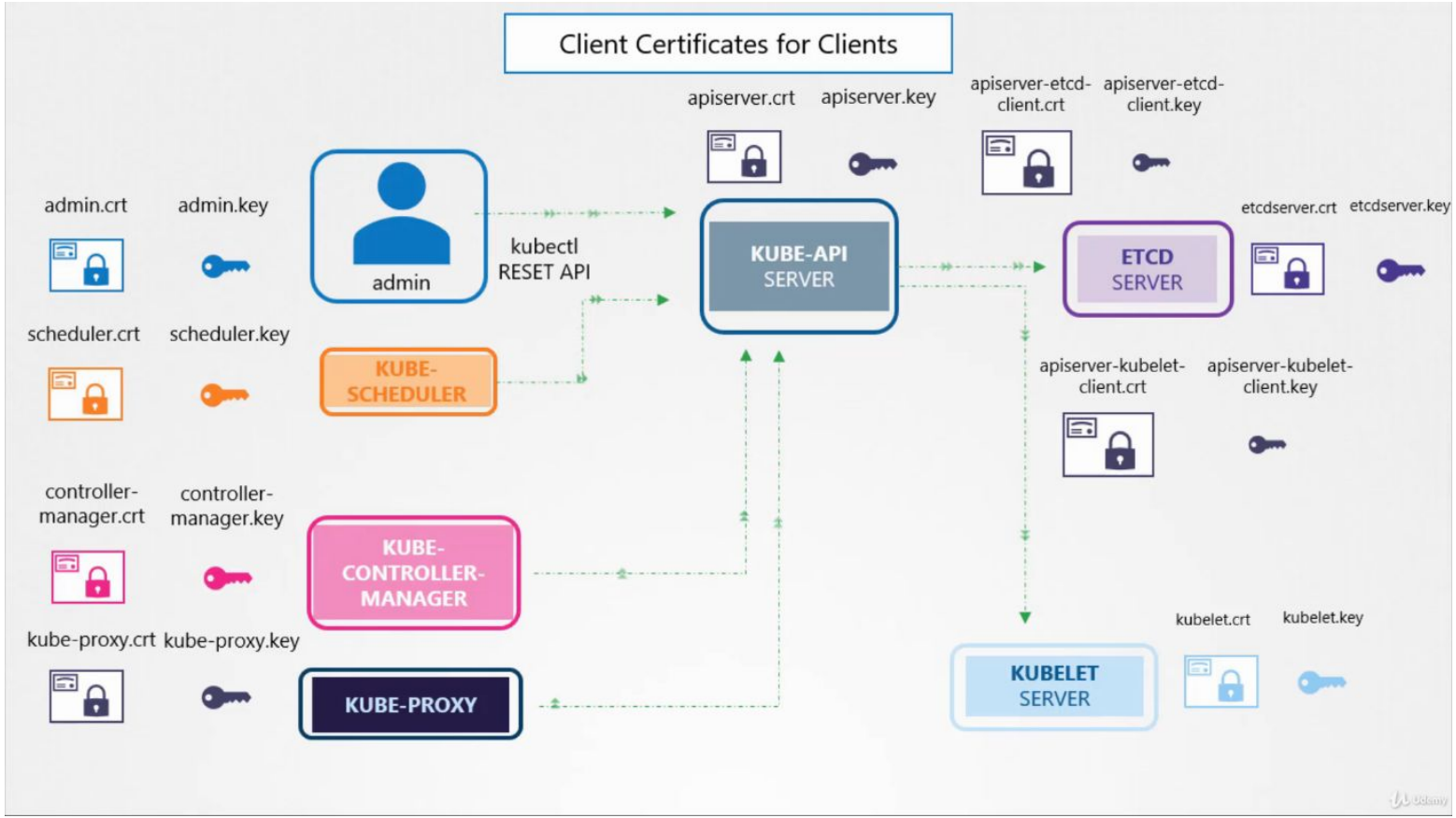
5. Disable account token

- 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.

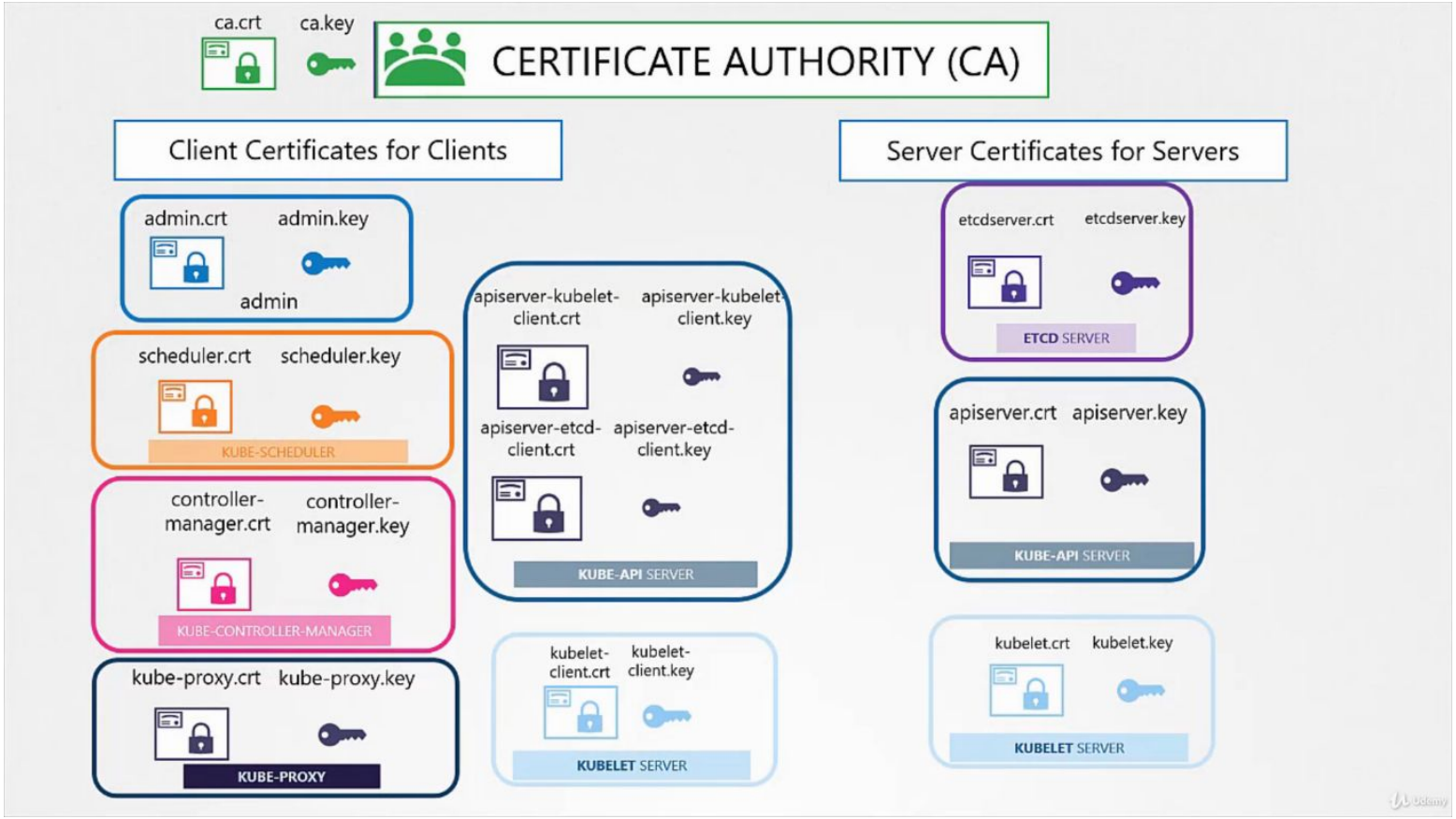
Certificates



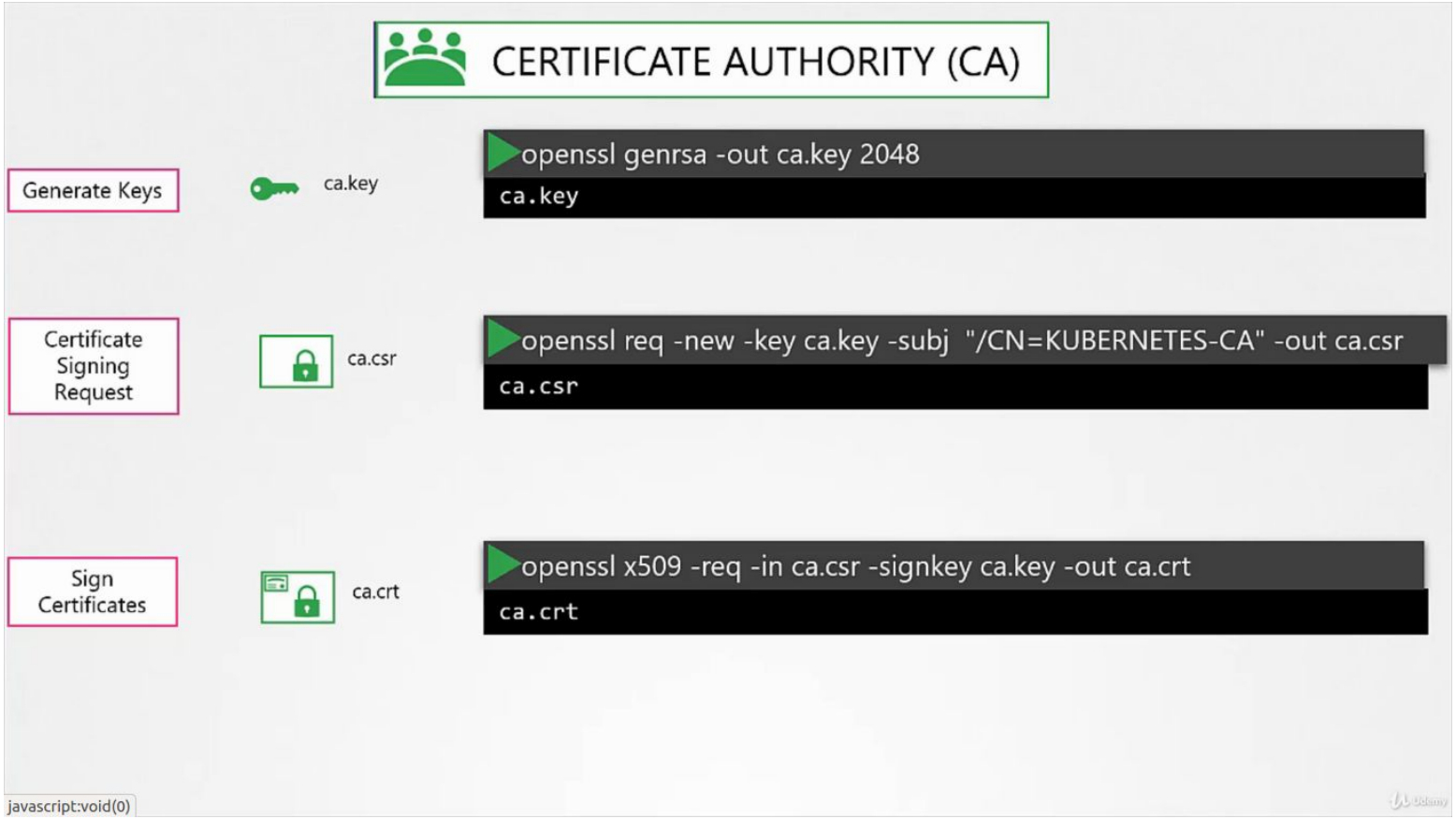
Kubernetes Certificates



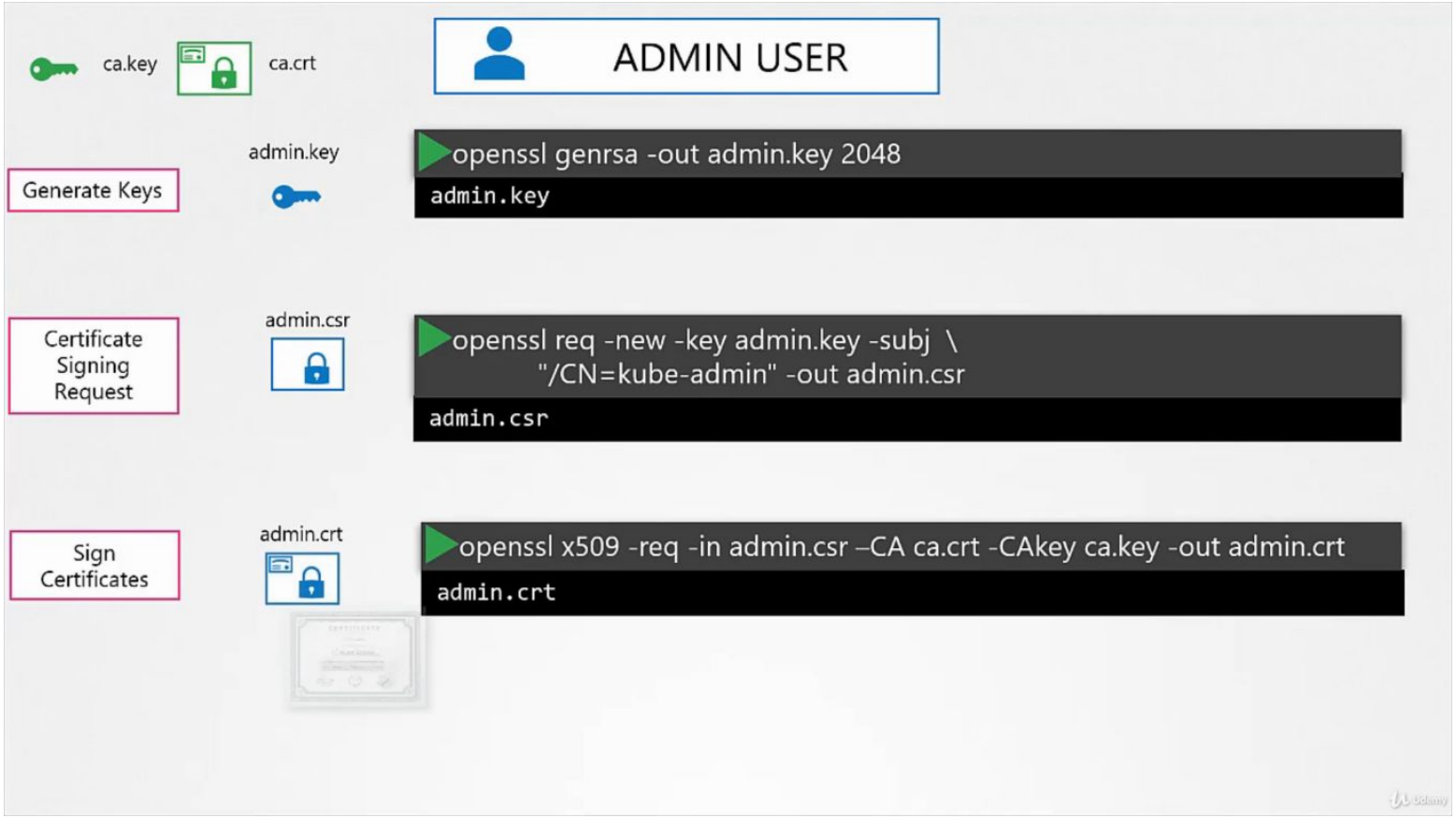
Kubernetes Certificates



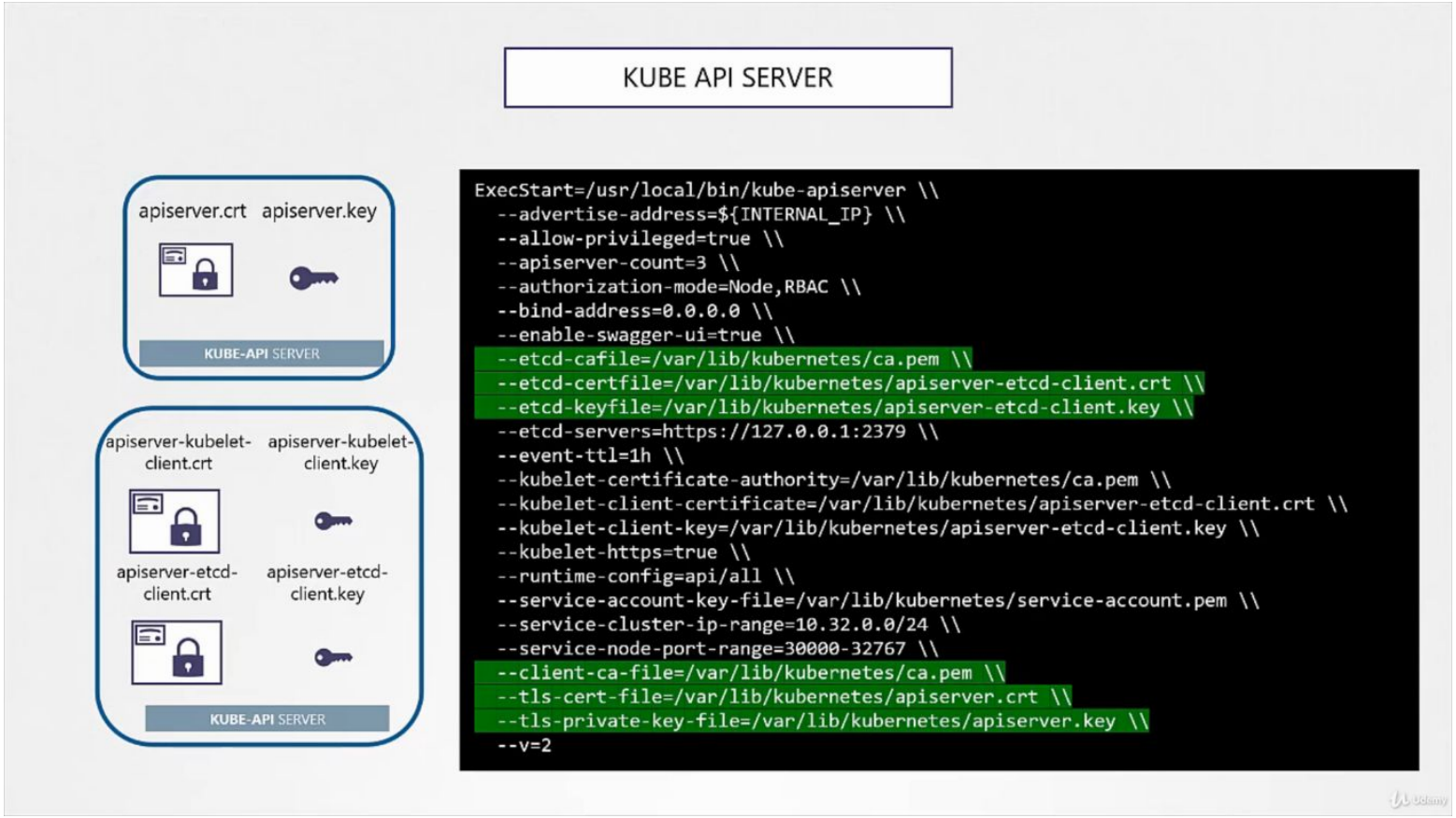
Kubernetes Certificates



Kubernetes Certificates



Kubernetes Certificates



View Kubernetes Certificates

/etc/kubernetes/pki/apiserver.crt

```
openssl x509 -in /etc/kubernetes/pki/apiserver.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 3147495682089747350 (0x2bae26a58f090396)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=kubernetes
        Validity
            Not Before: Feb 11 05:39:19 2019 GMT
            Not After : Feb 11 05:39:20 2020 GMT
        Subject: CN=kube-apiserver
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:d9:69:38:80:68:3b:b7:2e:9e:25:00:e8:fd:01:

                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
            X509v3 Subject Alternative Name:
                DNS:master, DNS:kubernetes, DNS:kubernetes.default,
DNS:kubernetes.default.svc, DNS:kubernetes.default.svc.cluster.local, IP
Address:10.96.0.1, IP Address:172.17.0.27
```

OR

use docker/kubectl logs

KubeConfig

```yaml
apiVersion: v1
kind: Config

clusters:
- name: my-kube-playground
  cluster:
    certificate-authority: ca.crt
    server: https://my-kube-playground:6443

contexts:
- name: my-kube-admin@my-kube-playground
  context:
    cluster: my-kube-playground
    user: my-kube-admin

users:
- name: my-kube-admin
  user:
    client-certificate: admin.crt
    client-key: admin.key
```