



# Kubernetes Security Best Practices

...

ianmlewis@

# Ian Lewis

- @IanMLewis
-  Google Cloud
-  Tokyo, Japan
- #kubernetes, #go, #python



# Kubernetes

- Container  
Orchestrator
- An operations  
framework



# Topics

- Security 101
- Runtime Security
- Host Security
- Network Security
- Threat detection
- Build Hygiene
- Image Hygiene
- SecOps



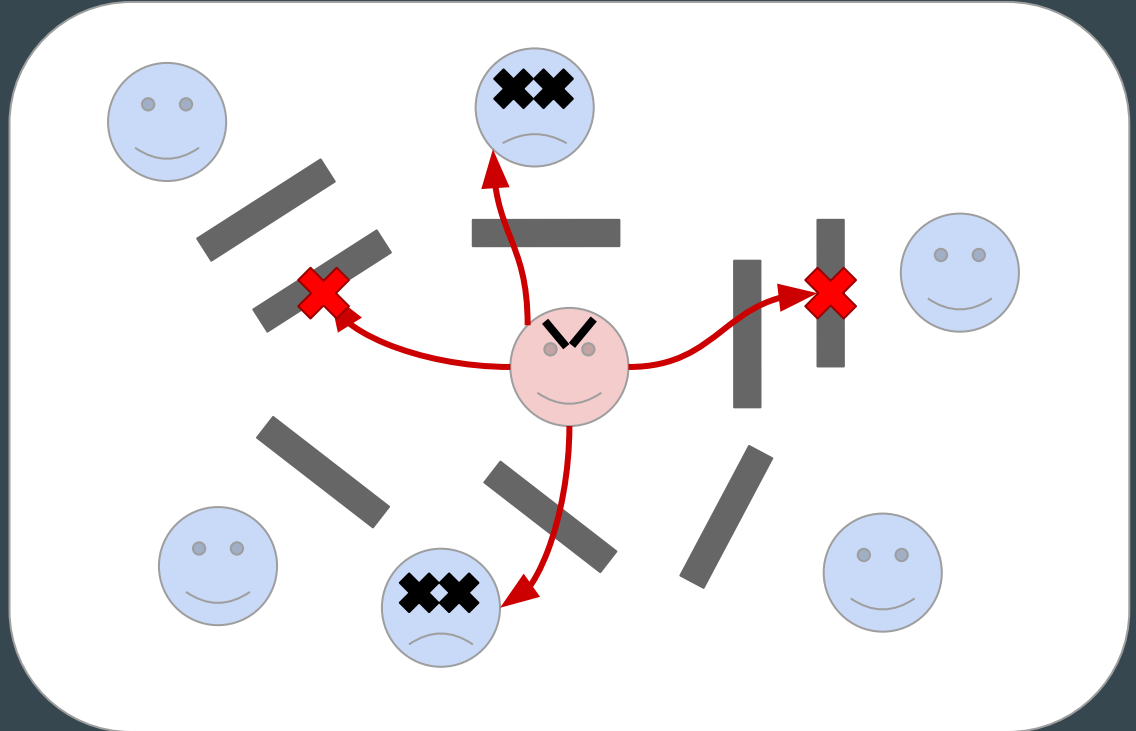
# Topics

- ✓ Security 101
- ✓ Runtime Security
- ✓ Host Security
- ✓ Network Security
- ✗ Threat detection
- ✗ Build Hygiene
- ✗ Image Hygiene
- ✗ SecOps



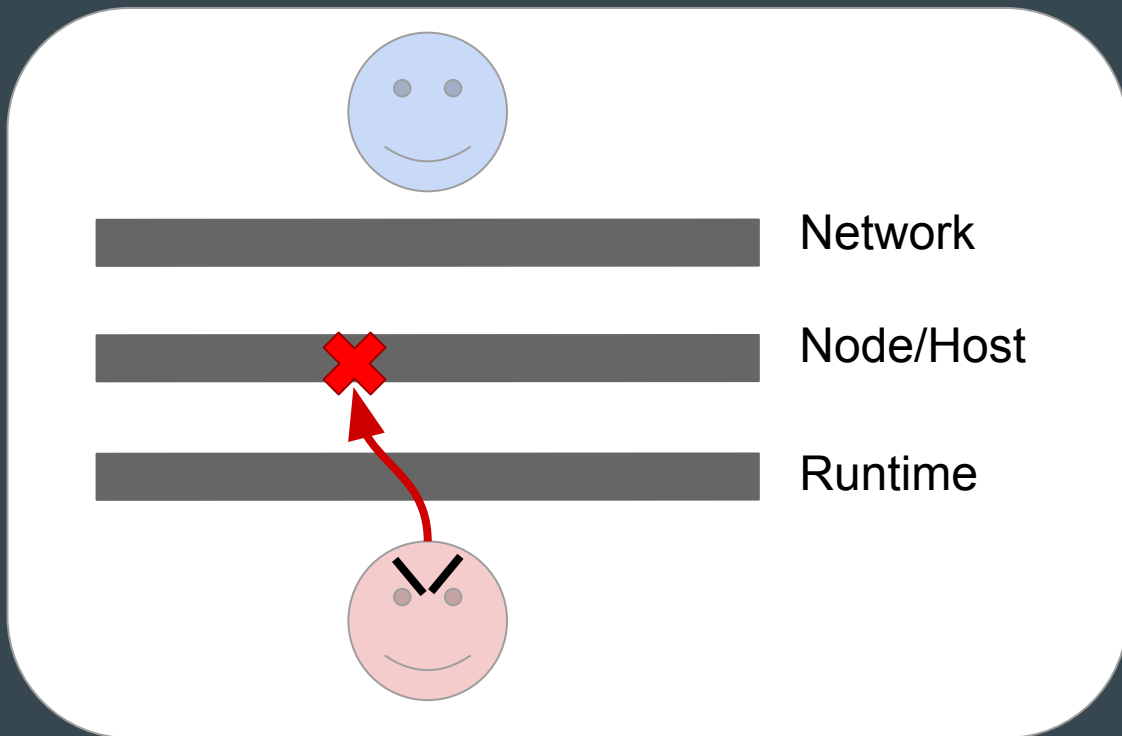
# Security 101

- Security is a spectrum
- Attackers can pick their targets
- Provide as many hurdles between the threat and asset
- Attackers can shift focus. "It doesn't matter how many locks are on your door if your window is open"



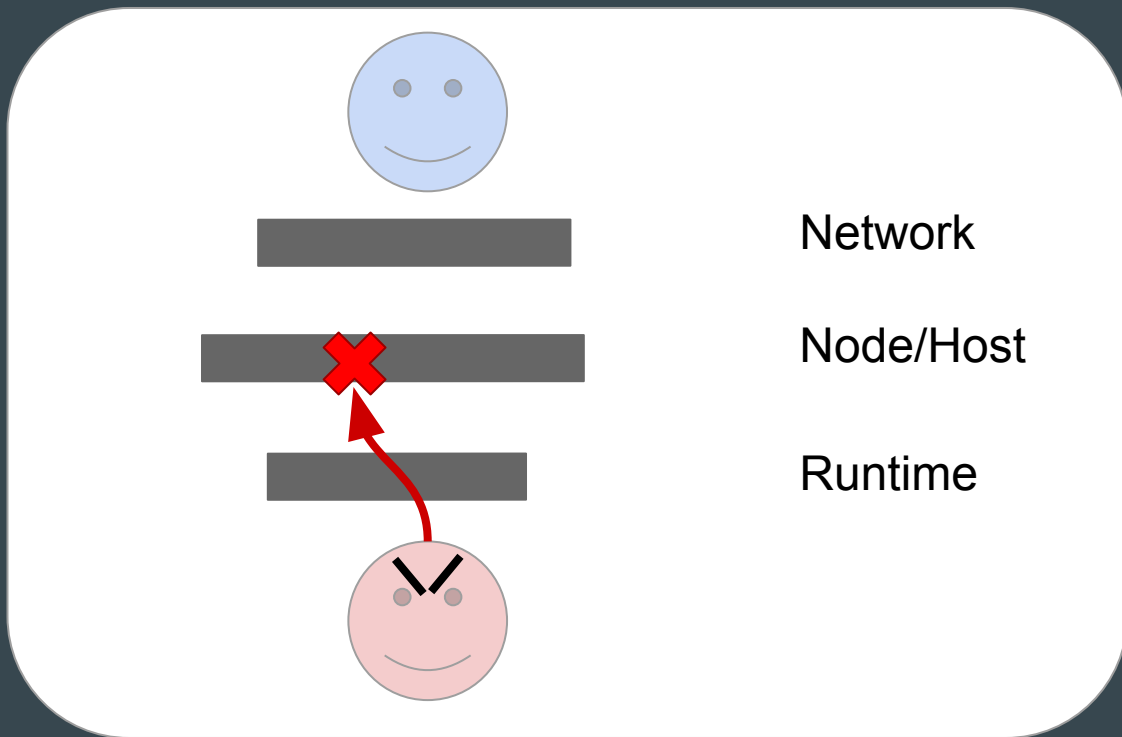
# Security 101

- Layered Security/Defence in Depth
- Good security is redundant (not DRY)



# Security 101

- Limit the attack surface





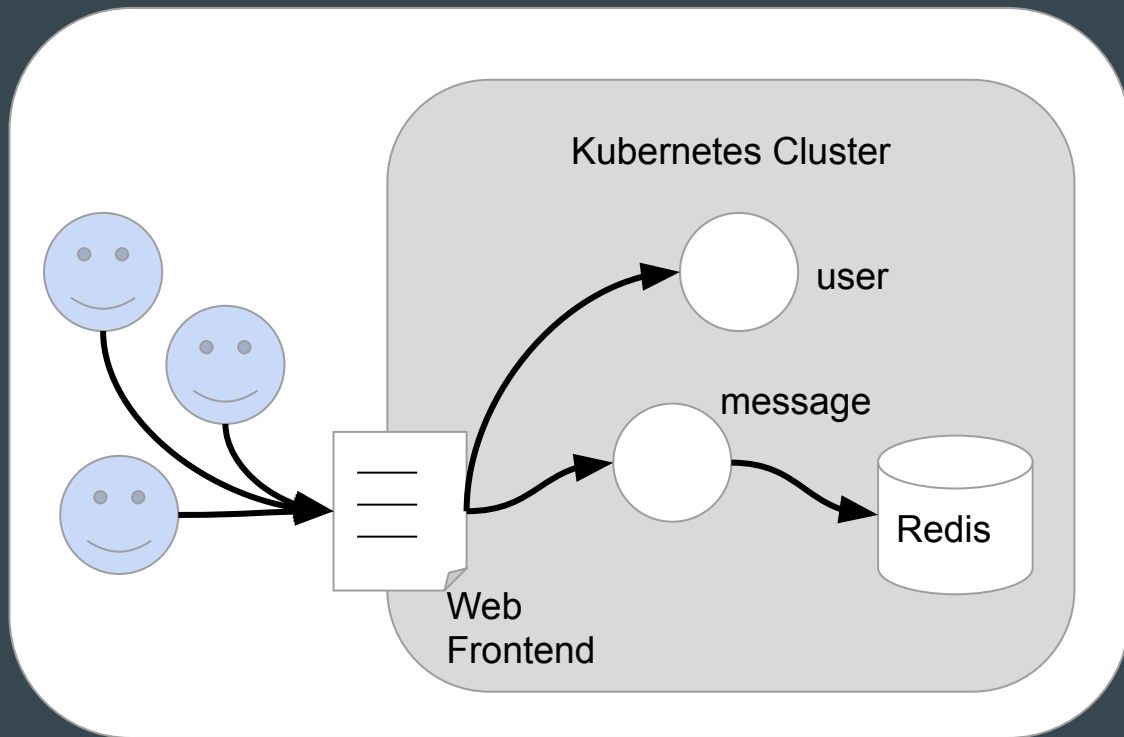
# Security 101

- Least Privilege
- Only give as much permission/privilege as is absolutely necessary



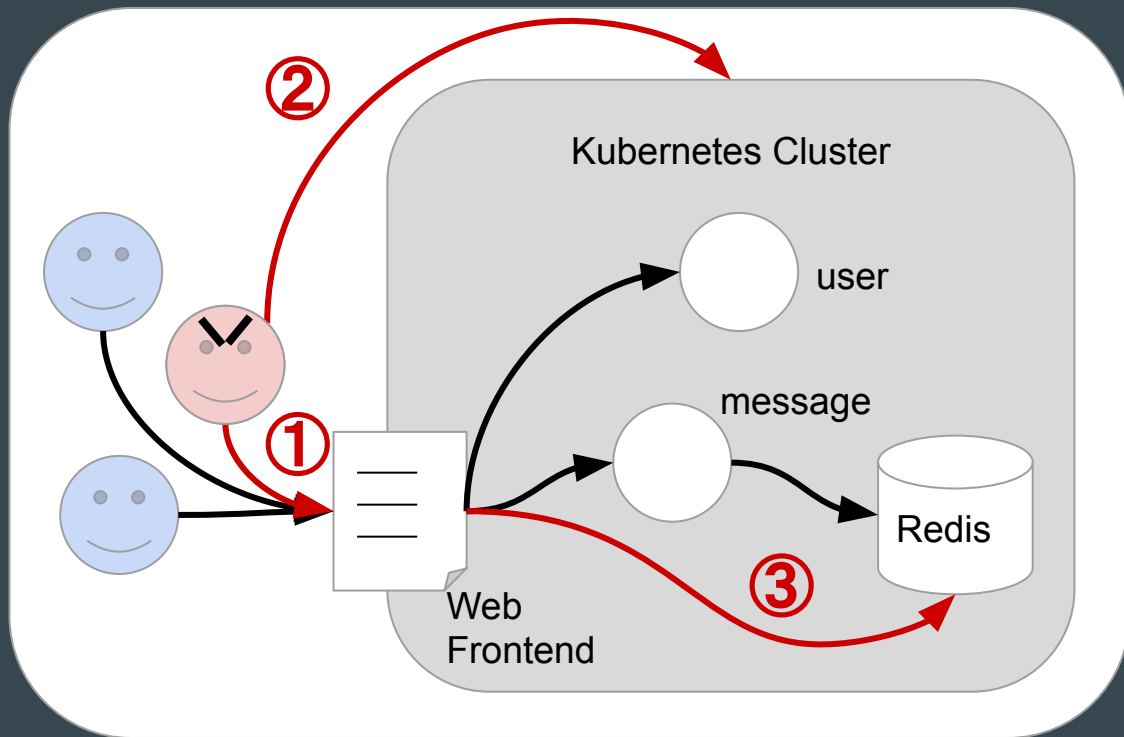
# Guestbook app

- Frontend
  - Serves web traffic
- Message
  - Stores/lists messages
- User
  - Authentication



# Kubernetes API Server

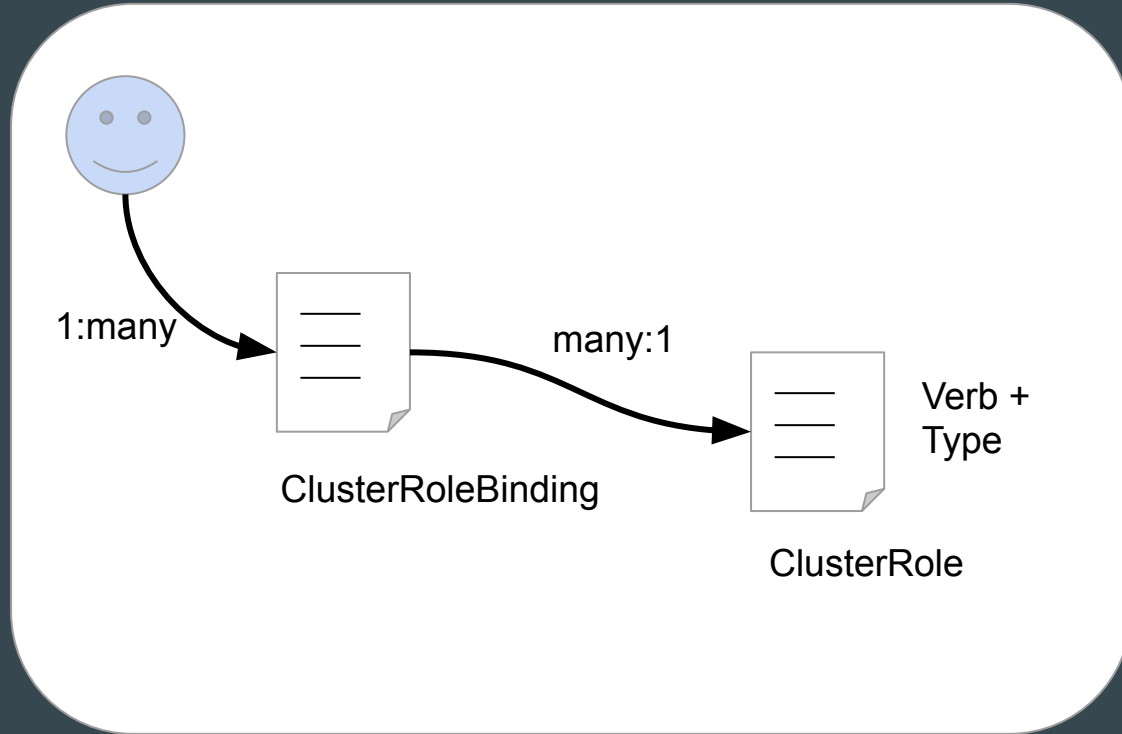
1. Get token from frontend Pod
2. Use token to attack cluster API server
3. Get secrets etc. to further attack



# Mitigate 1 & 2: RBAC

- Role Based Access Control
- Roles given to users
- Each role has permission to perform some operation
  - get secrets
  - update configmap
  - etc.
- RBAC settings apply to namespace

# Mitigate 1 & 2: RBAC



# Mitigate 2: API Server Firewall

- Restrict access to API server to certain IP addresses.
- GKE:
  - `gcloud container clusters create`  
`--enable-master-authorized-networks`  
`--master-authorized-networks=....`

# Mitigate 3: Network Policy

- Restrict access to database to only the Pods that require it
- Specify access via labels
- Implemented by Network solution: Calico, Weave, etc.

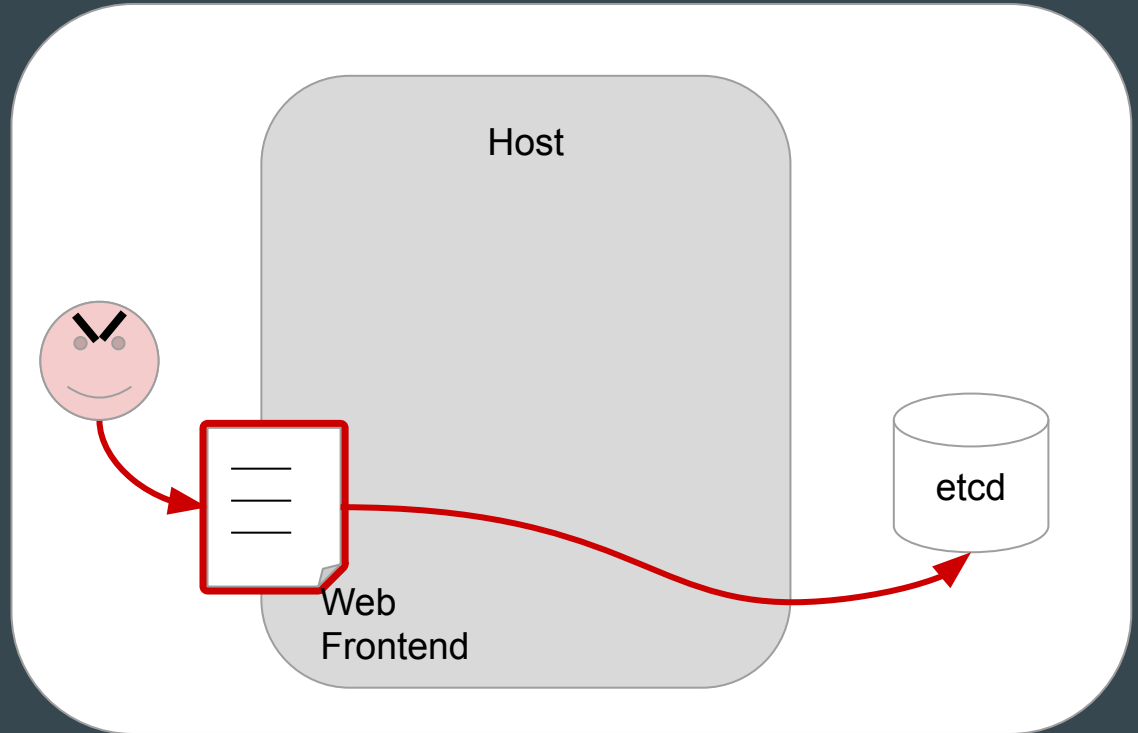
# NetworkPolicy

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: redis
spec:
  podSelector:
    matchLabels:
      name: redis
  ingress:
    - from:
      - podSelector:
          matchLabels:
            name: message
```



# Get access to cluster components

1. Manipulate cluster data in etcd

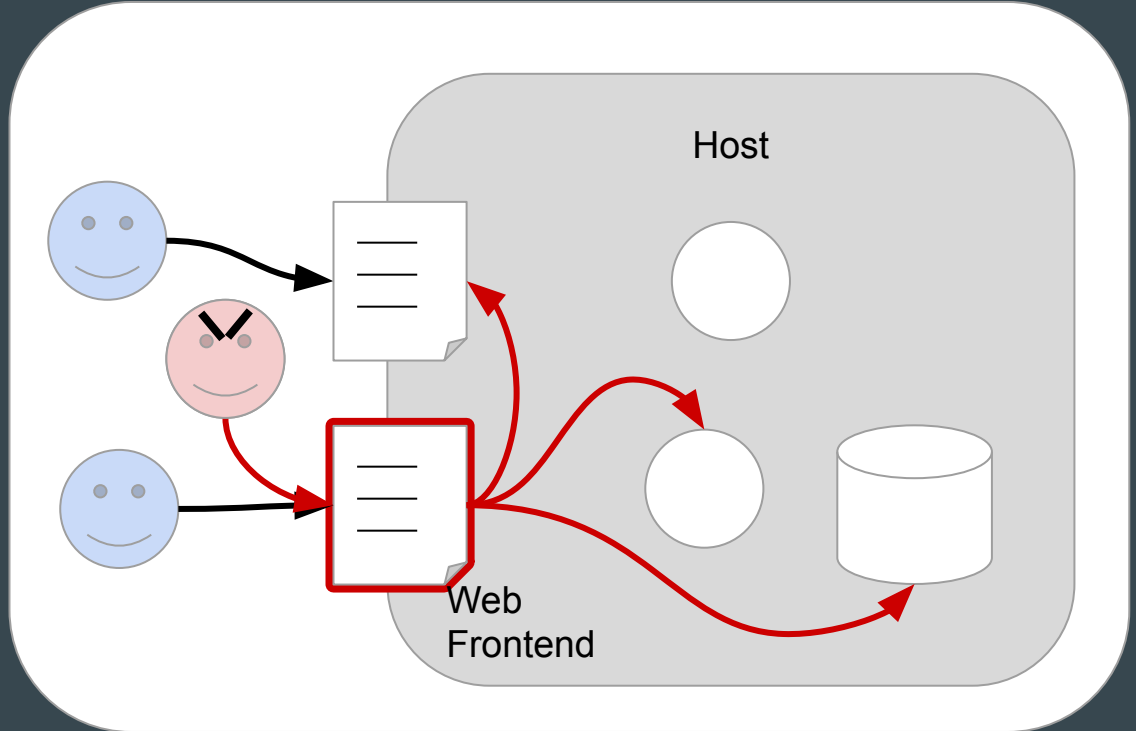


# Mitigate 1: Secure etcd

- Use authentication and firewalls to restrict access to etcd
- Encrypt data in etcd (encryption at rest)

# Get access to host

1. Break out of the container using container or kernel exploits
2. Attack the Kubelet
3. Attack other containers running on the same host



# Mitigate 1: Run as non-root

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
```

# Mitigate 1: Read only root filesystem

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    readOnlyRootFilesystem: true
```

# Mitigate 1: no\_new\_privs

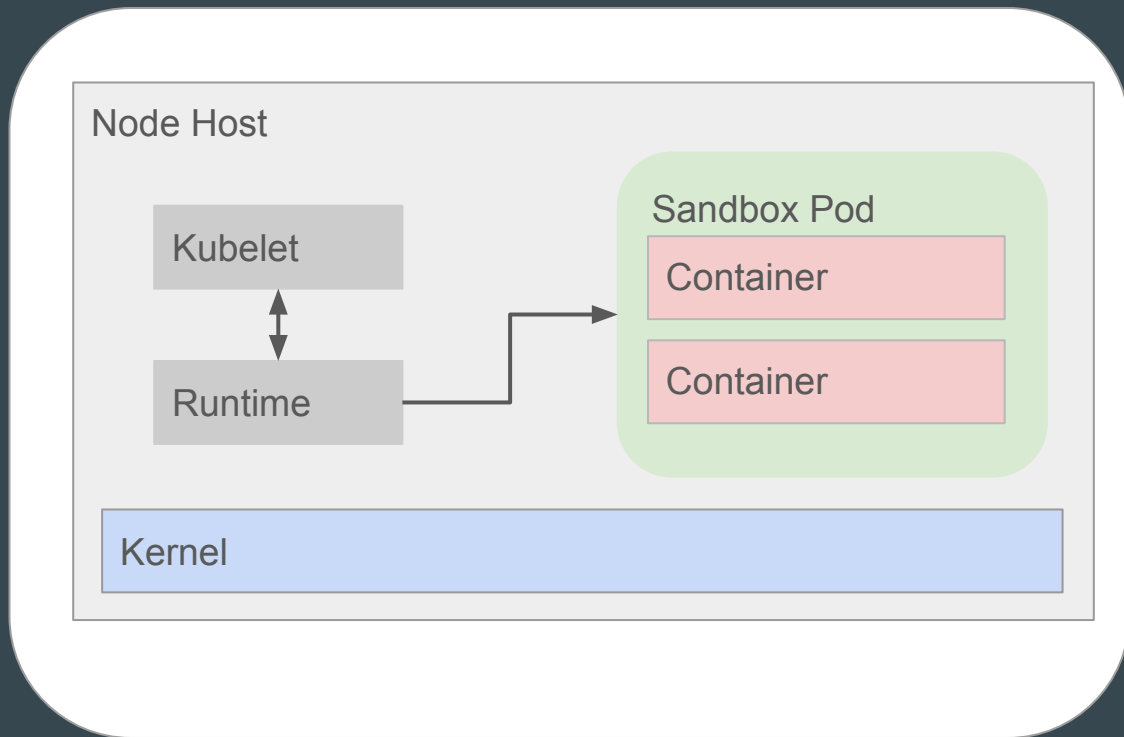
```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    allowPrivilegeEscalation: false
```

# Mitigate 1: Do them all

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    readOnlyRootFilesystem: true
    allowPrivilegeEscalation: false
```

# Mitigate 1: Sandboxed Pods

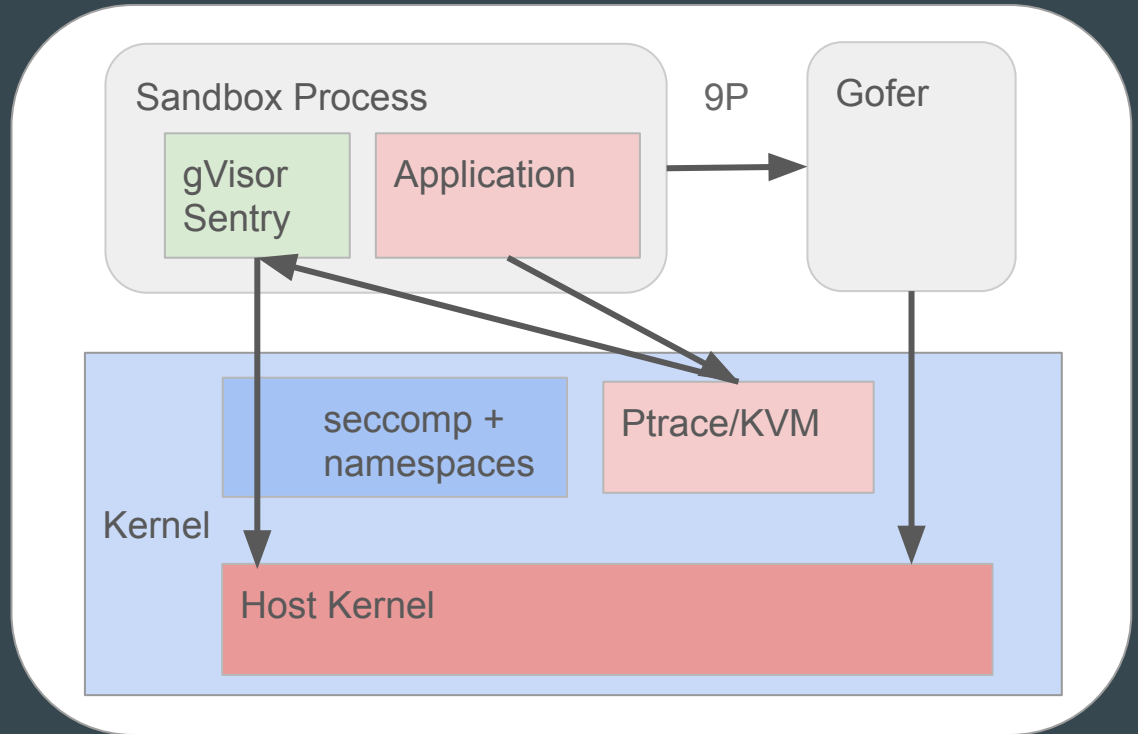
1. Pods are sandboxed from other Pods on the same host
2. Sandbox provides 2 layers of isolation: Sandbox + Container (Linux Kernel)
3. Examples: kata containers, gVisor







1. User space kernel
2. Intercepts and implements syscalls in userspace
3. Sandbox has low capabilities and runs with restricted seccomp filters



Mitigate 1:  
seccomp/  
AppArmor/  
SELinux



Your App

Mitigate 1:  
seccomp/  
AppArmor/  
SELinux



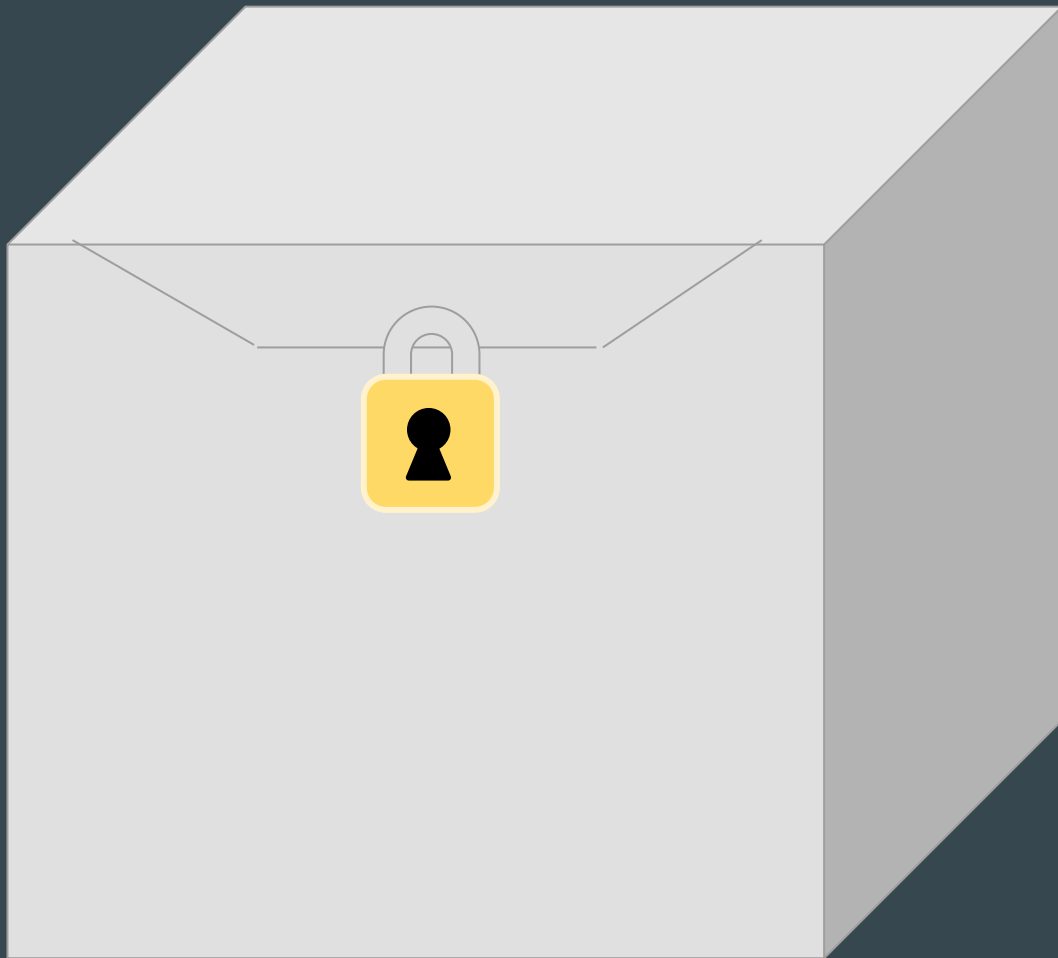
Container

Mitigate 1:  
seccomp/  
AppArmor/  
SELinux



seccomp

**Mitigate 1:**  
**seccomp/  
AppArmor/  
SELinux**



**AppArmor/  
SELinux**

# seccomp

apiVersion: v1

kind: Pod

metadata:

name: mypod

annotations:

**seccomp.security.alpha.kubernetes.io/pod: runtime/default**

...

# AppArmor

```
apiVersion: v1
```

```
kind: Pod
```

```
metadata:
```

```
  name: mypod
```

```
  annotations:
```

```
    container.apparmor.security.beta.kubernetes.io/hello: runtime/default
```

```
spec:
```

```
  containers:
```

```
    - name: hello
```

```
    ...
```

# SELinux

apiVersion: v1

kind: Pod

metadata:

  name: mypod

spec:

securityContext:

**seLinuxOptions:**

**level: "s0:c123,c456"**

containers:

  - name: hello

  ...

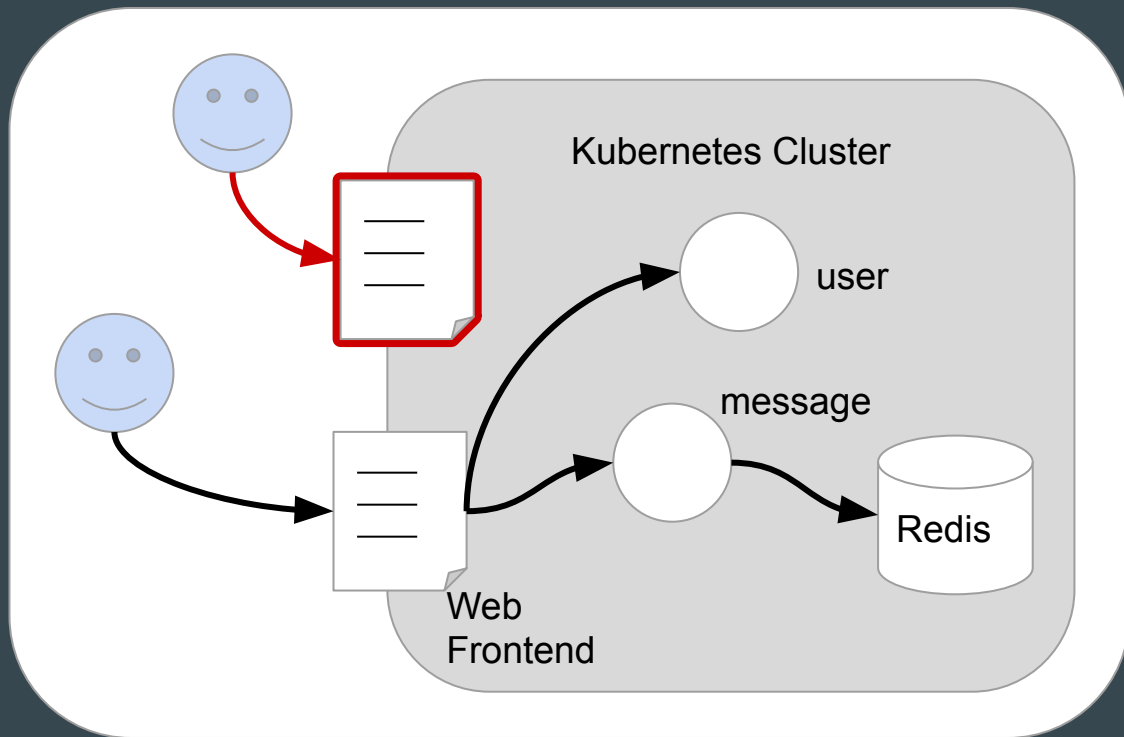


# Mitigate 2 & 3: Restrict Kubelet permissions

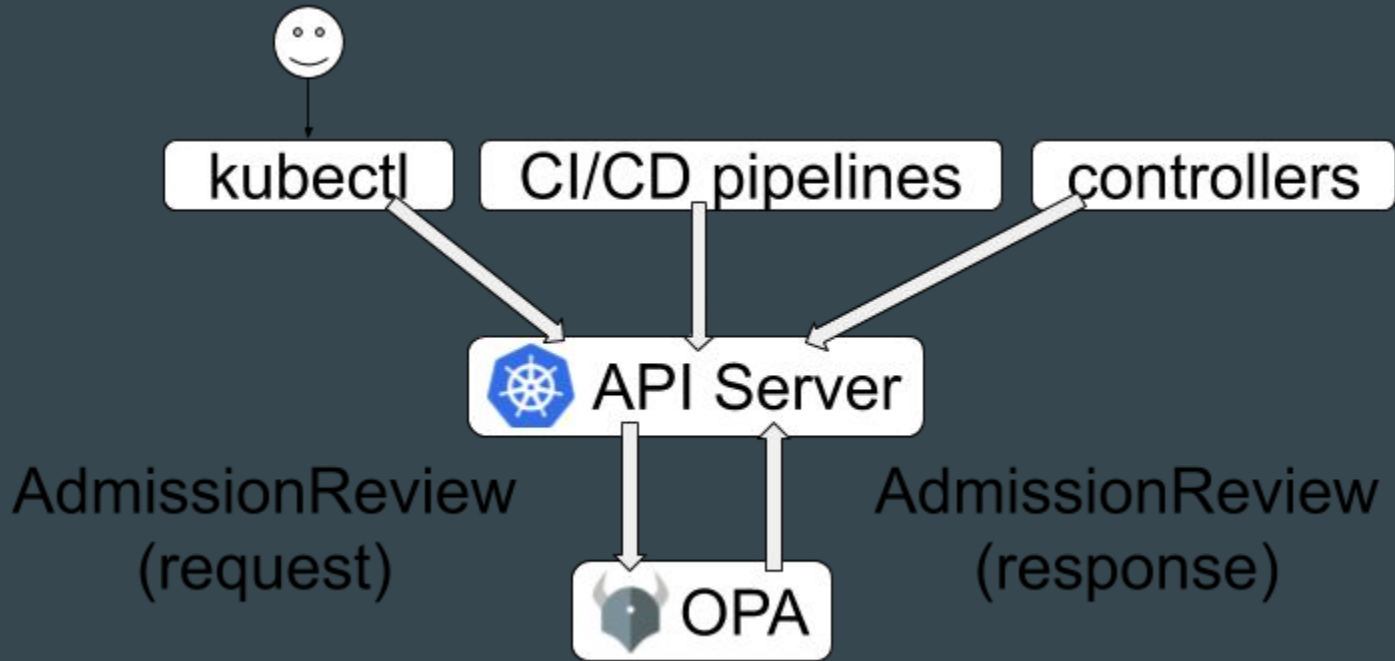
- RBAC for Kubelet:
  - `--authorization-mode=RBAC,Node`  
`--admission-control=...,NodeRestriction`
- Rotate Kubelet certs:
  - `kubelet ... --rotate-certificates`

# Unsecured Pods

- You follow the rules but others don't

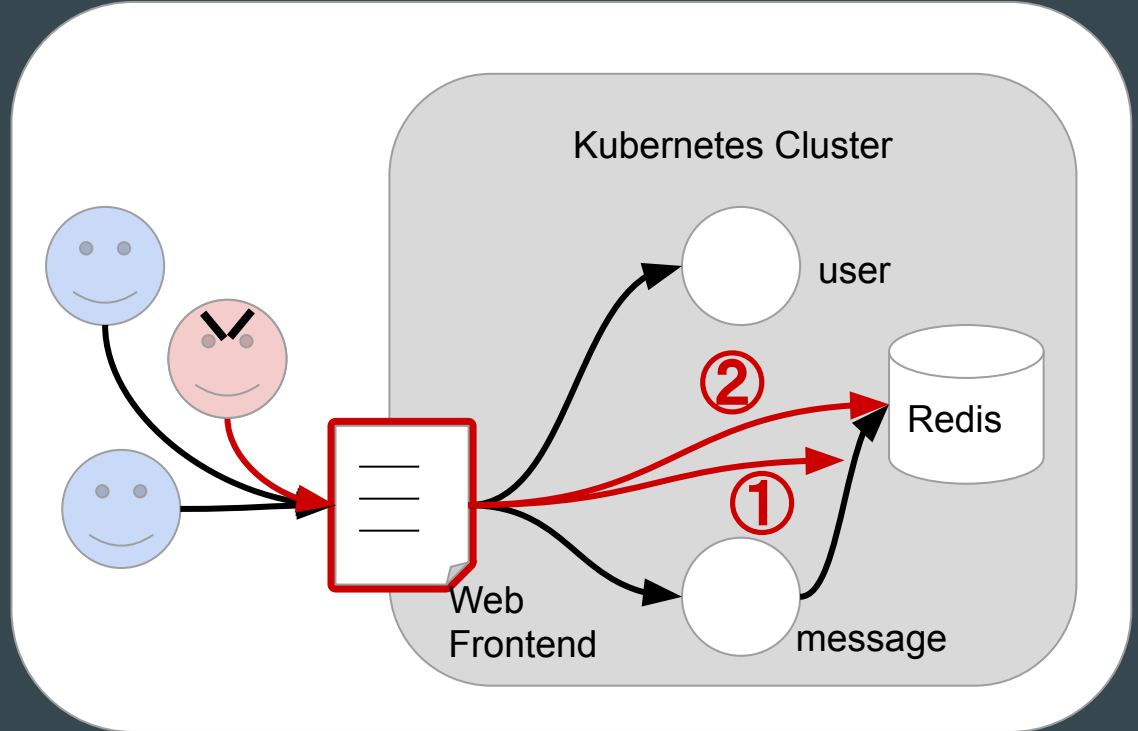


# Mitigate: Open Policy Agent



# Listening to Traffic

1. Sniffing or intercepting traffic on the network
2. Request forgery



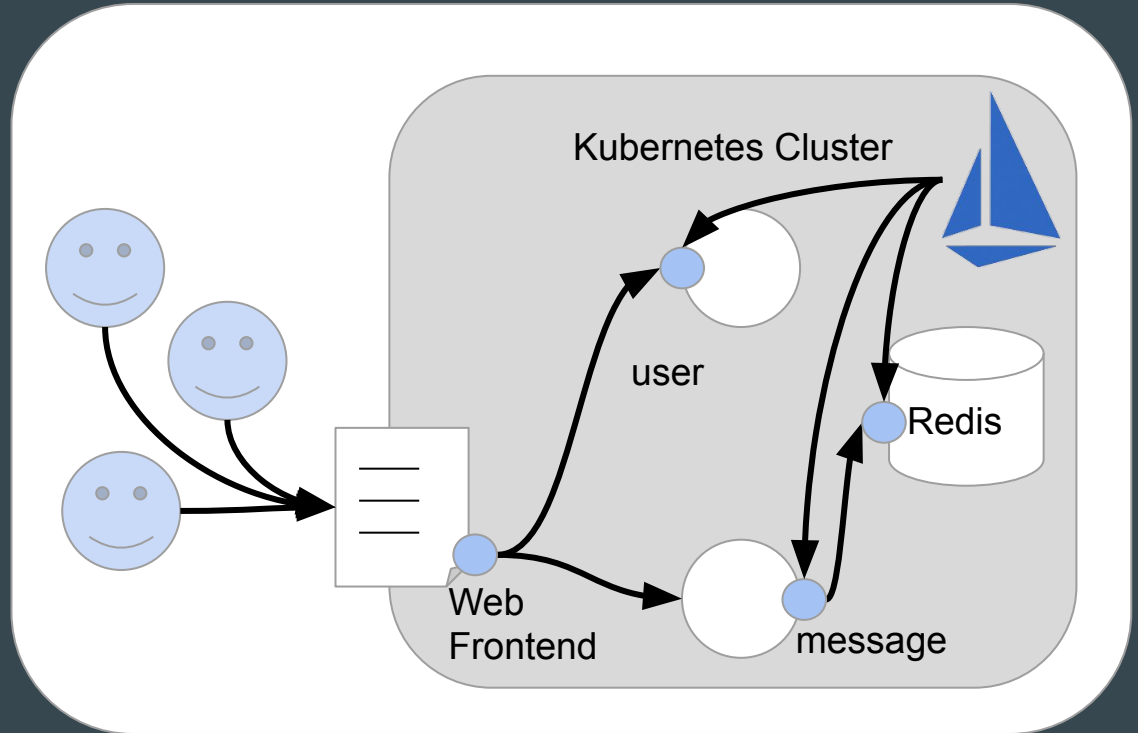
# istio

- Service mesh
- Includes Envoy proxy



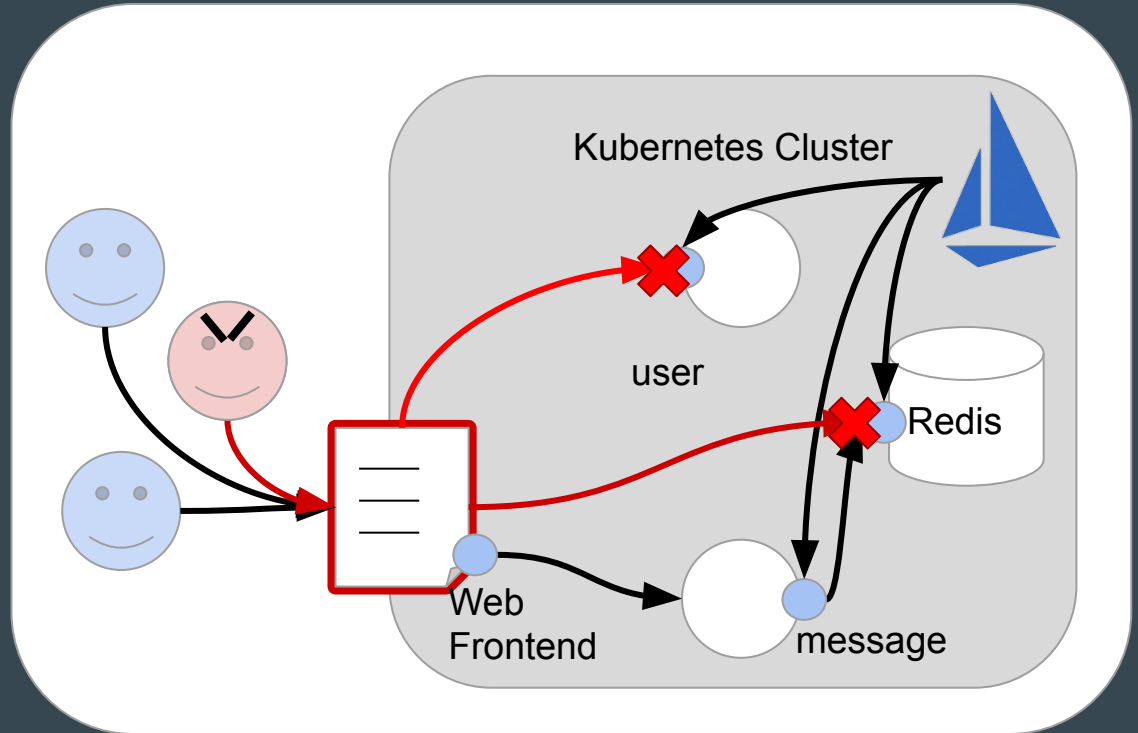
# istio

1. Proxy data between services
2. End-to-end encryption
3. Rolling certificates
4. Policy managed by central server




# istio

1. Proxy data between services
2. End-to-end encryption
3. Rolling certificates
4. Policy managed by central server



# Tips

1. Update Kubernetes early & often
2. Don't use admin for day-to-day work
3. Try benchmarking tools like kube-bench
4. Use managed services like  GKE



**Thanks!**