OWASP NZ 2023 – What Could Possibly Go Wrong in a K8s Cluster?

WILLIAM KOH

6 JULY 2023

Thank You to Our Sponsors and Hosts!

OWASP NEW ZEALAND owasp.org.nz

AppSec NZ appsec.org.nz

AUT UNIVERSITY TE WĀNANGA ARONUI O TAMAKI MAKAU RAU

DEFEND

fastly

QUANTUM SECURITY

DATACOM

aura INFORMATION SECURITY POWERED BY KORDIA

IriusRisk

dta Defence Technology Agency

planit an NRI company

CyberCX

tesserent

FIGHTING FOR FAIR FOR KIWI BUSINESS

safeadvisory.

Without them, this Conference couldn't happen.

I am…

Security Consultant @ WithSecure Singapore

- Booklover
- Collector
- Self-learner
- OSCP, CRT, CKA, CKS

LinkedIn:

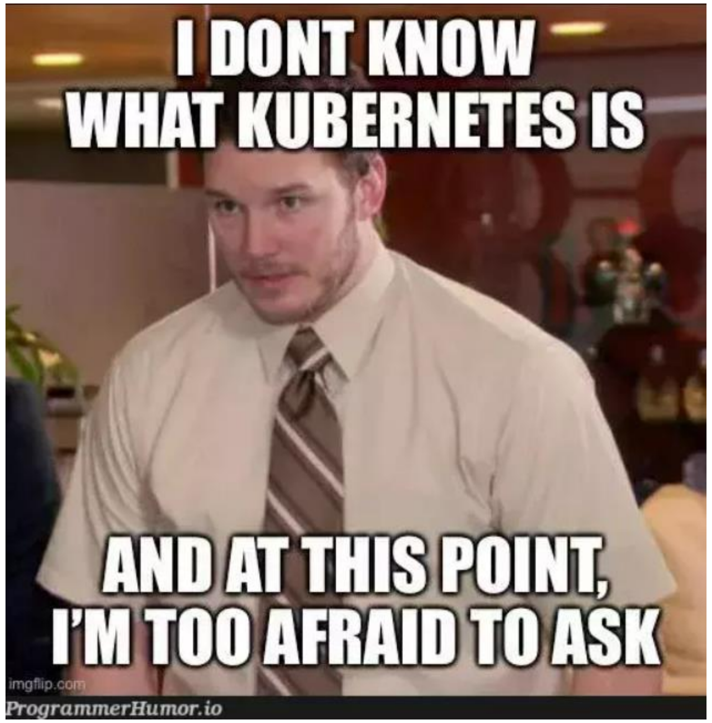
https://www.linkedin.com/in/w-xor/

Agenda

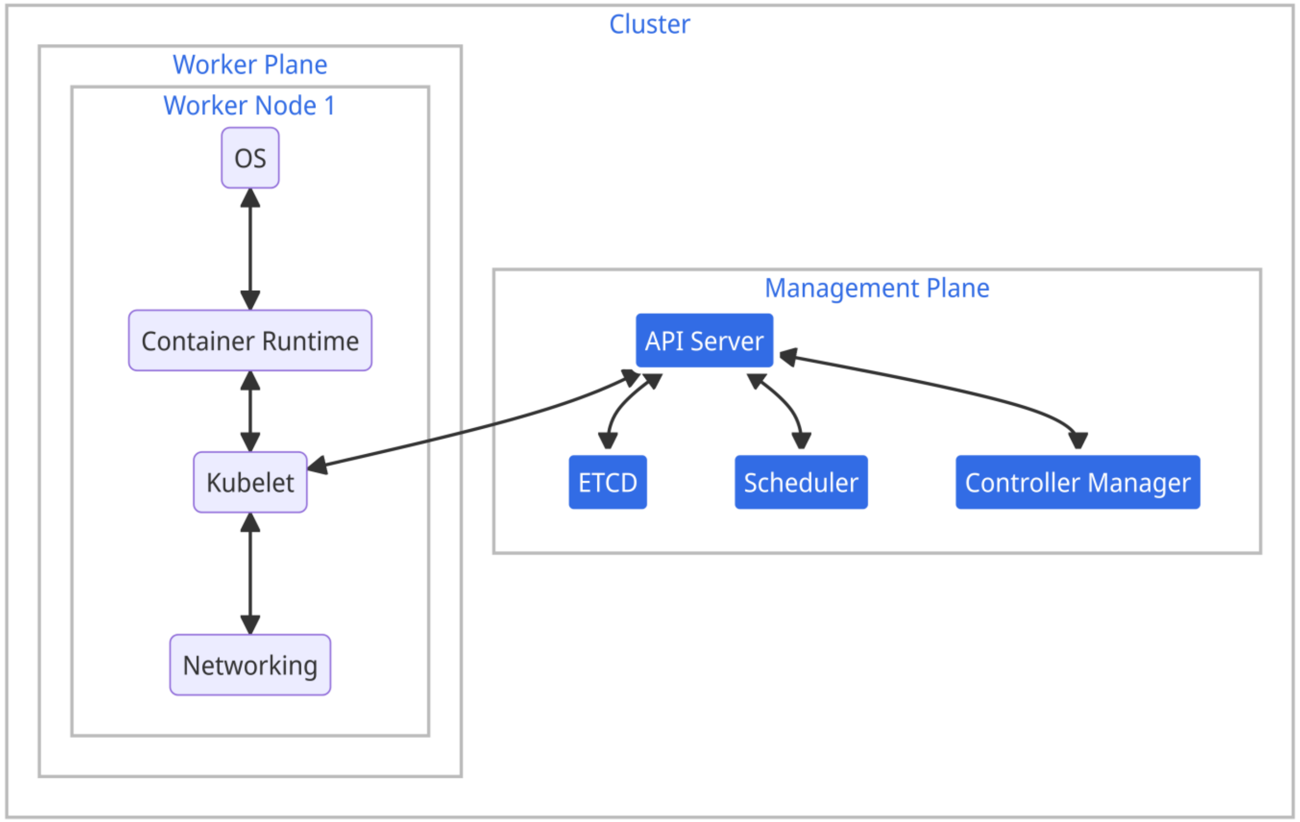
- Background
- Components within a K8s Cluster
- What Could Possibly Go Wrong?

What is Kubernetes?

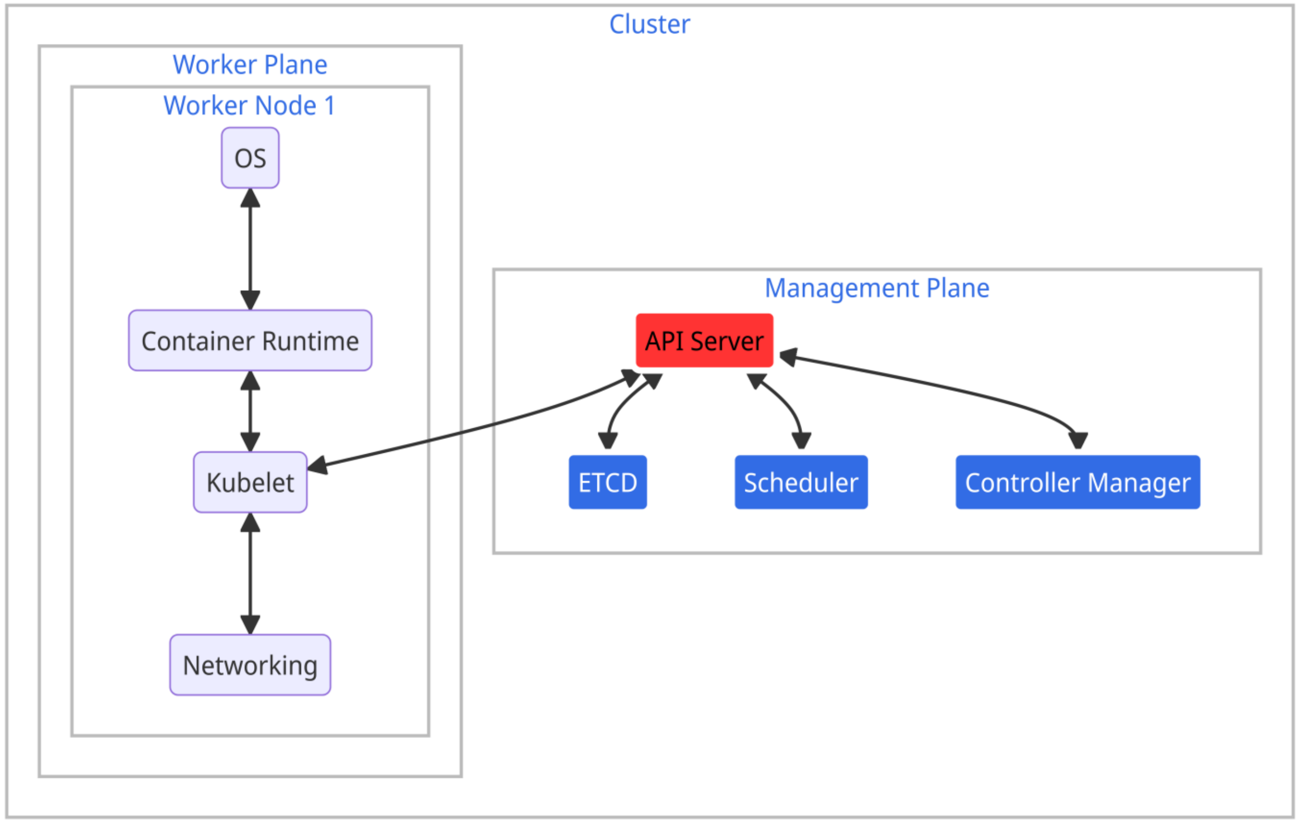
- It means Helmsman or “κυβερνήτης” in Greek
- Containers Orchestrator

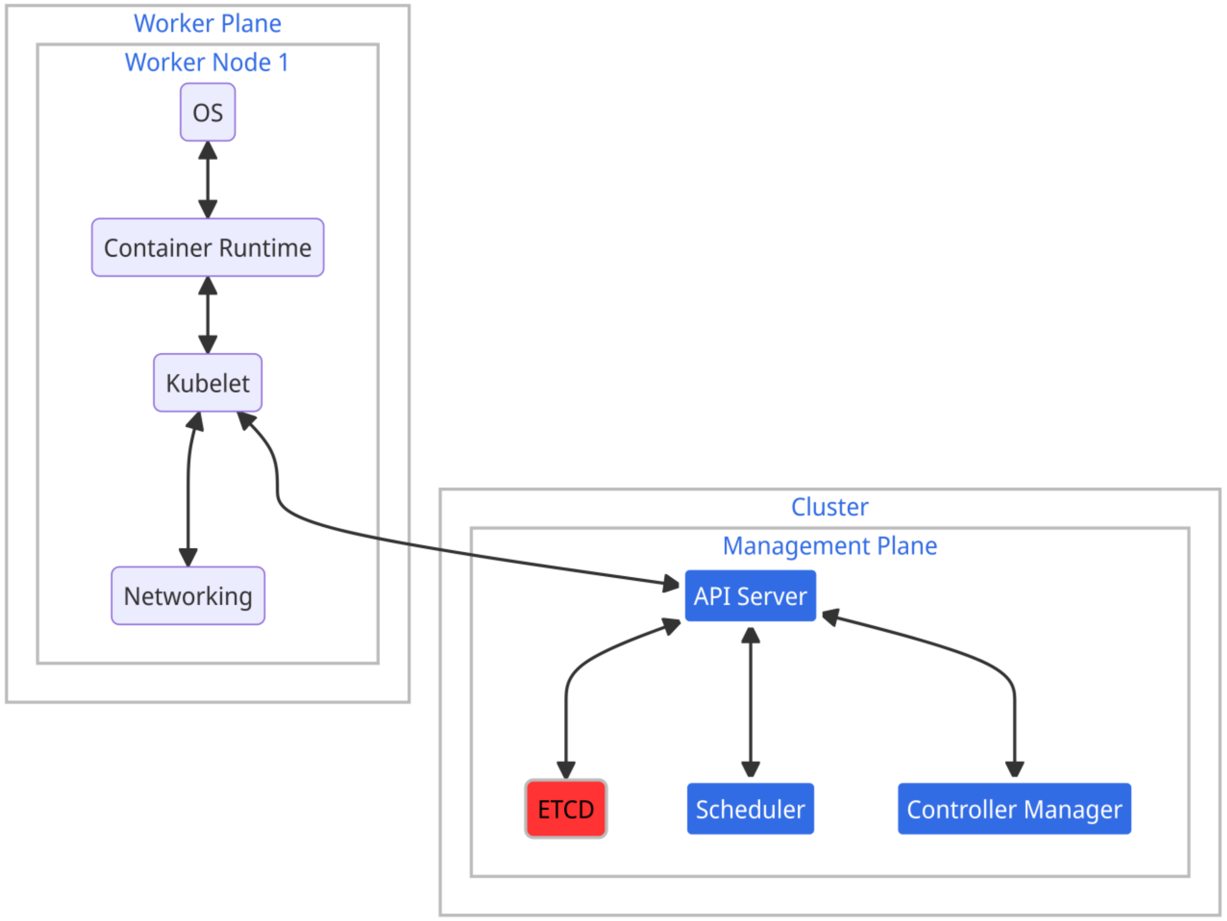


Kubernetes Cluster

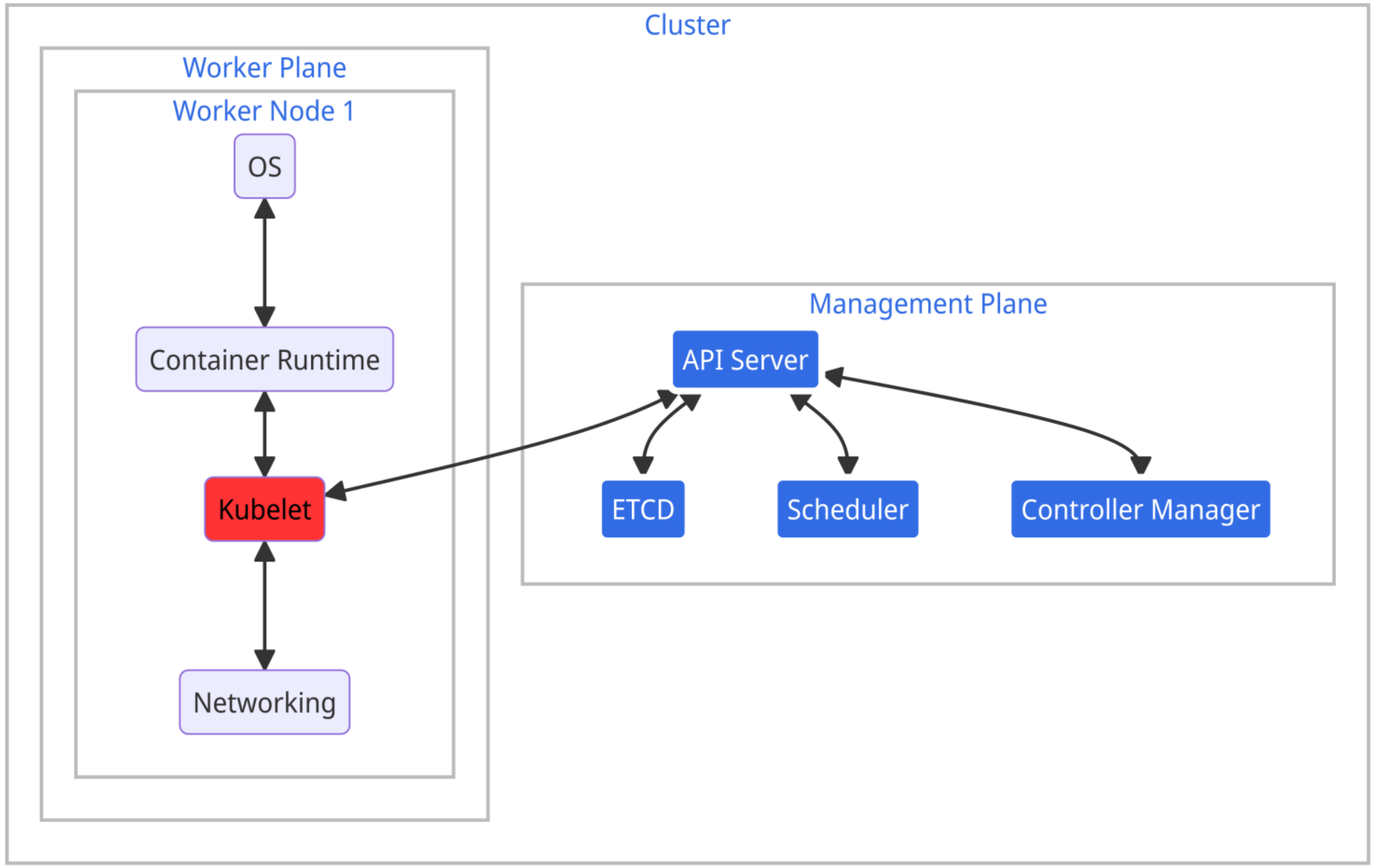


Kubernetes Cluster





Kubernetes Cluster



Kubernetes Cluster

WHAT COULD POSSIBLY GO WRONG?
38
makeameme.org

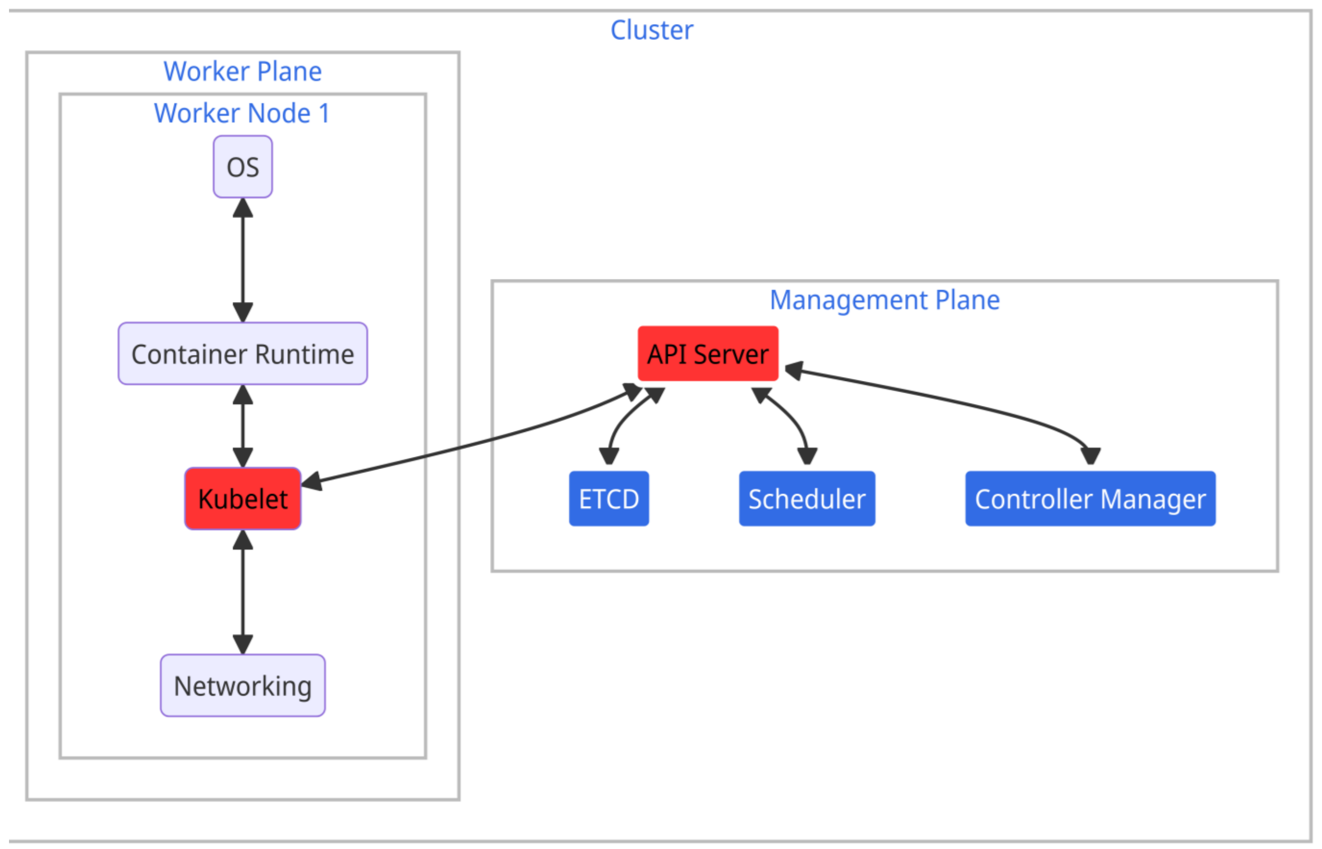
Networking

RBAC

Pod Security

Networking

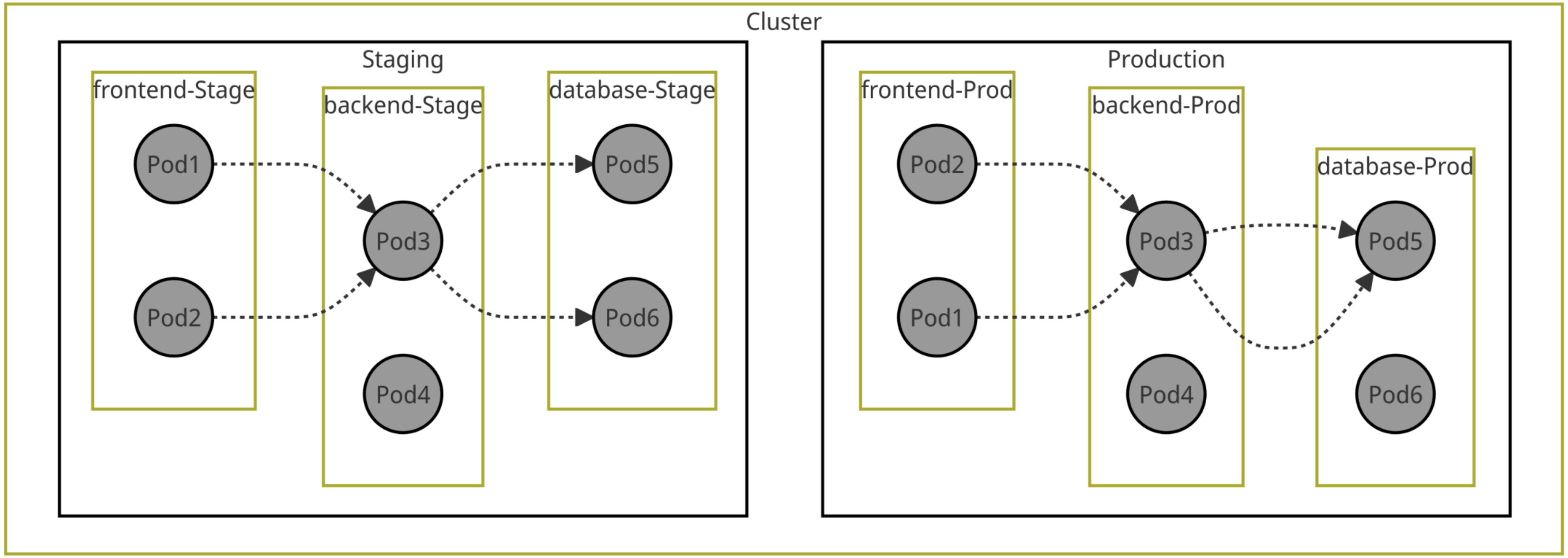
- Everything communicates via REST API
- Flat network
- No transport level encryption by default



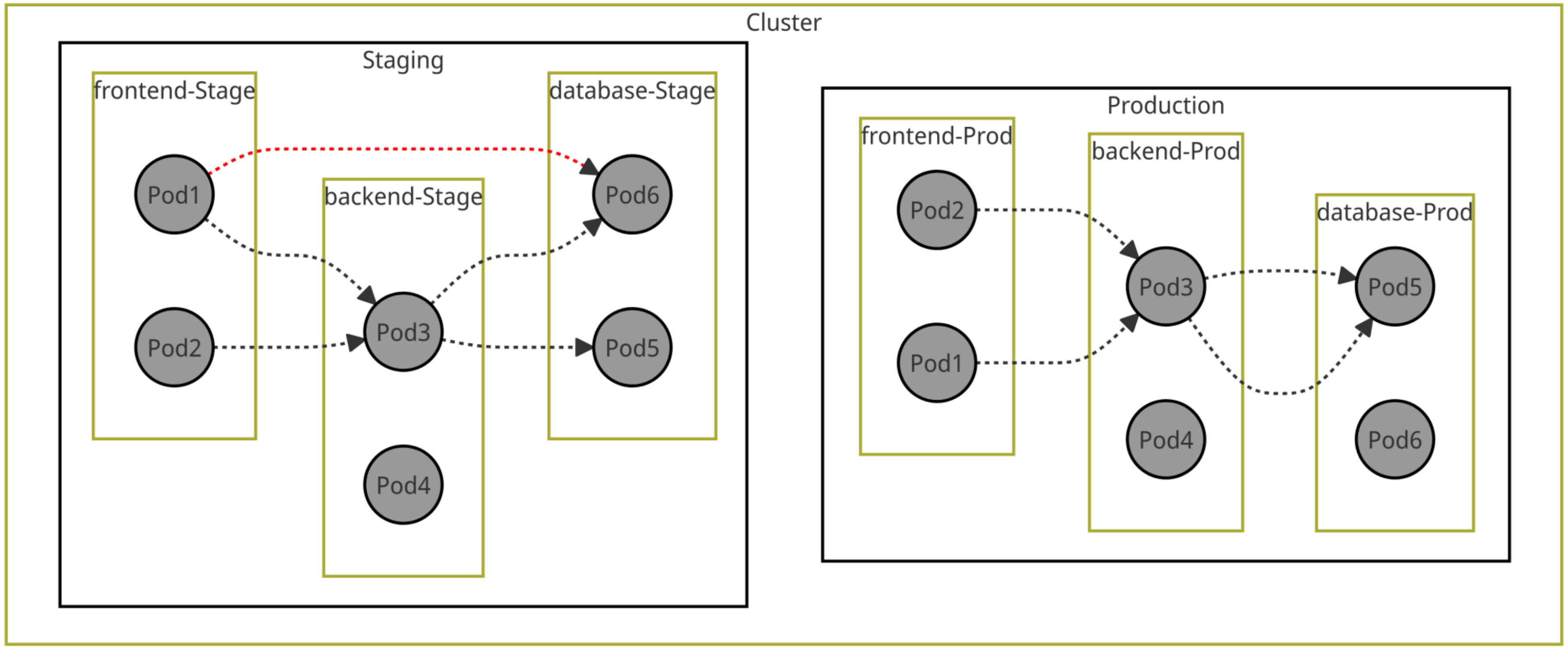
Possible Entry Points From The Outside

Networking

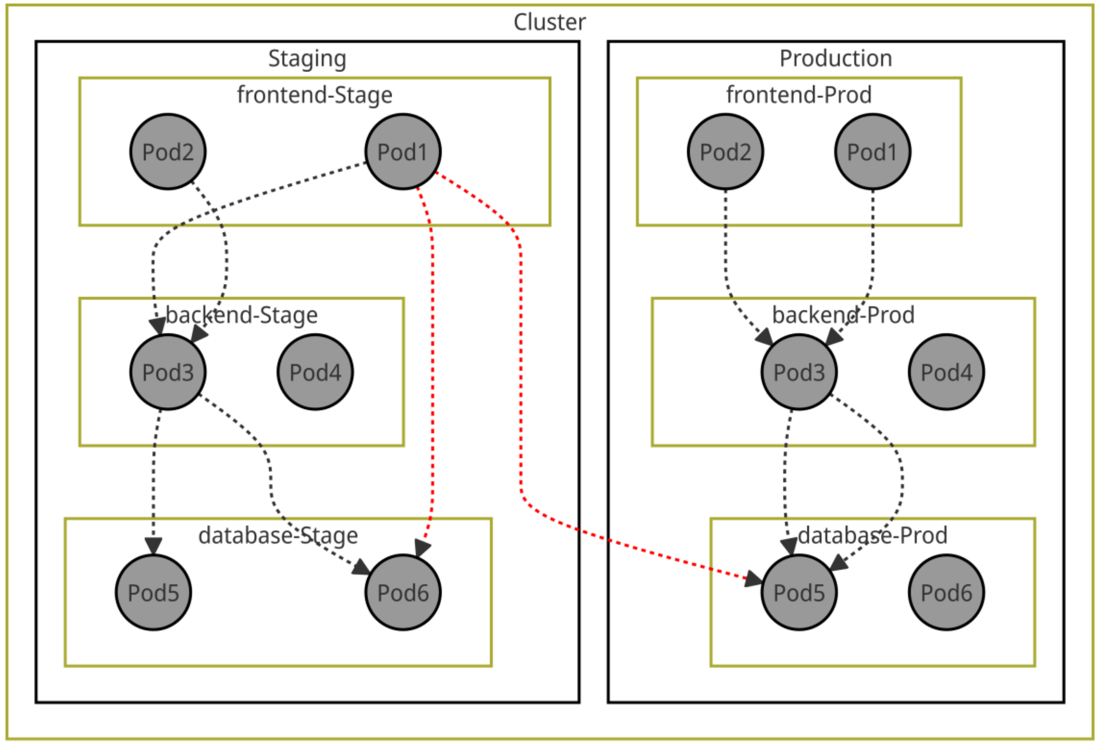
- (Un)authorised access to kubelet - /exec, /pods
- Privileged pods on the hunt
- RCE to the pods
- Privileged Service Account token
- Move laterally



Flat Network



Flat Network



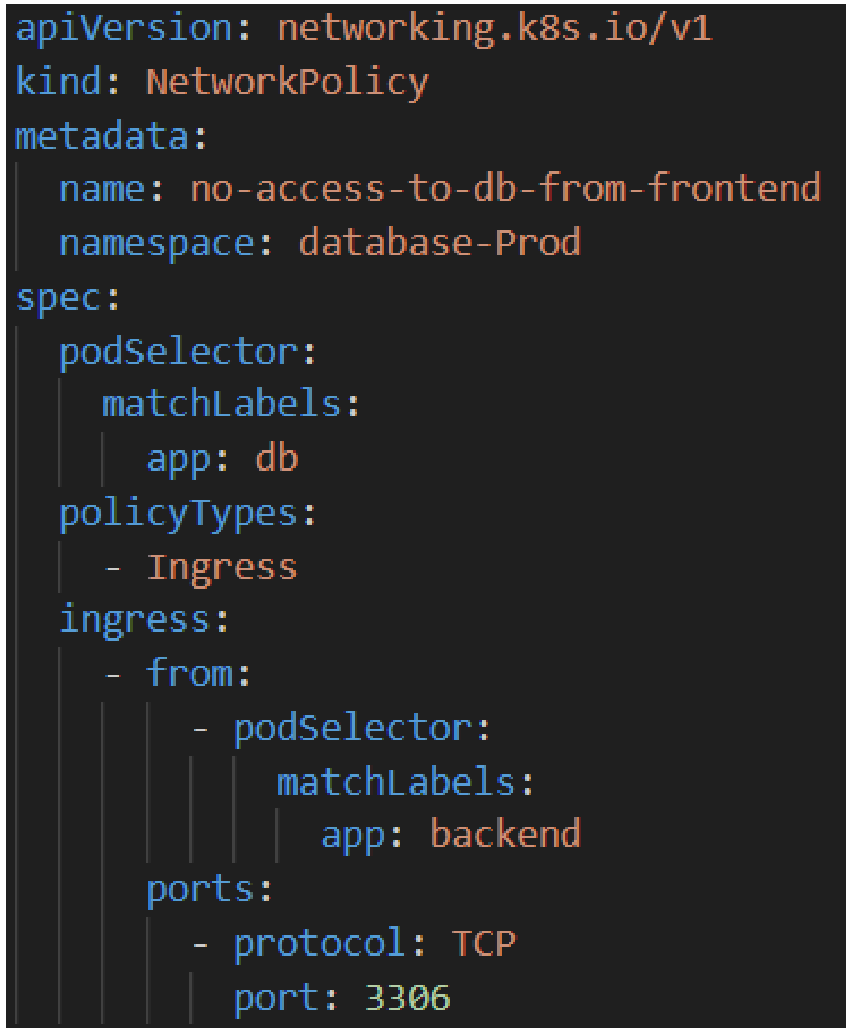
Flat Network

Network Policy

No worries! Network policy comes to our rescue!

Network Policy

- Can be defined through YAML manifest
- 2 types of policies: **Ingress** & **Egress**
- Enforce access controls based on several conditions
  - IP addresses
  - Namespace’s label
  - Pod’s label
  - Port



```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: no-access-to-db-from-frontend
  namespace: database-Prod
spec:
  podSelector:
    matchLabels:
      app: db
  policyTypes:
    - Ingress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              app: backend
      ports:
        - protocol: TCP
          port: 3306
```

Network Policy

This should be stopping the “frontend” pods from accessing the “database” pods in Production. Now it should be secure… Right?

Common Misconfigurations

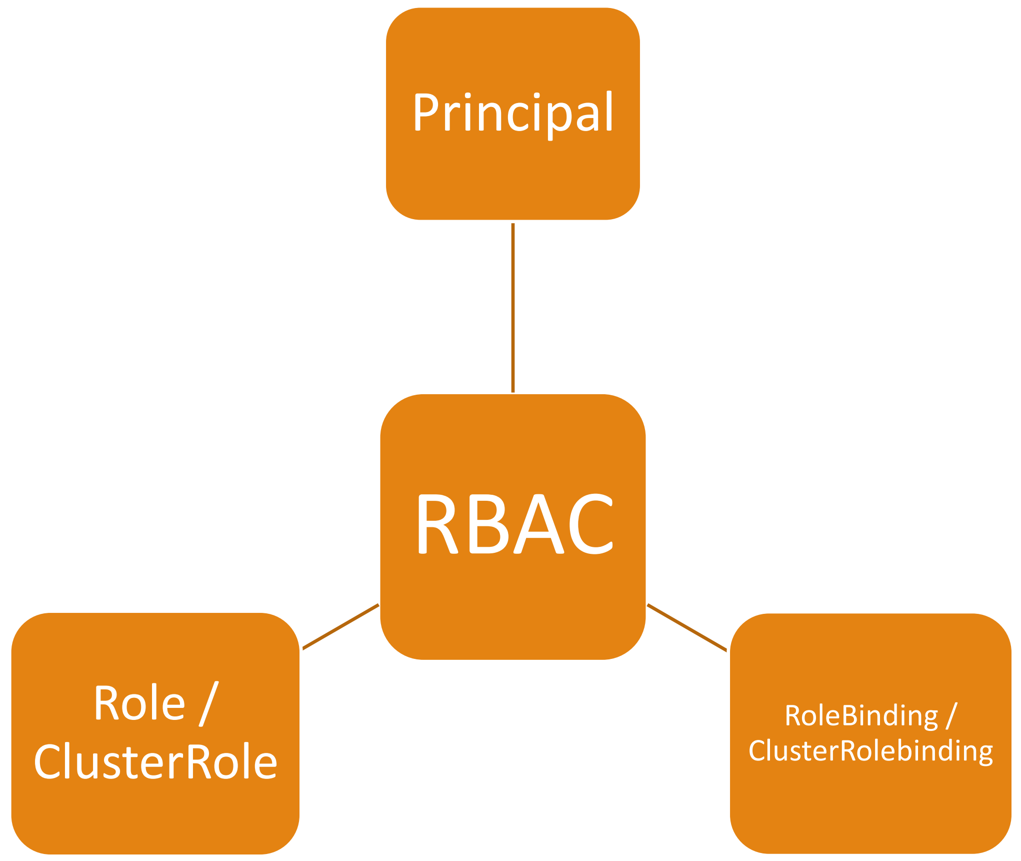
- Lack of network policies
- Network policies not being accurate / fine-grained

Networking

RBAC

Pod Security

RBAC – High-level Concept



RBAC – High-level Concept

Principal

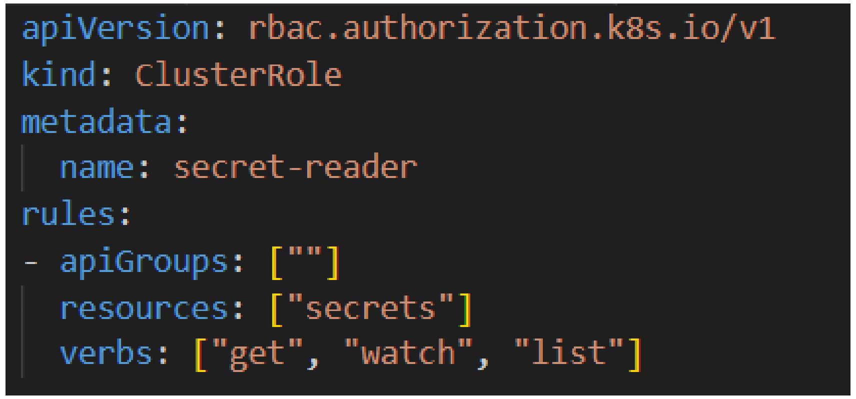
SERVICE ACCOUNT

USER / GROUP

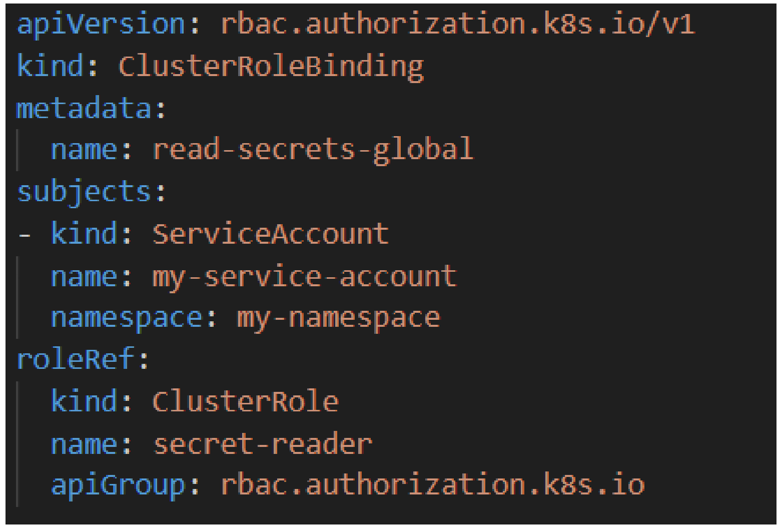
EXTERNAL USER

RBAC – High-level Concept

Role binding to a specific user / SA



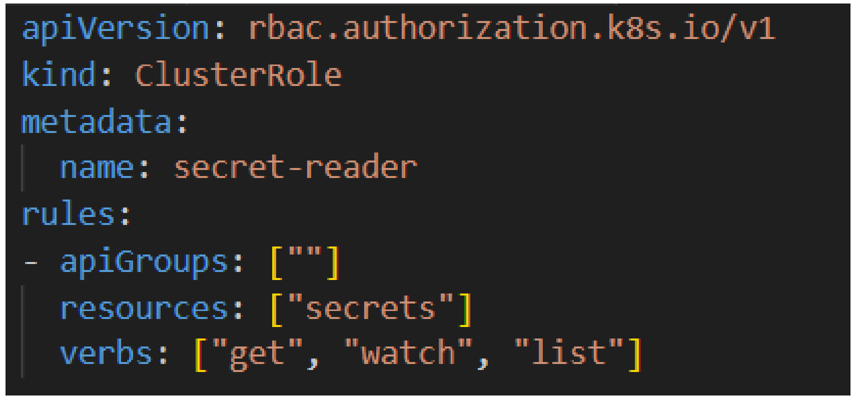
```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```



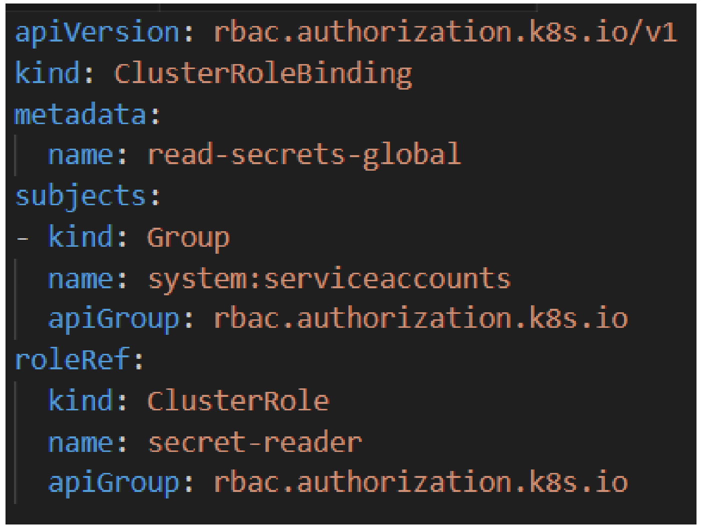
```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: ServiceAccount
  name: my-service-account
  namespace: my-namespace
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

RBAC – High-level Concept

Role binding to a group of users / SAs



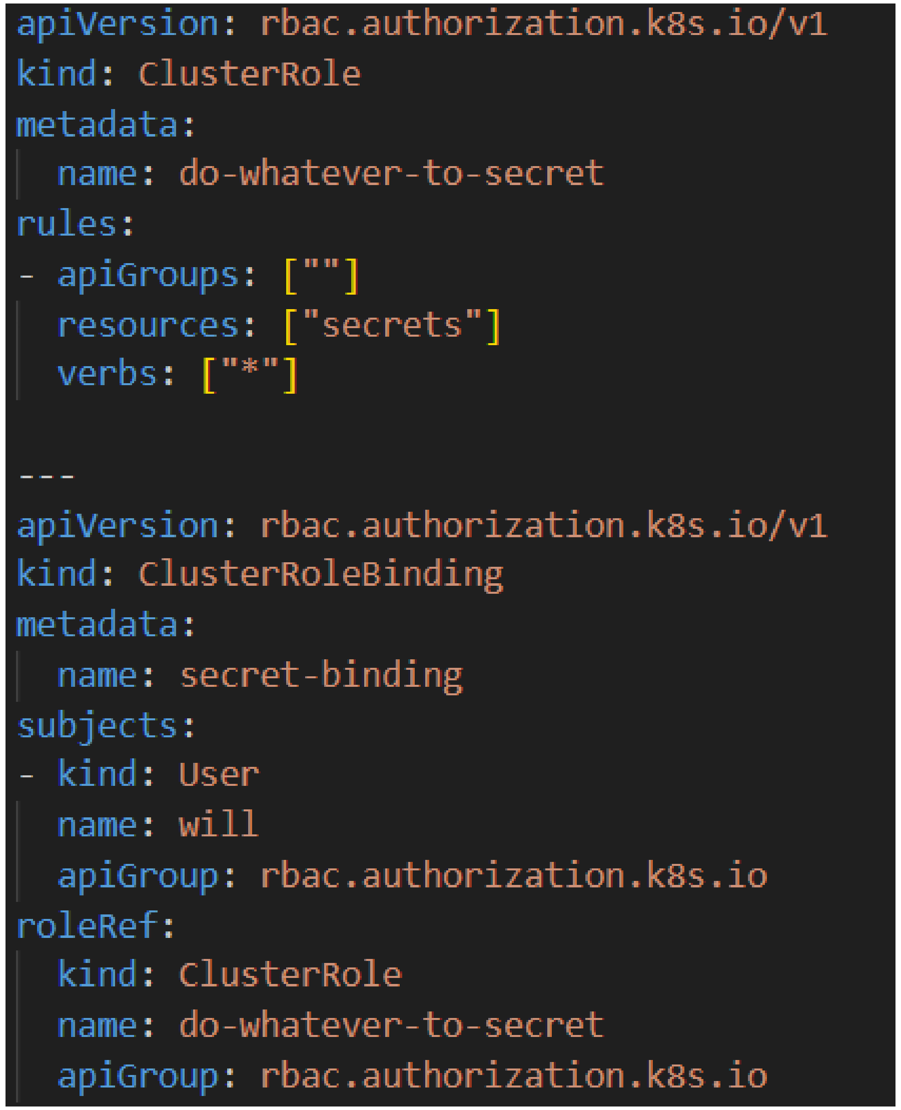
```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```



```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: system:serviceaccounts
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

Common Misconfigurations

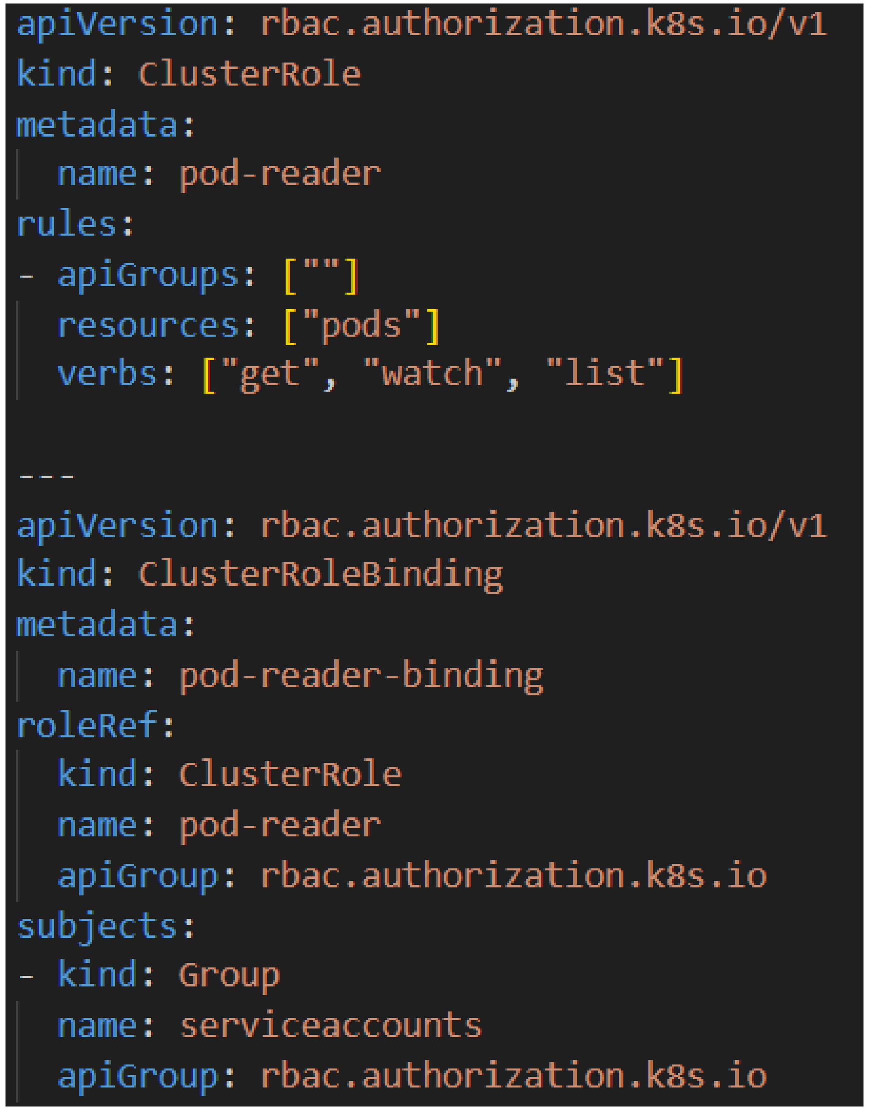
- Use of Wildcard
- Overly provisioned groups
- Cluster Role == Role


```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: do-whatever-to-secret
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["*"]

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: secret-binding
subjects:
- kind: User
  name: will
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: do-whatever-to-secret
  apiGroup: rbac.authorization.k8s.io
```

Use of Wildcard

- Does one really need all permissions?



```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: pod-reader-binding
roleRef:
  kind: ClusterRole
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  name: serviceaccounts
  apiGroup: rbac.authorization.k8s.io
```

Overly Provisioned groups

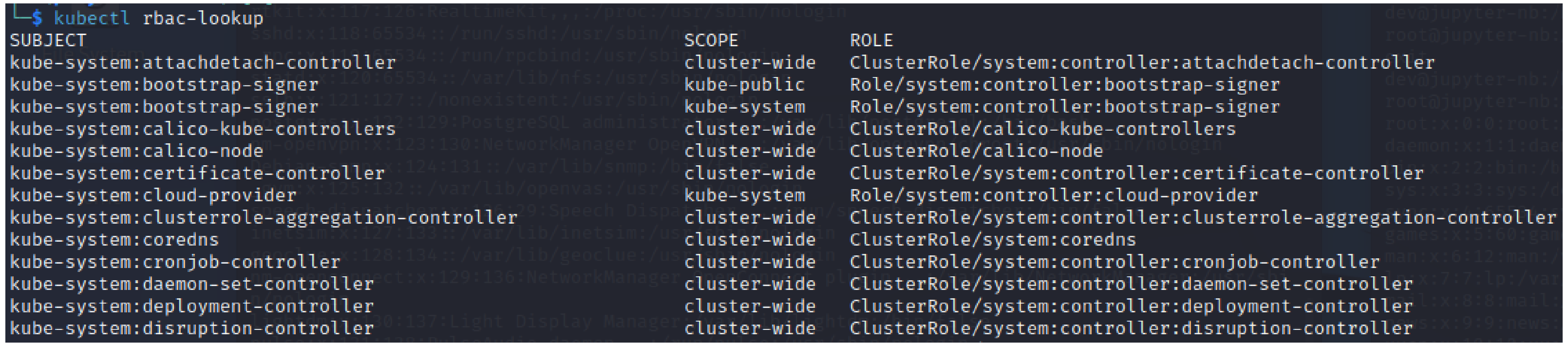
- Do all SAs require the same permission?

Cluster Role ≈ Role

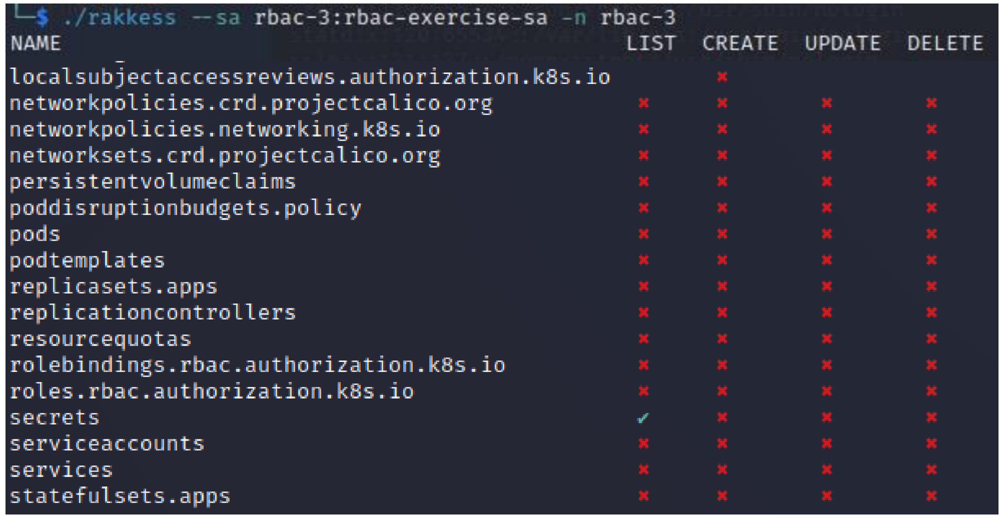
- Future-proof
- For convenience's sake
- Or sometimes people are just confused…

How to Find Out Overly Provisioned Principal?

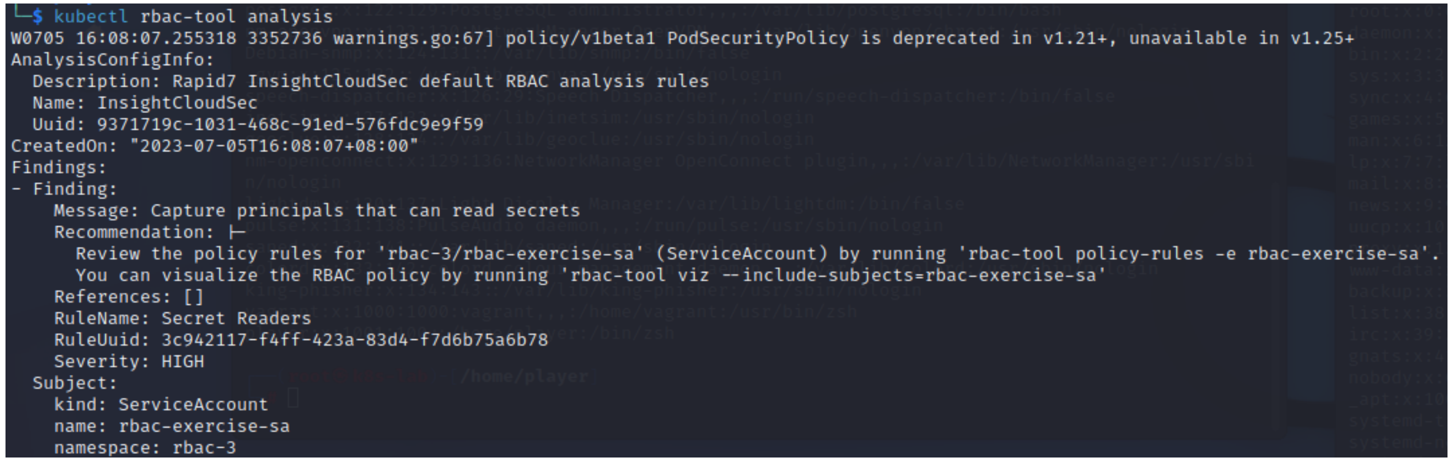
- rakkess
- rbac-tool
- rbac-lookup



```
└─$ kubectl rbac-lookup
SUBJECT                                             SCOPE          ROLE
kube-system:attachdetach-controller                 cluster-wide   ClusterRole/system:controller:attachdetach-controller
kube-system:bootstrap-signer                        kube-public    Role/system:controller:bootstrap-signer
kube-system:bootstrap-signer                        kube-system    Role/system:controller:bootstrap-signer
kube-system:calico-kube-controllers                 cluster-wide   ClusterRole/calico-kube-controllers
kube-system:calico-node                             cluster-wide   ClusterRole/calico-node
kube-system:certificate-controller                  cluster-wide   ClusterRole/system:controller:certificate-controller
kube-system:cloud-provider                          kube-system    Role/system:controller:cloud-provider
kube-system:clusterrole-aggregation-controller      cluster-wide   ClusterRole/system:controller:clusterrole-aggregation-controller
kube-system:coredns                                 cluster-wide   ClusterRole/system:coredns
kube-system:cronjob-controller                      cluster-wide   ClusterRole/system:controller:cronjob-controller
kube-system:daemon-set-controller                   cluster-wide   ClusterRole/system:controller:daemon-set-controller
kube-system:deployment-controller                   cluster-wide   ClusterRole/system:controller:deployment-controller
kube-system:disruption-controller                   cluster-wide   ClusterRole/system:controller:disruption-controller
```



```
└─$ ./rakkess --sa rbac-3:rbac-exercise-sa -n rbac-3
NAME                                         LIST  CREATE  UPDATE  DELETE
localsubjectaccessreviews.authorization.k8s.io      ✗
networkpolicies.crd.projectcalico.org         ✗     ✗       ✗       ✗
networkpolicies.networking.k8s.io             ✗     ✗       ✗       ✗
networksets.crd.projectcalico.org             ✗     ✗       ✗       ✗
persistentvolumeclaims                        ✗     ✗       ✗       ✗
poddisruptionbudgets.policy                   ✗     ✗       ✗       ✗
pods                                          ✗     ✗       ✗       ✗
podtemplates                                  ✗     ✗       ✗       ✗
replicasets.apps                              ✗     ✗       ✗       ✗
replicationcontrollers                        ✗     ✗       ✗       ✗
resourcequotas                                ✗     ✗       ✗       ✗
rolebindings.rbac.authorization.k8s.io        ✗     ✗       ✗       ✗
roles.rbac.authorization.k8s.io               ✗     ✗       ✗       ✗
secrets                                       ✓     ✗       ✗       ✗
serviceaccounts                               ✗     ✗       ✗       ✗
services                                      ✗     ✗       ✗       ✗
statefulsets.apps                             ✗     ✗       ✗       ✗
```



```
└─$ kubectl rbac-tool analysis
W0705 16:08:07.255318 3352736 warnings.go:67] policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25+
AnalysisConfigInfo:
  Description: Rapid7 InsightCloudSec default RBAC analysis rules
  Name: InsightCloudSec
  Uuid: 9371719c-1031-468c-91ed-576fdc9e9f59
CreatedOn: "2023-07-05T16:08:07+08:00"
Findings:
- Finding:
    Message: Capture principals that can read secrets
    Recommendation: ⊢
      Review the policy rules for 'rbac-3/rbac-exercise-sa' (ServiceAccount) by running 'rbac-tool policy-rules -e rbac-exercise-sa'.
      You can visualize the RBAC policy by running 'rbac-tool viz --include-subjects=rbac-exercise-sa'
    References: []
    RuleName: Secret Readers
    RuleUuid: 3c942117-f4ff-423a-83d4-f7d6b75a6b78
    Severity: HIGH
  Subject:
    kind: ServiceAccount
    name: rbac-exercise-sa
    namespace: rbac-3
```

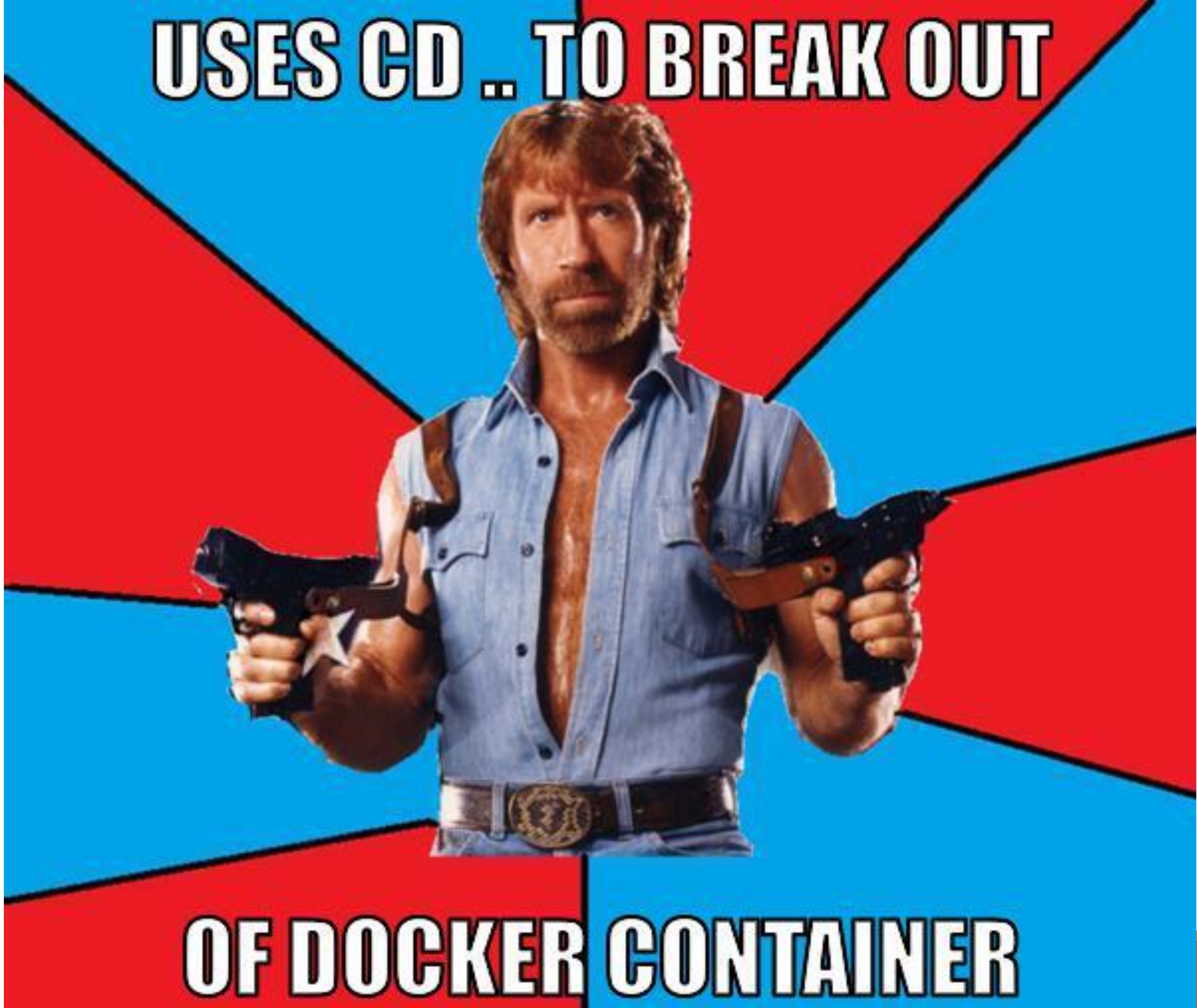
Networking

RBAC

Pod Security

Pod Security

What is the first thing that always pops up to people mind when it comes to pod security?



Pod Security

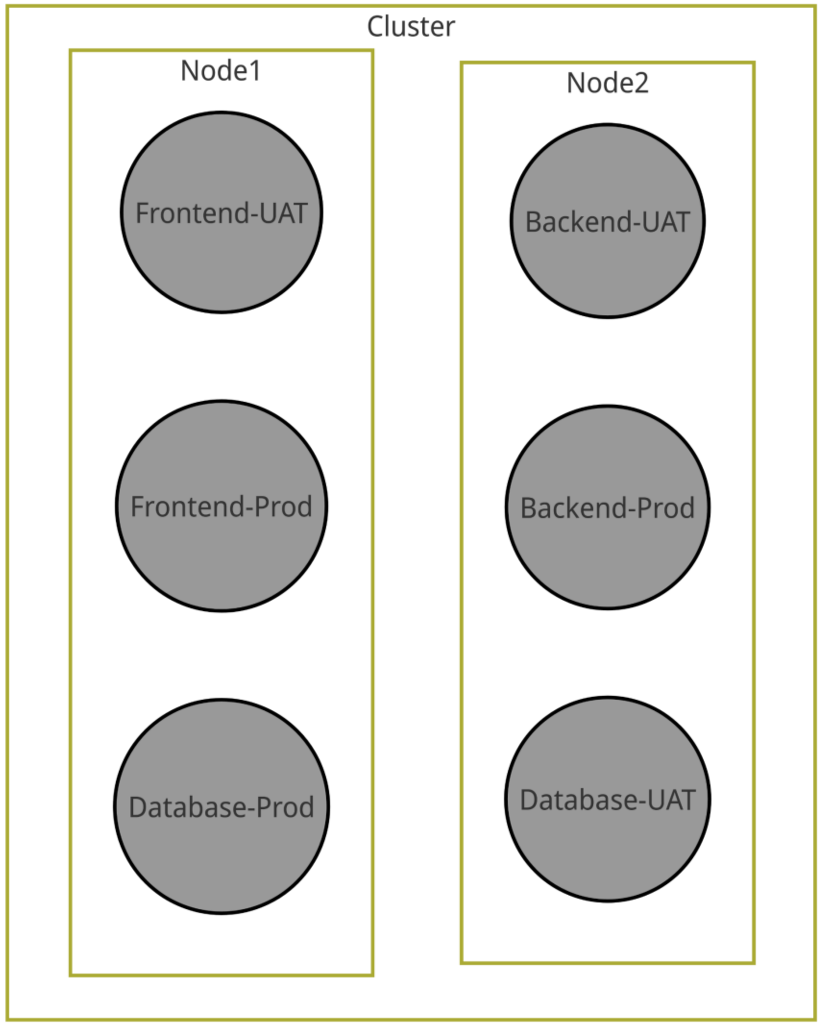
- A container is essentially a process running in isolation on a host
- Shares kernel resources / features of host operating system
  - Namespaces
  - Control Groups (cgroups)
  - Chroot
  - Capabilities
  - FileSystems

Security Context

- Defines how much privileges and level of access to kernel resources a pod or container is getting
- Some examples of security context:
  - allowPrivilegeEscalation
  - privileged
  - readOnlyRootFilesystem
  - runAsNonRoot

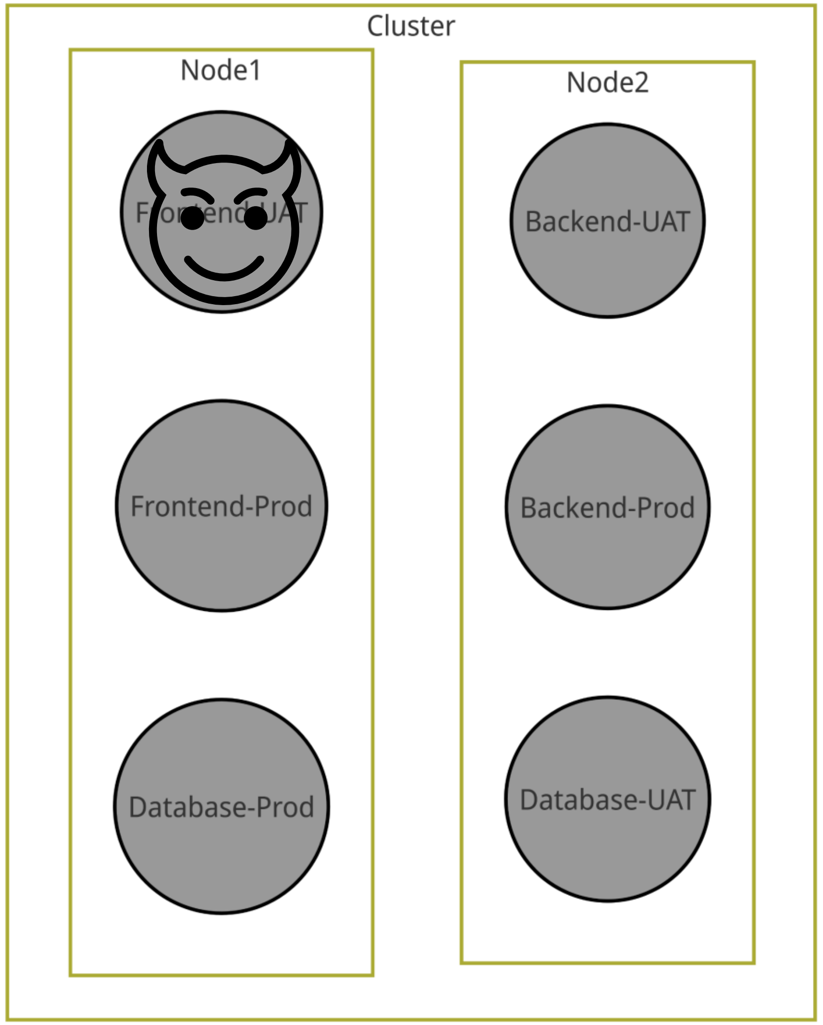
Pod (Container) Breakout

- Some techniques / misconfigurations could be utilized to break out a container:
  - Privileged pod (Overly permissive capabilities)
  - Kernel exploit
  - Host volumes mount



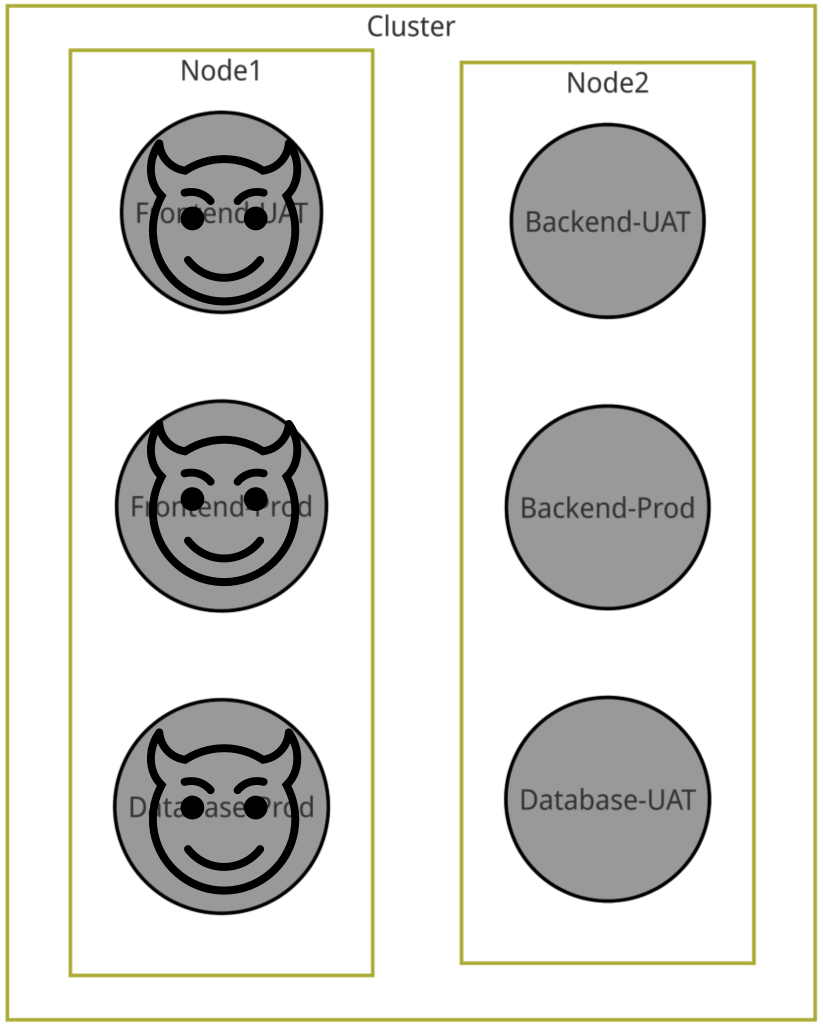
Pod Placement

- The blast radius could be even bigger with a failed pod placement strategy



Pod Placement

- The blast radius could be even bigger with a failed pod placement strategy

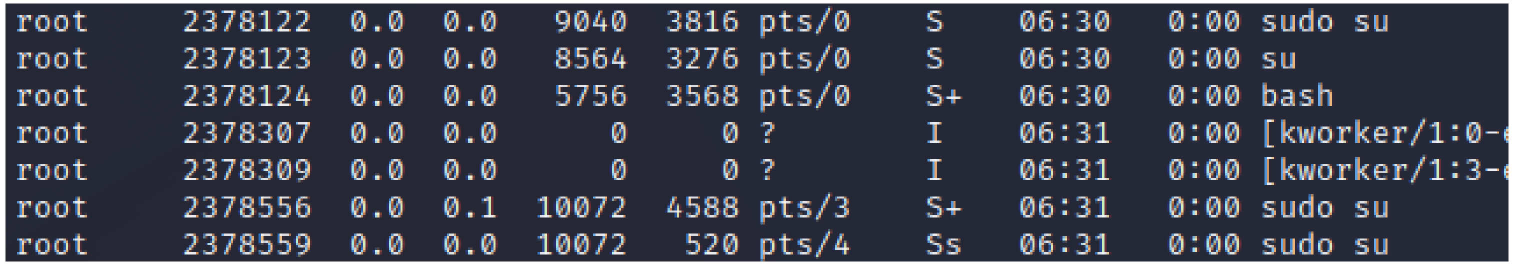


Pod Placement

- The blast radius could be even bigger with a failed pod placement strategy

Observability & Visibility

Do you think you can see what's happening inside a pod?

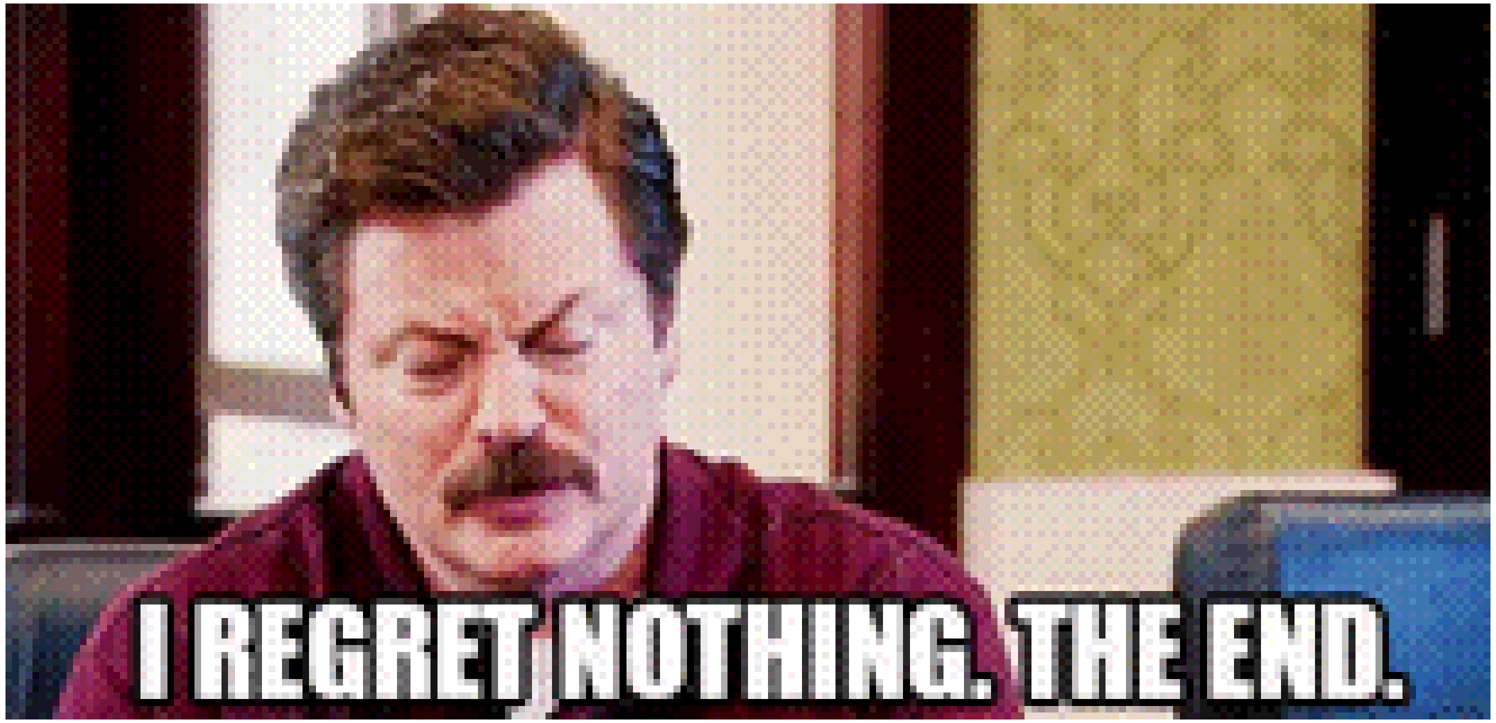


```
root      2378122  0.0  0.0   9040  3816 pts/0    S    06:30   0:00 sudo su
root      2378123  0.0  0.0   8564  3276 pts/0    S    06:30   0:00 su
root      2378124  0.0  0.0   5756  3568 pts/0    S+   06:30   0:00 bash
root      2378307  0.0  0.0      0     0 ?        I    06:31   0:00 [kworker/1:0-
root      2378309  0.0  0.0      0     0 ?        I    06:31   0:00 [kworker/1:3-
root      2378556  0.0  0.1  10072  4588 pts/3    S+   06:31   0:00 sudo su
root      2378559  0.0  0.0  10072   520 pts/4    Ss   06:31   0:00 sudo su
```

Who executed those commands?

Secure Pods!

- Pod Security Admission
- Linux Kernel Security Modules: Seccomp, AppArmor
- nodeSelector
- Observability and visibility solutions, e.g., Falco, Cilium


I REGRET NOTHING. THE END.

Thank you!