

# Seguridad en Kubernetes

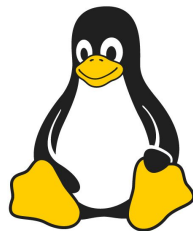
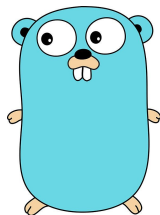


# # whoami

## Carlos Gaona

Cloud Security Engineer

MercadoLibre





# Seguridad en Kubernetes

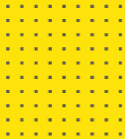


- Intro a Kubernetes
- RBAC
- Hardening
  - Control Plane
  - Containers
  - Admission Controllers
- Networking
  - Network Policies
- Preguntas



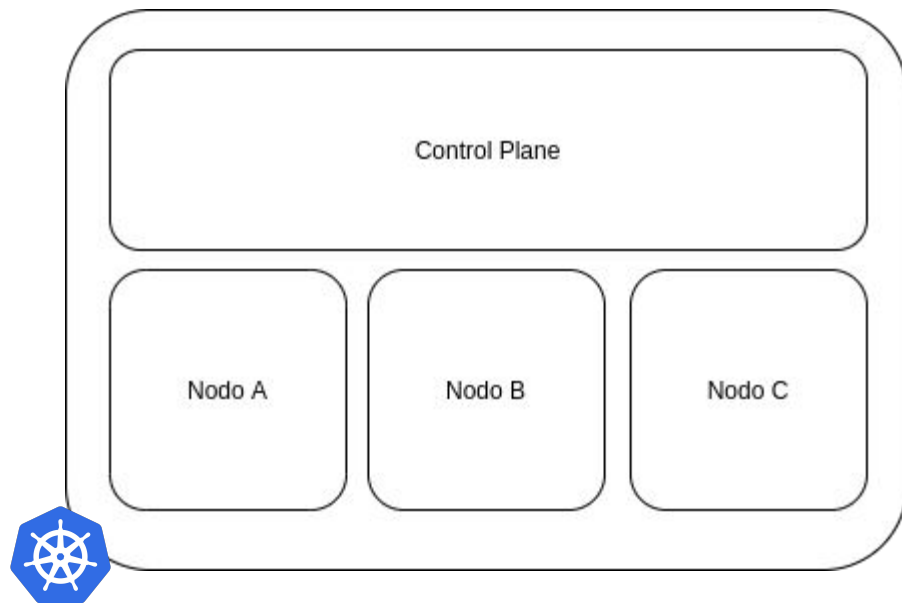
# Intro

## Qué es Kubernetes?

- Orquestador de Contenedores.
  - Creado por Google - Open Sourced 2014
- 

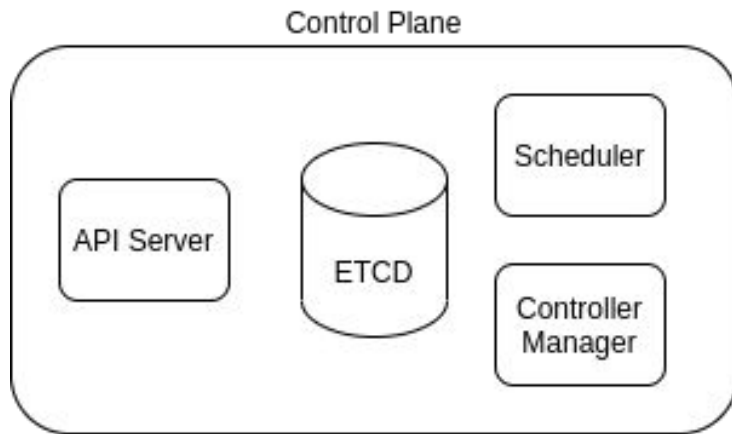
# Intro

## Cluster



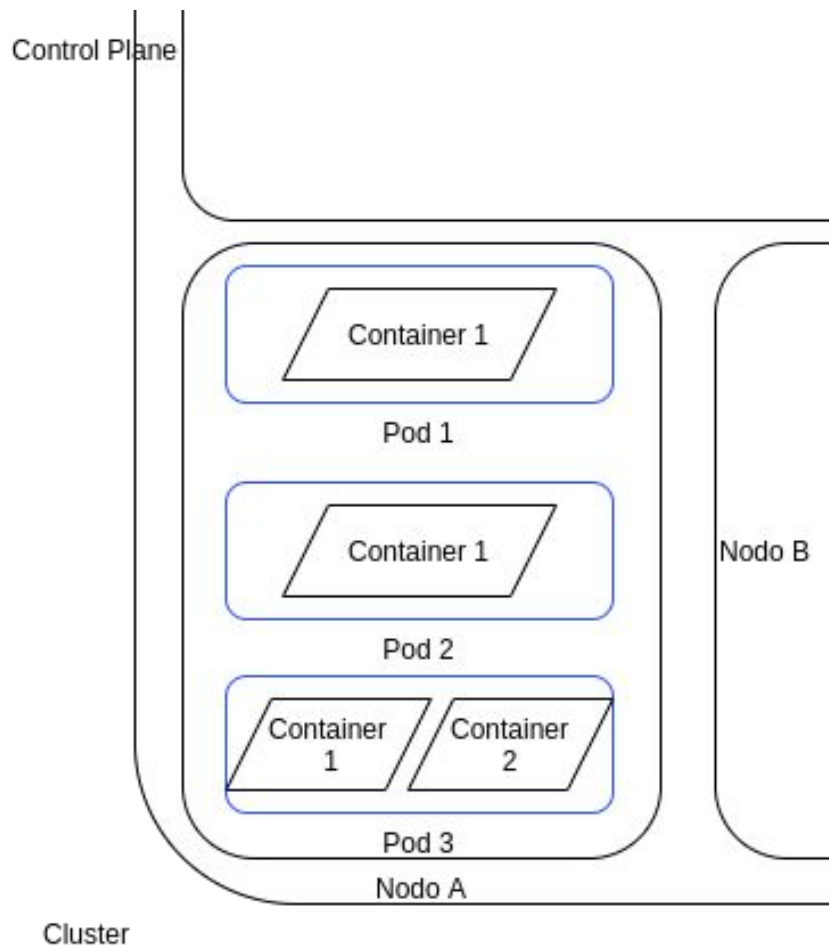
# Intro

## Control Plane



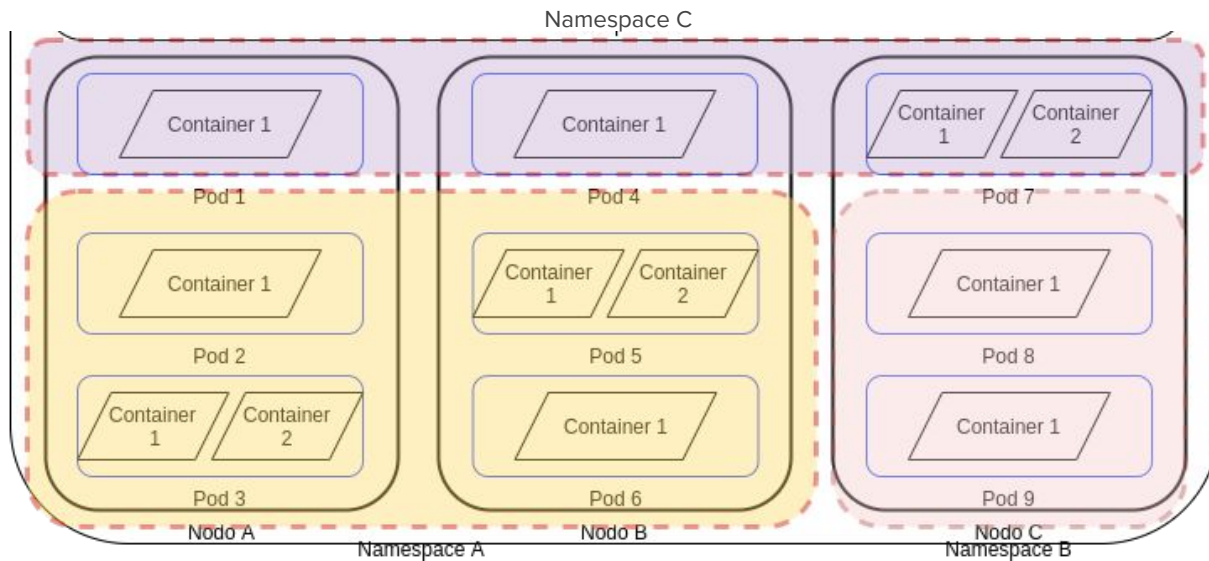
# Intro

Control Plane



# Intro

## Namespaces





# Intro

## kubectl y manifests

```
apiVersion: v1
kind: Pod
metadata:
  name: static-web
spec:
  containers:
    - name: web
      image: nginx
      ports:
        - name: web
          containerPort: 80
          protocol: TCP
```

# Intro

## kubectl y manifests

```
$ kubectl apply -f pod.yaml
```

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
static-web	1/1	Running	0	10s

# RBAC

Role Based Access Control

¿Puede \_\_\_\_\_ ?  
sujeto                      acción                      objeto

# RBAC

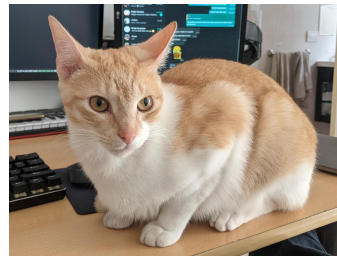
Role Based Access Control

¿Puede Pochoclo dormir en el sillón?

sujeto

acción

objeto



# RBAC

## Usuarios y Roles

Scope del Rol: 'Global'

Nombre del Rol: 'Mascota Malcriada'

Permisos:

- 'Dormir en el sillón'

# RBAC

## Usuarios y Roles



Usuario: Pochoclo  
Roles:  
- Mascota Malcriada  
Scope: Departamento A



# RBAC

## Usuarios y Roles

Scope del Rol: 'Local'

Nombre del Rol: 'Mascota Peleadora'

Permisos:

- 'Pelear con Aceituna'



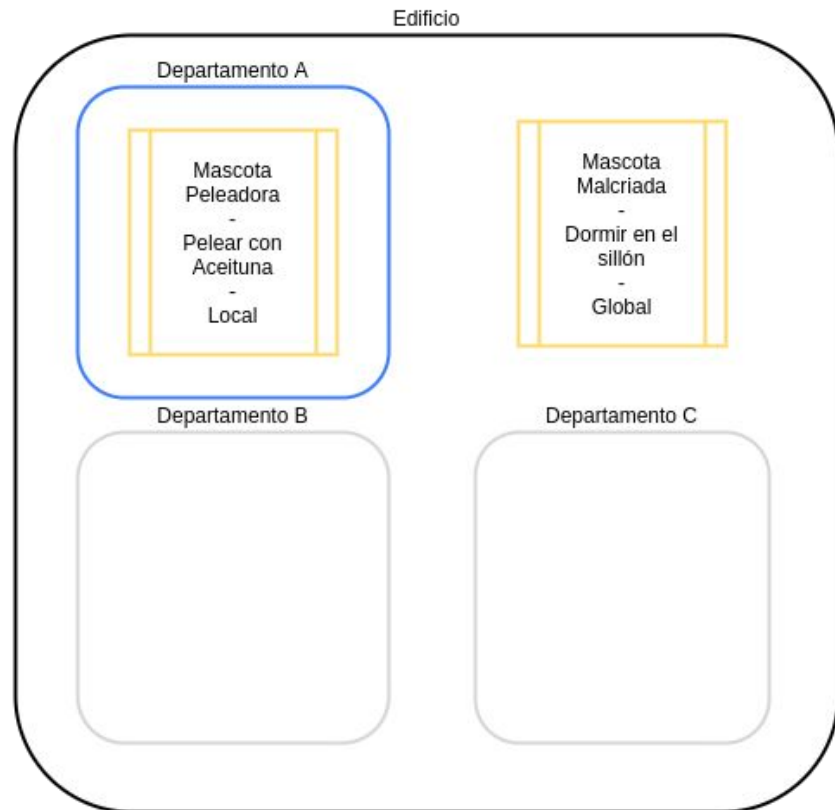
# RBAC

## Usuarios y Roles



Usuario: Pochoclo  
Roles:

- Mascota Malcriada  
Scope: Departamento A
- Mascota Peleadora  
Scope Departamento A





# RBAC

## Usuarios y Roles

Scope del Rol: 'Global'

Nombre del Rol: 'Mascota Confianzada'

Permisos:

- 'Tomar Agua'

# RBAC

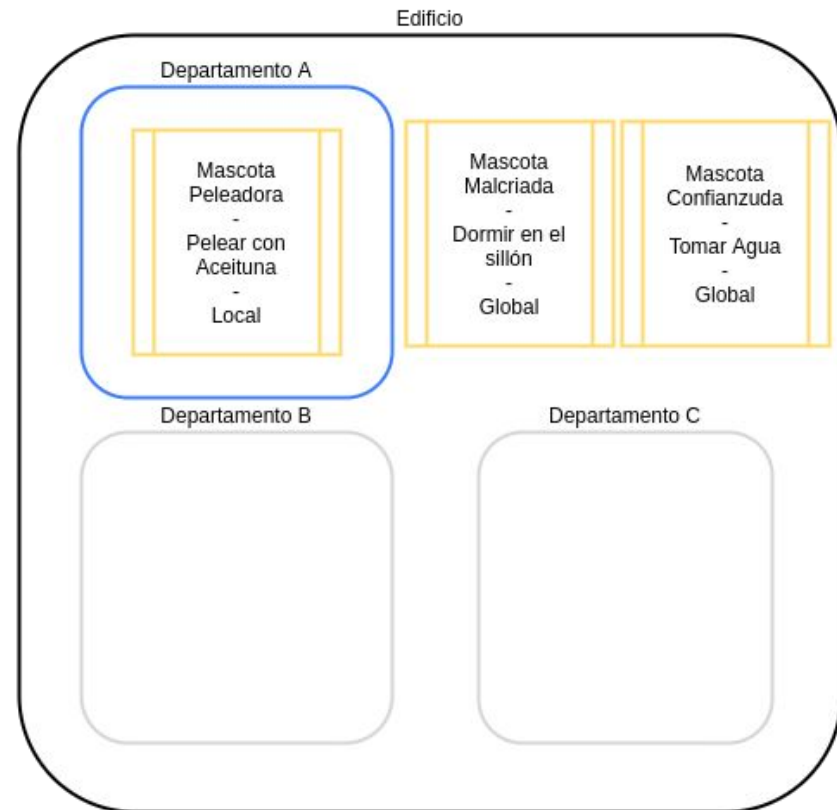
## Usuarios y Roles



Usuario: Pochoclo

Roles:

- Mascota Malcriada  
Scope: Departamento A
- Mascota Peleadora  
Scope Departamento A
- Mascota confianzuda  
Scope: Global





# RBAC

## Usuarios y Roles



### **Roles Globales:**

Disponibles en todos los departamentos del edificio.

Pueden ser asumidos en departamentos específicos (localmente) o para todos a la vez (globalmente).

### **Roles Locales:**

Disponibles sólo en un departamento y puede ser asumido sólo en ese departamento (localmente).

# RBAC

## Usuarios y Roles

### **Roles Globales:**

Disponibles en todos los ~~departamentos del edificio.~~  
namespaces del clúster.

Pueden ser asumidos ~~para departamentos~~ en  
namespaces específicos (localmente) o en todos a la  
vez (globalmente).

### **Roles Locales:**

Disponibles sólo en un ~~departamento~~ namespace y  
puede ser asumido sólo en ese ~~departamento~~  
namespace (localmente).



# RBAC

Usuarios y Roles



>> **RBAC Usuarios y Roles Demo**



# RBAC

## Usuarios y Roles



## Resumen

- Podemos crear roles globales o locales.
- Podemos asociar los roles con usuarios.
- Podemos dar permisos por acción y por recurso.



# RBAC

## ServiceAccounts y Roles



## ServiceAccounts

Los *ServiceAccounts* sirven para asociar permisos a aplicaciones que necesitan hacer uso de otros recursos del clúster a través del *APIServer*

# RBAC

## ServiceAccounts y Roles

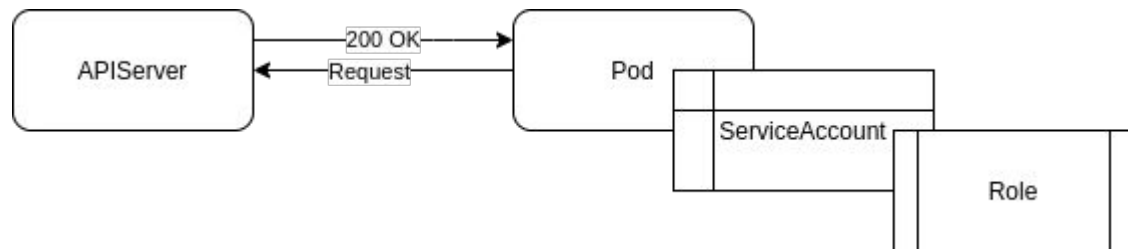
### Default ServiceAccount

- Cualquier Pod que se despliegue en el clúster por defecto va a usar un 'Default' ServiceAccount.
- Hay uno por *Namespace*
- No tiene permisos.



# RBAC

## ServiceAccounts y Roles





# RBAC

>>

**ServiceAccounts Demo**





# RBAC

## ServiceAccounts y Roles



## Recomendaciones

- No cambiar los permisos del default SA.
- Crear un SA por aplicación, haciendo uso del principio de mínimos privilegios
- Auditar el uso de permisos con herramientas como [audit2rbac](#)



# RBAC

## ServiceAccounts y Roles



## Resumen

- Un default ServiceAccount por Namespace.
- Varios Pods/Deployments pueden usar el mismo.
- Pods/Deployments pueden usar un SA a la vez.

# Audit

```
~ » kubectl logs kube-apiserver-minikube -n kube-system | grep pochoclo | jq .
{
  "kind": "Event",
  "apiVersion": "audit.k8s.io/v1",
  "level": "Metadata",
  "auditID": "ce936ebc-413c-4c34-8a5d-bbc0c6fcdabb6",
  "stage": "RequestReceived",
  "requestURI": "/api/v1/namespaces/default/secrets?limit=500",
  "verb": "list",
  "user": {
    "username": "pochoclo",
    "groups": [
      "group1",
      "system:authenticated"
    ]
  },
  "sourceIPs": [
    "192.168.49.1"
  ],
  "userAgent": "kubectl/v1.21.1 (linux/amd64) kubernetes/5e58841",
  "objectRef": {
    "resource": "secrets",
    "namespace": "default",
    "apiVersion": "v1"
  },
  "requestReceivedTimestamp": "2021-06-07T00:21:16.819397Z",
  "stageTimestamp": "2021-06-07T00:21:16.819397Z"
}
```



# Audit

## Verbosity



- **Metadata**  
User, timestamp, resource, verb
- **Request**  
Metadata + Body Request
- **RequestResponse**  
Metadata + Body Requests + Body Response



# Audit

## Verbosity



```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
  # Loguear Request response para 'pods'
  - level: RequestResponse
    resources:
      - group: ""
        resources: ["pods"]
  # Loguear sólo metadata para 'secrets'
  - level: Metadata
    resources:
      - group: ""
        resources: ["secrets"]
  # No loguear nada para configMaps llamados 'controller-leader'
  - level: None
    resources:
      - group: ""
        resources: ["configmaps"]
        resourceNames: ["controller-leader"]
```



# Audit

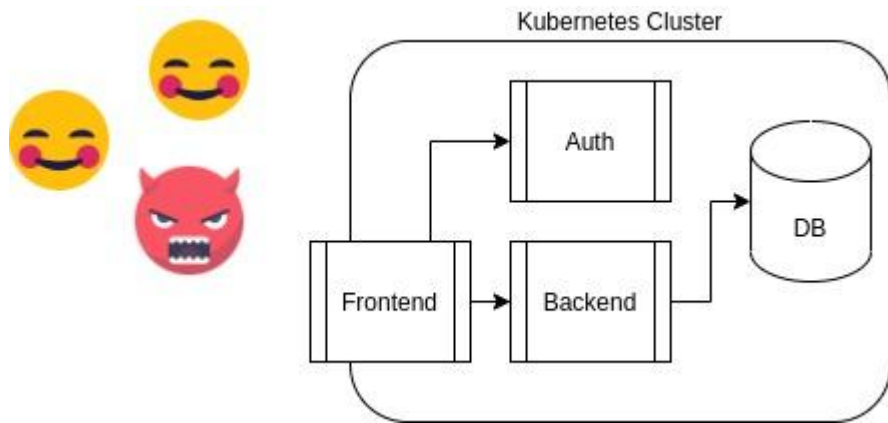
## Backends



- Log
- Webhook

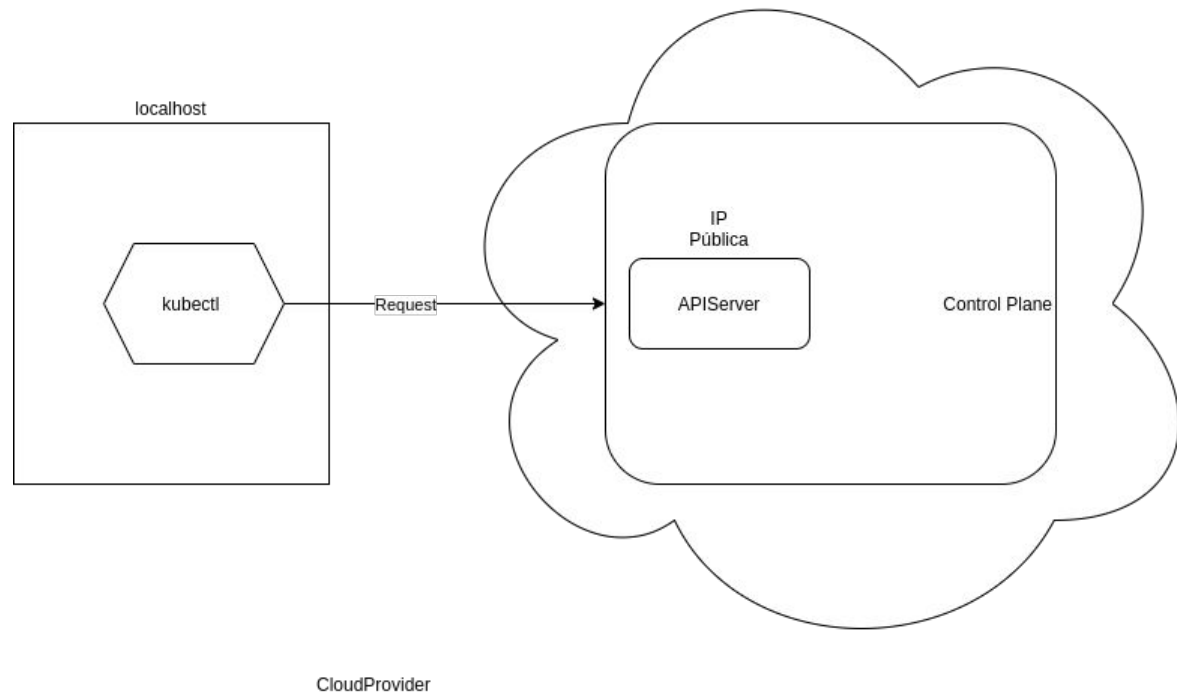


# Hardening

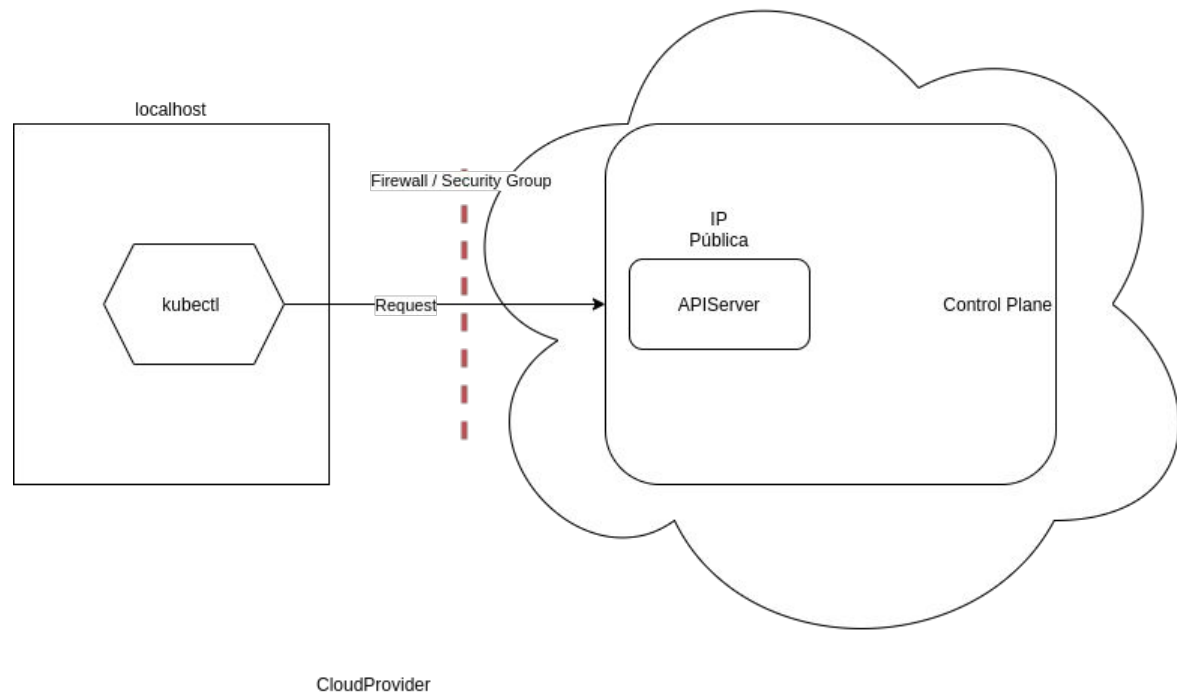


# Hardening

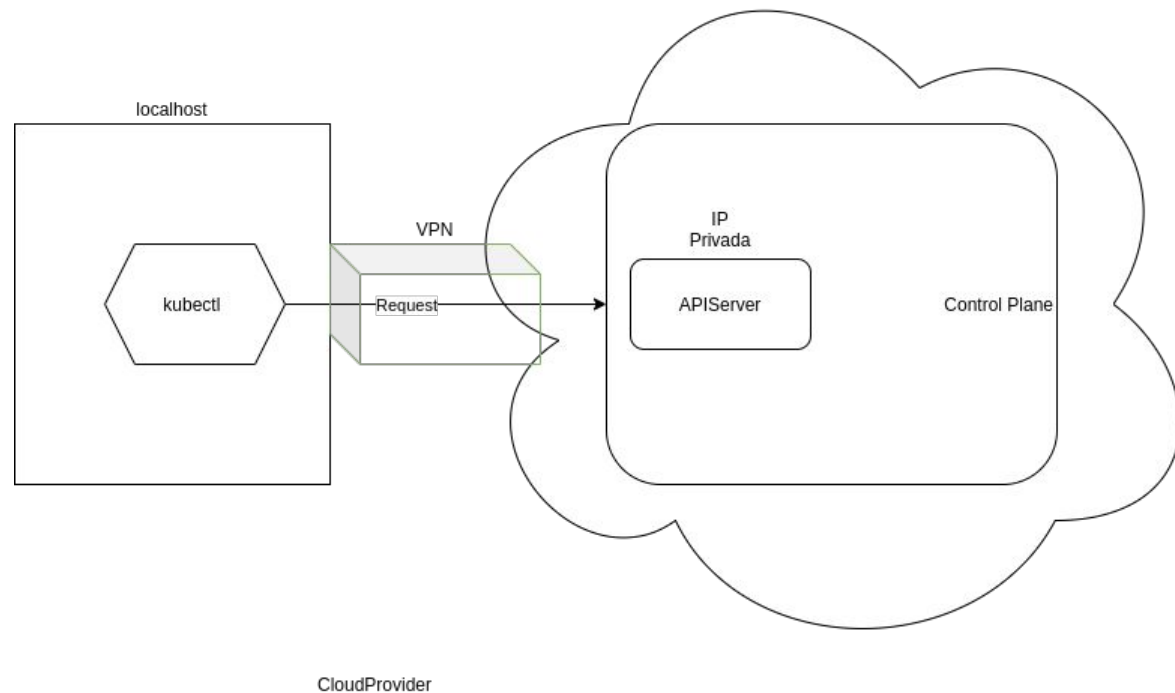
## Control Plane



# Control Plane



# Control Plane

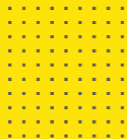




# Containers

## Non-root users.

```
spec:  
  securityContext:  
    runAsUser: 1000
```

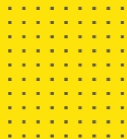




# Containers

## ReadOnly RootFilesystem

```
spec:  
  securityContext:  
    runAsUser: 1000  
    readOnlyRootFilesystem: true
```



# Containers

## !Privilege Escalation

```
spec:
  securityContext:
    runAsUser: 1000
    readOnlyRootFilesystem: true
    allowPrivilegeEscalation: false
```



# Admission Controllers

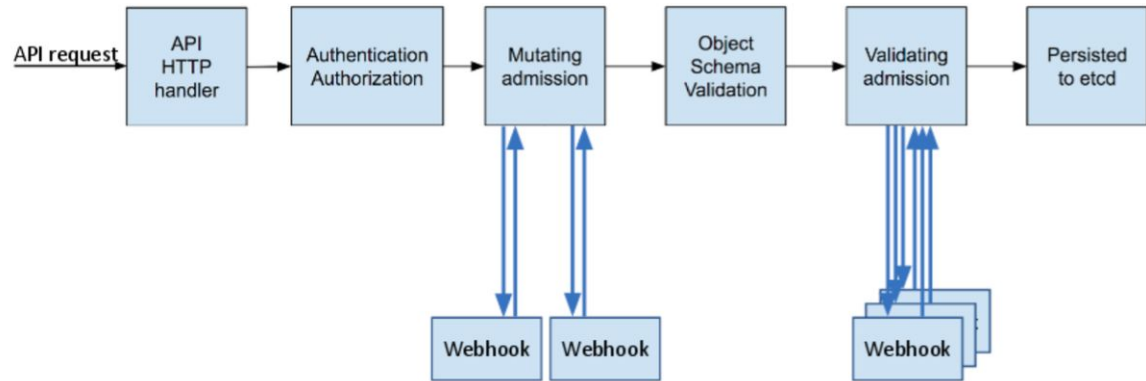


Los **Admission Controllers** son aplicaciones que controlan el despliegue de recursos en un cluster en base a determinadas reglas.

Su funcionamiento se basa en dos recursos de Kubernetes: **ValidationWebhooks** y **MutationWebhooks**



# Admission Controllers



# Admission Controllers

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
```

```
...
```

```
webhooks:
```

```
- name: my-webhook.example.com
```

```
  rules:
```

```
    - operations: ["CREATE", "UPDATE"]
```

```
      apiGroups: ["apps"]
```

```
      apiVersions: ["v1", "v1beta1"]
```

```
      resources: ["deployments", "replicasets"]
```

```
      scope: "Namespaced"
```

```
...
```

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
```

```
...
```

```
webhooks:
```

```
- name: my-webhook.example.com
```

```
  rules:
```

```
    - operations: ["CREATE"]
```

```
      apiGroups: ["*"]
```

```
      apiVersions: ["*"]
```

```
      resources: ["*"]
```

```
      scope: "*"

```

```
...
```



# Admission Controllers



## Ejemplos

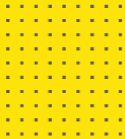
<https://github.com/FairwindsOps/polaris>

[https://github.com/cargaona/image-cloner/blob/master/pkg/webhooks/validate\\_daemonset\\_webhook.go](https://github.com/cargaona/image-cloner/blob/master/pkg/webhooks/validate_daemonset_webhook.go)



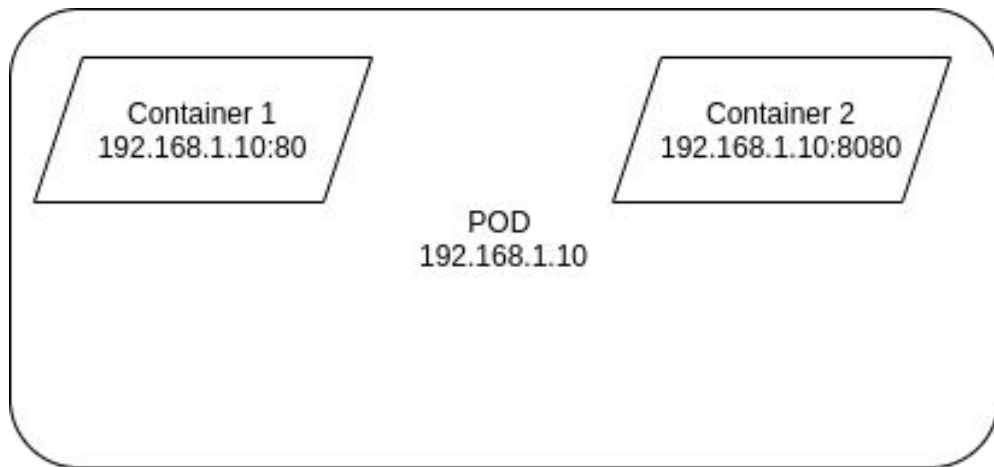
# Networking

## Kubernetes Networking Model

- Container-To-Container (Mismo pod)
  - Pod-to-Pod
  - Pod-to-Service
- 

# Networking

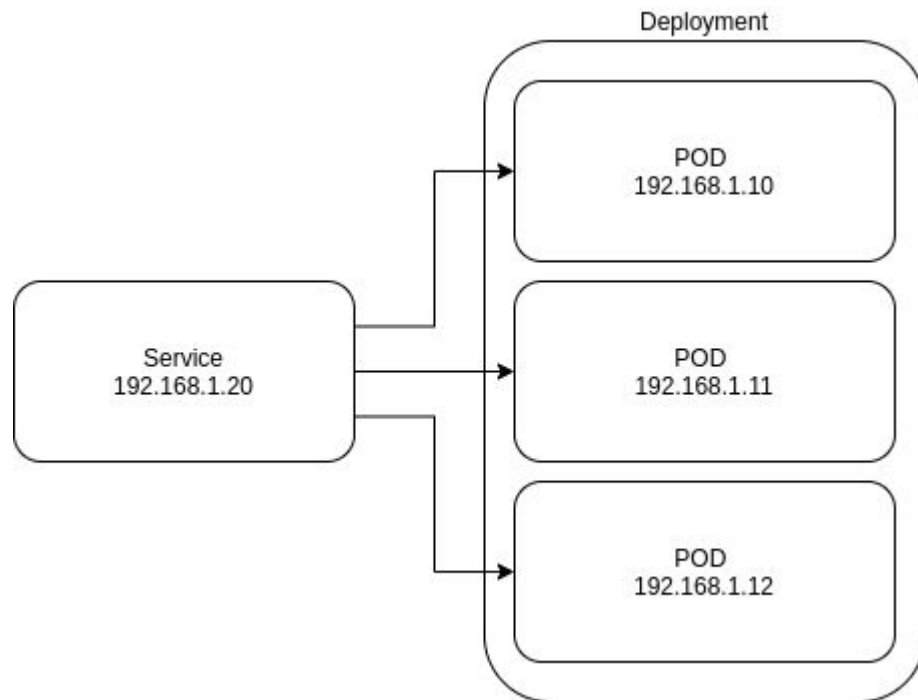
## Container to Container



# Networking

## Service

- **NodePort**
- **ClusterIP**
- **LoadBalancer**





# Networking



>> **Kubernetes Networking Demo**



# Networking



## Resumen

- Múltiples soluciones de Networking (Calico, Cilium, Flannel, etc)
- Múltiples metodologías (iptables/virt/ebpf)
- Containers comparten IP y se comunican por localhost.
- Services sirven como 'load-balancers' dentro del cluster.
- Por defecto, todos los pods pueden alcanzar a otros pods en cualquier parte del clúster.





# **Network Policies**



**>> Kubernetes Network Policies Demo**



# Network Policies



## Resumen

- Nos dejan controlar tráfico entre Pods.
- Nos dejan controlar tráfico entre Pods según su namespace.
- Nos dejan controlar tráfico entre Pods/Services según su puerto.
- Por defecto, una Policy vacía es un DenyAll.
- Por el momento, sólo podemos agregar AllowRules.
- No nos dejan auditar conexiones aceptadas o negadas.

.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....

.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....

>> Preguntas