Curso de kubernetes

Septiembre 2020

Unidad 6

Administración de kubernetes

Índice

Cuestiones previas

- Integración continua en k8s: Argo
  - https://argoproj.github.io/
  - CNCF incubating project
  - https://blog.argoproj.io/introducing-argo-rollouts-59dd0fad476c
- Software para repositorio propio de Helm: harbor o chartmuseum
- Aplicación nativa cloud no necesita almacenamiento permanente
  - El almacenamiento permanente en IaaS o contenedores se considera una solución intermedia
  - Utiliza almacenamiento de objetos si es necesario
  - Quizás no es adecuada una aplicación nativa cloud o de microservicios

Kubernetes en producción

¿Cluster de clusters?

- Un cluster de k8s se puede dividir en múltiples clusters virtuales mediante el uso de espacios de nombres y así dividir diferentes proyectos
- Alternativamente se puede crear un cluster independiente para cada proyecto
- Un nodo de k8s puede ser una máquina física, virtual o instancia en nube. Cada caso tiene sus ventajas e inconvenientes
- Es necesario usar uno o varios métodos de instalación y actualización

Kubernetes from scratch

- Las distribuciones no proporcionan paquetes.
  - No, Debian parece que tampoco: https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=970717
- Kubernetes incluye cientos de dependencias de terceros:
  - https://sources.debian.org/src/kubernetes/1.19.3-1/vendor/
- El proyecto proporciona binarios que se pueden usar directamente
- Hay que gestionar directamente los parches de seguridad/actualizaciones
  - La versión 1.18 actual es la 1.18.10
  - La versión 1.18.1 se publicó el 8 de Abril de 2020
- Sí puede ser una opción para entornos en producción desarrollando herramientas auxiliares

Kubernetes. Distribuciones e instaladores

- Queremos instalar kubernetes en entornos en producción en nube privada, virtualización tradicional o máquinas físicas
- Descartamos proveedores de nube pública
- Descartamos las soluciones que no sean software libre
- Seleccionamos las siguientes opciones de landscape.cncf.io

Canonical Charmed Kubernetes

- https://ubuntu.com/kubernetes/features
- Orientado a usar el ecosistema de Canonical: MaaS, Juju, ...
- Desarrollado por Canonical

Gravity

- https://github.com/gravitational/gravity
- Desarrollado por Gravitational Inc.
- Sistema propio de instalación actualización desarrollado en Go
- Implementa Teleport: ssh gateway para usuarios y soporte remoto
- Enfoque “open core”: Gravity Enterprise/Gravity Community

k3s

- https://github.com/rancher/k3s
- Desarrollado por Rancher, pero adoptado por CNCF
- Orientado a usar en Edge Computing. Permite montar clusters en equipos con muy pocos recursos, x86 o arm
- CNI: Flannel con VXLAN (opcionalmente canal o calico)
- Almacenamiento local de volúmenes
- Puede utilizar otro almacén diferente a etcd: SQLite, PostgreSQL, … [1]

Kubespray

- https://github.com/kubernetes-sigs/kubespray
- Desarrollado por CNCF
- Para nube pública o servidores físicos (¿Virtuales?)
- Conjunto de playbooks de Ansible
- CNI: cni-plugins, calico, canal, cilium, contiv, flanneld, kube-ovn, kube-router, multus, ovn4nfv, weave

Magnum. Integración en OpenStack

- Kubernetes as a Service (COE as a Service)
- Cada usuario de OpenStack puede desplegar su propio cluster de k8s
- Se puede integrar la autenticación de k8s con keystone a través de tokens:
  https://github.com/dims/k8s-keystone-auth/blob/master/kube-system/k8s-keystone-auth.yaml
  https://git.garr.it/cloud/charms/kubernetes-keystone
- No sólo k8s. También soporta Swarm y Mesos

Metalk8s

- https://github.com/scality/metalk8s
- Desarrollado por Scality (RING: almacanamiento de objetos)
- Orientada al despliegue en servidores físicos (baremetal)
- Desarrollado en Python
- CNI: Calico

OpenShift (OKD)

- https://github.com/openshift/okd
- Orientada al desarrollo y puesta en producción (PaaS)
- Creación y ciclo de vida completo de aplicaciones sobre contenedores
- OpenShift SDN: OpenvSwith

RKE: Rancher Kubernetes Engine

- https://github.com/rancher/rke
- Distribución que se ejecuta completamente sobre docker
- Desarrollada por Rancher Labs
- Escrito en Go
- Gestionada normalmente con Rancher
- CNI: Flannel, Calico, Canal o Weave

Suse CaaS Platform

- https://www.suse.com/es-es/products/caas-platform/
- Desarrollada por Suse
- Escrito en Go
- Implementa Cilium (CNI basado en eBGP)

Otras opciones a considerar

- Kubevirt: Orquestación de contenedores y MV
- Contenedores “seguros” con características de MV (MicroMV):
  - Kata containers
  - Firecracker

Espacios de nombres

Espacios de nombres (*namespaces*)

Los Namespaces nos permiten aislar recursos para el uso por los distintos usuarios del cluster, para trabajar en distintos proyectos. A cada namespace se le puede asignar una cuota y definir reglas y políticas de acceso.

```
kubectl get ns
NAME              STATUS   AGE
default           Active   24d
kube-node-lease   Active   24d
kube-public       Active   24d
kube-system       Active   24d
```

- Por defecto se utiliza el espacio de nombres default
- Se pueden definir nuevos espacios de nombres directamente o desde un fichero yaml:

```
kubectl create ns <nombre>
```

```yaml
apiVersion: v1
kind: Namespace
metadata:
 name: <nombre>
```

- Se pueden crear nuevos recursos asociados a un espacio de nombres:

```
kubectl create deployment nginx --image=nginx -n <nombre>
```

- Cuando se borra un espacio de nombres, se eliminan todos los recursos asociados (los volúmenes no están asociados a ningún espacio de nombres)

Usuarios

- Es conveniente definir usuarios diferentes al admin, incluso cuando sólo se utilice el espacio de nombres default
- Se pueden definir usuarios y grupos:

user:jose
group: dev, tech-lead

user: juan
group: dev

user: alberto
group: sre, tech-lead

user: marta
group: sre, devops

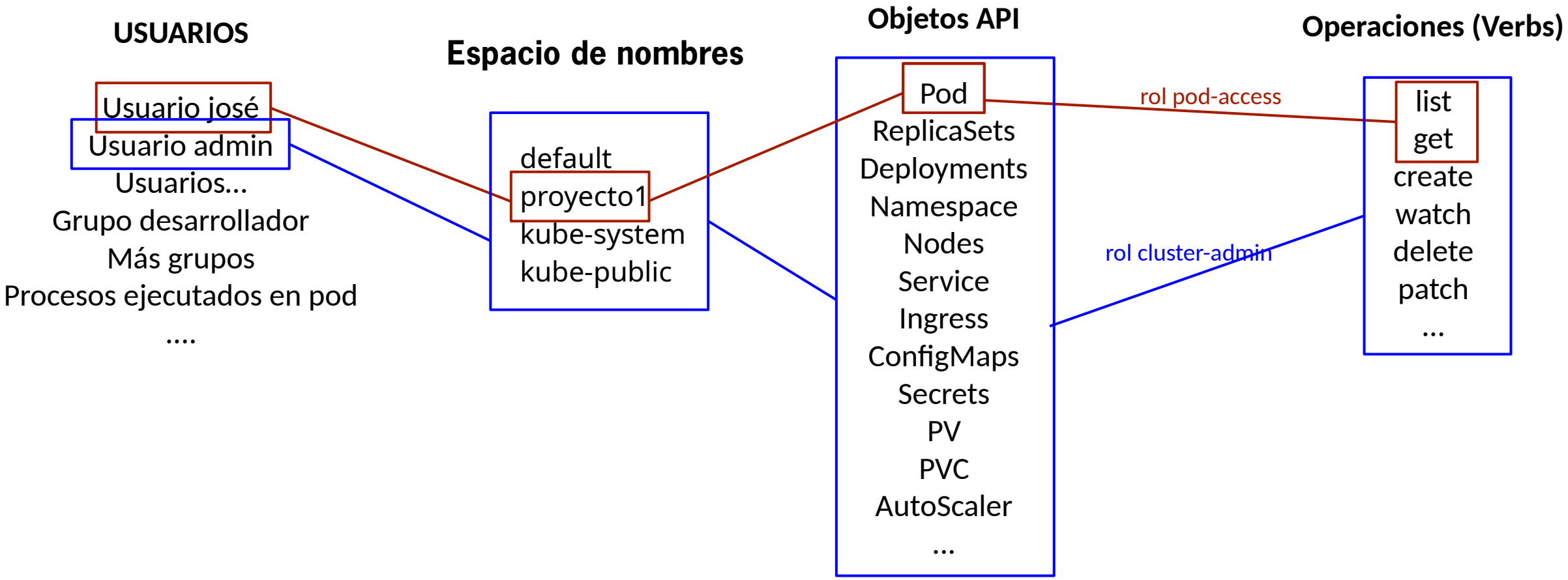
- No existe algo similar a: ~~`kubectl create user pepe`~~
- Se gestionan los usuarios mediante:
  - Certificados X.509
  - Tokens
  - Autenticación básica
  - oAuth2

Autenticación con certificados X.509

- Se configura una autoridad certificadora que se usa para firmar
- No hay ubicación estándar para los certificados, /etc/kubernetes/pki, /var/lib/kubernetes/, ...
- Dos campos esenciales:
  - CN (Common Name): k8s lo utiliza para el usuario
  - O (Organization): k8s lo utiliza para el grupo
- Ejemplo2:
  - Creamos el usuario “usuario1” del grupo “desarrollo” y el espacio de nombres proyecto1
  - Configuramos un contexto en el cliente kubectl para que se conecte como usuario1 al espacio de nombres proyecto1
- Alternativamente se puede crear la configuración en un fichero kubeconfig y usarlo mediante la variable de entorno KUBECONFIG

RBAC

RBAC (Role Based Access Control)



RBAC. Roles y asociaciones

- Role: Conjunto de permisos (se asociarán a un espacio de nombres)
- Clusterrole: Conjunto de permisos en todo el cluster (cualquier ns)
- RoleBinding: Asociación de un rol con un usuario o grupo en un espacio de nombres
- ClusterRoleBinding: Asociación de un rol con un usuario o grupo en todo el cluster

- Los nombres deben ser “Path Segment Names”
- Los verbos están relacionados con el método de la API: “Determine the Request Verb”
- API Groups: core, apps, … vienen definidos en la API de kubernetes
- Ejemplo 3:
  - Creamos un rol para que pueda manejar despliegues
  - Asignamos ese rol al usuario1 en proyecto1
  - Alternativamente asignamos ese rol al grupo desarrollo en proyecto1

RBAC. Roles habituales

El usuario admin tiene todos los privilegios en todos los espacios de nombres porque tiene asignado el clusterrole cluster-admin (tiene que asignarse desde un nodo controlador)

```
kubectl describe clusterrole cluster-admin
Name:         cluster-admin
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
  Resources  Non-Resource URLs  Resource Names  Verbs
  ---------  -----------------  --------------  -----
  *.*        []                 []              [*]
             [*]                []              [*]
```

Podemos ver los roles y asignaciones existentes:

```
kubectl get clusterroles
kubectl get clusterrolebindings
kubectl get roles -all-namespaces
Kubectl get rolebindings --all-namespaces
```

RBAC. ServiceAccount

- Cuando quien debe autenticarse frente a la API es un proceso que se ejecuta en un Pod, se utiliza una cuenta de servicio, no de usuario
- El serviceAccountName se especifica en el pod/despliegue correspondiente
- Al definir un ServiceAccount, se genera un token que se almacena en el cluster
- Ejemplo: coredns

Cuotas y límites

Cuotas

- Por cada namespace podemos indicar una cuotas de uso de recursos utilizando el objeto ResourceQuotas. En la cuota podemos limitar el uso de:
  - Hardware: CPU y memoria
  - Recursos de la API
  - Almacenamiento
  - ...

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
 name: team-vision-resquota
 namespace: team-vision
spec:
 hard: # Quotas go here
   requests.cpu: 100
   requests.memory: 400Gi
   pods: 6
   configmaps: 5
   persistentvolumeclaims: 3
   requests.storage: 500Gi
```

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: database
spec:
 containers:
 - name: db
   image: mysql
   resources: # requests here
     requests:
       memory: "64Mi"
       cpu: "250m"
```

Límites

- Necesitamos un objeto que nos permita limitar el uso de recurso (memoria y CPU) de un pod. Vamos a usar un objeto Limits. Casos de uso:
  - Si ponemos un request.cpu=100, significa que la suma de núcleos de CPU no puede superar 100, ¿pero si queremos limitar que cualquier pod no use más de un núcleo?
  - Queremos limitar el uso de recursos:

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: database
spec:
 containers:
 - name: db
   image: mysql
   resources:
      requests:
        memory: 64Mi
        cpu: 250m
      limits:
        memory: 128Mi
        cpu: 500m
```

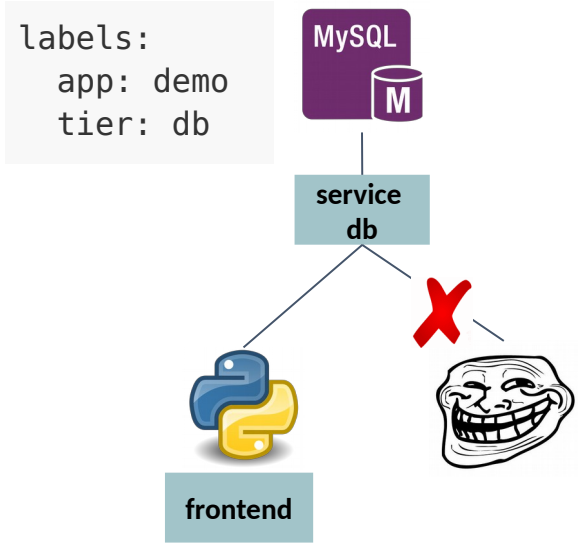
Podemos limitar el uso de recursos en el namespace:

```yaml
kind: LimitRange
metadata:
 name: mem-min-max-demo-lr
spec:
 limits:
 - max:
     cpu: 1
     memory: 512Mi
   type: Container
```

Políticas de red

Políticas de red

- Control de acceso a dirección IP (L3) o puerto (L4) de un pod
- Gestionado por el plugin de CNI (no todos la soportan) (!)
- Hay políticas de ingress y egress
- Network Policy




```yaml
apiVersion: networking.k8s.io/
v1
 kind: NetworkPolicy
 metadata:
   name: db-frontend-allow
 spec:
   podSelector:
     matchLabels:
       app: demo
       tier: db
   ingress:
     - from:
       - podSelector:
           matchLabels:
             app: demo
             tier: frontend
         ports:
         - protocol: TCP
           port: 3306
```

Los pod destinos, la base de datos

Los pod que pueden acceder, la app frontend

Puerto al que se puede conectar

CNI

Container Networking

There’s no such thing as container networking. It’s just networking.
Anyone who tells you differently is either trying to sell you something or doesn’t understand how networking works

Justin Garrison: “No SDN Kubernetes”

CNI: Container Network Interface

- Proyecto auspiciado por la CNCF
- https://github.com/containernetworking
- Interfaz única para redes de contenedores
- No solo para k8s. También lo utilizan Rocket y Mesos. Docker no
- Proporciona plugin básicos: https://github.com/containernetworking/plugins
- Otros plugins adicionales para CNI:
  - Flannel
  - Calico
  - Canal
  - Weave
  - Cilium
  - ...

CNI. Plugins

- Soluciones de SDN que añaden funcionalidad adicional
- Gestionan automáticamente el enrutamiento entre nodos
- Aislan los pods de diferentes espacios de nombres
- Normalmente utilizan una red “overlay” entre todos los nodos, con tecnologías propias o existentes (OpenvSwitch, VXLAN, túneles, etc.) ← MTU (!)
- Implementan o no políticas de red
- Pueden implementar cifrado del tráfico
- Se pueden integrar con otros componentes, p. ej. Istio
- El ecosistema de kubernetes es amplísimo, el tiempo determinará qué opciones permanecen

CNI. Flannel

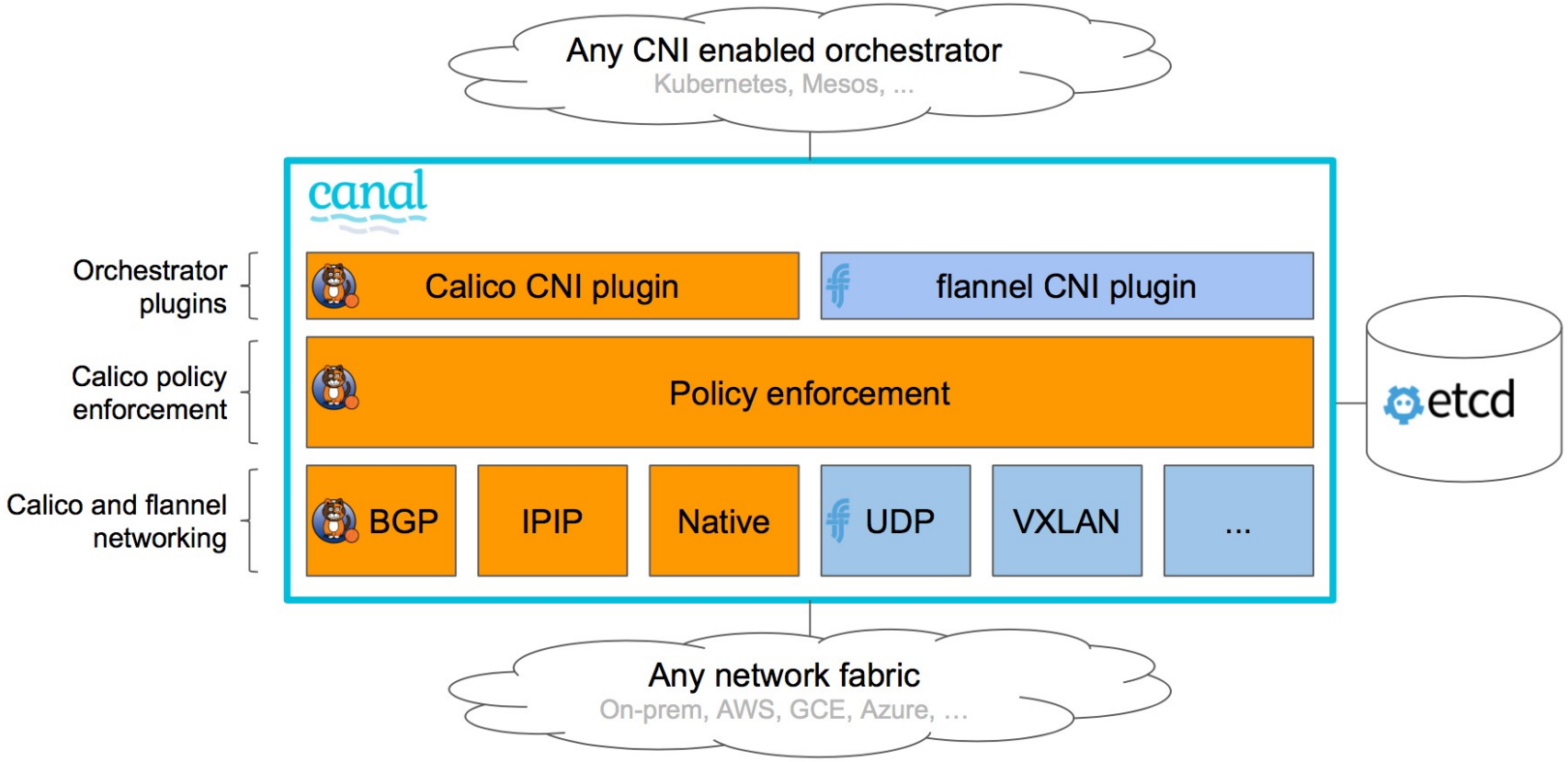
- Desarrollo original de CoreOS
- Proyecto maduro
- Utiliza etcd como almacén. Integración simple con k8s
- L3
- Red overlay. Varios modos, pero habitualmente VXLAN
- No implementa políticas de red

CNI. Calico

- L3
- Enrutamiento dinámico con BIRD/BGP
- eBPF
- Existe versión “Enterprise”
- Red overlay. Varios modos, principalmente VXLAN
- Implementa políticas de red
- Se le achaca mayor complejidad
- Buena integración con Istio

CNI. Canal

- Fusión de Flannel y Calico
- Simpleza de flannel con seguridad de Calico



CNI. Weave

- Instalación muy sencilla
- VXLAN con OpenvSwitch
- Política de red
- Cifrado completo opcional

CNI. Comparativa de plugins

Benchmark results of Kubernetes network plugins