1. Instala Multus (opcional, para múltiples redes)

Multus permite que un pod tenga más de una interfaz de red (multi-homed), útil para appliances, firewalls, balanceadores, gateways, etc. Instálalo si necesitas conectar pods a varias redes físicas o VLANs (por ejemplo, mediante bridges y NADs).

Instalación:

```
kubectl apply -f https://raw.githubusercontent.com/k8snetworkplumbingwg/multus-cni/master/deployments/multus-daemonset.yml
```

Verifica:

```
kubectl get pods -n kube-system | grep multus
```

2. (Opcional) Quita el taint del nodo master para poder programar pods en él

Por defecto, Kubernetes no programa pods de usuario en el nodo principal (control-plane). Elimina este bloqueo para poder desplegar aplicaciones ahí.

```
kubectl taint nodes --all node-role.kubernetes.io/control-plane-
kubectl taint nodes --all node-role.kubernetes.io/master-
```

Nota: Uso de taints en nodos control-plane (alta disponibilidad y ejecución de cargas)

¿Qué es un taint?

- Un **taint** en Kubernetes es una marca especial que se pone a un nodo para **evitar que los pods se programen ahí**, salvo que declaren una "toleration" explícita.
- Se usa para reservar nodos solo para tareas especiales (por ejemplo, el control-plane).
- Por defecto, los nodos control-plane llevan un taint:
    - `node-role.kubernetes.io/control-plane:NoSchedule`

¿Por qué quitar el taint?

- Si quieres que **los nodos control-plane puedan ejecutar pods de usuario** (además del plano de control), necesitas **quitar el taint**.
- Esto es común en clústeres pequeños o medianos, donde **todos los nodos cumplen doble función** (alta disponibilidad y ejecución de cargas).

Comandos para quitar el taint de todos los nodos control-plane:

```
kubectl taint nodes --all node-role.kubernetes.io/control-plane-
kubectl taint nodes --all node-role.kubernetes.io/master-
```

- El `-` final indica "quitar".
- Ejecuta ambos para máxima compatibilidad entre versiones.

Comando para añadir el taint (dejar el nodo solo como control-plane):

```
kubectl taint nodes NOMBRE_DEL_NODO node-role.kubernetes.io/control-
plane=:NoSchedule
```

- Así, ese nodo **solo ejecuta el plano de control** (salvo pods con toleration específica).

---

3. Test rápido de Multus (NAD + pod con 2 interfaces)

Puedes comprobar que Multus funciona creando una red secundaria y un pod de prueba con dos interfaces (una por defecto, una secundaria).

En la carpeta `multus/` de tu repositorio, debes tener:

- `multus/nad-br-servicios.yaml` (NetworkAttachmentDefinition)
- `multus/test-multus-pod.yaml` (pod Alpine multi-homed)

Despliega la NAD:

```
kubectl apply -f multus/nad-br-servicios.yaml
```

Despliega el pod de test:

```
kubectl apply -f multus/test-multus-pod.yaml
```

Comprueba las interfaces:

```
kubectl exec -it multus-test -- sh
ip a
```

- El pod debe mostrar una interfaz extra (además de la de Flannel), conectada a tu red secundaria (`br-servicios`, etc.).

Para limpiar:

```
kubectl delete pod multus-test
```

> **Nota:** Puedes crear tantas NADs (NetworkAttachmentDefinition) como bridges/VLANs quieras conectar a pods específicos (con Multus), ideal para appliances de red, gateways, SDN, pruebas de seguridad, etc.

---

4. Instalación y configuración de MetalLB (LoadBalancer local)

MetalLB permite asignar IPs flotantes de tu red LAN a servicios `LoadBalancer`, igual que hacen los clústeres en la nube. Es fundamental si quieres exponer servicios como ingress, dashboards, etc., accesibles desde tu red local.

---

a) Instala MetalLB

```
kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.14.5/config/manifests/meta
llb-native.yaml
```

Esto crea el namespace `metallb-system` y los pods necesarios.

---

b) Declara múltiples pools de IP

Puedes crear varios pools (por ejemplo, uno para producción y otro para test, o para diferentes VLANs/segmentos). Los pools se definen en el objeto `IPAddressPool`.

Ejemplo: `metallb/ipaddresspool.yaml`

```
```

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: pool-produccion
  namespace: metallb-system
spec:
  addresses:
    - 192.168.1.100-192.168.1.110
---
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: pool-lab
  namespace: metallb-system
spec:
  addresses:
    - 192.168.2.100-192.168.2.110
```

Ejemplo: `metallb/l2advertisement.yaml`

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: advert-all
  namespace: metallb-system
spec: {}
```

Kustomization:

```
resources:
  - ipaddresspool.yaml
  - l2advertisement.yaml
```

Para aplicar:

```
kubectl apply -k metallb/
```

---

c) Asigna un pool concreto a un Service (anotaciones)

Por defecto, los Services `LoadBalancer` usan cualquier pool disponible. Para forzar el uso de un pool específico, **añade esta anotación al Service**:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: ejemplo-prod
  annotations:
    metallb.universe.tf/address-pool: pool-produccion
spec:
  selector:
    app: ejemplo
  ports:
    - port: 80
      targetPort: 80
  type: LoadBalancer
```

- Cambia `pool-produccion` por el nombre de pool que quieras usar.

Al crear el Service, MetalLB le asignará una IP **solo de ese pool**.

---

d) Comprobar el resultado

```
kubectl get svc
```

Verás la IP asignada en la columna `EXTERNAL-IP`.

> **Notas:**
>
> - Puedes definir tantos pools como necesites, uno por segmento/VLAN/uso.
> - Puedes versionar los manifiestos de MetalLB en una carpeta específica del repositorio (`metallb/`).
