

Setting up Nginx as a Reverse Proxy for Apache using K8s (istio enabled)

1. Instalar kind:

<https://kind.sigs.k8s.io/docs/user/quick-start/#installation>

2. Crear archivo de configuración de cluster (cluster-config.yaml):

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  extraPortMappings:
  - containerPort: 30000
    hostPort: 30000
    protocol: TCP
```

NOTAS:

- el extraPortMappings es importante para entornos que usen Docker Desktop, porque sino la configuración de kind impide luego acceder a contenidos del servidor Apache (u otro) a través de localhost:30000 (u otro puerto mayor que el 30000). Para más documentación: <https://kind.sigs.k8s.io/docs/user/configuration/#extra-port-mappings>

- Se podrían añadir workers con la línea: `-role: worker`

3. Crear el clúster y comprobar estado:

```
kind create cluster --config=cluster-config.yaml --name=trainingPath
```

```
kubectl cluster-info
```

NOTA: para lo último, debería salir algo así:

```
tracert@DESKTOP-H70CEFS:/mnt/f/EscritorioPC/Drive_Accenture/trainingPathProject$ kubectl cluster-info
Kubernetes control plane is running at https://127.0.0.1:32769
KubeDNS is running at https://127.0.0.1:32769/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

4. Archivos de configuración de los pods (Apache y Nginx):

NOTA: Tanto el service como el deployment están en el mismo yaml, para ahorrar en kubectl apply -f. Se podría usar kustomize también para ahorrar tiempo de despliegue.

nginx.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  type: NodePort
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
    nodePort: 30000
  selector:
    app: nginx
---
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
          volumeMounts:
            - name: nginx-conf
              mountPath: /etc/nginx/
      volumes:
        - name: nginx-conf
          configMap:
            name: nginx-conf
```

apache-yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: apache
  labels:
    app: apache
spec:
  ports:
    - name: http
      port: 8080
      targetPort: 80
  selector:
    app: apache
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apache
spec:
  selector:
    matchLabels:
      app: apache
  replicas: 1
  template:
    metadata:
      labels:
        app: apache
    spec:
      containers:
        - name: apache
          image: httpd:latest
          ports:
            - containerPort: 80
          volumeMounts:
            - name: html
```

```
    mountPath: /usr/local/apache2/htdocs/
  volumes:
  - name: html
    configMap:
      name: html
```

5. Crear el ConfigMap para configurar nginx como reverse proxy (nginx-conf.yaml)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-conf
data:
  nginx.conf: |
    events {
      worker_connections 1024;
    }

    http {
      server {
        listen 80;
        server_name localhost;

        location / {
          index index.html;
          proxy_pass http://apache:8080/;
          proxy_set_header Host $host;
          proxy_set_header X-Real-IP $remote_addr;
          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        }
      }
    }
  }
```

NOTA:

- La directiva `worker_connections` controla el número máximo de conexiones simultáneas que pueden ser manejadas por un worker process de Nginx. Se debe escalar al tráfico, 1024 es un ejemplo.
- Está mapeado el puerto 80 de apache al 8080 para evitar conflictos con el pod de nginx.

6. Aplicar configuraciones al clúster:

```
kubectl apply -f nginx.yaml
kubectl apply -f apache.yaml
kubectl apply -f nginx-conf.yaml
```

NOTA: nos debería aparecer esto:

```
tracert@DESKTOP-H70CEFS:/mnt/f/EscritorioPC/Drive_Accenture/trainingPathProject$ kubectl apply -f nginx.yaml
service/nginx created
deployment.apps/nginx created
tracert@DESKTOP-H70CEFS:/mnt/f/EscritorioPC/Drive_Accenture/trainingPathProject$ kubectl apply -f apache.yaml
service/apache created
deployment.apps/apache created
tracert@DESKTOP-H70CEFS:/mnt/f/EscritorioPC/Drive_Accenture/trainingPathProject$ kubectl apply -f nginx-conf.yaml
configmap/nginx-conf created
```

Tras un `kubectl get pods`:

```
tracert@DESKTOP-H70CEFS:/mnt/f/EscritorioPC/Drive_Accenture/trainingPathProject$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
apache-8444cf55c5-lxdbf             1/1     Running   0           39m
nginx-dc584d69f-dnk72               1/1     Running   0           95s
```

Importante comprobar que `nginx.conf` está bien configurado:

`kubectl exec <nombre de tu pod de nginx> -- cat /etc/nginx/nginx.conf`

```
tracert@DESKTOP-H70CEFS:/mnt/f/EscritorioPC/Drive_Accenture/trainingPathProject$ kubectl exec nginx-dc584d69f-wddrw -- cat /etc/nginx/nginx.conf
events {
    worker_connections 1024;
}

http {
    server {
        listen 80;
        server_name localhost;

        location / {
            index index.html;
            proxy_pass http://apache:8080/;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        }
    }
}
```

7. Si tenemos un HTML personalizado lo podemos usar para verlo a través de nginx y asegurarnos de que todo está funcionando bien hasta aquí.

Los ficheros que dejo se llaman `custom.html` y `custom.css`, pero pueden ser cualquier otros. Es un html de Dragon ball que están bastante guay.

- Creamos el ConfigMap de los ficheros html y css:

```
kubectl create configmap html --from-file=custom.html --from-file=custom.css
```

NOTA: También se deja el fichero `html.yaml` como ConfigMap a aplicar con `kubectl apply -f html.yaml`. Es equivalente al comando anterior.

- Actualizamos el `apache.yaml`, creando un punto de montaje y especificando el volumen a utilizar:

```
...
spec:
  containers:
  - name: apache
    image: httpd:latest
    ports:
    - containerPort: 80
    volumeMounts:
    - name: html
      mountPath: /usr/local/apache2/htdocs/
  volumes:
  - name: html
    configMap:
      name: html
```

```
spec:
  containers:
  - name: apache
    image: httpd:latest
    ports:
    - containerPort: 80
    volumeMounts:
    - name: html
      mountPath: /usr/local/apache2/htdocs/
  volumes:
  - name: html
    configMap:
      name: html
```

- Aplicamos las configuraciones:

```
kubectl apply -f apache.yaml
```

- Comprobamos que todo se ha cargado y va correctamente el reverse proxy:

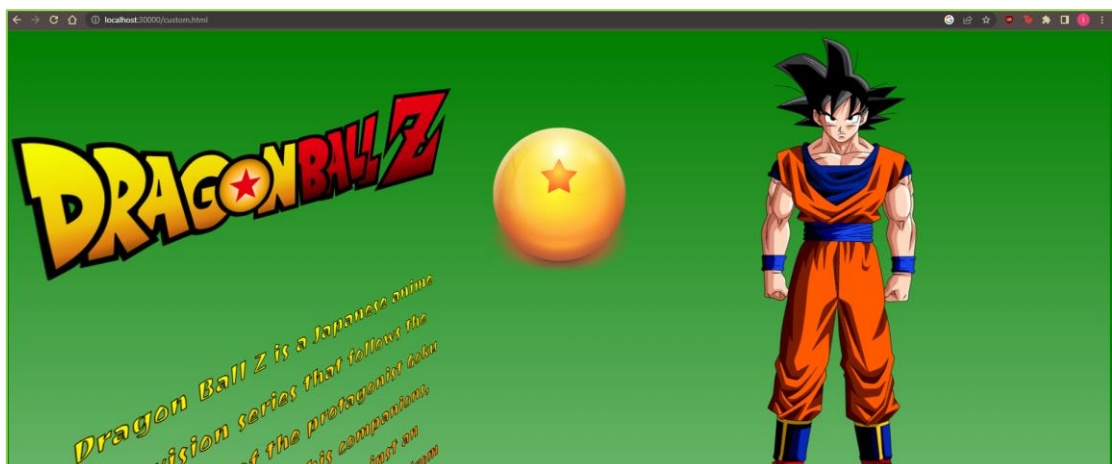
```
kubectl exec <nombre de tu pod de nginx> -- curl localhost/custom.html
```

```
tracert@DESKTOP-H70CEFS:/mnt/f/EscritorioPC/Drive_Accenture/trainingPathProject$ kubectl exec nginx-dc584d69f-wddrw -- curl
localhost/custom.html
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total   Spent    Left   Speed
  0     0    0     0     0     0     0     0     0     0  --:--:-- --:--:-- --:--:--    0<head>
<link href="https://fonts.googleapis.com/css?family=Ravi+Prakash" rel="stylesheet">
<link href="custom.css" rel="stylesheet" type="text/css"/>
</head>
<div class="grad">
  <div class="container1">
    <div class="left">
      <div class="scroll">
        
        <p>Dragon Ball Z is a Japanese anime television series that follows the adventures of the protagonist Goku who, along
with his companions, defends the Earth against an assortment of villains ranging from intergalactic space fighters and con
querors, unnaturally powerful androids and nearly indestructible creatures. </p>
        <p> Work for Training Path</p>
        <p> Enrique Fluxia - Ivan Perez <p>
      </div>
    </div>
  </div>
  <div class="container2">
    <div class="goku">
    </div>
  </div>
</div>
100   939  100   939    0     0   916k    0  --:--:-- --:--:-- --:--:--   916k
```

8. Hasta aquí deberíamos tener esto configurado así:

```
tracert@DESKTOP-H70CEFS:/mnt/f/EscritorioPC/Drive_Accenture/trainingPathProject$ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
apache-8465855477-8qzzk  1/1     Running   0          22m
nginx-dc584d69f-wddrw    1/1     Running   0          5m
tracert@DESKTOP-H70CEFS:/mnt/f/EscritorioPC/Drive_Accenture/trainingPathProject$ kubectl get services
NAME                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
apache              ClusterIP   10.96.122.45  <none>        8080/TCP         30m
kubernetes           ClusterIP   10.96.0.1     <none>        443/TCP          39m
nginx               NodePort    10.96.185.1   <none>        80:30000/TCP     20m
tracert@DESKTOP-H70CEFS:/mnt/f/EscritorioPC/Drive_Accenture/trainingPathProject$ kubectl get configmaps
NAME    DATA   AGE
html    2       22m
nginx-conf  1       30m
tracert@DESKTOP-H70CEFS:/mnt/f/EscritorioPC/Drive_Accenture/trainingPathProject$ kubectl exec nginx-dc584d69f-wddrw -- curl
localhost
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total   Spent    Left   Speed
  0     0    0     0     0     0     0     0     0     0  --:--:-- --:--:-- --:--:--    0<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Fi
nal//EN">
<html>
  <head>
    <title>Index of </title>
  </head>
  <body>
    <h1>Index of </h1>
    <ul><li><a href="..2023_03_08_12_26_47.776213457/"> ..2023_03_08_12_26_47.776213457/</a></li>
    <li><a href="..data/"> ..data/</a></li>
    <li><a href="custom.css"> custom.css</a></li>
    <li><a href="custom.html"> custom.html</a></li>
    </ul>
  </body></html>
100   385  100   385    0     0   375k    0  --:--:-- --:--:-- --:--:--   375k
```

Y a través de localhost:30000 accederíamos a nuestra página web custom.html:



9. Si hasta aquí todo ha ido bien, genial, es lo que debería ser. Sino, he dejado un fichero `kustomization.yaml` que utiliza `kustomize` para levantar todo automáticamente. Si algo había ido mal, sigue los siguientes pasos:

- `kind delete cluster --name=trainingPath` (o el nombre que le hayas puesto). Si no te acuerdas, utiliza `kind get clusters`
- `kind create cluster --config=cluster-config.yaml --name=<nombre-que-quieras>`
- `kubectl apply -k .`

Todo listo. Haz las comprobaciones anteriores (`nginx.conf`, `localhost:30000`, etc.)

NOTA:

- Cuidado con copiar y pegar, los guiones no siempre se copian bien.
- Ya sé que hubiera venido bien esto antes, pero tenemos que repasar todo desde el principio 😊.

----- K8S DASHBOARD -----

Lo que se comenta en esta sección no es la configuración de Istio, sino que es la configuración del dashboard de K8s para `kind`. Está muy bien para ver métricas y ver el estado del clúster (más info en <https://istio.io/latest/docs/setup/platform-setup/kind/#setup-dashboard-ui-for-kind>):

1. Desplegamos el deployment del dashboard:

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml
(si te da algún error añade --validate=false)
```

2. Verificamos que está disponible el pod y que se han creado bien:

```
kubectl get pod -n kubernetes-dashboard
```

```
tracert@DESKTOP-H70CEFS:/mnt/f/EscritorioPC/Drive_Accenture/trainingPathProject$ kubectl get deployments -n kubernetes-dashboard
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
dashboard-metrics-scraper          1/1     1             1           2m9s
kubernetes-dashboard               1/1     1             1           2m9s
```

3. Creamos un `ServiceAccount` y un `ClusterRoleBinding` para dar accesos de administrador al clúster:

```
kubectl create serviceaccount -n kubernetes-dashboard admin-user
```

```
kubectl create clusterrolebinding -n kubernetes-dashboard admin-user --clusterrole
cluster-admin --serviceaccount=kubernetes-dashboard:admin-user
```

4. Generamos el token que nos pedirá luego para iniciar sesión:

```
token=$(kubectl -n kubernetes-dashboard describe secret $(kubectl -n kubernetes-
dashboard get secret | awk '/^admin-user/{print $1}')) | awk '{ $1=="token:"{print $2}' }
```

5. Comprobamos que se ha almacenado bien en la variable `token`:

```
echo $token
```

6. Podemos acceder al Dashboard por CLI escribiendo:

```
kubectl proxy
```

Ahora podemos acceder desde el navegador en <http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/#/login>

Kubernetes Dashboard

☒ Token

Cada cuenta de servicio tiene un Secret asociado a un Bearer Token que puede usarse para iniciar sesión en el tablero. Para saber más sobre cómo configurar y utilizar Bearer Tokens, referirse a la sección [Autenticación](#).




☐ Kubeconfig

Por favor, selecciona el fichero kubeconfig que has creado para configurar el acceso al cluster. Para encontrar más información sobre cómo configurar y usar el fichero kubeconfig, referirse a la sección [Configurar el acceso a varios clústeres](#).

Ingresar token *

Iniciar sesión

7. Accedemos al dashboard del clúster desde la web, una vez introducido el token (ej. Servicios, etc.):

Servicios				
Nombre	Etiquetas	Tipo	IP cluster	Endpoints Internos
 apache	<div>app: apache</div>	ClusterIP	10.96.158.151	apache:8080 TCP apache:0 TCP
 nginx	-	NodePort	10.96.161.199	nginx:80 TCP nginx:30000 TCP
 kubernetes	<div>component: apiserver</div> <div>provider: kubernetes</div>	ClusterIP	10.96.0.1	kubernetes:443 TCP kubernetes:0 TCP

JENKINS

1. Para completar el clúster, vamos a introducir Jenkins también. Crearemos un servicio de tipo NodePort mapeado al hostPort 30001, como veremos. El fichero yaml que contiene el deployment y el service es:

jenkins.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: jenkins
spec:
  type: NodePort
  selector:
    app: jenkins
  ports:
    - name: http
      port: 8080
      targetPort: 8080
      nodePort: 30001
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jenkins
spec:
  selector:
    matchLabels:
      app: jenkins
  replicas: 1
  template:
    metadata:
      labels:
        app: jenkins
    spec:
      containers:
        - name: jenkins
          image: jenkins/jenkins:ls
```

```
ports:
  - containerPort: 8080
  - containerPort: 50000
```

2. En este punto, nos estaremos preguntando que dónde vamos a exponer Jenkins. La idea sería hacerlo a través de nginx en "location jenkins/" y acceder a través de localhost:30000/jenkins -solo habría que modificar nginx-conf.yaml y agregar

```
location /jenkins {
    proxy_pass http://<jenkinsPodUsedIp>:30001/;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
}
```

En mi caso, no puedo hacerlo así por problemas de compatibilidad de kind con wsl2. Para ello, he modificado el fichero cluster-config.yaml, escribiendo los extraPortMappings que necesitaremos más adelante. Además, he añadido un par de workers y he añadido una etiqueta en el máster para agregar un ingress controller más adelante:

cluster-config.yaml

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  kubeadmConfigPatches:
  - |
    kind: InitConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "ingress-ready=true"
  extraPortMappings:
  - containerPort: 80
    hostPort: 80
    protocol: TCP
  - containerPort: 443
    hostPort: 443
    protocol: TCP
  - containerPort: 8080
    hostPort: 8080
    protocol: TCP
  - containerPort: 30000
    hostPort: 30000
    protocol: TCP
  - containerPort: 30001
    hostPort: 30001
    protocol: TCP
- role: worker
- role: worker
```

3. Ahora bien, puedes seguir con tu cluster y agregar Jenkins a través de nginx. Si prefieres asegurarte de que todo vaya bien, te recomiendo que utilices el fichero autoDeploy.py, que te despliega todo lo realizado hasta ahora automáticamente (nginx, apache, Jenkins y dashboard). Solo hay que ejecutar en el directorio donde están todos los ficheros yaml:

python3 autoDeploy.py

Esperamos un poco, obtenemos el token para el dashboard manualmente y con kubectl proxy accedemos a él para ver todos nuestros recursos.

4. Finalmente, podemos acceder a nginx a través de localhost:30000 y a Jenkins a través de localhost:30001. Lo de nginx ya sabemos cómo va, pasemos a Jenkins:

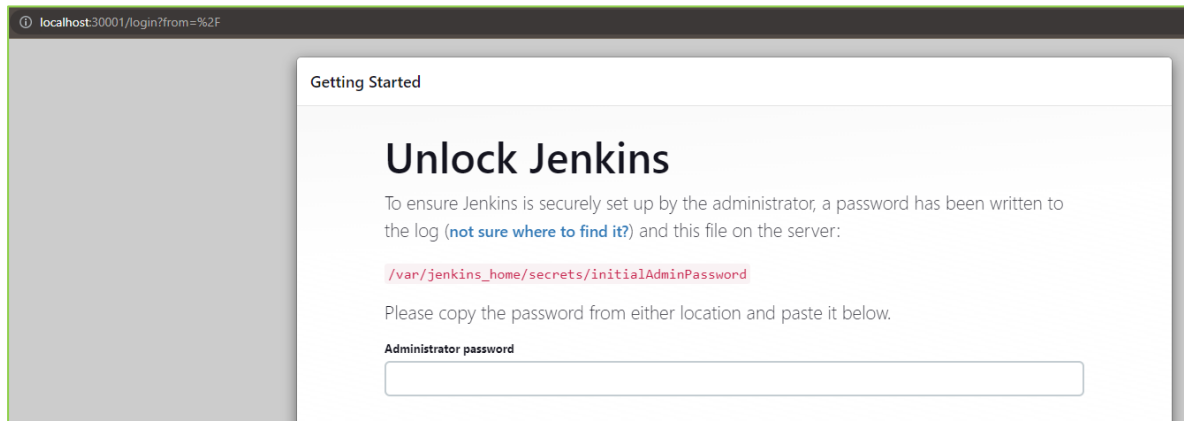
- **Unlock Jenkins:** necesitamos la contraseña inicial. Ejecutamos:

```
kubectl get pods (para obtener el nombre del pod de Jenkins)
```

```
kubectl exec <nombre-pod-jenkins> -- cat  
/var/jenkins_home/secrets/initialAdminPassword
```

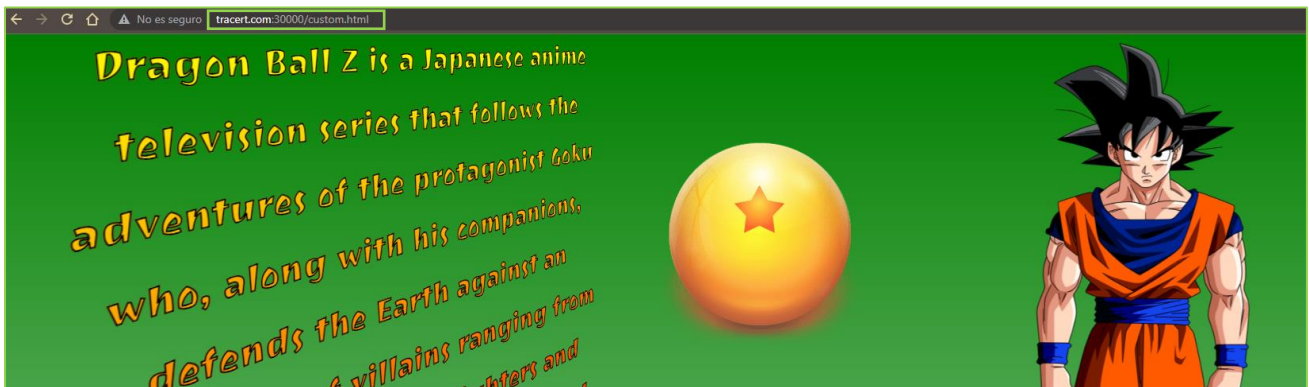
- Nos vamos a localhost:30001 e introducimos la contraseña inicial:

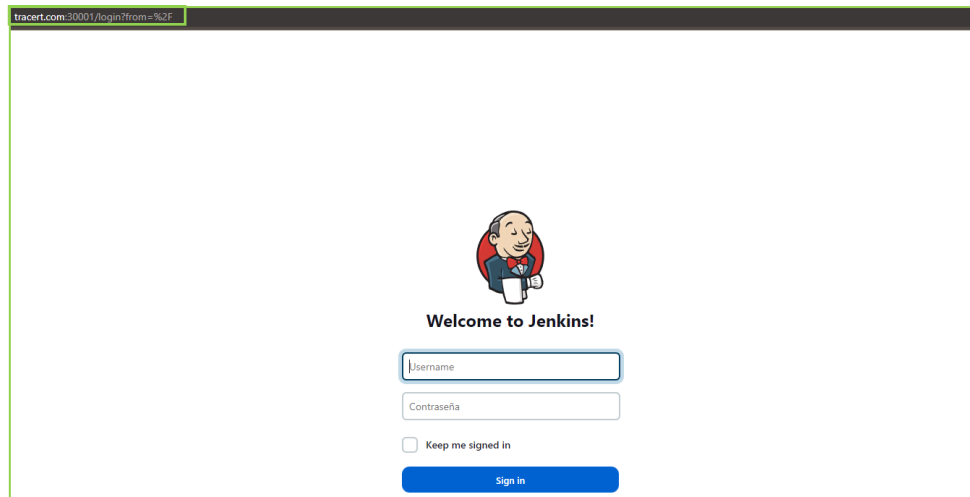
```
tracert@DESKTOP-H70CEFS:/mnt/f/EscritorioPC/Drive_Accenture/trainingPathProject$ kubectl exec jenkins-9b9b8b97f-ltk8v --  
cat /var/jenkins_home/secrets/initialAdminPassword  
34b41d3467af4312b8c1e92645483b93
```



- ¡Instala los plugins recomendados, crea tu usuario de administrador y listo!

NOTA: si quieres utilizar un nombre más llamativo que "localhost", puedes modificar el fichero /etc/hosts de tu máquina añadiendo la línea 127.0.0.1 <nombre-de-dominio-guay>. Obtendrás algo como esto:





NOTA:

- Siempre es recomendable trabajar con diferentes namespaces (ej, pre, pro, dev, etc.). No lo hacemos en esta breve práctica para no complicar más el asunto, pero lo suyo sería separar Jenkins de los otros servicios, etc.

Pasamos a Istio. Por mi poca experiencia y la documentación que he leído, Istio da muchos problemas trabajando con un clúster en local ya que, entre otras cosas, no gestionamos tráfico del exterior. Por ello, las configuraciones que vamos a hacer no son excesivamente complejas, sino que son más bien ilustrativas de cuál es la utilidad del servicio.

Antes de nada:

- El "Gateway" es un componente de Istio que actúa como entrypoint para el tráfico externo que ingresa al clúster.
- El "VirtualService" es un objeto de Istio que permite definir cómo el tráfico debe ser dirigido desde el Gateway hacia los servicios back-end en el clúster.

10. Instalación de Istio en clúster:

- Descarga e instalación de Istio:

```
curl -L https://istio.io/downloadIstio | sh -
```

```
cd istio-*
```

```
export PATH=$PWD/bin:$PATH
```

- Comprobar que los CRDs están instalados con:

```
kubectl get crds | grep 'istio.io\|certmanager.k8s.io' | wc -l
```

- Si obtenemos 0, ejecutar:

```
istioctl install --set profile=default
```

- Comprobar que funciona todo y que se ha creado un ingress Gateway y un pod de istio:

```
kubectl get pods -n istio-system
```

```
tracert@DESKTOP-H70CEFS:/mnt/f/EscritorioPC/Drive_Accenture/trainingPathProject/istio-1.17.1$ kubectl get pods -n istio-system
NAME                                READY   STATUS    RESTARTS   AGE
istio-ingressgateway-85666b77c8-f9sn6 1/1     Running   0           51s
istiod-789d599647-6nckc               1/1     Running   0           68s
```

- Habilitar Istio para el namespace en el que están desplegados los pods de Apache, Nginx y jenkins:

```
kubectl label namespace default istio-injection=enabled
```

11. Para hacer una configuración sencilla de Istio en este escenario, podríamos agregar un balanceador de carga para el servicio de nginx utilizando el control de tráfico de Istio. Para ello:

- Generar un balanceador de carga para el servicio de nginx. Creamos un VirtualService y un Gateway en Istio para nginx:

loadBalancerNginx.yaml:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: nginx-vs
spec:
  hosts:
  - "*"
  gateways:
  - istio-gateway
  http:
  - match:
    - uri:
        prefix: /
      route:
      - destination:
          host: nginx
          port:
            number: 80
  ---
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: istio-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"

```

- `kubectl apply -f loadBalancerNginx.yaml`

NOTAS:

- Teóricamente deberíamos poder acceder a nginx a través de `http://<nodeIP>:30000`. En realidad, por trabajar en local kind nos pone las cosas más difíciles ya que nuestro entorno no soporta balanceadores de carga externos¹. Esto no se arregla sin modificar el fichero de configuración del kubelet, el cluster-config, etc. Si

¹ Más info en <https://istio.io/latest/docs/tasks/traffic-management/ingress/ingress-control/#using-node-ports-of-the-ingress-gateway-service>

se soportara un balanceador externo, el procedimiento² sería modificar los exports de la siguiente forma (en mi caso para wsl2):

```
export INGRESS_NAME=istio-ingressgateway
export INGRESS_NS=istio-system

export INGRESS_HOST=127.0.0.1

#check
kubectl get svc -n istio-system
echo "INGRESS_HOST=$INGRESS_HOST, INGRESS_PORT=$INGRESS_PORT"

export GATEWAY_URL=$INGRESS_HOST:$INGRESS_PORT
```

El fichero, por si quieres trastear con él, es:

```
nginx-nodeport.yaml:

apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: nginx-nodeport-vs
spec:
  hosts:
    - "*"
  gateways:
    - istio-gateway
  http:
    - match:
        - port: 30000
      route:
        - destination:
            host: nginx
            port:
              number: 80
```

- Es posible que haya que reiniciar los deployments (o hacer un delete de los pods) para que tengan istio inyectado:

```
kubectl rollout restart deployment nginx

kubectl rollout restart deployment apache

kubectl delete pods --all
```

- Comprobar que todo funciona bien con:

```
istioctl analyze
```

```
tracert@DESKTOP-H70CEFS:/mnt/f/EscritorioPC/Drive_Accenture/trainingPathProject/istio-1.17.1$ istioctl analyze
No validation issues found when analyzing namespace: default.
```

```
kubectl get gateway

kubectl get virtualService
```

² Más info en <https://istio.io/latest/docs/examples/bookinfo/#determine-the-ingress-ip-and-port>

```
tracert@DESKTOP-H70CEFS:/mnt/f/EscritorioPC/Drive_Accenture/trainingPathProject/istio-1.17.1$ kubectl get vs
kubectl get gateway
NAME          GATEWAYS    HOSTS    AGE
nginx-nodeport-vs [istio-gateway] [*]      26m
nginx-vs      [istio-gateway] [*]      29m
NAME          AGE
istio-gateway 29m
```

KIALI

Suponiendo que ya tenemos un balanceador de carga que redirige todo hacia el reverse-proxy de nginx, procedemos a ver la siguiente herramienta. Kiali es una herramienta de tracking/dashboards de Istio que proporciona visualizaciones y análisis de la topología de servicios de Istio y la información de tráfico de red. Se suele usar siempre que Istio esté configurado (como Grafana si está Prometheus o Kibana si está Elastic-Search configurado). Vamos a instalarlo:

12. Instalar kiali:

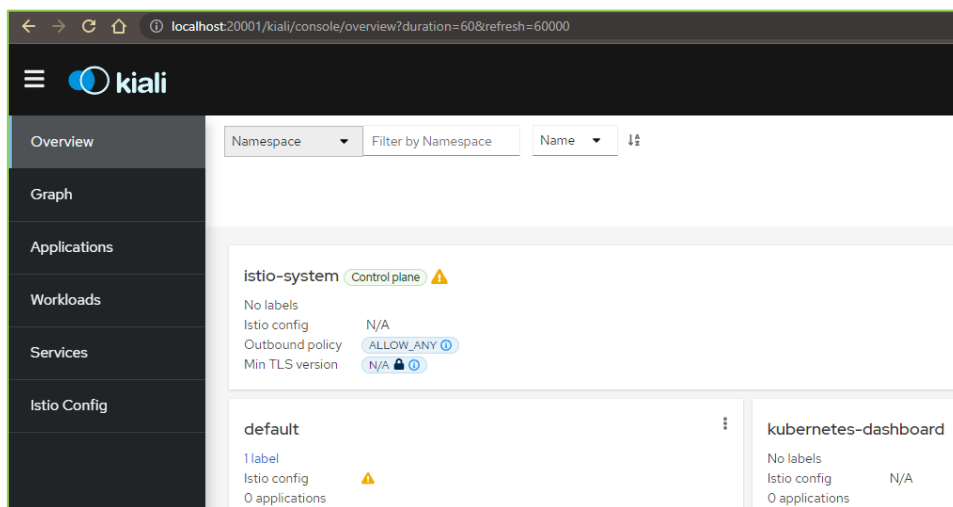
```
kubectl apply -f https://raw.githubusercontent.com/istio/istio/release-1.17/samples/addons/kiali.yaml --validate=false
```

13. Chequear instalación:

```
kubectl -n istio-system get svc kiali
```

14. Usar el dashboard:

```
istioctl dashboard kiali
```



CONCLUSIÓN:

Istio es un servicio muy completo, que se puede integrar con prometheus, el propio dashboard de Kubernetes, y muchísimos servicios y plataformas más. Mejoraremos las configuraciones de Istio en las siguientes prácticas donde, entre otras cosas, instalaremos Prometheus a través de Istio, y nos centraremos más en la monitorización del clúster.

NOTA IMPORTANTE: Todo el escenario se puede montar ejecutando el fichero autoDeploy.py, por si en algún momento hay algún problema. El fichero acepta un solo input, que es el nombre del clúster. Si quieres dejarlo por defecto, el nombre es "trainingPath" (pulsa Enter).

```
python3 autoDeploy.py
```

```
tracert@DESKTOP-H70CEFS:/mnt/f/EscritorioPC/Drive_Accenture/trainingPathProject$ python3 autoDeploy.py
Introduce your cluster name (default 'trainingpath'):
```