CONTENIDOS

1 DESPLIEGUE DE APLICACIONES EN CONTENEDORES (2 HORAS)

- Introducción a los contenedores
- Arquitectura de microservicios
- Tecnologías subyacentes y diferencias entre ellas: docker, cri-o, LXC, ...
- Ciclo de vida en el despliegue de aplicaciones con docker

2 INTRO A KUBERNETES (2 HORAS)

- Características, historia, estado actual del proyecto kubernetes (k8s)
- Arquitectura básica de k8s
- Alternativas para instalación simple de k8s: minikube, kubeadm, k3s
- Instalación con minikube
- Instalación y uso de kubectl
- Despliegue de aplicaciones con k8s

3 DESPLIEGUE DE APLICACIONES CON K8S (1:30 HORAS)

- Pods
- ReplicaSet: Tolerancia y escalabilidad
- Deployment: Actualizaciones y despliegues automáticos

4 COMUNICACIÓN ENTRE SERVICIOS Y ACCESO DESDE EL EXTERIOR (1:30 HORAS)

- Services
- DNS
- Ingress
- Ejemplos de uso y despliegues

5 CONFIGURACIÓN DE APLICACIONES (1 HORA)

- Variables de entorno
- ConfigMaps
- Secrets
- Ejemplo de despliegue parametrizado

6 ALMACENAMIENTO (1:30 HORAS)

- Consideraciones sobre el almacenamiento
- PersistentVolume
- PersistentVolumeClaim
- Ejemplo de despliegue con volúmenes

CONTENIDOS

7 OTROS TIPOS DE DESPLIEGUES (1:30 HORAS)

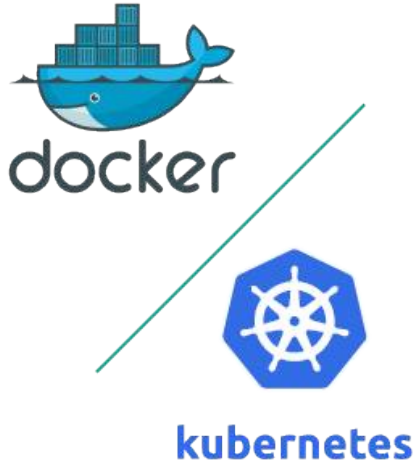
- StatefulSet
- DaemonSet
- AutoScale
- Helm

8 ADMINISTRACIÓN BÁSICA (1 HORA)

- Namespaces
- Usuarios
- RBAC
- Cuotas y límites

9 INSTALACIÓN PASO A PASO (4 HORAS)

- Consideraciones previas: Requisitos hardware, arquitectura física y lógica, entornos y herramientas para el despliegue
- Instalación completa componente a componente en múltiples nodos



MÓDULO 4. COMUNICACIÓN ENTRE SERVICIOS Y ACCESO DESDE EL EXTERIOR

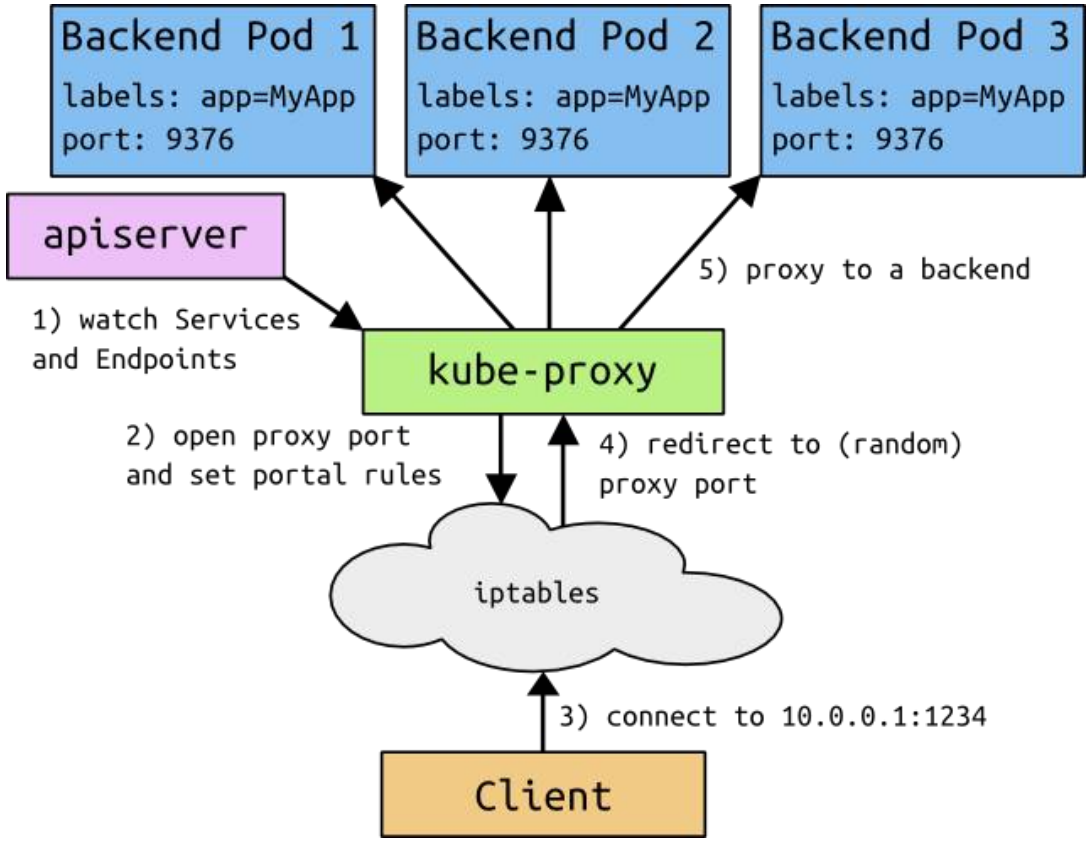
Services

Los servicios (services) nos permiten acceder a nuestra aplicaciones.

- Un servicio es una abstracción que define un conjunto de pods que implementan un micro-servicio. (Por ejemplo el *servicio frontend*).
- Ofrecen una dirección virtual (CLUSTER-IP) y un nombre que identifica al conjunto de pods que representa, al cual nos podemos conectar.
- La conexión al servicio se puede realizar desde otros pods o desde el exterior (mediante la generación aleatoria de un puerto).

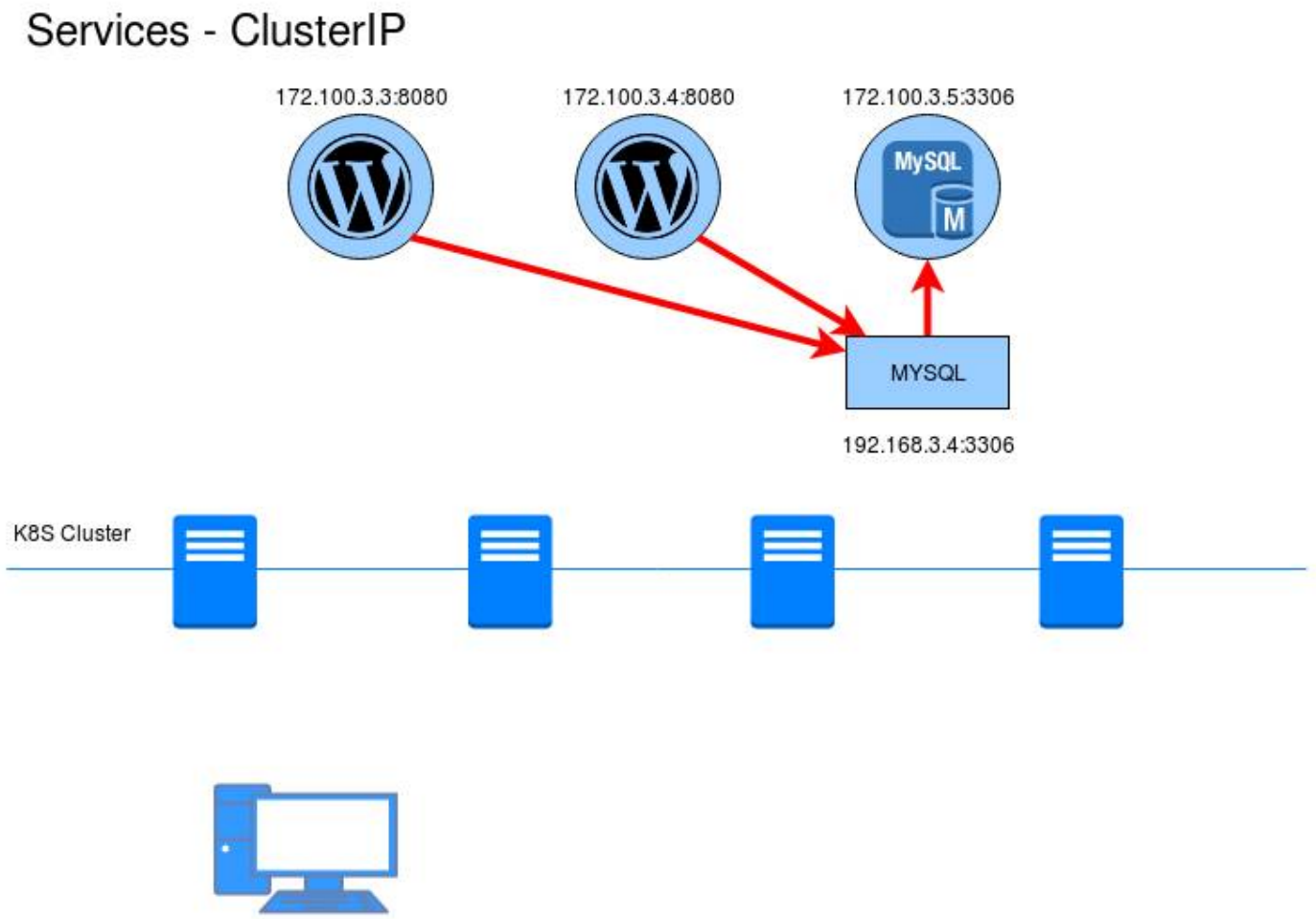
Services

- Los servicios se implementan con iptables.
- El componente kube-proxy de Kubernetes se comunica con el servidor de API para comprobar si se han creado nuevos servicios.
- Cuando se crea un nuevo servicio, se le asigna una nueva ip interna virtual (IP-CLUSTER) que permite conexiones desde otros pods.
- Además podemos habilitar el acceso desde el exterior, se abre un puerto aleatorio que permite que accediendo a la IP del cluster y a ese puerto se acceda al conjunto de pods.
- Si tenemos más de un pod el acceso se hará siguiendo una política round-robin.



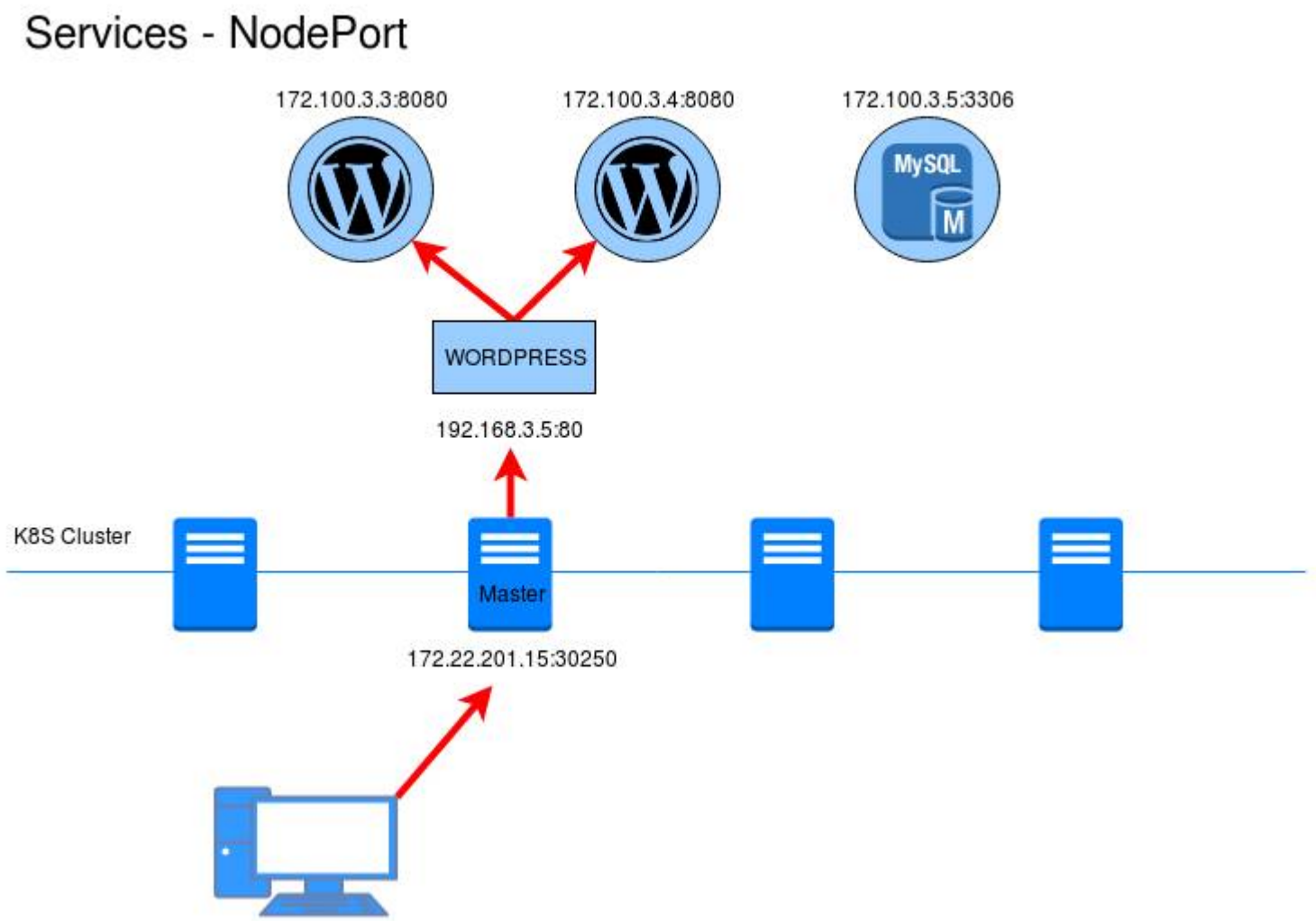
Services ClusterIP

- **ClusterIP:** Solo permite el acceso interno entre distintos servicios. Es el tipo por defecto. Podemos acceder desde el exterior con la instrucción kubectl proxy, puede de ser gran ayuda para los desarrolladores.



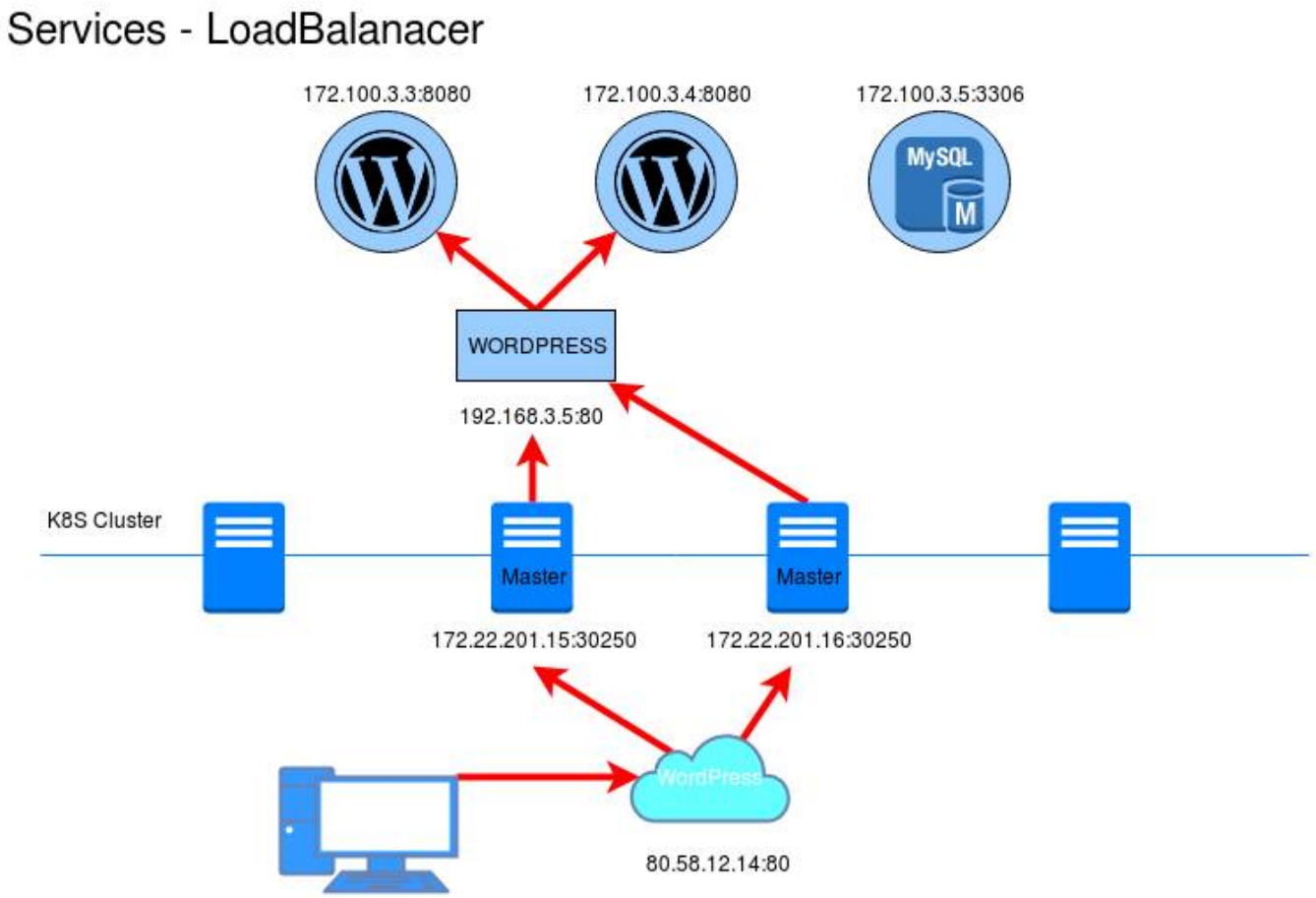
Services NodePort

- **NodePort:** Abre un puerto, para que el servicio sea accesible desde el exterior. Por defecto el puerto generado está en el rango de 30000:40000. Para acceder usamos la ip del servidor master del cluster y el puerto asignado.



Services NodePort

- **LoadBalancer**: Este tipo sólo está soportado en servicios de cloud público (GKE, AKS o AWS). El proveedor asignara un recurso de balanceo de carga para el acceso a los servicios. si usamos un cloud privado, como OpenSatck necesitaremos un plugin para configurar el funcionamiento.



EJEMPLO 1: Describiendo objetos k8s: Services ClusterIP

```yaml
apiVersion: v1
kind: Service
metadata:
 name: nginx
 namespace: default
spec:
 type: ClusterIP
 ports:
 - name: http
   port: 80
   targetPort: http
 selector:
   app: nginx
```

```
kubectl create -f deployment.yaml
```

```
kubectl create -f service_ci.yaml
```

También podríamos haber creado el servicio sin usar el fichero yaml, de la siguiente manera:

```
kubectl expose deployment/nginx --port=80 --type=ClusterIP
```

Podemos ver el servicio que hemos creado:

```
kubectl get svc
```

Puede ser bueno acceder desde exterior, por ejemplo en la fase de desarrollo de una aplicación para probarla:

```
kubectl proxy
```

Y accedemos a la URL:

```
http://localhost:8001/api/v1/namespaces/<NAMESPACE>/
services/<SERVICE NAME>:<PORT NAME>/proxy/
```

Para más información acerca de los Services puedes leer:
la documentación de la API y la guía de usuario.

EJEMPLO 1: Describiendo objetos k8s: Services NodePort

```yaml
apiVersion: v1
kind: Service
metadata:
 name: nginx
 namespace: default
spec:
 type: NodePort
 ports:
 - name: http
   port: 80
   targetPort: http
 selector:
   app: nginx
```

```
kubectl create -f service_np.yaml
```

También podríamos haber creado el servicio sin usar el fichero yaml, de la siguiente manera:

```
kubectl expose deployment/nginx --port=80 --type=NodePort
```

Podemos ver el servicio que hemos creado:

```
kubectl get svc
```

Desde el exterior accedemos a:

```
http://<IP_MASTER>:<PUERTO_ASIGNADO>
```

Para más información acerca de los Services puedes leer:
la documentación de la API y la guía de usuario.

DNS

Existe un componente de Kubernetes llamado KubeDNS, que ofrece un servidor DNS para que los pods puedan resolver diferentes nombres de recursos (servicios, pods, ...) a direcciones IP.

- Cada vez que se crea un nuevo servicio se crea un registro de tipo A con el nombre `servicio.namespace.svc.cluster.local.`

Veamos el **Ejemplo 2**:

```
kubectl create -f busybox.yaml
kubectl exec -it busybox -- nslookup nginx
kubectl exec -it busybox -- wget http://nginx
```

Comprobando el servidor DNS:

```
kubectl get pods --namespace=kube-system -o wide
kubectl get services --namespace=kube-system
kubectl exec -it busybox -- cat /etc/resolv.conf
```

Balanceo de carga

Hemos creado una imagen docker que nos permite crear un contenedor con una aplicación PHP que muestra el nombre del servidor donde se ejecuta, el fichero `index.php`:

```
<?php echo "Servidor:"; echo gethostname();echo "\n"; ?>
```

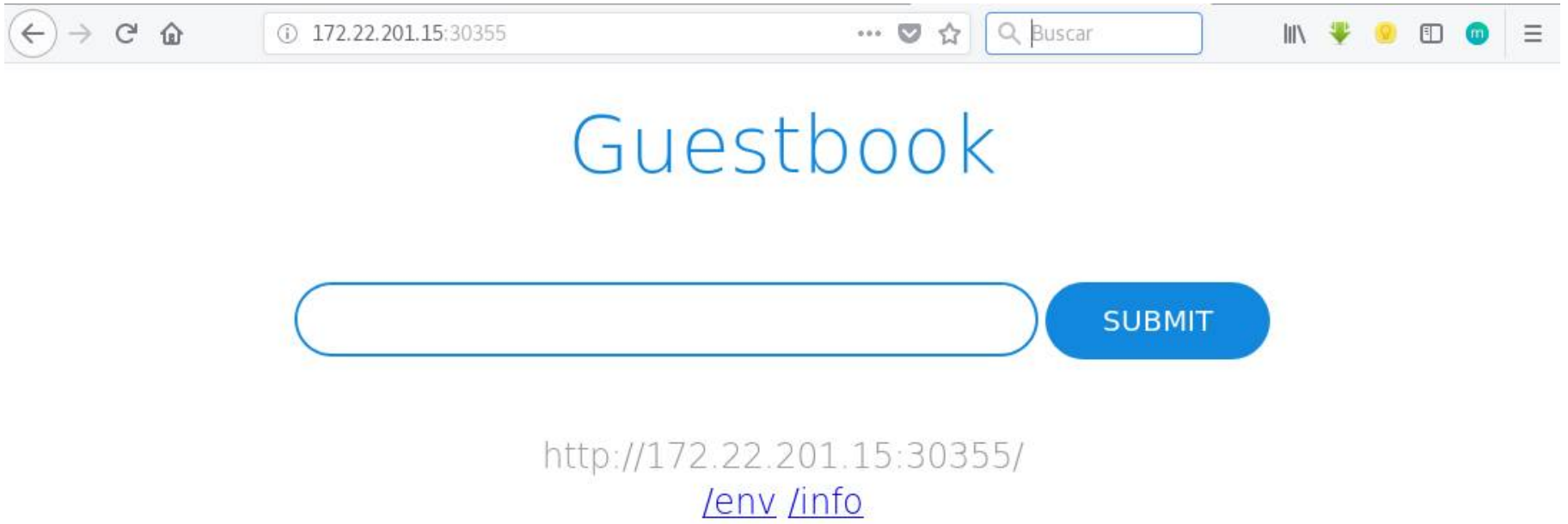
Si tenemos varios pod de esta aplicación, el objeto Service balancea la carga entre ellos:

```
kubectl create deployment pagweb --image=josedom24/infophp:v1
kubectl expose deploy pagweb --port=80 --type=NodePort
kubectl scale deploy pagweb --replicas=3
```

Al acceder hay que indicar el puerto asignado al servicio:

```
for i in `seq 1 100`; do curl http://192.168.99.100:32376; done
Servidor:pagweb-84f6d54fb7-56zj6
Servidor:pagweb-84f6d54fb7-mdvfn
Servidor:pagweb-84f6d54fb7-bhz4p
```

EJEMPLO 3: Desplegando la aplicación GuestBook (Parte 2)



EJEMPLO 4: Despliegue canary

- Creamos el servicio y el despliegue de la primera versión (5 réplicas):

```
kubectl create -f service.yaml
kubectl create -f deploy1.yaml
```

- En un terminal, vemos los pods:

```
watch kubectl get pod
```

- En otro terminal, accedemos a la aplicación:

```
service=$(minikube service my-app --url)
while sleep 0.1; do curl "$service"; done
```

- Desplegamos una réplica de la versión 2, para ver si funciona bien:

```
kubectl create -f deploy2.yaml
```

- Una vez que comprobamos que funciona bien, podemos escalar y eliminar la versión 1:

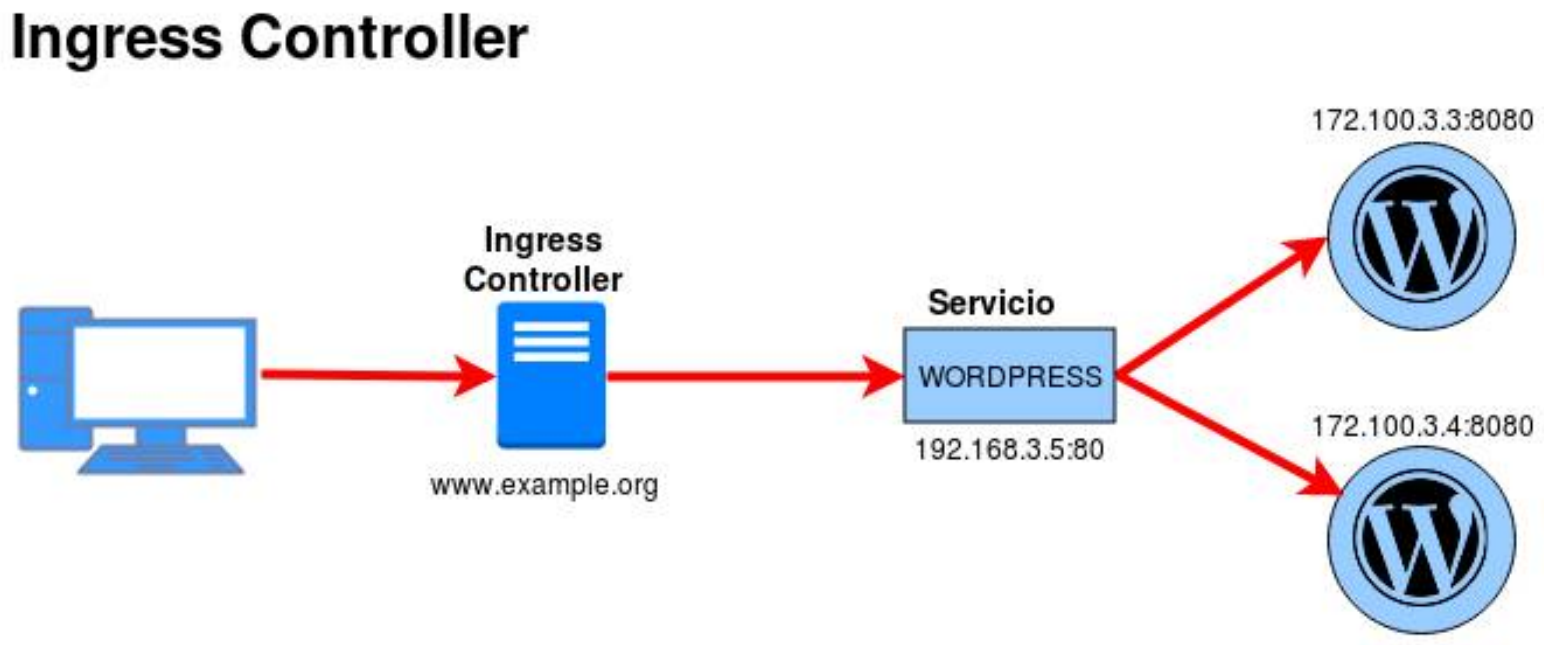
```
kubectl scale --replicas=5 deploy my-app-v2
kubectl delete deploy my-app-v1
```

Ingress Controller

Hasta ahora tenemos dos opciones principales para acceder a nuestras aplicaciones desde el exterior:

1. Utilizando servicios del tipo *NodePort*: Esta opción no es muy viable para entornos de producción ya que tenemos que utilizar puertos aleatorios desde 30000-40000.
2. Utilizando servicios del tipo *LoadBalancer*: Esta opción sólo es válida si trabajamos en un proveedor Cloud que nos cree un balanceador de carga para cada una de las aplicaciones, en cloud público puede ser una opción muy cara.

La solución puede ser utilizar un Ingress controller que nos permite utilizar un proxy inverso (HAproxy, nginx, traefik,...) que por medio de reglas de encaminamiento que obtiene de la API de Kubernetes nos permite el acceso a nuestras aplicaciones por medio de nombres.



EJEMPLO 5: Trabajando con Ingress

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
 name: nginx
spec:
 rules:
 - host: nginx.192.168.99.100.nip.io
   http:
     paths:
     - path: /
       backend:
         serviceName: nginx
         servicePort: 80
```

```
kubectl create -f nginx-ingress.yaml

kubectl get ingress
```

EJEMPLO 6: Desplegando la aplicación LetsChat