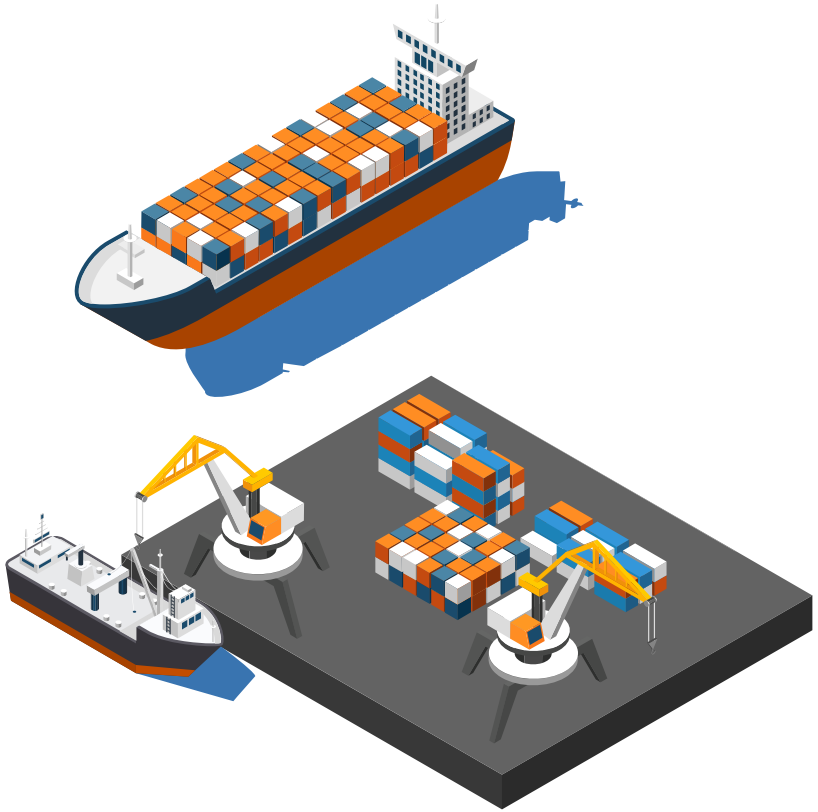


Alberto García

alberto@rootdesdezero.com

Kubernetes Práctico

- Introducción a Kubernestes
- Conceptos básicos
- Administración básica de Kubernetes
- Demos

DevOps

DevOps es un conjunto de prácticas que automatizan los procesos entre los equipos de desarrollo de software y TI para que puedan compilar, probar y publicar software con mayor rapidez y fiabilidad. El concepto de DevOps se basa en establecer una cultura de colaboración entre equipos que, tradicionalmente, trabajaban en grupos aislados. Entre sus ventajas se incluyen el aumento de la confianza y de la velocidad de publicación de software, la capacidad de solucionar incidencias críticas rápidamente y una mejor gestión del trabajo imprevisto.

Una revolución en marcha

El software es ahora la pieza clave de todo negocio. Y cuando cualquier empresa es una empresa de software, tus aplicaciones son las que te diferencian de la competencia.
La percepción de lo que es una buena aplicación cambia con el tiempo, especialmente a medida que los mercados evolucionan y los competidores redefinen su estrategia.
Por lo tanto, cuanto más rápido puedas actualizar tus aplicaciones, mejor. Gracias al cloud, la disponibilidad y la escalabilidad de la infraestructura que las soporta ya no es un problema.

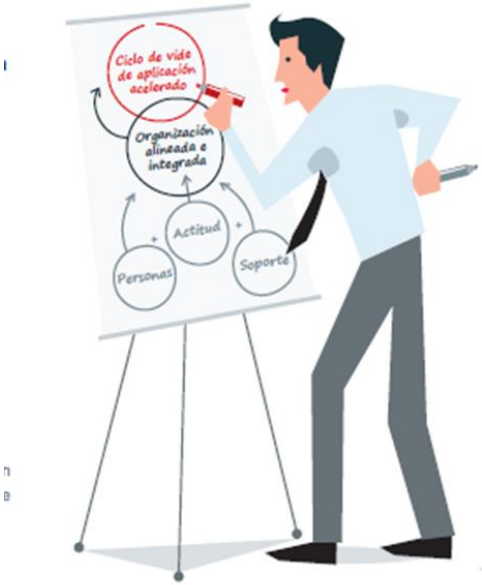
Hoy, el modo en que tu equipo técnico colabora con todas las partes implicadas en el ciclo de vida de las aplicaciones es lo que marca la diferencia.

DevOps

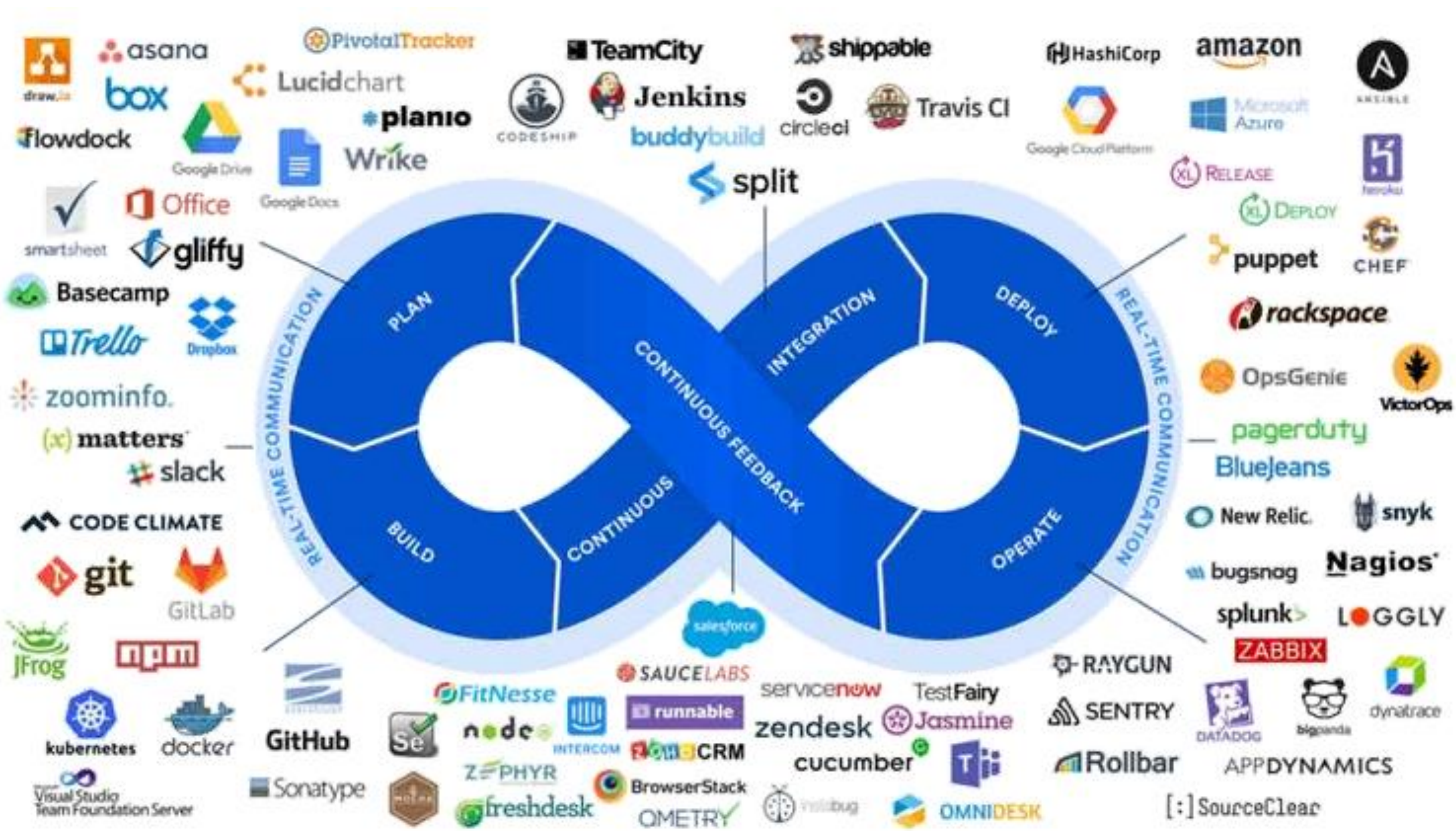
El papel de DevOps

La finalidad de DevOps es crear las condiciones adecuadas para la colaboración entre los departamentos de desarrollo y de operaciones. Sin embargo, antes de implementar una estrategia DevOps hay que tener varias cosas en cuenta.

¿Cuentas con la infraestructura adecuada para soportar este enfoque? ¿Y las herramientas? ¿Sabes por dónde empezar?



DevOps


PLAN
BUILD
CONTINUOUS
INTEGRATION
DEPLOY
OPERATE
CONTINUOUS FEEDBACK
REAL-TIME COMMUNICATION
REAL-TIME COMMUNICATION
asana
box
flowdock
Lucidchart
PivotalTracker
planio
Wrike
Google Drive
Google Docs
Office
smartsheet
gliffy
Basecamp
Trello
Dropbox
zoominfo.
matters
slack
CODE CLIMATE
git
GitLab
JFrog
npm
kubernetes
docker
GitHub
Sonatype
Visual Studio Team Foundation Server
TeamCity
Jenkins
buddybuild
CODESHIP
shippable
circleci
Travis CI
split
HashiCorp
amazon
Google Cloud Platform
Microsoft Azure
ANSIBLE
heroku
XL RELEASE
XL DEPLOY
puppet
CHEF
rackspace
OpsGenie
VictorOps
pagerduty
BlueJeans
New Relic.
snyk
bugsnag
Nagios
splunk
LOGGLY
ZABBIX
RAYGUN
SENTRY
Rollbar
DATADOG
bigpanda
dynatrace
APPDYNAMICS
[:] SourceClear
salesforce
SAUCELABS
FitNesse
runnable
node
INTERCOM
CRM
ZEPHYR
freshdesk
BrowserStack
QMETRY
servicenow
TestFairy
zendesk
Jasmine
cucumber
OMNIDESK

Introducción

- Docker es un proyecto open source que ha revolucionado la manera de desarrollar software gracias a la sencillez con la que permite gestionar contenedores. Los contenedores LXC (LinuX Containers) son un concepto relativamente antiguo y utilizado desde hace tiempo por grandes empresas como Amazon o Google, pero cuyo gestión era complicada. Sin embargo, Docker define APIs y herramientas de línea de comandos que hacen casi trivial la creación, distribución y ejecución de contenedores. De ahí que el lema de Docker sea: “Build, Ship and Run. Any application, Anywhere” y se haya convertido en una herramienta fundamental tanto para desarrolladores como para administradores de sistemas.

Introducción

- Podríamos definir un contenedor Docker como una máquina virtual ligera, que corre sobre un sistema operativo Linux pero con su propio sistema de ficheros, su propio espacio de usuarios y procesos, sus propias interfaces de red… por lo que se dice que son sistemas aislados. Las características principales de Docker son su portabilidad, su inmutabilidad y su ligereza:

Introducción

- **Portabilidad**

Un contenedor Docker es ejecutado por lo que se denomina el Docker Engine, un demonio que es fácilmente instalable en prácticamente todas las distribuciones Linux. Un contenedor ejecuta una imagen de docker, que es una representación del sistema de ficheros y otros metadatos que el contenedor va a utilizar para su ejecución. Una vez que hemos generado una imagen de Docker, ya sea en nuestro ordenador o vía una herramienta externa, esta imagen podrá ser ejecutada por cualquier Docker Engine, independientemente del sistema operativo y la infraestructura que haya por debajo.

Introducción

- **Inmutabilidad**

Una aplicación la componen tanto el código fuente como las librerías del sistema operativo y del lenguaje de programación necesarias para la ejecución de dicho código. Estas dependencias dependen a su vez del sistema operativo donde nuestro código va a ser ejecutado, y por esto mismo ocurre muchas veces aquello de que “no sé, en mi máquina funciona”. Sin embargo, el proceso de instalación de dependencias en Docker no depende del sistema operativo, si no que este proceso se realiza cuando se genera una imagen de docker. Es decir, una imagen de docker (también llamada repositorio por su parecido con los repositorios de git) contiene tanto el código de la aplicación como las dependencias que necesita para su ejecución. Una imagen se genera una vez y puede ser ejecutada las veces que sean necesarias, y siempre ejecutará con las misma versión del código fuente y sus dependencias, por lo que se dice que es inmutable. Si unimos inmutabilidad con el hecho de que Docker es portable, decimos que Docker es una herramienta fiable, ya que una vez generada una imagen, ésta se comporta de la misma manera independientemente del sistema operativo y de la infraestructura donde se esté ejecutando.

Introducción

- **Ligereza**

Los contenedores Docker corriendo en la misma máquina comparten entre ellos el sistema operativo, pero cada contenedor es un proceso independiente con su propio sistema de ficheros y su propio espacio de procesos y usuarios (para este fin Docker utiliza cgroups y namespaces, recursos de aislamiento basados en el kernel de Linux). Esto hace que la ejecución de contenedores sea mucho más ligera que otros mecanismos de virtualización. Comparemos por ejemplo con otra tecnología muy utilizada como es Virtualbox. Virtualbox permite del orden de 4 ó 5 máquinas virtuales en un ordenador convencional, mientras que en el mismo ordenador podremos correr cientos de containers sin mayor problema, además de que su gestión es mucho más sencilla.

Contenedores en cluster

- **Orquestadores de contenedores**

https://docs.docker.com/engine/swarm/

http://kubernetes.io/

http://mesos.apache.org/

Contenedores en cluster

- **Orquestadores de contenedores**
  - Software encargado de **gestionar la ejecución** de contenedores en un **cluster** de máquinas
  - Pueden tratar **varios contenedores** como una única unidad lógica (**aplicación**)
  - **Servicios ofrecidos**
    - **Redes aisladas** para contenedores de la misma app
    - Políticas de **reinicio** en caso de **caída** del servicio
    - Políticas de **replicación** de contenedores por carga

Contenedores en cluster

- **Servicios y contenedores**
  - **Un contenedor con todos los servicios:** Una aplicación se puede empaquetar como una imagen docker con todos los servicios incluidos (web, BBDD, caché, etc…).
  - **Un contenedor por servicio:** También puede empaquetarse como muchas imágenes diferentes. Al arrancar habrá varios contenedores comunicados entre sí por red

Contenedores en cluster

- **Un contenedor con todos los servicios**
  - **Ventajas**
    - Más fácil de crear y probar (sólo un Dockerfile)
    - La comunicación de los servicios es siempre por localhost (más sencilla)
    - Se descarga de forma automática con un comando (ideal para distribuir)

Contenedores en cluster

- **Un contenedor con todos los servicios**
  - **Desventajas**
    - Un contenedor sólo puede escalar verticalmente (con una máquina más potente), pero no horizontalmente (usando varias máquinas)
    - No es tolerante a fallos (un error en un servicio afecta a todos los demás)

Contenedores en cluster

- **Un contenedor por servicio**
  - **Ventajas**
    - Las aplicaciones pueden escalar horizontalmente en un cluster
    - Cada contenedor puede estar en una máquina diferente
    - Se pueden clonar servicios que necesitan más potencia de cómputo en varias máquinas (web, servicios *stateless*)
    - Es tolerante a fallos (un servicio se cae y se puede reiniciar)

Contenedores en cluster

- **Un contenedor por servicio**
  - **Desventajas**
    - Puede ser más difícil de crear si tu código está en varios contenedores
    - Para webs sencillas se puede tener un contenedor para la BBDD y otros para la web (o varios)
    - Existen muchas formas diferentes de gestionar una aplicación formada por varios servicios
      - Una sola máquina: **docker-compose**
      - Cluster: **orquestadores de contendores**

Que es Kubernetes

Kubernetes es un orquestador de contenedores, y que se ha convertido en el estándar *de facto* para desplegar aplicaciones sobre todo en entornos cloud.

Kubernetes es un proyecto *open source*, desarrollado en un principio por Google. Es una evolución de Borg, el orquestador que utiliza Google en producción, pero adaptado a usar contenedores Docker. Por tanto, es un orquestador que nace con muchos años de experiencia de Google ejecutando contenedores en producción, y que por tanto, es muy potente y escala muy bien.

La funcionalidad principal de Kubernetes es lanzar y gestionar contenedores en un clúster, permitiendo la ejecución de muchos contenedores en cada nodo. Y lo hace de un modo declarativo.

Es decir, a Kubernetes le vamos a decir el número de instancias de un contenedor que queremos ejecutar, y Kubernetes se va a encargar de ejecutar ese número de instancias de la mejor manera posible. Si un contenedor empieza a funcionar mal, lo reemplazará. Y si en ese momento no se pueden ejecutar todos los contenedores deseados por falta de recursos, por ejemplo, esperará a que haya recursos disponibles y en ese momento seguirá creando contenedor hasta alcanzar el número deseado

Que es Kubernetes

Probablemente **el aspecto más potente de Kubernetes es su ecosistema**. Parece que todo el mundo se ha puesto de acuerdo para convertirlo en el estándar de despliegue de aplicaciones en el cloud, y no hay herramienta que se precie que no tenga una versión para correr en Kubernetes.

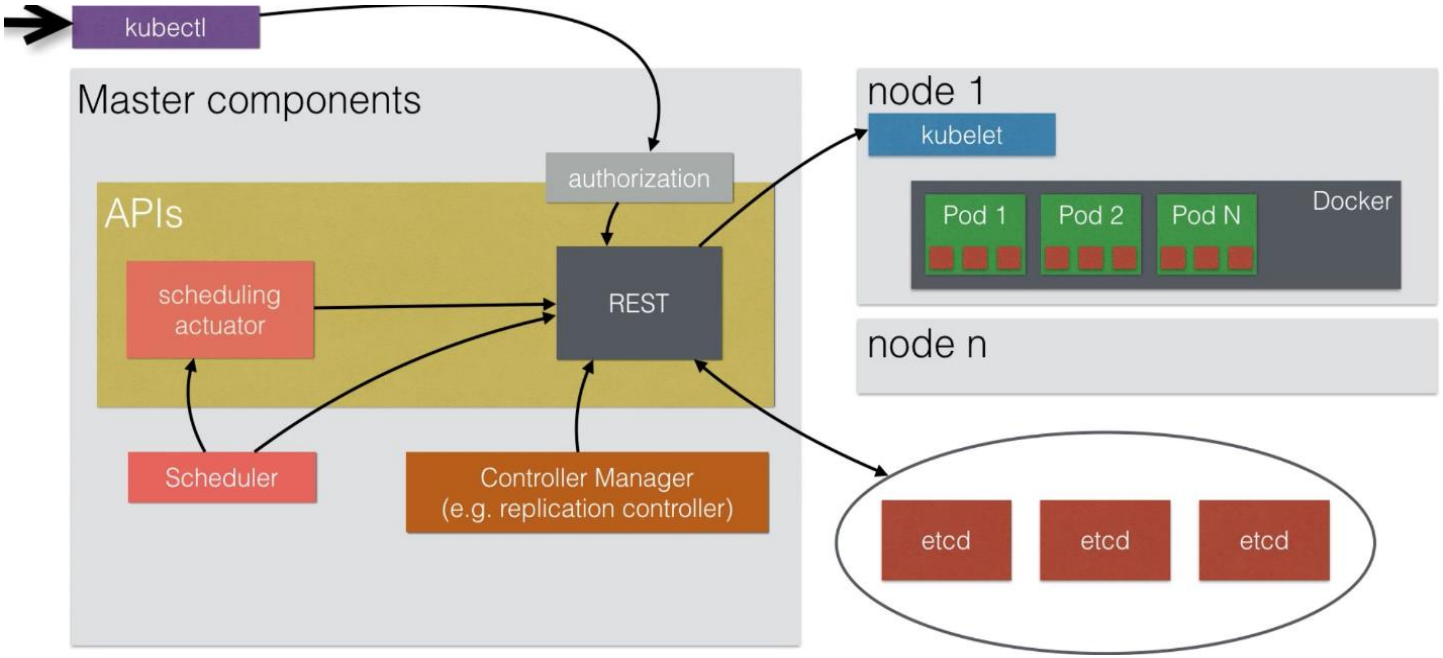
De hecho, Kubernetes ha conseguido hacer realidad el concepto de multi-cloud. Si empaquetamos nuestra aplicación para correr en Kubernetes, nos va a ser relativamente sencillo lanzar nuestra aplicación en un Kubernetes corriendo en AWS, o en Azure, o en Google Cloud, o en un clúster *on-premises*.

Esta característica es de vital importancia, sobre todo para las empresas que hace software enterprise que ejecuta en la infraestructura del cliente

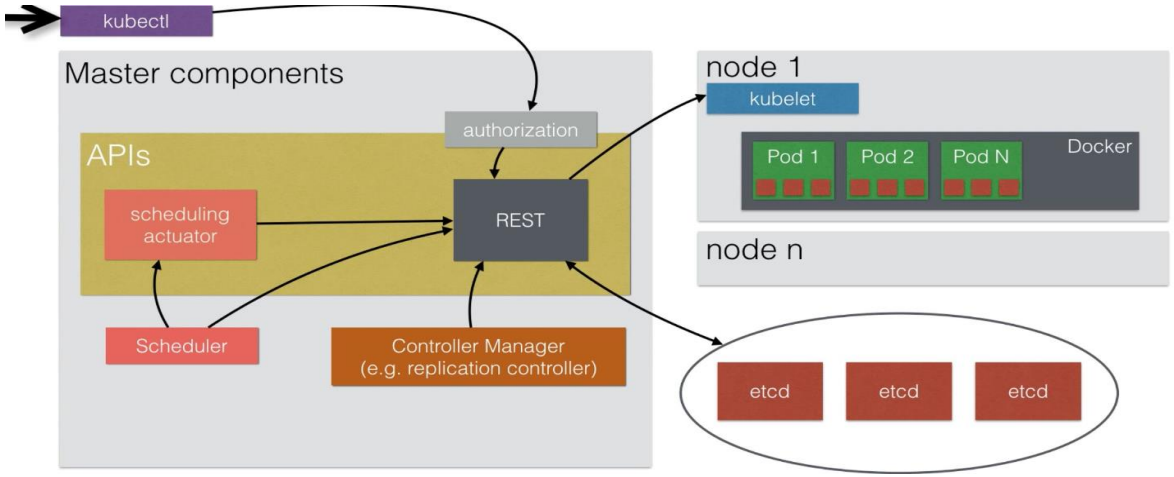
Arquitectura Básica Kubernetes

Kubernetes divide los nodos de un clúster en *master nodes* y en *worker nodes*. Los master nodes son la capa de control y en principio no ejecutan contenedores de usuario. Los worker nodes son los responsables de ejecutar los contenedores de usuario.

Aunque esta división es la más habitual, cuando corremos Kubernetes en local se suele crear un clúster de un solo nodo, que actúa a la vez como master y como worker node.



Arquitectura Básica Kubernetes

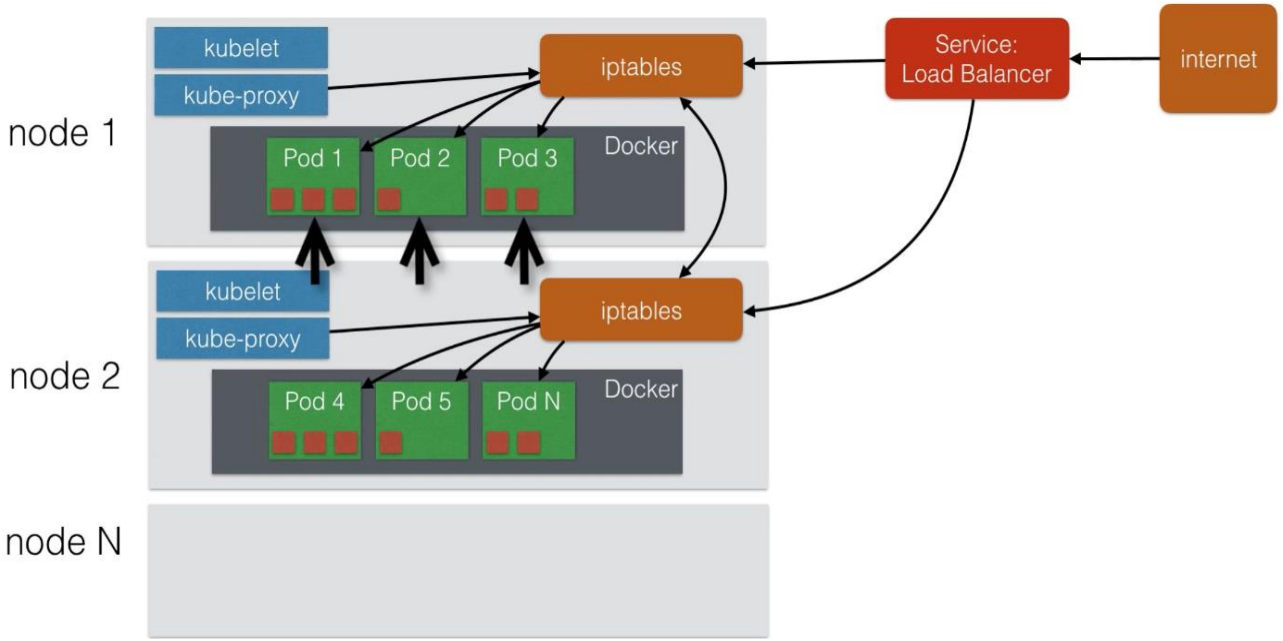


En el master node la lógica de la capa de control se fundamenta en **`etcd`**, una base de datos distribuida que nos asegura la consistencia de los datos usando el protocolo RAFT. Sobre **`etcd`** tenemos la capa REST, que es a través de la que pasan todos los accesos a la base de datos. Esa capa REST se expone vía un módulo de autenticación, para que podamos invocarla, por ejemplo, usando comandos **`kubectl`**.

El resto de la lógica que corre en los master nodes son controladores, que son funciones que se ejecutan en bucle y que comprueban si el estado deseado para el clúster se corresponde con el estado actual, o si por el contrario hay que crear o eliminar contenedores.

Los masters siempre serán impares normalmente tres nodos, si el cluster es muy grande podríamos utilizar cinco.

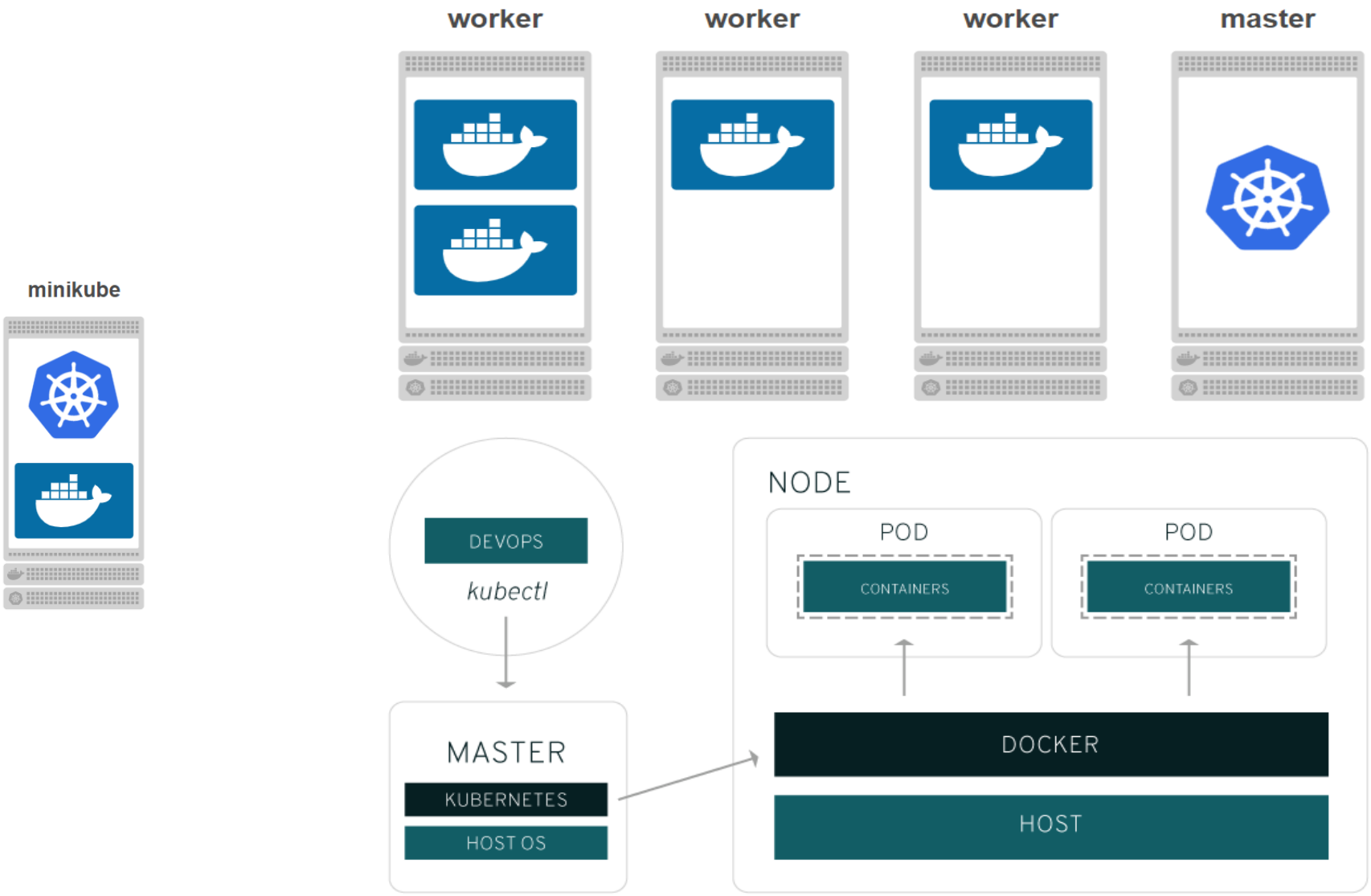
Arquitectura Básica Kubernetes



Los **worker nodes**, además de los contenedores de usuario, tienen principalmente dos componentes, el **`kubelet`** y el **`kube-proxy`**.

El **`kubelet`** se encarga de chequear el estado de **etcd** vía la capa REST para ver qué contenedores tiene asignados para ejecutar, y en base a ello, crear o eliminar contenedores en el nodo que gestiona. Por su parte, el **`kube-proxy`** se encarga de gestionar la red y el *service discovery* entre contenedores creando reglas de *iptables*.

Arquitectura Básica Kubernetes



Arquitectura Básica Kubernetes

Distribuciones para la instalación de Kubernetes en local

Hay muchas maneras de tener un clúster de Kubernetes corriendo.

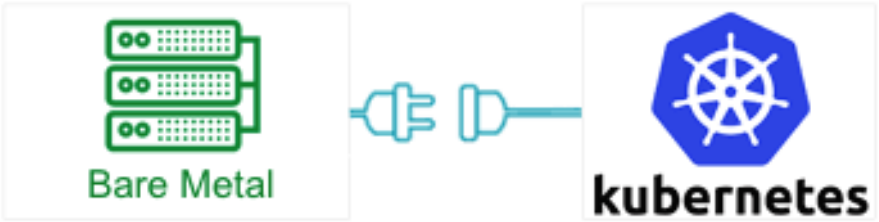
En primer lugar tenemos las instalaciones de Kubernetes en local, donde destacamos Minikube y MicroK8S.

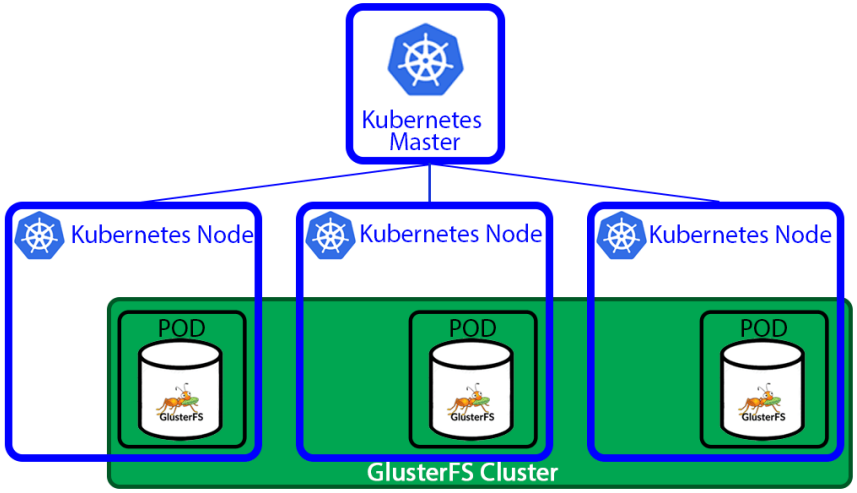
Cuando queremos lanzar clústers de Kubernetes en el cloud, una opción interesante es Kops. Kops es un proyecto *open source* que nos permite lanzar clústers de Kubernetes en multitud de proveedores de cloud.

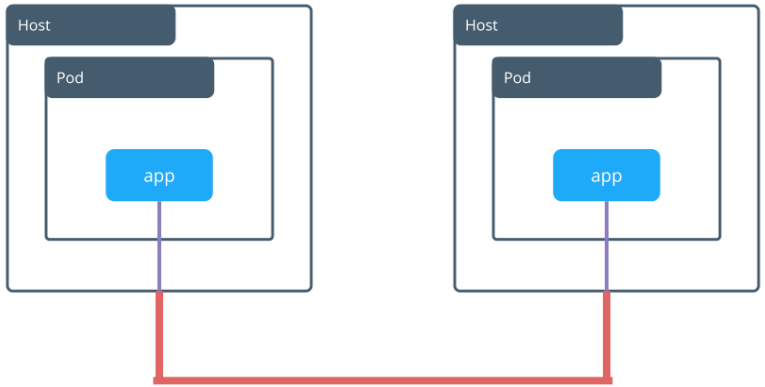
Si necesitas lanzar un clúster de Kubernetes en el cloud, yo recomendaría las soluciones nativas de cada proveedor, como son AKS (de Azure), EKS (de AWS), KUBERNETES (de Google).

Arquitectura Básica Kubernetes

Kubernetes on Premise/Bare Metal







Kubernetes Arquitectura

¿Qué es el nodo maestro, y qué hace?

Cuando le pide a Kubernetes que cree una implementación, utiliza kubectl una herramienta de línea de comandos para interactuar con el clúster.
kubectl envía una solicitud de API con el contenido del archivo YAML a la API de Kubernetes.
La API almacena el estado del recurso en una base de datos.

Etcd: Es el servicio de disponibilidad de cluster desarrollado por CoreOS y utilizado tambien por Centos/REDHAT. El demonio etcd es el encargado del almacenamiento distribuido.Utiliza una clave llave/valor que puede ser distribuida y leída por múltiples nodos. Kubernetes utiliza etcd para guardar la configuración y el estado del cluster leyendo la metadata de los nodos.

API server: API Server de Kubernetesse encarga de validar y configurar los datos de los objetos api que incluyen los pods, servicios, replicationcontrollers,etc. El servidor de API sirve para operaciones REST y proporciona al fronten del estado compartido del clúster a través del cual interactúan todos los demás componentes.

Kubernetes Arquitectura

ControllerManager Server:

Es un servicio usado para manejar el proceso de replicación de las tareas definidas. Los detalles de estas operaciones son escritos en el etcd, donde el controllermanager observa los cambios y cuanto un cambio es detectado, el controllermanager lee la nueva información y ejecuta el proceso de replicación hasta alcanzar el estado deseado

SchedulerServer:

Es el servicio que se encarga balancear la carga de los servidores y elegir el nodo mas saludable(procesamiento, memoria,etc) a la hora de deployar un pod. Tambien se utiliza para leer los requisitos de un pod, analizar el ambiente del clustery seleccionar el nodo más performance. Este servicio adicionalmente cumple la función de monitorear el estado de los recursos de los nodos en realtime.

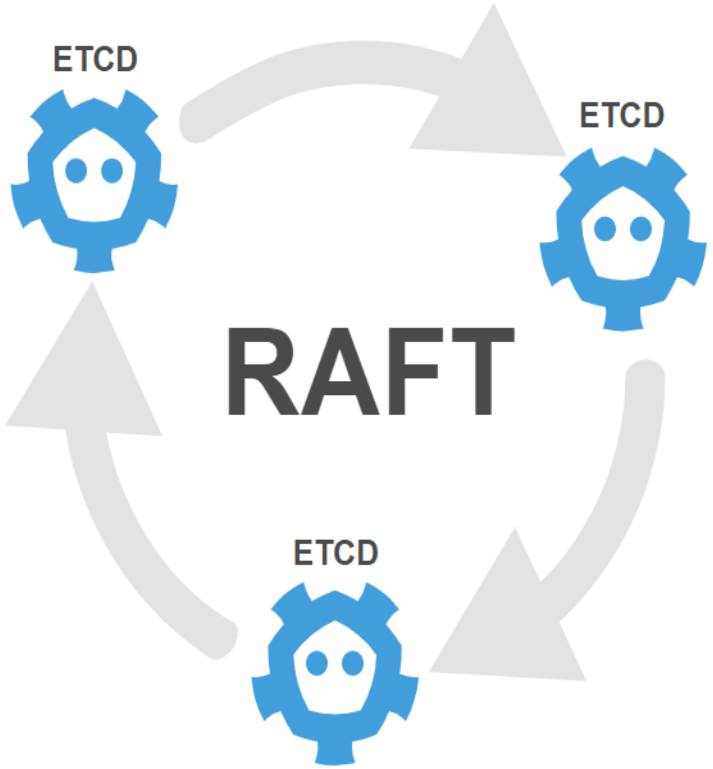
Kubernetes

etcd - Almacén clave-valor

etcd es un almacén de clave-valor.
La razón por la que es tan popular y se usa en Kubernetes es que está diseñado para ejecutarse en sistemas distribuidos.

Cuando inicias una colección de instancias de etcd, comienzan a hablar entre sí y forman una red.

Eventualmente eligen a un líder y el resto de los casos se convierten en seguidores.
Antes de escribir cualquier valor en el disco, las bases de datos acuerdan ese valor primero y luego lo persisten.
Las bases de datos de etcd coordinan la lectura y escritura utilizando el protocolo RAFT



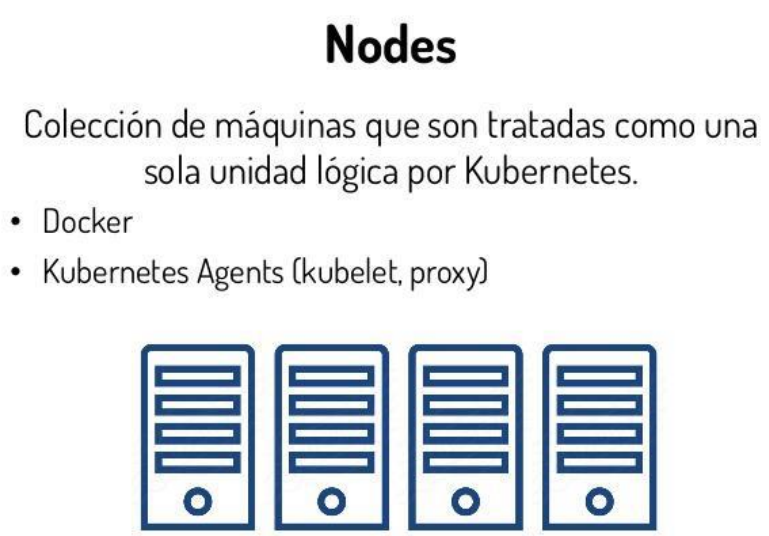
Kubernetes

Kubernetes Minion

Un minión es un nodo/servidor que forma parte de un Cluster, este nos aporta su Computo (procesamiento,memoria,etc) y los suma a los recursos del Cluster. Para poder lograr esto el minion utiliza los siguientes demonios que lo mantienen en contacto con el Master.

Kubelet: Este servicio se encarga de obtener las instrucciones del Master y realizar los cambios correspondientes dentro del nodo.Otra de sus funciones manejar la creación y configuración de los pods(imagenes,volumenes,etc).

Kube-Proxy: Este servicio monitorea los Servicios y Endpoints levantados en el master. Nos provee la apertura de un puerto especifico/aleatorio en el pod para que pueda ser accedido y configura las reglas de firewalls requeridas.



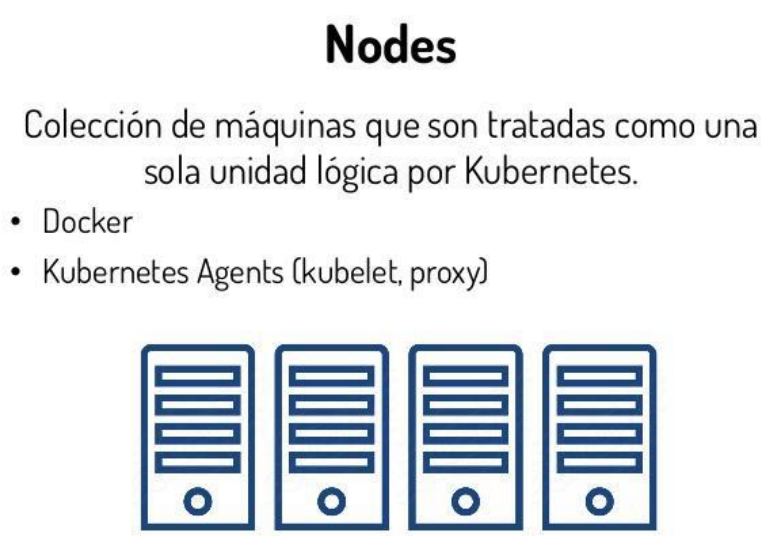
Kubernetes

Kubernetes Minion

Un minión es un nodo/servidor que forma parte de un Cluster, este nos aporta su Computo (procesamiento,memoria,etc) y los suma a los recursos del Cluster. Para poder lograr esto el minion utiliza los siguientes demonios que lo mantienen en contacto con el Master.

Kubelet: Este servicio se encarga de obtener las instrucciones del Master y realizar los cambios correspondientes dentro del nodo.Otra de sus funciones manejar la creación y configuración de los pods(imagenes,volumenes,etc).

Kube-Proxy: Este servicio monitorea los Servicios y Endpoints levantados en el master. Nos provee la apertura de un puerto especifico/aleatorio en el pod para que pueda ser accedido y configura las reglas de firewalls requeridas.



Kubernetes Conceptos Básicos

Pod

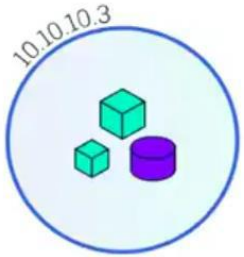
La unidad fundamental de despliegue en Kuberneteses el Pod. Un pod sería el equivalente a la mínima unidad funcional de la aplicación.

En general, un pod contendrá únicamente un contenedor, aunque no tiene que ser así: si tenemos dos contenedores que actúan de forma conjunta, podemos desplegarlos dentro de un solo pod. Dentro de un pod todos los contenedores se pueden comunicar entre ellos usando localhost, por lo que es una manera sencilla de desplegar en Kubernetes .

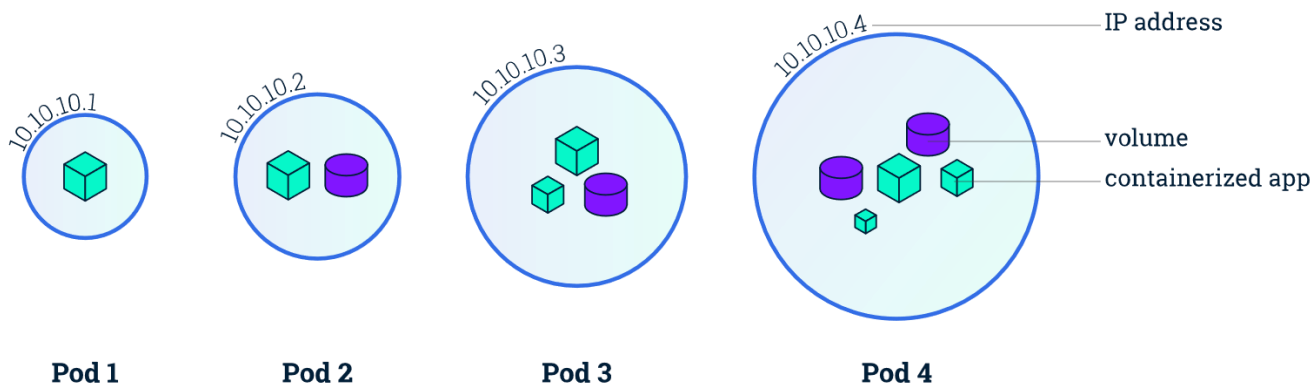
En este sentido, todos los contenedores dentro de un pod se podría decir que están instaladas en una mismo equipo (como un stack LAMP).
Sin embargo, un pod es un elemento no-durable, es decir, que puede fallar o ser eliminado en cualquier momento. Por tanto, no es una buena idea desplegar pods individuales en Kubernetes.

Kubernetes Pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: example
spec:
  containers:
  - image: busybox
    volumeMounts:
    - mountPath: /busy
      name: test
    name: busy
  - image: busybox
    volumeMounts:
    - mountPath: /box
      name: test
    name: box
  volumes:
  - name: test
    emptyDir: {}
```



emptyDir: persistent storage only until pod is deleted



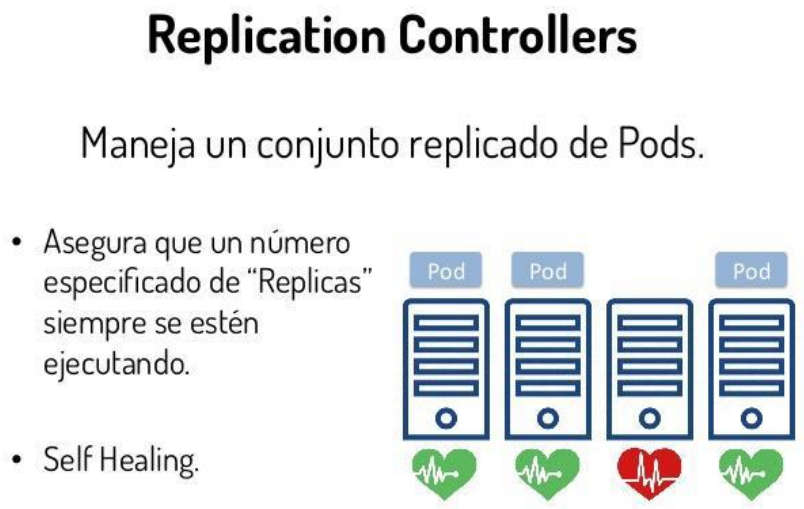
Kubernetes Conceptos Básicos

Replication Controller

Replication Controllerse encarga de mantener un determinado número de réplicas del pod en el clúster.

El Replication Controller asegura que un determinado número de copias -réplicas-del pod se encuentran en ejecución en el clúster en todo momento. Por tanto, si alguno de los pods es eliminado, el ReplicationControllerse encarga de crear un nuevo pod. Para ello, el ReplicationController incluye una plantilla con la que crear nuevos pods.

Así, el Replication Controller define el estado deseado de la aplicación: cuántas copias de mi aplicación quiero tener en todo momento en ejecución en el clúster. Modificando el número de réplicas para el Replication Controller podemos escalar(incrementar o reducir) el número de copias en ejecución en función de las necesidades.



Kubernetes Conceptos Básicos

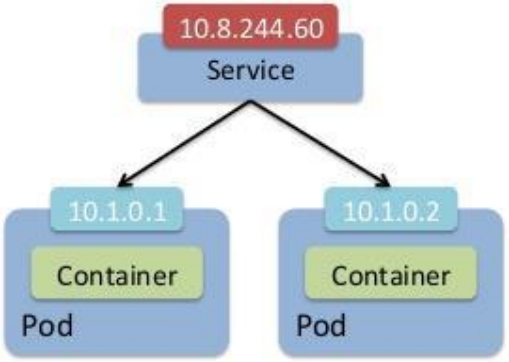
Services: Es el microservicio de una aplicación la cual fue deployada en un pod o ReplicationController y se puede conectar a otro servicio para Lograr la aplicación final.

Un ejemplo de esto puede ser un replicationcontroller llamado WordPress(nos realiza un deployde 3 containers/pod), el cual es asociado a un servicio con el mismo nombre y se conecta al replicationcontrol a través de un Label/Etiqueta.
Este servicio llamado WordPress ,lo podemos conectar a otro servicio llamado MYSQL(Mismo proceso que WordPress con el ReplicationController).

Services

Service Discovery para los Pods.

- Endpoints persistentes para los Pods.
- Backend dinámico basado en Labels.



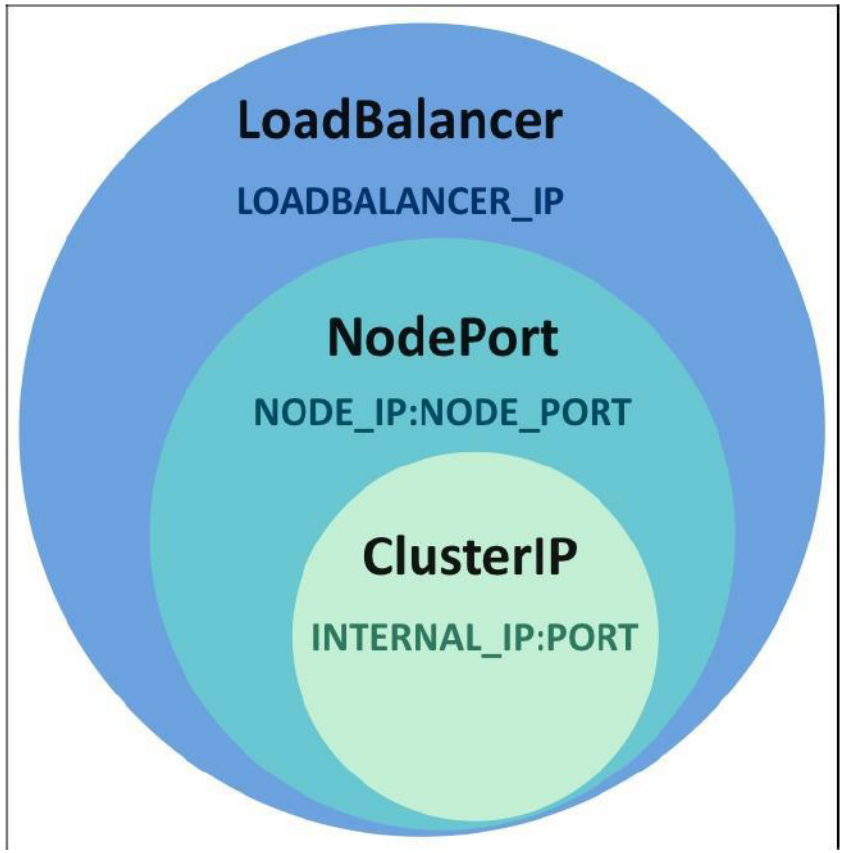
Kubernetes Services

Type Services

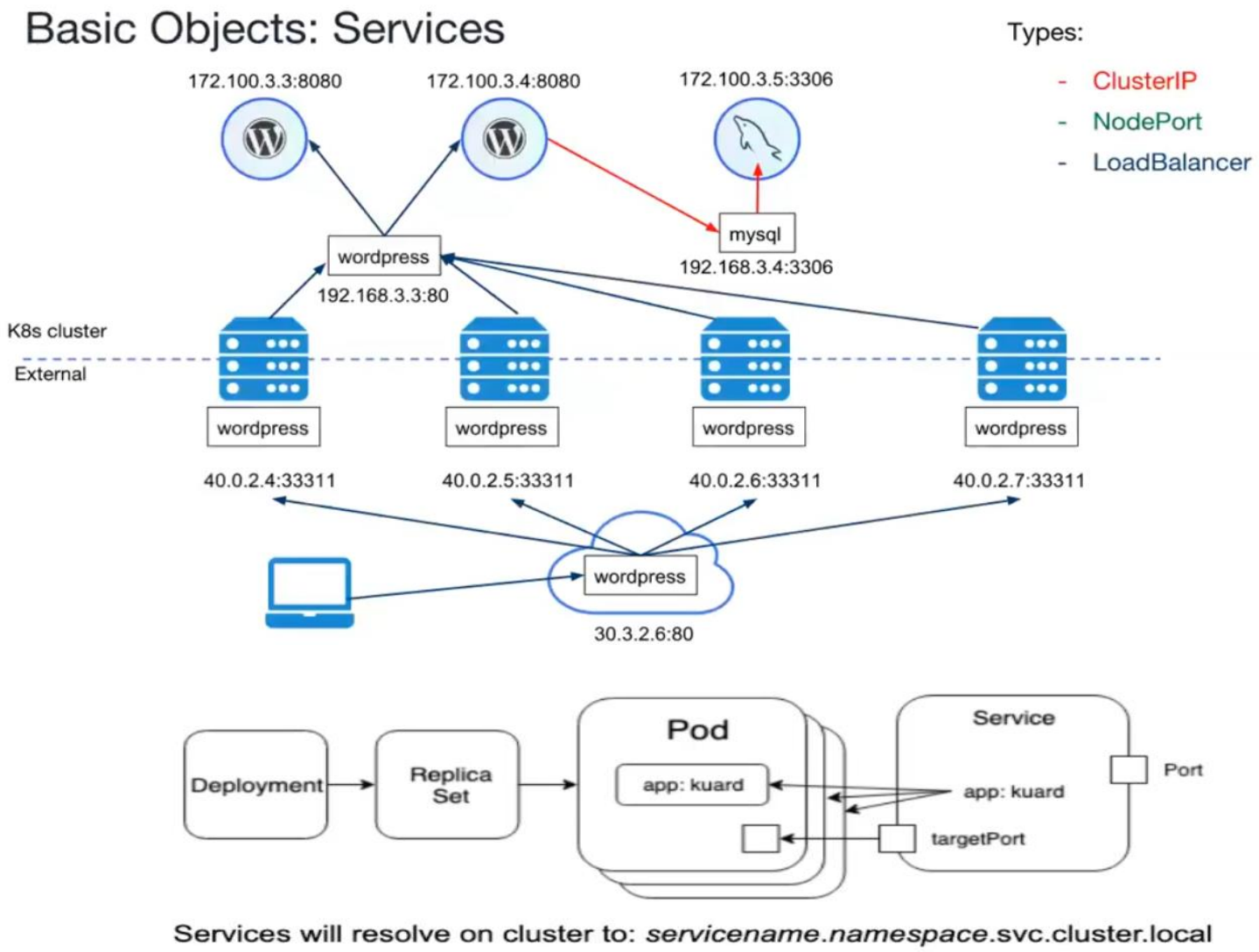
ClusterIP: Exposes the service on a cluster-internal IP. Choosing this value makes the service only reachable from within the cluster. This is the default ServiceType

NodePort: Exposes the service on each Node's IP at a static port (the NodePort). A ClusterIP service, to which the NodePort service will route, is automatically created. You'll be able to contact the NodePort service, from outside the cluster, by requesting <NodeIP>:<NodePort>.

LoadBalancer: Exposes the service externally using a cloud provider's load balancer. NodePort and ClusterIP services, to which the external load balancer will route, are automatically created.



Kubernetes Services



Kubernetes Labels

Las Labels son un mecanismo de consulta y filtrado muy potente que nos ofrece Kubernetes, pero con el que hay que tener especial atención porque es una fuente común de errores en la configuración de nuestras aplicaciones.

Los **Labels** son pares clave/valor que podemos asociar a cualquier objeto de Kubernetes.

Por ejemplo, podemos añadir la clave `environment` en todos mis objetos, y darle el valor `dev`, `staging` o `prod` según el entorno en el que estemos ejecutando.

Esto nos va a permitir hacer consultas filtrando por el valor de estos labels, y así refinar los resultados que recibo.

Pero muy importante, las Labels sirven para muchas más cosas. Ya hemos visto como usarlas para mapear los Pods a los que afecta un Replica Controller, y también para decidir entre qué Pods hace balanceo de carga un Service. Otro uso común es para seleccionar los nodos en los que un Pod puede ser ejecutado.

Otro tipo de labels son de tipo node Selector, para que etiquete mis nodos y despliegue los objetos de kubernetes en estos nodos.

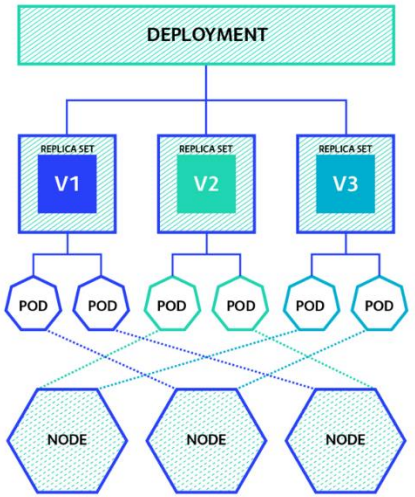
Filtrando pods, por label:

```
~$ kubectl get pods -l app=nginx
```

Kubernetes Deployments

Los Deployments son una abstracción muy útil sobre el concepto de Replica Controllers, y es el objeto que se utiliza como norma general para desplegar nuestras aplicaciones.

Sobre la abstracción del **Replica Controller** ofrece ***rolling updates***. De esta manera, si tenemos 4 instancias de un Pod y queremos desplegar una nueva versión de nuestra aplicación, el Deployment se encarga de ir sustituyendo uno a uno los Pods antiguos por los nuevos, de tal manera que no nos veamos afectados por un *downtime*. Además, el Deployment mantiene un histórico de versiones de los Pods que ha ejecutado, permitiendo hacer *rollbacks* a versiones pasadas si detectamos un problema en producción.



Kubernetes

Useful Commands

Command	Description
kubectl get deployments	Get information on current deployments
kubectl get rs	Get information about the replica sets
kubectl get pods --show-labels	get pods, and also show labels attached to those pods
kubectl rollout status deployment/helloworld-deployment	Get deployment status
kubectl set image deployment/helloworld-deployment k8s-demo=k8s-demo:2	Run k8s-demo with the image label version 2
kubectl edit deployment/helloworld-deployment	Edit the deployment object
kubectl rollout status deployment/helloworld-deployment	Get the status of the rollout
kubectl rollout history deployment/helloworld-deployment	Get the rollout history
kubectl rollout undo deployment/helloworld-deployment	Rollback to previous version
kubectl rollout undo deployment/helloworld-deployment --to-revision=n	Rollback to any version version

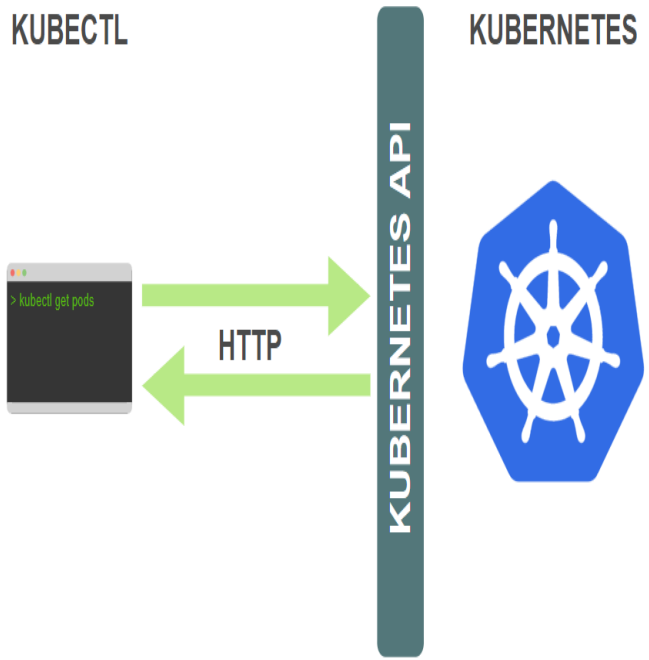
Kubernetes kubectl

¿Qué es kubectl?

Desde el punto de vista de un usuario, kubectl es un comando para controlar Kubernetes. Le permite realizar todas las operaciones posibles de Kubernetes.
Desde un punto de vista técnico, kubectl es un cliente para la **API de Kubernetes** .

La API de Kubernetes es una **API REST de HTTP** . Esta API es la **interfaz de usuario** real de Kubernetes . Kubernetes está totalmente controlado a través de esta API.

Esto significa que cada operación de Kubernetes está expuesta como un punto final de API y puede ejecutarse mediante una solicitud HTTP a este punto final.
En consecuencia, el trabajo principal de kubectl es realizar solicitudes HTTP a la API de Kubernetes:



Kubernetes kubectl

Kubernetes Cheat Sheet

What is Kubernetes?

Kubernetes is a platform for managing containerized workloads. Kubernetes orchestrates computing, networking and storage to provide a seamless portability across infrastructure providers.

Viewing Resource Information

Nodes

```
$ kubectl get no
$ kubectl get no -o wide
$ kubectl describe no
$ kubectl get no -o yaml
$ kubectl get node --selector=[label_name]
$ kubectl get nodes -o jsonpath='{.items[*].status.addresses[?(@.type=="ExternalIP")].address}'
$ kubectl top node [node_name]
```

Pods

```
$ kubectl get po
$ kubectl get po -o wide
$ kubectl describe po
$ kubectl get po --show-labels
$ kubectl get po -l app=nginx
$ kubectl get po -o yaml
$ kubectl get pod [pod_name] -o yaml --export
$ kubectl get pod [pod_name] -o yaml --export > nameoffile.yaml
$ kubectl get pods --field-selector status.phase=Running
```

Namespaces

```
$ kubectl get ns
$ kubectl get ns -o yaml
$ kubectl describe ns
```

Deployments

```
$ kubectl get deploy
$ kubectl describe deploy
$ kubectl get deploy -o wide
$ kubectl get deploy -o yaml
```

Services

```
$ kubectl get svc
$ kubectl describe svc
$ kubectl get svc -o wide
$ kubectl get svc -o yaml
$ kubectl get svc --show-labels
```

DaemonSets

```
$ kubectl get ds
$ kubectl get ds --all-namespaces
$ kubectl describe ds [daemonset_name] -n [namespace_name]
$ kubectl get ds [ds_name] -n [ns_name] -o yaml
```

Events

```
$ kubectl get events
$ kubectl get events -n kube-system
$ kubectl get events -w
```

Logs

```
$ kubectl logs [pod_name]
$ kubectl logs --since=1h [pod_name]
$ kubectl logs --tail=20 [pod_name]
$ kubectl logs -f -c [container_name] [pod_name]
$ kubectl logs [pod_name] > pod.log
```

Service Accounts

```
$ kubectl get sa
$ kubectl get sa -o yaml
$ kubectl get serviceaccounts default -o yaml > ./sa.yaml
$ kubectl replace serviceaccount default -f ./sa.yaml
```

ReplicaSets

```
$ kubectl get rs
$ kubectl describe rs
$ kubectl get rs -o wide
$ kubectl get rs -o yaml
```

Roles

```
$ kubectl get roles --all-namespaces
$ kubectl get roles --all-namespaces -o yaml
```

Secrets

```
$ kubectl get secrets
$ kubectl get secrets --all-namespaces
$ kubectl get secrets -o yaml
```

ConfigMaps

```
$ kubectl get cm
$ kubectl get cm --all-namespaces
$ kubectl get cm --all-namespaces -o yaml
```

Ingress

```
$ kubectl get ing
$ kubectl get ing --all-namespaces
```

PersistentVolume

```
$ kubectl get pv
$ kubectl describe pv
```

PersistentVolumeClaim

```
$ kubectl get pvc
$ kubectl describe pvc
```

Kubernetes kubectl

Kubernetes Cheat Sheet

page 2

Viewing Resource Information (cont.)

StorageClass

```
$ kubectl get sc
$ kubectl get sc -o yaml
```

Multiple Resources

```
$ kubectl get svc, po
$ kubectl get deploy, no
$ kubectl get all
$ kubectl get all --all-namespaces
```

Changing Resource Attributes

Taint

```
$ kubectl taint [node_name] [taint_name]
```

Labels

```
$ kubectl label [node_name] disktype=ssd
$ kubrectl label [pod_name] env=prod
```

Cordon/Uncordon

```
$ kubectl cordon [node_name]
$ kunectl uncordon [node_name]
```

Drain

```
$ kubectl drain [node_name]
```

Nodes/Pods

```
$ kubectl delete node [node_name]
$ kubectl delete pod [pod_name]
$ kubectl edit node [node_name]
$ kubectl edit pod [pod_name]
```

Deployments/Namespaces

```
$ kubectl edit deploy [deploy_name]
$ kubectl delete deploy [deploy_name]
$ kubectl expose deploy [deploy_name]
--port=80 --type=NodePort
$ kubectl scale deploy [deploy_name]
--replicas=5
$ kubectl delete ns
$ kubectl edit ns [ns_name]
```

Services

```
$ kubectl edit svc [svc_name]
$ kubectl delete svc [svc_name]
```

DaemonSets

```
$ kubectl edit ds [ds_name] -n kube-system
$ kubectl delete ds [ds_name]
```

Service Accounts

```
$ kubectl edit sa [sa_name]
$ kubectl delete sa [sa_name]
```

Annotate

```
$ kubectl annotate po [pod_name]
[annotation]
$ kubectl annotate no [node_name]
```

Adding Resources

Creating a Pod

```
$ kubectl create -f [name_of_file]
$ kubectl apply -f [name_of_file]
$ kubectl run [pod_name] --image=nginx
--restart=Never
$ kubectl run [pod_name]
--generator=run-pod/v1 --image=nginx
$ kubectl run [pod_name] --image=nginx
--restart=Never
```

Creating a Service

```
$ kubectl create svc nodeport [svc_name]
--tcp=8080:80
```

Creating a Deployment

```
$ kubectl create -f [name_of_file]
$ kubectl apply -f [name_of_file]
$ kubectl create deploy [deploy_name]
--image=nginx
```

Interactive Pod

```
$ kubectl run [pod_name] --image=busybox
--rm -it --restart=Never -- sh
```

Output YAML to a File

```
$ kubectl create deploy [deploy_name]
--image=nginx --dry-run -o yaml >
deploy.yaml
$ kubectl get po [pod_name] -o yaml --export
> pod.yaml
```

Getting Help

```
$ kubectl -h
$ kubectl create -h
$ kubectl run -h
$ kubectl explain deploy.spec
```

Requests

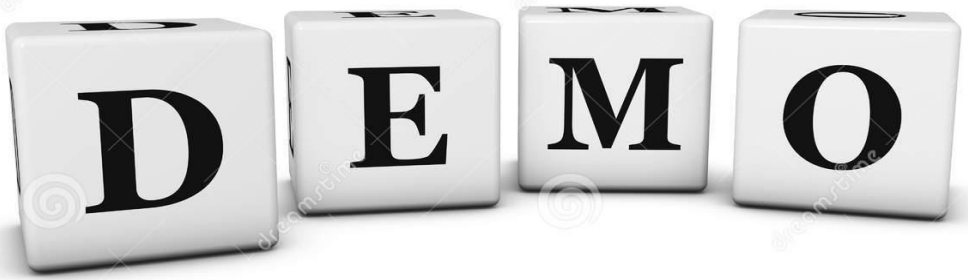
API Call

```
$ kubectl get --raw /apis/metrics.k8s.io/
```

Cluster Info

```
$ kubectl config
$ kubectl cluster-info
$ kubectl get componentstatuses
```

Kubernetes



Kubernetes Ingress

Ingress

https://kubernetes.io/docs/concepts/services-networking/ingress/

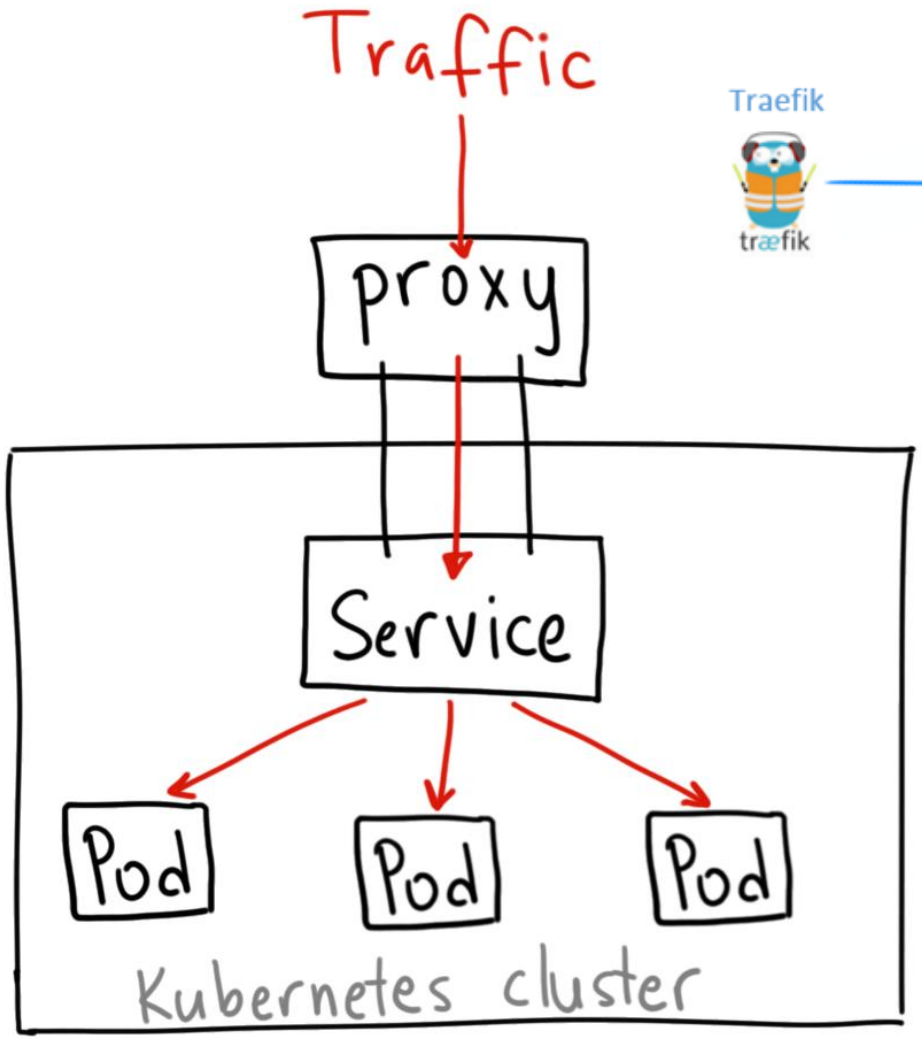
Cuando tenemos que exponer nuestros servicios a tráfico externo hemos visto que podemos crear un servicio de tipo Load Balancer, pero esta solución puede resultar demasiado rígida y costosa. La alternativa es utilizar un Ingress.

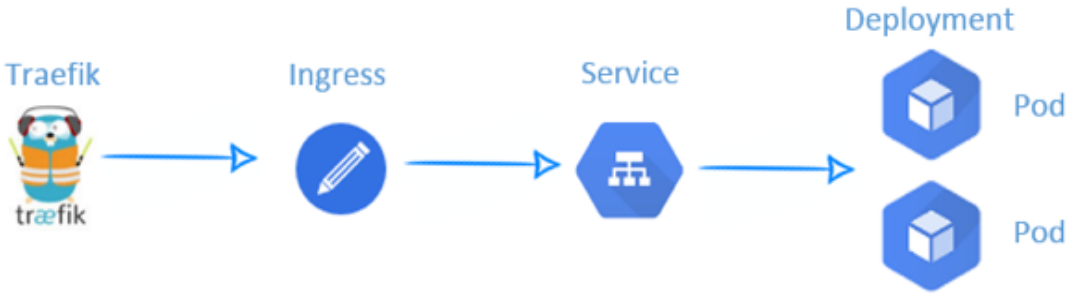
Un Ingress nos permite definir rustas de entrada a nuestro servicio de una manera programática. Existen numerosos Ingress Controllers, como el de Nginx, HAProxy o traefik, cada uno de ellos permitiendo distintas configuraciones.

Por ejemplo, con el Nginx Ingress Controller, podemos definir a dónde redirigir una petición en base al dominio solicitado, o al `path` solicitado dentro de esa petición.

Hay aplicaciones que dependen mucho de la ruta donde se expongan, entonces como programadores o administradores tenemos que saber en que rutas o dominios se expone su aplicación.

Kubernetes Ingress





Kubernetes

Kubernetes Namespacess

Los ***Namespaces*** (espacios de nombres) en Kubernetes permiten establecer un nivel adicional de separación entre los contenedores que comparten los recursos de un clúster.

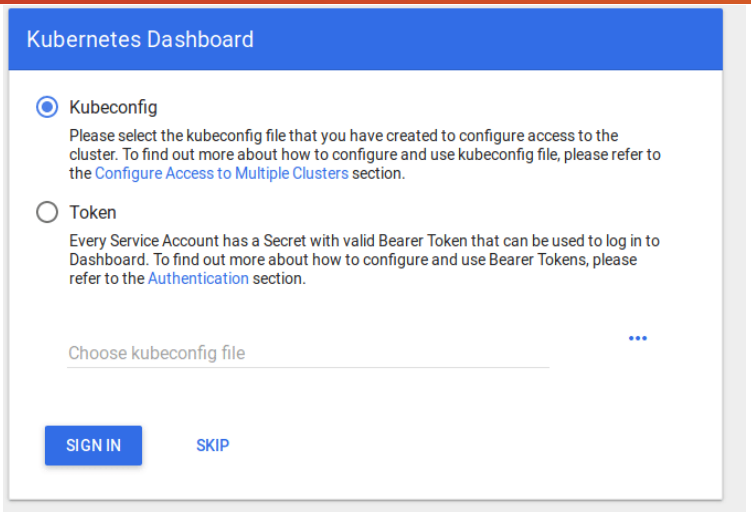
Esto es especialmente útil cuando diferentes grupos de trabajo usan el mismo clúster y existe el riesgo potencial de colisión de nombres de los *pods*, etc usados por los diferentes equipos.

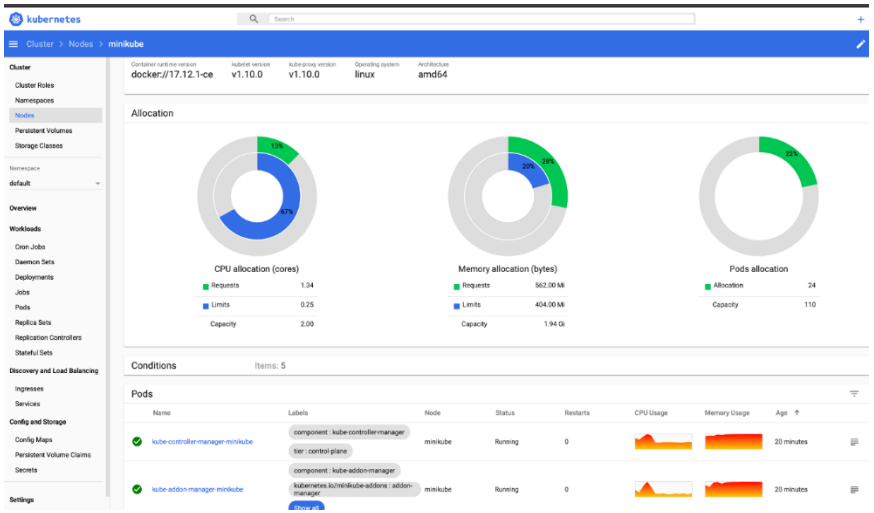
Los espacios de nombres también facilitan la creación de cuotas para limitar los recursos disponibles para cada *namespace*. Puedes considerar los espacios de nombres como clústers *virtuales* sobre el clúster físico de Kubernetes. De esta forma, proporcionan separación lógica entre los entornos de diferentes equipos.

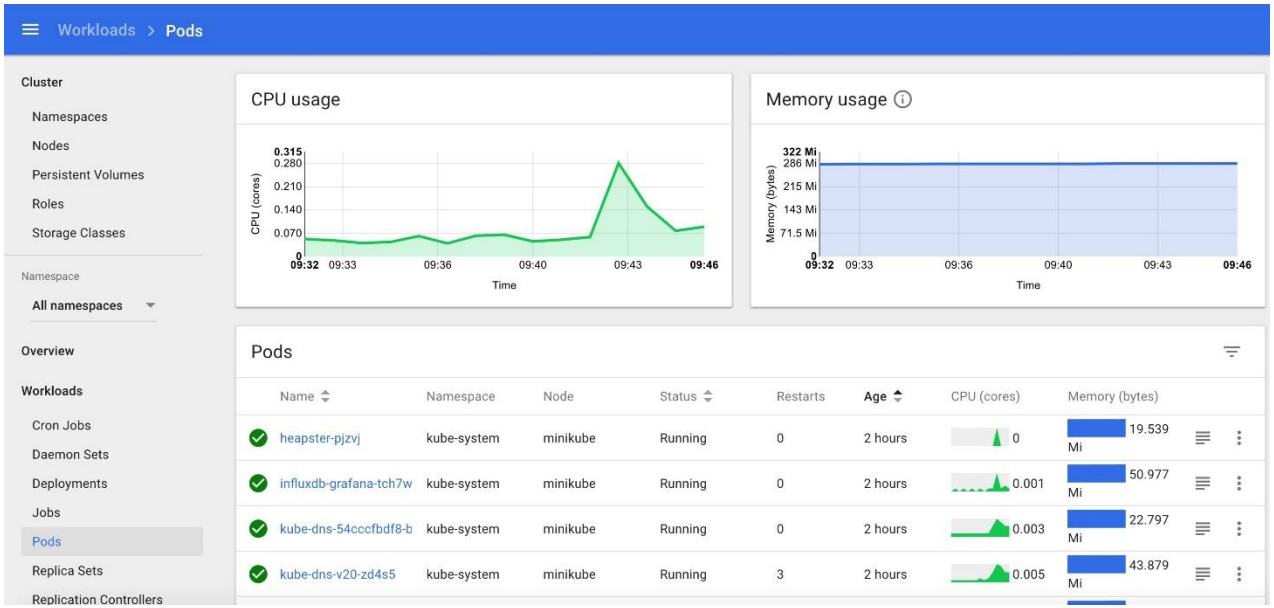
Kubernetes proporciona dos namespaces por defecto**: kube-system** y **default**.

Los objetos “de usuario” se crean en el espacio de nombres **default**, mientras que los de “sistema” se encuentran en **kube-system**.

Kubernetes Dashboard



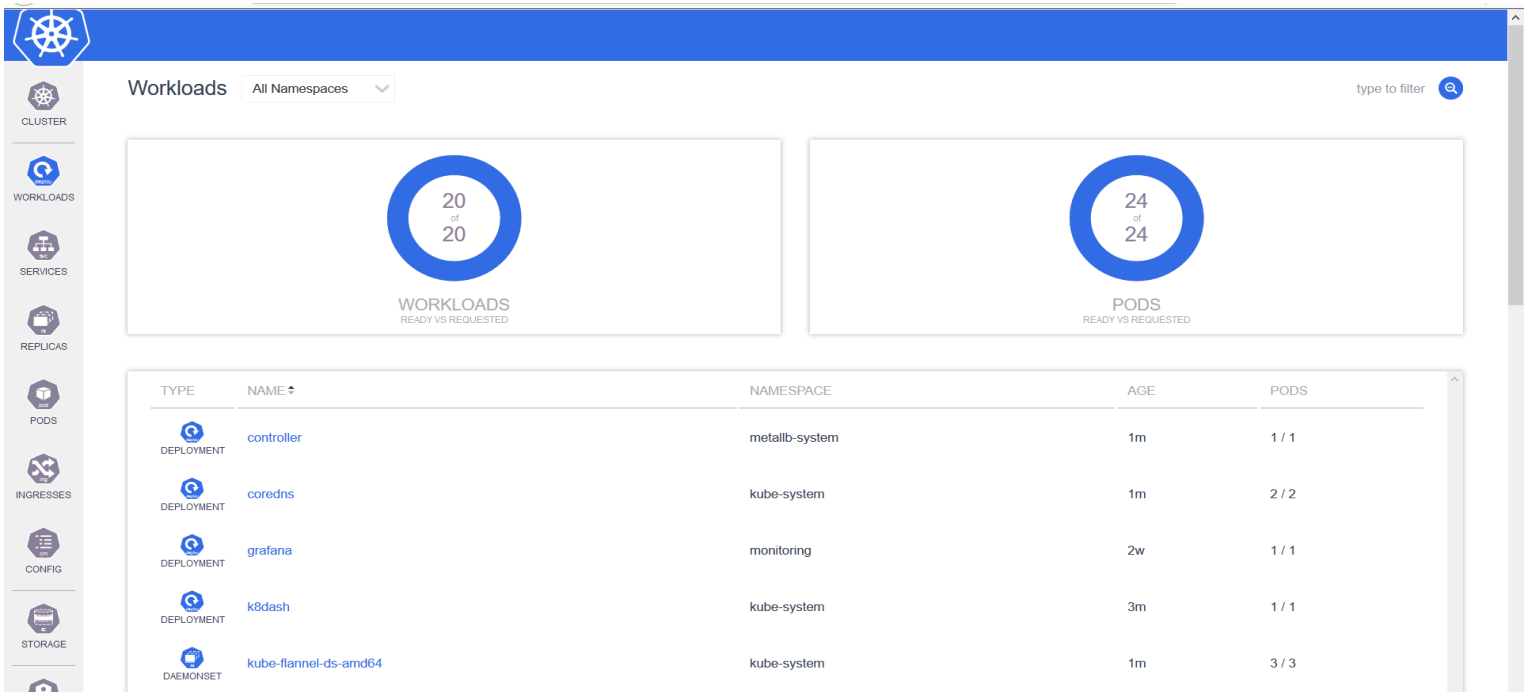




Kubernetes Dashboard

K8Dash - Kubernetes Dashboard

https://github.com/herbrandson/k8dash



Kubernetes

GRACIAS


THANK
YOU
GRACIAS
ARIGATO
SHUKURIA
BOLZÏN
MERCI
BÏYAN
SHUKRIA
DANKSCHEEN
TASHAKKUR ATU
YAQHANYELAY
SUKSAMA
EKHMET
MEHRBANI
GRAZIE
MAAKE
KOMAPSUMNIDA
JUSPAXAR
GOZAIMASHITA
EFCHARISTO
PALDIES
TINGKI
SPASSIBO
SNACHALHUYA
NUHUN
CHALTU
WABEEJA
MAITEKA
YUSPAGARÀTAM
HUI
GUI
HATUR
EKOJU
SIKOMO
MAKETAI
MINMONCHAR
SPASIBO
DENKAUJA
NENACHALHYA
UNALCHÉESH
MERSI
ATTO
ANHA
DHANYABAAD
LAH
SANCO
MERASTAWHY
GAEJTHO
TAVTAPUCH
MEDAWAGSE
AGUYJE
FAKAAUE
BAIKA