Docker y Kubernetes

En busca de la alta disponibilidad

$ whoami

¡Hola! Mi nombre es **Óscar de Arriba**

- Desarrollo principalmente en Elixir y Ruby.
- Experiencia gestionando sistemas de alta concurrencia.
- Actualmente trabajando en Bizneo HR
- Tengo dos gatas que son monísimas

¿Qué es esto de los contenedores?

Introducción a los contenedores

Los contenedores son la forma de empaquetar y distribuir software que está de moda desde hace varios años.

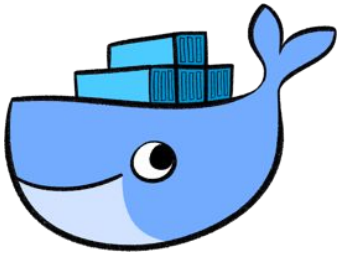
- Permite empaquetar un programa compilado con todo lo necesario para funcionar.
- Tiene un toolset que facilita su uso en *prácticamente* cualquier plataforma.
- Añade una capa extra de seguridad.
- Chroot

Los contenedores **no** son sólo Docker

La tecnología de contenedores se basa en una serie de funcionalidades incluidas en el kernel de Linux, que permiten crear entornos virtuales aislados.

Los contenedores son:

- Ejecución controlada del programa con recursos limitados
- Sistema de archivos virtual (*layered*)
- Interfaces virtuales de red

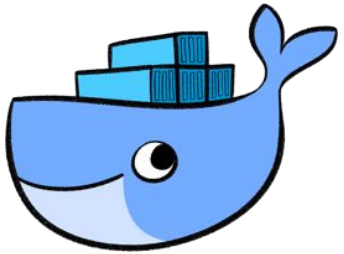


Los contenedores **no** son sólo Docker

Múltiples gestores de contenedores:

- Docker
- LXD
- rkt

El más popular es Docker: gran aceptación, toolset conocido y disponible en otras plataformas (OSX y Windows)



DEMO

Los contenedores en producción

Los contenedores por sí mismos no están listos para producción:

- Múltiples servicios - múltiples contenedores coordinados
- Escalabilidad
- Alta disponibilidad

Docker Compose: permite coordinar múltiples contenedores, pero no es *production ready*.

Kubernetes

Kubernetes

Software de orquestación de contenedores

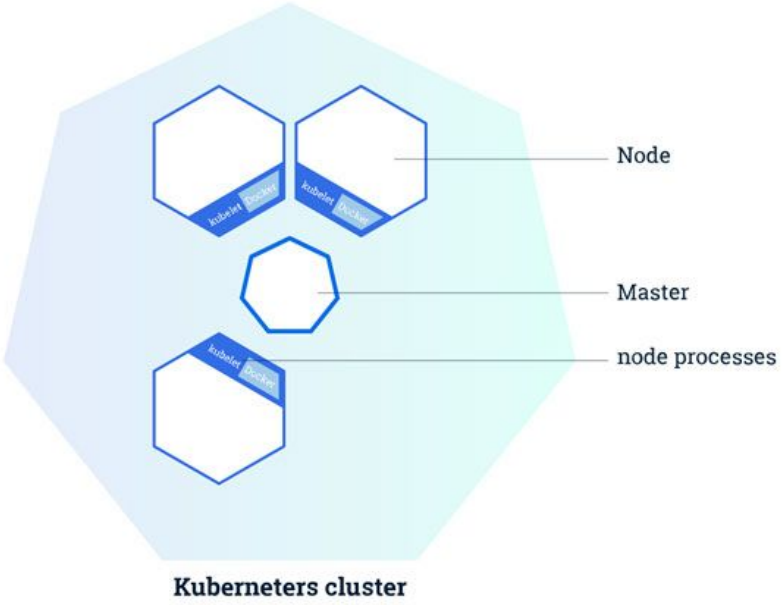
- Liberado por Google en 2014.
- Contenedores **distribuidos** entre servidores.
- Puede coordinar diferentes tipos de contenedores.
- Gestión de logs y métricas de servicios.
- Todo Kubernetes puede ejecutarse desde contenedores Docker.

Hay alternativas: Docker Swarm, Rancher, Apache Mesos...

Kubernetes: Estructura

Un cluster de Kubernetes está compuesto por varios servidores:

- Nodo(s) Master: controla la ejecución de procesos y la configuración.
- Nodos Worker: ejecuta los contenedores y se encargan de recibir toda la carga y el tráfico.

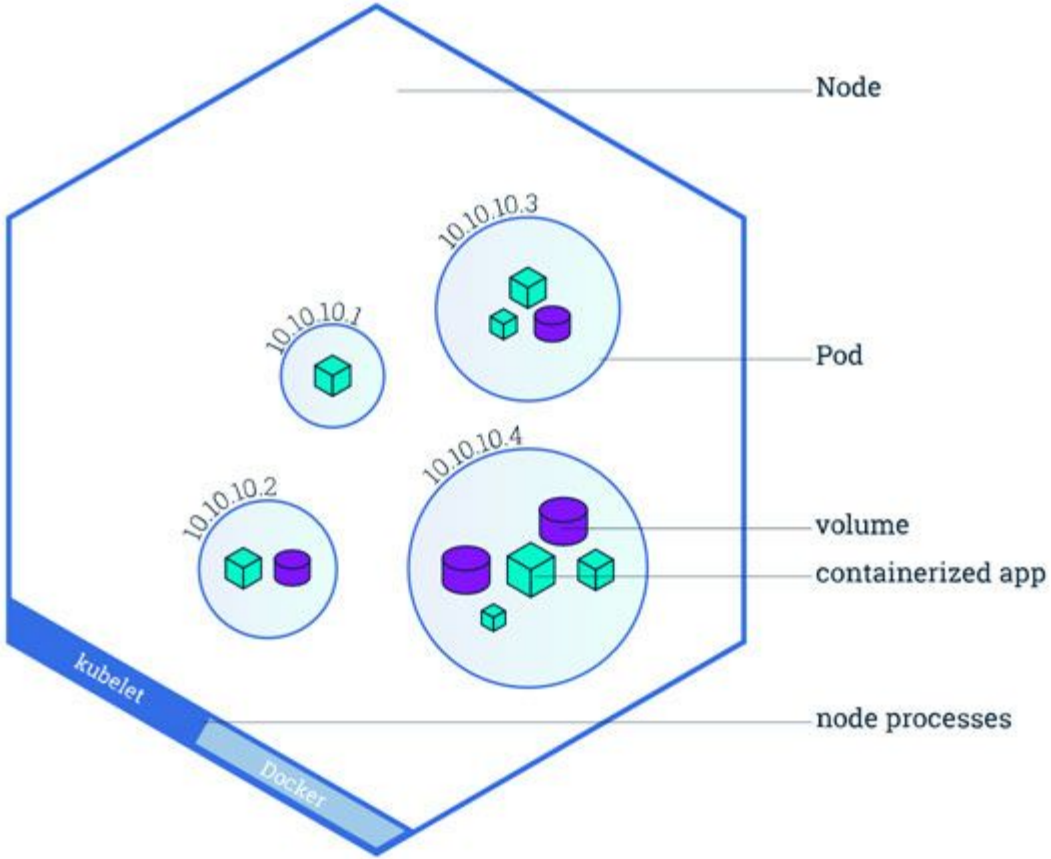


Kubernetes: Organización

Entre todos los nodos forman una red privada virtual para comunicar todos los contenedores entre sí.

Los contenedores se despliegan definiendo **deployments**:

- Múltiples contenedores formando un **pod**.
- Todos los servicios expuestos por los contenedores de un pod se exponen en una misma IP interna dentro del cluster.
- Puede haber varias copias (**replicas**) de un pod en diferentes nodos físicos.


Node
10.10.10.3
10.10.10.1
Pod
10.10.10.4
10.10.10.2
volume
containerized app
kubelet
Docker
node processes

Kubernetes: Organización

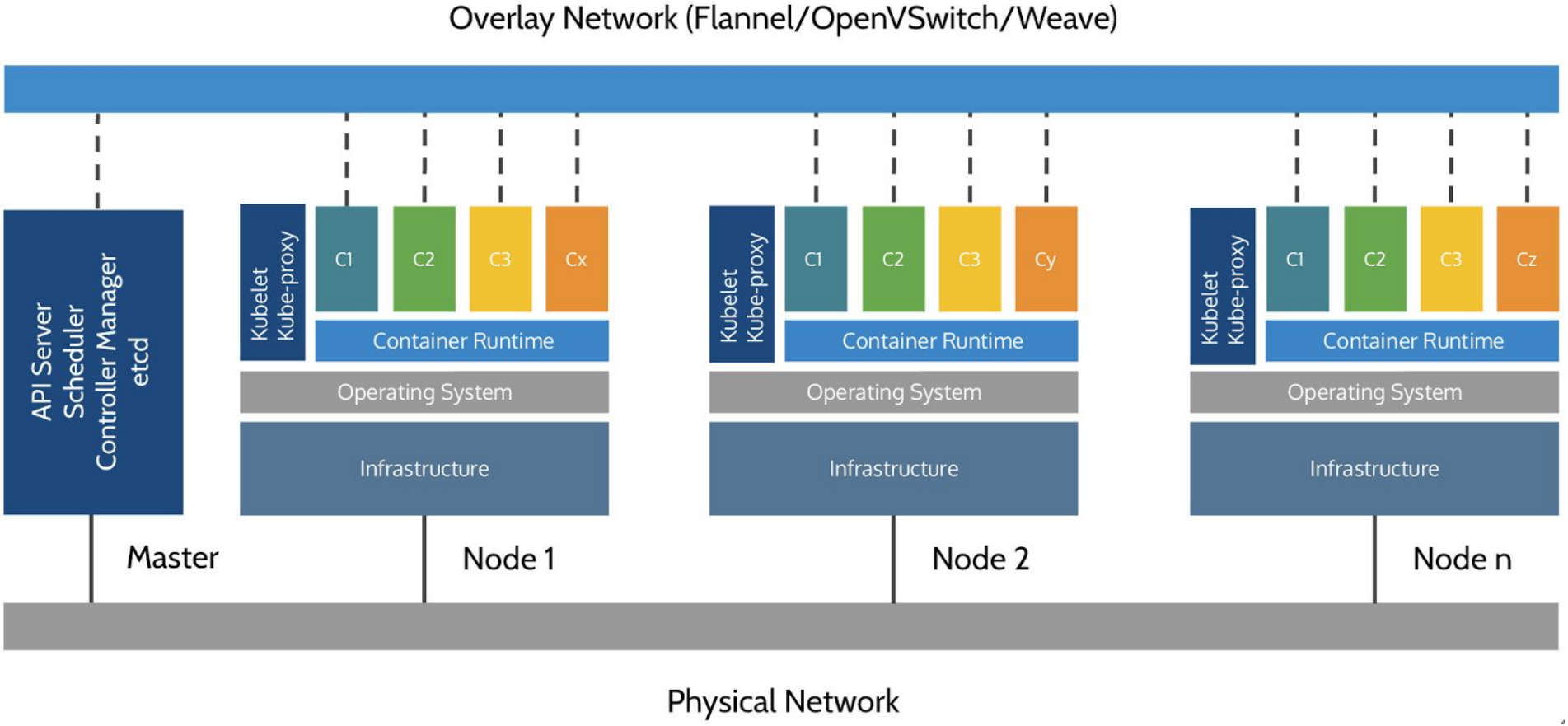
Para exponer puertos se definen Servicios:

- Se define un nombre del servicio, los pods que tienen el servicio y en qué puertos.
- Permite mapear esos puertos a Internet
- Cada servicio genera un nombre de dominio interno para acceder a él:
  - *my-svc.default.svc.cluster.local A 10.0.2.48, 10.0.3.25, 10.0.4.67*

Hay otros elementos dentro de un cluster de Kubernetes (como los Configmap, los Secrets).

Kubernetes: Servicios internos

- **kubelet**: servicio en cada worker que controla los contenedores que corren en ese nodo.
- **kube-proxy**: en cada nodo worker, hace redirección de puertos entre la red y los contenedores en base a los servicios definidos.
- **kube-dns**: servicio DNS interno para descubrir las IPs internas de los contenedores.
- **API Server**: servicio que corre en los nodos master y se encarga de la gestión de:
  - salud, disponibilidad y reparto de carga de los nodos
  - gestión por parte de los administradores del cluster
- **etcd**: base de datos clave valor de alta disponibilidad para almacenar toda la configuración y el estado del cluster.


Overlay Network (Flannel/OpenVSwitch/Weave)
API Server
Scheduler
Controller Manager
etcd
Kubelet
Kube-proxy
C1
C2
C3
Cx
Container Runtime
Operating System
Infrastructure
Kubelet
Kube-proxy
C1
C2
C3
Cy
Container Runtime
Operating System
Infrastructure
Kubelet
Kube-proxy
C1
C2
C3
Cz
Container Runtime
Operating System
Infrastructure
Master
Node 1
Node 2
Node n
Physical Network

Alta disponibilidad

Alta disponibilidad: definición

Alta disponibilidad (HA): característica de un sistema que intenta asegurar el nivel de rendimiento operacional acordado (generalmente el *uptime*) por un periodo superior al normal.

https://en.wikipedia.org/wiki/High_availability

Alta disponibilidad del 100%, ¿es posible?

NO.

Alta disponibilidad del 100%, ¿por qué no?

Siempre habrá comunicaciones o procesos que estén en progreso durante el fallo:

- Una petición web que esté ejecutándose cuando se caiga el servidor.
- Una conexión streaming cuando se cae el enlace de fibra.
- Un proceso en background mientras se produce un *kernel panic*.

Todos estos casos acabarían con una falta de disponibilidad, aunque sólo afecte al usuario/proceso en cuestión y se recupere con la siguiente petición/ejecución.

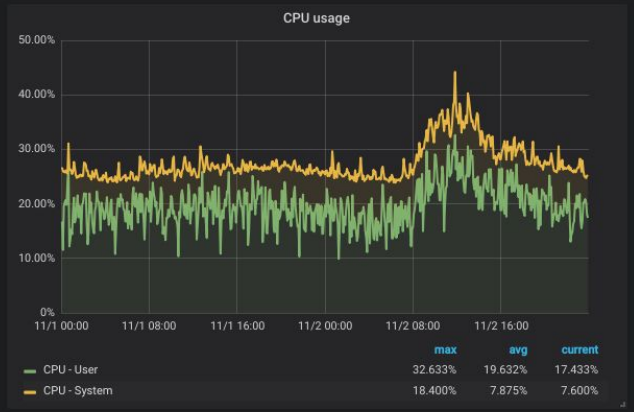
Alta disponibilidad: objetivos y soluciones

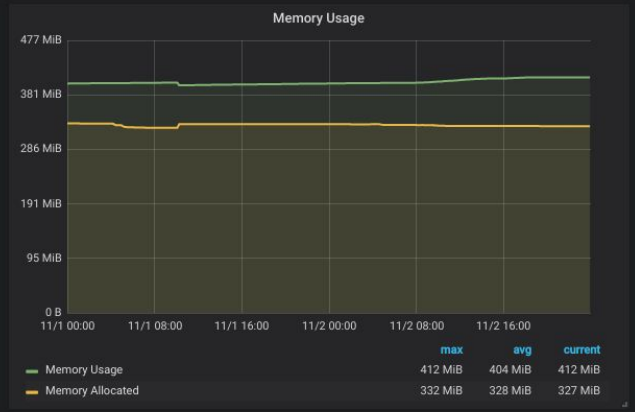
- Objetivos:
  - Poder cumplir los objetivos marcados (*n* nueves).
  - Que el sistema responda en el tiempo esperado sea cual sea su carga.
- Soluciones:
  - Monitorización
  - Escalado

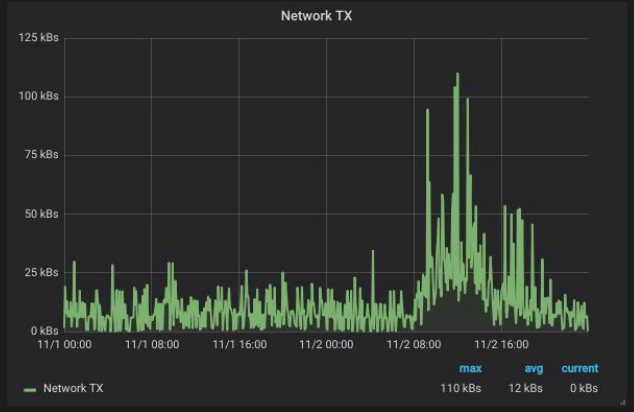
Redis

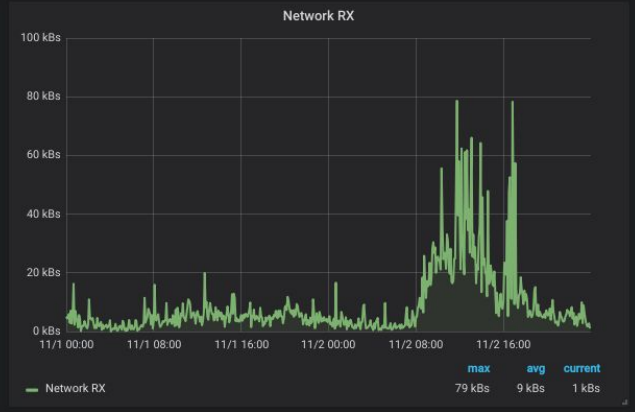
System

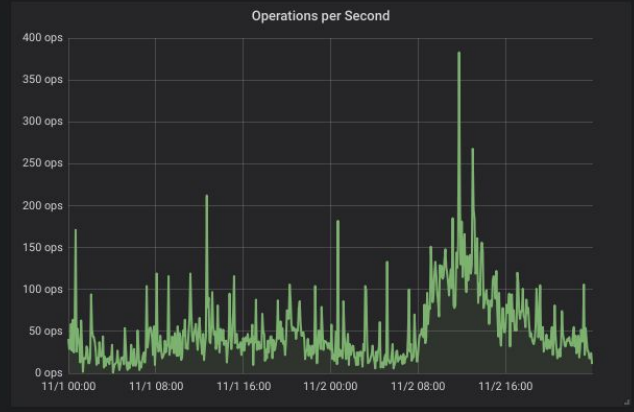
now-3d/d to now-2d/d Refresh every 5s







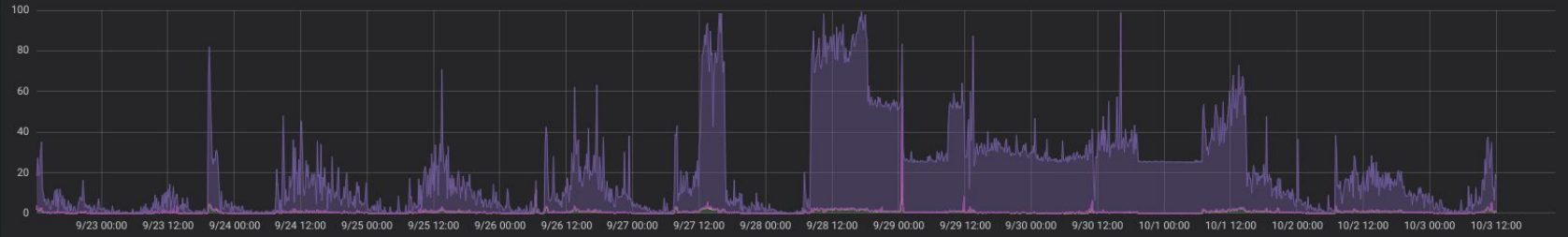




Hosts

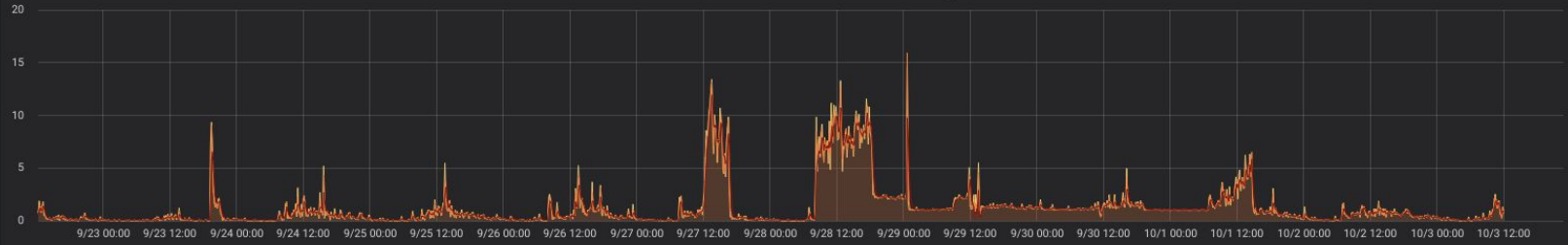
Sep 22, 2018 12:18:13 to Oct 3, 2018 22:45:37

CPU usage



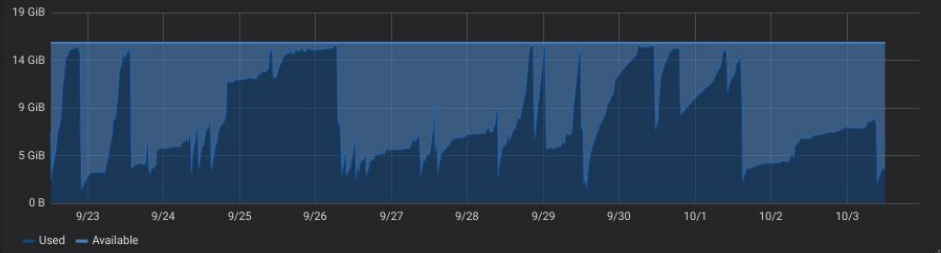
	min	max	avg
system	0	26	1
nice	0	1	0
irq	0	0	0
softirq	0	1	0
iowait	0	31	0
steal	0	0	0
guest	0	0	0
user	0	98	19

Load average

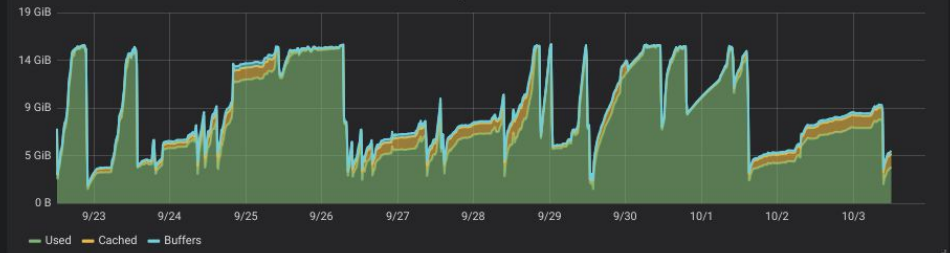


	min	max	avg
Load 1m	0	14.57	1.17
Load 5m	0.01	15.93	1.17
Load 15m	0.05	11.93	1.18

Memory



Memory Distribution



Alta disponibilidad en Kubernetes

HA con Kubernetes

Algunos comportamientos vienen por defecto:

- Si un proceso se muere, el contenedor muere y K8s lo reinicia.
- Si un nodo worker desaparece o se vuelve *unhealthy*, K8s arranca los contenedores que tenía en otro nodo para continuar aguantando el tráfico.
- El tráfico se reparte entre todos los nodos aunque sólo entre por uno de ellos.
- Monitorización de uso de recursos (CPU y RAM) de contenedores.

HA con Kubernetes: despliegues

Los despliegues en Kubernetes se puede realizar usando ***rolling deployments***:

- Conviven contenedores con la versión nueva y la vieja al mismo tiempo.
- Se van renovando poco a poco, apagándose de forma controlada y sin tráfico.
- Puede tener inconvenientes:
  - Migraciones
  - Software clusterizado

HA con Kubernetes: Mejoras

Podemos comprobar cuándo un contenedor está en funcionamiento

- Hay dos tipos de pruebas:
  - Readiness probe: prueba a realizar para saber cuándo el contenedor está listo tras arrancar.
  - Liveness probe: prueba a realizar periódicamente para ver si el contenedor sigue en activo.
- Pueden ser de múltiples tipos:
  - Que un puerto TCP esté abierto
  - Que una petición HTTP responda con un status code concreto.
- Si un contenedor tarda mucho en estar listo, o deja de responder a los *liveness probes*, se reinicia.

HA con Kubernetes: Mejoras

Podemos limitar el uso de recursos por contenedor

- Se mide en Megabytes de RAM y en miliCPUs (100% de un core == 1000m)
- Se asignan “tramos” de recursos (el contenedor empieza con un tramo) y un límite máximo:
  - Incrementos de 64 Mb de RAM y 100m
  - Límite de 512 Mb de RAM y 2000m
- Si el contenedor llega al límite y sigue pidiendo recursos de forma sostenida, se reinicia.

HA con Kubernetes: Mejoras

También podemos autoescalar los contenedores:

- Podemos definir escalados en base a métricas del sistema (número de requests, uso de CPU, uso de RAM, etc...).
- El escalado tiene un número mínimo y máximo de réplicas.
  - el número de réplicas es en global al cluster, no por nodo
- Hay un tiempo entre escalados y desescalados para evitar el desajuste de re-arrancar contenedores demasiado rápido.

HA con Kubernetes: Mejoras

Hasta ahora hemos hablado de escalar contenedores, pero el número de recursos disponibles es acotado: sólo tenemos x nodos worker.

Algunos proveedores (como Google Cloud) permiten el autoescalado de nodos worker para soportar situaciones como un aumento de RAM y de número de réplicas por encima de las capacidades del cluster.

DEMO

HA con Kubernetes: Resumen

- Podemos ejecutar nuestros software en un entorno controlado.
- Podemos hacer que nuestro software tenga más capacidad escalándolo horizontalmente (más réplicas).
- Podemos limitar su uso de recursos (evitar *memory leaks*).
- Si nuestro software falla, el contenedor se reinicia sólo.

Conclusiones

Cosas que sí hay que hacer

- Conocer nuestro software: recursos que necesita, servicios de los que depende, rangos de funcionamiento aceptables.
- Buscar puntos de fallo y situaciones límite.
- Limitar el tiempo de caída estableciendo protecciones como el límite de recursos o las *liveness probes*.
- Prepararse para picos de tráfico con autoescalado de contenedores y/o nodos.

Cosas que sí hay que hacer

Y lo más importante, asumir que nuestro sistema **en algún momento se va a caer o va a fallar** porque los sistemas no son perfectos.

Y lo mejor que podemos hacer es intentar ser proactivos para que esto ocurra durante el menor tiempo posible.

FIN

¡Gracias!

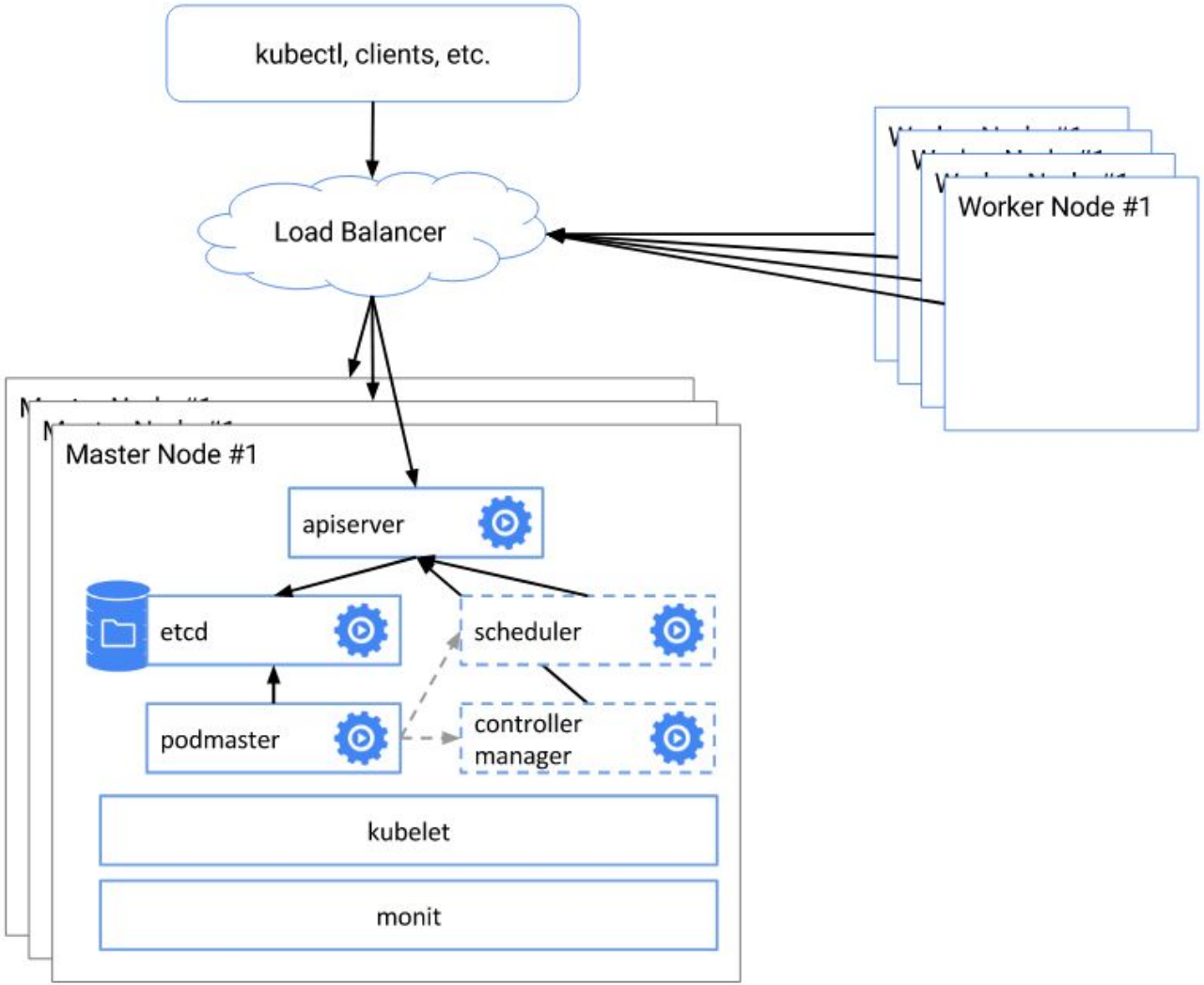
Todo lo que quisiste saber de Kubernetes y nunca te han contado

Desastres en Kubernetes

- ¿Qué pasa *exáctamente* si se cae un nodo worker?
  - Hay un tiempo de espera hasta dar el nodo por perdido.
  - Cuando se pierde un nodo, se deja de dirigir tráfico a él y se arrancan sus contenedores en otros nodos.
- ¿Qué pasa exáctamente si se cae el nodo master?
  - Los workers siguen funcionando con sus contenedores, aunque la capacidad del clúster se degrada: se pierde la capacidad de descubrir en qué nodos tenemos qué contenedores.
  - Cada nodo sigue pudiendo atender las peticiones de los contenedores que tiene o de otros que conoce de antes de la caída.
  - Los comando de administración no funcionan.
  - El autoescalado deja de funcionar.
  - Alguna alarma *debería* sonar.

Evitar desastres en Kubernetes

- ¿Puedo tener tener alta disponibilidad del nodo master?
  - Sí, levantando varios nodos master. Pero requiere ciertas cosas:
    - Kubernetes usa una cadena de certificados autogenerados, que deben ser iguales en todos los nodos master.
    - Los workers deben conocer todos los nodos master para poder comunicarse con uno que esté sano (un Load Balancer serviría).
  - Todos los nodos master deben apuntar al mismo **etcd.**


kubectl, clients, etc.
Load Balancer
Worker Node #1
Master Node #1
apiserver
etcd
scheduler
podmaster
controller
manager
kubelet
monit

WTF: ¿qué era **etcd**?

Etcd es una base de datos clave-valor en la que kubernetes guarda todo:

- Estado de los nodos
- Contenedores en cada nodo
- Configuración de deployments
- Estado de autoescalados
- Eventos que han ocurrido en el clúster
- Tokens, claves y secretos

Evitar desastres en Kubernetes

- ¿Es importante etcd? ¿Puedo tener alta disponibilidad?
  - Es tan importante, que si se pierde todo el cluster dejará de funcionar y toda la configuración tendrá que ser regenerada.
  - **etcd está preparado para tener alta disponibilidad**, incluso para soportar particiones de red.
  - **Podemos y debemos** tener alta disponibilidad y backups periódicos de etcd.
  - https://coreos.com/etcd/docs/latest/v2/clustering.html

¿Puede alguien hacerme esto?

La mayoría de entornos gestionados de Kubernetes (Google Cloud, AWS, DigitalOcean, etc...) se encargan de la gestión y administración de los nodos (tanto workers como masters), por lo que **todo esto está cubierto.**