Curso de kubernetes

Septiembre 2020

Ponentes

José Domingo Muñoz Rodríguez
Alberto Molina Coballes

Índice del curso

https://github.com/iesgn/curso_kubernetes_2020

Unidad 1

Despliegue de aplicaciones en contenedores

Introducción a k8s

Índice

- Contenedores
- Microservicios
- Tecnologías subyacentes. LXC, docker
- Kubernetes (k8s)

Contenedores

Contenedores

- ¿Para qué sirve un sistema operativo?¿Qué es un proceso?
- Compartir o no compartir, ésa es la cuestión: .so., dependencias
- ¿Qué es un contenedor y para qué se utiliza?
- Precedentes en linux
  - chroot
  - OpenVZ
  - Linux vservers
- Precedentes en otros sistemas operativos: FreeBSD Jails, Solaris Zones, etc.

Contenedores

- El gran hito: inclusión de cgroups y namespaces en el kérnel linux (a partir de 2007)
- **cgroups** (límite de memoria, cpu, I/O o red para un proceso y sus hijos)

  https://wiki.archlinux.org/index.php/Cgroups
- **cgroupsv2** (rootless containers)

  https://medium.com/nttlabs/cgroup-v2-596d035be4d7
- **namespaces**: proporcionan un punto de vista diferente a un proceso (interfaces de red, procesos, usuarios, etc.)

  http://laurel.datsi.fi.upm.es/~ssoo/SOA/namespaces.html
- Todo esto unido a la expansión de linux en el centro de datos ha provocado la explosión en el uso de contenedores de los últimos años.
- Cgroups, namespaces, and beyond: what are containers made from?

Despliegue de apliacaciones

Microservicios

Microservicios. Aplicación monolítica

- Todos los componentes en el mismo nodo
- Escalado vertical
- Arquitectura muy sencilla
- Suele utilizarse un solo lenguaje de programación (o un conjunto pequeño de ellos)
- No pueden utilizarse diferentes versiones de un lenguaje de programación a la vez
- Interferencias entre componentes en producción
- Complejidad en las actualizaciones. Puede ocasionar paradas en producción
- Infraestructura estática y fija por años
- Típicamente la aplicación no es tolerante a fallos

Microservicios. Aplicación distribuida

- Idealmente un componente por nodo
- Escalado horizontal
- Arquitectura más compleja
- Menos interferencias entre componentes
- Mayor simplicidad en las actualizaciones
- Diferentes enfoques no excluyentes: SOA, cloud native, microservicios, ..

Microservicios. SOA

- SOA: Service Oriented Architecture
- Servicios independientes
- Múltiples tecnologías, lenguajes y/o versiones interactuando
- Comunicación vía SOAP
- Uso de XML, XSD y WSDL
- Colas de mensajes
- Se relaciona con aplicaciones corporativas
- Se le achaca mucha complejidad y no ha terminado de extenderse

Microservicios. Cloud Native Application

- Énfasis en la adaptación de la infraestructura a la demanda
- Uso extensivo de la elasticidad: Infraestructura dinámica
- Aplicaciones resilientes
- Elasticidad horizontal
- Automatización
- Puede ser complejo de implementar en una aplicación

Microservicios

- Deriva del esquema SOA
- No existe una definición formal, ni WSDL, XSD, etc. Solución más pragmática
- Servicios llevados a la mínima expresión (idealmente un proceso por nodo): Microservicios
- Comunicación vía HTTP REST y colas de mensajes, ¿gRPC?
- Relacionado con procesos ágiles de desarrollo, facilita enormemente la actualizaciones de versiones, llegando incluso a la entrega continua o despliegue continuo.
- Suele implementarse sobre contenedores
- Aumento de la latencia entre componentes
- Puede utilizar características de “cloud native application”

Microservicios. Ejemplo

- OpenStack
  - Cada componente es un microservicio que puede ejecutarse en un nodo independiente
  - kolla-ansible: Despliegue de OpenStack con ansible en múltiples contenedores docker https://elatov.github.io/2018/01/openstack-ansible-and-kolla-on-ubuntu-1604/

Tecnologías de contenedores

LXC

- Características
  - Espacios de nombres del kérnel
  - Apparmor y SELinux
  - Chroots (pivot root)
  - Kernel capabilities
  - CGroups
- Comienza su desarrollo en 2008
- Licencia LGPL
- Desarrollado principalmente por Canonical

  http://linuxcontainers.org

LXC

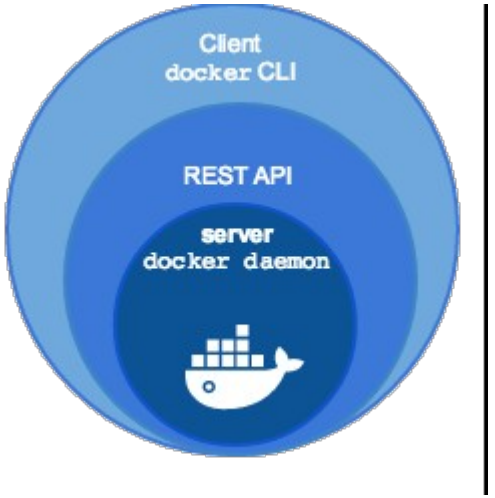
- Pertenece a los denominados contenedores de sistemas
- Gestiona contenedores directamente sin “adornos” y a bajo nivel
- No compite con docker sino con otros sistemas de virtualización
- No hay nuevos conceptos, es otro sistema de virtualización en la que todos los contenedores tienen el mismo kérnel
- No hay nuevos paradigmas de uso
- Utiliza pivot root para definir el directorio raíz del contenedor en un directorio
- No hay que definir un LXCFile ni nada que se parezca ;)
- Para acceder al contenedor utilizamos ssh(!)
- Instalación simple: apt install lxc
- LXD: LXC + demonio + CLI unificado + imágenes

Docker

- “docker”: estibador
- Pertenece a los denominados contenedores de aplicaciones
- Gestiona contenedores a alto nivel proporcionando todas las capas y funcionalidad adicional
- Nuevo paradigma. Cambia completamente la forma de desplegar y distribuir una aplicación
- Docker: build, ship and run
- Lo desarrolla la empresa Docker, Inc.
- Instalación y gestión de contenedores simple
- El contenedor ejecuta un comando y se para cuando éste termina, no es un sistema operativo al uso, ni pretende serlo
- Escrito en go
- Software libre (ha ido cambiando con el tiempo)

Docker

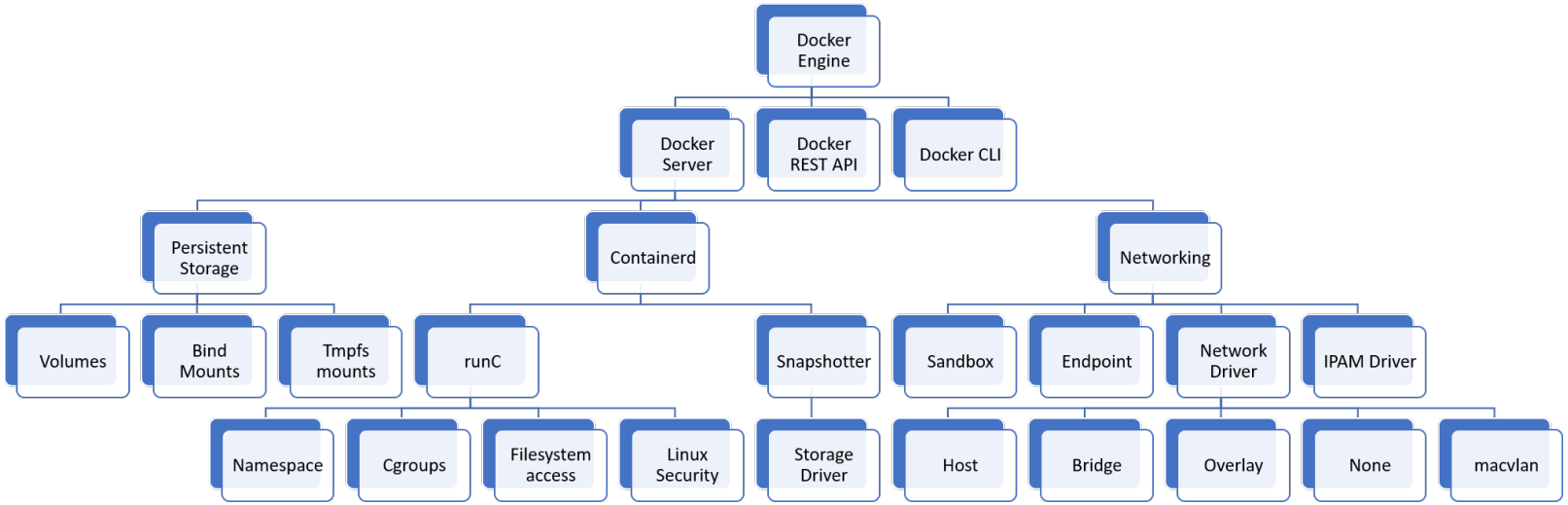
- docker engine
  - demonio docker
  - docker API
  - docker CLI
- docker-machine
  - Gestiona múltiples docker engine
- docker compose
  - Para definir aplicaciones que corren en múltiples contenedores
  - Ejemplo: https://github.com/bitnami/bitnami-docker-wordpress/blob/master/docker-compose.yml
- docker swarm
  - Orquestador de contenedores



Docker. Evolución del proyecto

- El dilema de docker inc. entre el éxito y el negocio
- OCI: Open Containers Initiative
- runtime-spec: http://www.github.com/opencontainers/runtime-spec
- image-spec: http://www.github.com/opencontainers/image-spec
- distribution-spec: https://github.com/opencontainers/distribution-spec
- El cambio en docker
  - Moby (proyecto de comunidad) ← docker.io de debian
  - docker CE (docker engine proporcionado por Docker inc)
  - docker EE (docker engine + servicios de Docker inc)
  - runc (OCI) https://github.com/opencontainers/runc
  - containerd (CNCF) https://github.com/containerd/containerd

Docker. Componentes



- Si vamos a utilizar un orquestador diferente a docker swarm, ¿necesitamos docker engine o containerd?
- runc es equivalente a lxc
- Tanto runc como containerd son proyectos de software libre hoy en día independientes de docker inc.

Alternativas a docker

- rkt, inicialmente desarrollado por CoreOS. Actualmente dentro de la Cloud Native Computing Foundation: https://github.com/rkt/rkt y enfocado a ser una alternativa a containerd
- cri-o. Creado por Red Hat como alternativa a containerd https://cri-o.io/ y pensado solo para funcionar integrado en kubernetes
- Podman. Creado por Red Hat como alternativa a docker https://podman.io
- Pouch. Desarrollado por Alibaba como alternativa a docker. https://pouchcontainer.io/#/
- ¿Micromáquinas virtuales?
  - Kata containers. MVs ligeras para para proporcionar mayor aislamiento.
  - https://katacontainers.io/
  - Nemu: https://github.com/intel/nemu
  - Firecraker (AWS): https://github.com/firecracker-microvm/firecracker

Ciclo de vida de una aplicación en docker

- Docker ha tenido un gran éxito en el desarrollo de software en su uso como plataforma de desarrollo, pruebas y puesta en producción y como alternativa en el empaquetamiento, distribución y despliegue de aplicaciones
- Ciclo de vida tipo en docker
  - Se crea un Dockerfile con las dependencias necesarias: Sistema base, runtime, etc.
  - Se añade la propia aplicación, típicamente desde un repositorio git
  - Se construye la imagen del contenedor con la aplicación y se sube a un registro público o privado
  - Cada vez que se crea una nueva versión de la aplicación, se crea una nueva imagen de docker
  - ¿Qué hacemos con los cambios entre versiones?
  - ¿Cómo hacemos los cambios en producción?
  - Respuestas muy diferentes dependiendo de las características y requisitos de la aplicación
    - Aplicación en un contenedor
    - Aplicación en múltiples contenedores
    - Aplicación totalmente distribuida en múltiples nodos

Ciclo de vida de una aplicación en docker

- Docker ha tenido un gran éxito en el desarrollo de software en su uso como plataforma de desarrollo, pruebas y puesta en producción y como alternativa en el empaquetamiento, distribución y despliegue de aplicaciones
- Ciclo de vida tipo en docker
  - Se crea un Dockerfile con las dependencias necesarias: Sistema base, runtime, etc.
  - Se añade la propia aplicación, típicamente desde un repositorio git
  - Se construye la imagen del contenedor con la aplicación y se sube a un registro público o privado
  - Cada vez que se crea una nueva versión de la aplicación, se crea una nueva imagen de docker
  - ¿Qué hacemos con los cambios entre versiones?
  - ¿Cómo hacemos los cambios en producción?
  - Respuestas muy diferentes dependiendo de las características y requisitos de la aplicación
    - Aplicación en un contenedor
    - Aplicación en múltiples contenedores
    - Aplicación totalmente distribuida en múltiples nodos

Limitaciones de docker

- ¿Cómo se balancea la carga entre múltiples contenedores iguales?
- ¿Cómo se conectan contenedores que se ejecuten en diferentes demonios de docker?
- ¿Se puede hacer una actualización de una aplicación sin interrupción?
- ¿Se puede variar a demanda el número de réplicas de un determinado contenedor?
- ¿Es posible mover la carga entre diferentes nodos?
- Las respuestas a estas preguntas y otras similares tiene que venir de un orquestador de contenedores

¿Qué orquestador de contenedores?

- Hace algunos años había tres alternativas para la orquestación. Todos proyectos de software libre
  - Docker swarm
  - Apache Mesos
  - Kubernetes
  - ...
- Hoy en día se acepta generalmente que el vencedor ha sido kubernetes. ¿Por qué?
  - Proyecto de fundación con gran cantidad de empresas implicadas
  - Los recelos iniciales del control de Google se disiparon con la CNCF
  - La versión inicial estaba prácticamente lista para poner en producción
  - Se ha desarrollado un interesante ecosistema de aplicaciones complementarias

Kubernetes

El proyecto kubernetes

- Gestiona el despliegue de aplicaciones sobre contenedores, automatizando el despliegue, con énfasis en la escalabilidad y controlando todo el ciclo de vida:
  - Despliega aplicaciones rápidamente
  - Escala las aplicaciones al vuelo
  - Integra cambios sin interrupciones
  - Permite limitar los recursos a utilizar
- Es un proyecto centrado en la puesta en PRODUCCIÓN de contenedores y por tanto indicada su gestión para administradores de sistemas y personal de equipos de operaciones.
- Afecta a los desarrolladores, ya que las aplicaciones deben adaptarse para poder desplegarse en kubernetes

El proyecto kubernetes

- Para desplegarse en nube: pública, privada o híbrida. Puede instalarse también en servidores físicos, pero no es su entorno natural
- Extensible: Módulos, plugins, adaptable, ...
- Autoreparable

- Proyecto comenzado por Google en 2014, ahora gestionado por Cloud Native Computing Foundation
- Permite gestionar un clúster de nodos en los que desplegar aplicaciones sobre contenedores
- kubernetes es una palabra del griego antiguo que significa “timonel”
- Habitualmente se escribe k8s
- Es muy importante el ecosistema que se está desarrollando alrededor

El proyecto kubernetes

- Primera versión: 7 junio 2014
- Última versión: 26 agosto 2020 (1.19). Poco a poco ralentizándose la publicación de nuevas versiones
- Desarrollado en Go
- Licencia Apache 2.0
- https://github.com/kubernetes/kubernetes
- Kubernetes está inspirado en Borg, software de Google encargado de la gestión de los servicios de Google
- Casi 3000 personas han contribuido
- Más de 94000 commits en 6 años

El ecosistema

- Una de los aspectos más interesantes de k8s es la cantidad de desarrollo que tiene alrededor, un ecosistema en ebullición con desarrollo de múltiples aplicaciones de las que pocas sobreviven suficiente tiempo

https://landscape.cncf.io/

Arquitectura básica

- ~~minion~~ ~~node~~ worker. Componente de un clúster de k8s
- Puede ser una máquina virtual o física
- Tiene los servicios para ejecutar contenedores
- Es gestionado por los componentes ~~master~~ controller
- Cada nodo tiene:
  - Direcciones: Hostname, IP externa, IP interna
  - Condición: Campo que describe el estado (listo, sin red, sin disco, ...)
  - Capacidad: Describe los recursos disponibles
  - Info: Información general (kernel, sw instalado, versiones)
- Nodo controlador es el que contiene componentes para controlar otros nodos. Se suele denominar master (ejem ...)

Componentes del nodo controlador

- **kube-apiserver** Gestiona la API de k8s
- **etcd** Almacén clave-valor que guarda la configuración del clúster
- **kube-scheduler** Selecciona el nodo donde ejecutar los pods
- **kube-controller-manager** Ejecuta los controladores
- **docker/rkt/containerd/...** Ejecuta los contenedores del master
- **cloud-controller-manager** Ejecuta los controladores que interactúan con el proveedor de nube:
  - nodos
  - enrutamiento
  - balanceadores
  - volúmenes

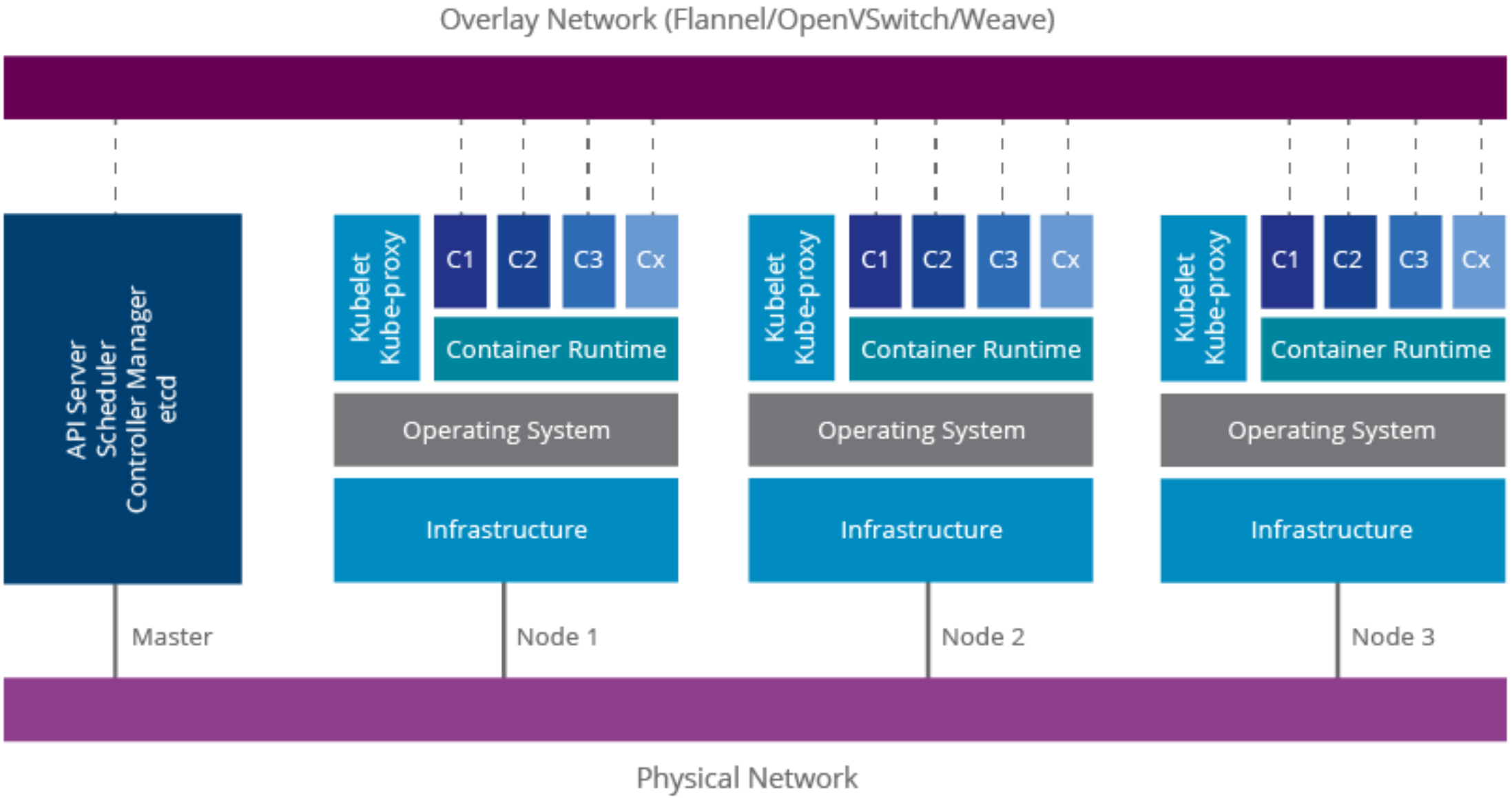
Complementos (addons)

Son pods y servicios que proporcionan funcionalidad al clúster

- **Cluster DNS** Proporciona registros DNS para los servicios de k8s (compatible con el DNS de la organización)
- **Web UI** Interfaz web para el manejo de k8s
- **Container Resource Monitoring** Recoge métricas de forma centralizada. Múltiples opciones: prometheus, sysdig
- **Cluster-level Logging** Almacena los logs de los contenedores

Componentes de los worker

- **kubelet** Vigila los pods asignados a su nodo
- **kube-proxy** Permite la conexión a través de la red
- **docker/rkt/containerd/...** Ejecuta los contenedores
- **supervisord** Monitoriza y controla kubelet y docker
- **fluentd** Proporciona logs de contenedores a cluster-level logging



Kubernetes architecture

Alternativas de instalación simple

- Opciones de instalación y uso: https://kubernetes.io/docs/setup/
- ¿No se puede instalar desde paquetes de la distribución?

Proyectos propios de k8s	Ecosistema
Minikube	Charmed Distribution of kubernetes (CDK on LXD)
Kind: Kubernetes IN Docker	Docker Desktop
	Minishift
	MicroK8s
	k3s

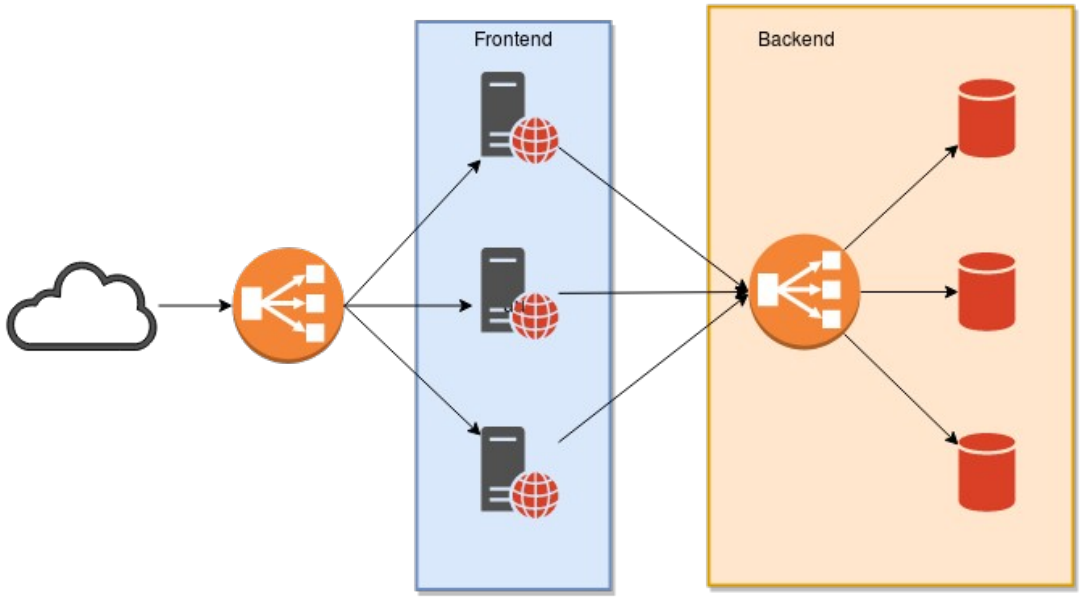
kubectl

- kubectl es la herramienta principal para gestionar los clusters de k8s
- Desarrollada en go
- Paquete debian: https://packages.debian.org/search?keywords=kubernetes-client
- Instalación directa de binarios:
  https://kubernetes.io/docs/tasks/tools/install-kubectl/#install-kubectl-binary-with-curl-on-linux

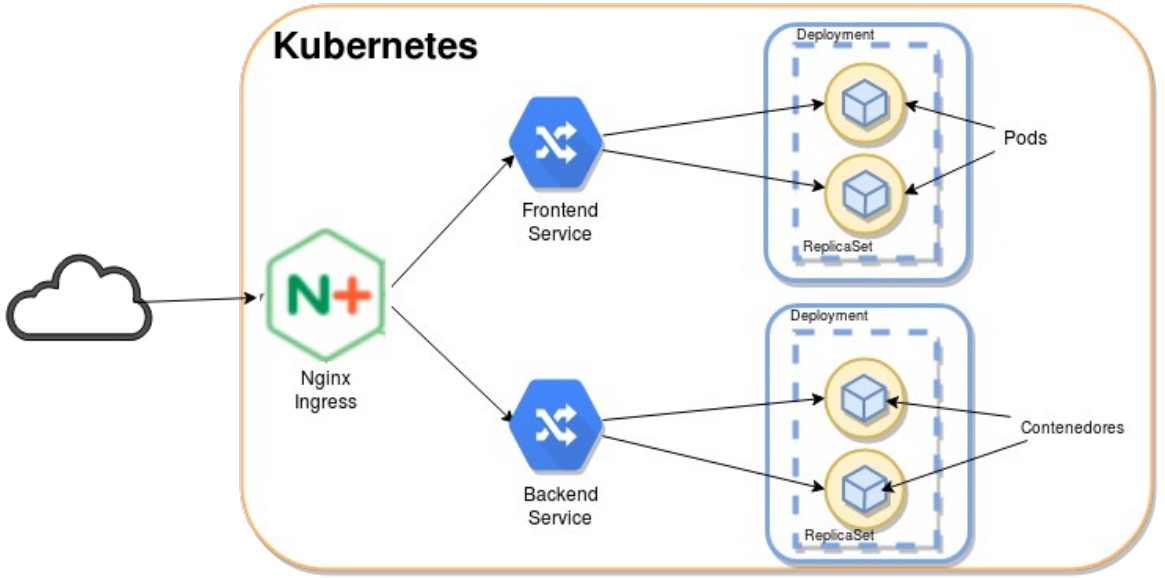
Configuración de kubectl

- kubectl puede configurarse para gestionar distintos clústers mediante el uso de contextos
- kubectl config
- La configuración se guarda en ~/.kube/config
- Uso de kubectl
- Con “kubectl proxy” puede interactuarse con la API directamente utilizando las credenciales del usuario

Despliegue de aplicaciones



- Máquinas virtuales
- Balanceadores de carga



- Pods
- Deployments
- Services
- ReplicaSets
- Ingress

Estrategias de despliegue

DEPLOYMENT STRATEGIES

When it comes to production, a ramped or blue/green deployment is usually a good fit, but proper testing of the new platform is necessary.

Blue/green and shadow strategies have more impact on the budget as it requires double resource capacity. If the application lacks in tests or if there is little confidence about the impact/stability of the software, then a canary, a/b testing or shadow release can be used.

If your business requires testing of a new feature amongst a specific pool of users that can be filtered depending on some parameters like geolocation, language, operating system or browser features, then you may want to use the a/b testing technique.

Container Solutions

Strategy	ZERO DOWNTIME	REAL TRAFFIC TESTING	TARGETED USERS	CLOUD COST	ROLLBACK DURATION	NEGATIVE IMPACT ON USER	COMPLEXITY OF SETUP
RECREATE version A is terminated then version B is rolled out	✗	✗	✗	■□□	■■■	■■■	□□□
RAMPED version B is slowly rolled out and replacing version A	✓	✗	✗	■□□	■■■	■□□	■□□
BLUE/GREEN version B is released alongside version A, then the traffic is switched to version B	✓	✗	✗	■■■	□□□	■■□	■■□
CANARY version B is released to a subset of users, then proceed to a full rollout	✓	✓	✗	■□□	■□□	■□□	■■□
A/B TESTING version B is released to a subset of users under specific condition	✓	✓	✓	■□□	■□□	■□□	■■■
SHADOW version B receives real world traffic alongside version A and doesn't impact the response	✓	✓	✗	■■■	□□□	□□□	■■■

https://github.com/ContainerSolutions/k8s-deployment-strategies