CESTE
Escuela Internacional de Negocios

DOCKER + KUBERNETES

Profesor: Ginés Carrascal de las Heras

DOCKER + KUBERNETES

Contenedores

Fundamentos: Docker
Orquestación: Kubernetes

DOCKER + KUBERNETES

Conceptos básicos de contenedores

DOCKER

Conceptos fundamentales de Docker.

Ejecución de contenedores

Mapeo de puertos

Almacenamiento:

- Volúmenes

Creación de imágenes:

- Docker files, capas, variables

Gestión de imágenes:

- Container Registry

Docker composer

Practica Docker con un modelo IA

Despliegue en cloud

DOCKER + KUBERNETES

Orquestación de Contenedores

KUBERNETES

Conceptos básicos de Kubernetes

Kubernetes en la nube

Pods

Despliegues declarativos en Kubernetes.

Servicios

Configuración: ConfigMaps y Secretos

Ingress

Healthchecks

Conceptos avanzados

- Helm
- Prometheus y Grafana
- Open Shift

UNIDAD TEMATICA

CONCEPTOS BÁSICOS DE CONTENEDORES: DOCKER

DOCKER

Contenedores ligeros y portables para las aplicaciones

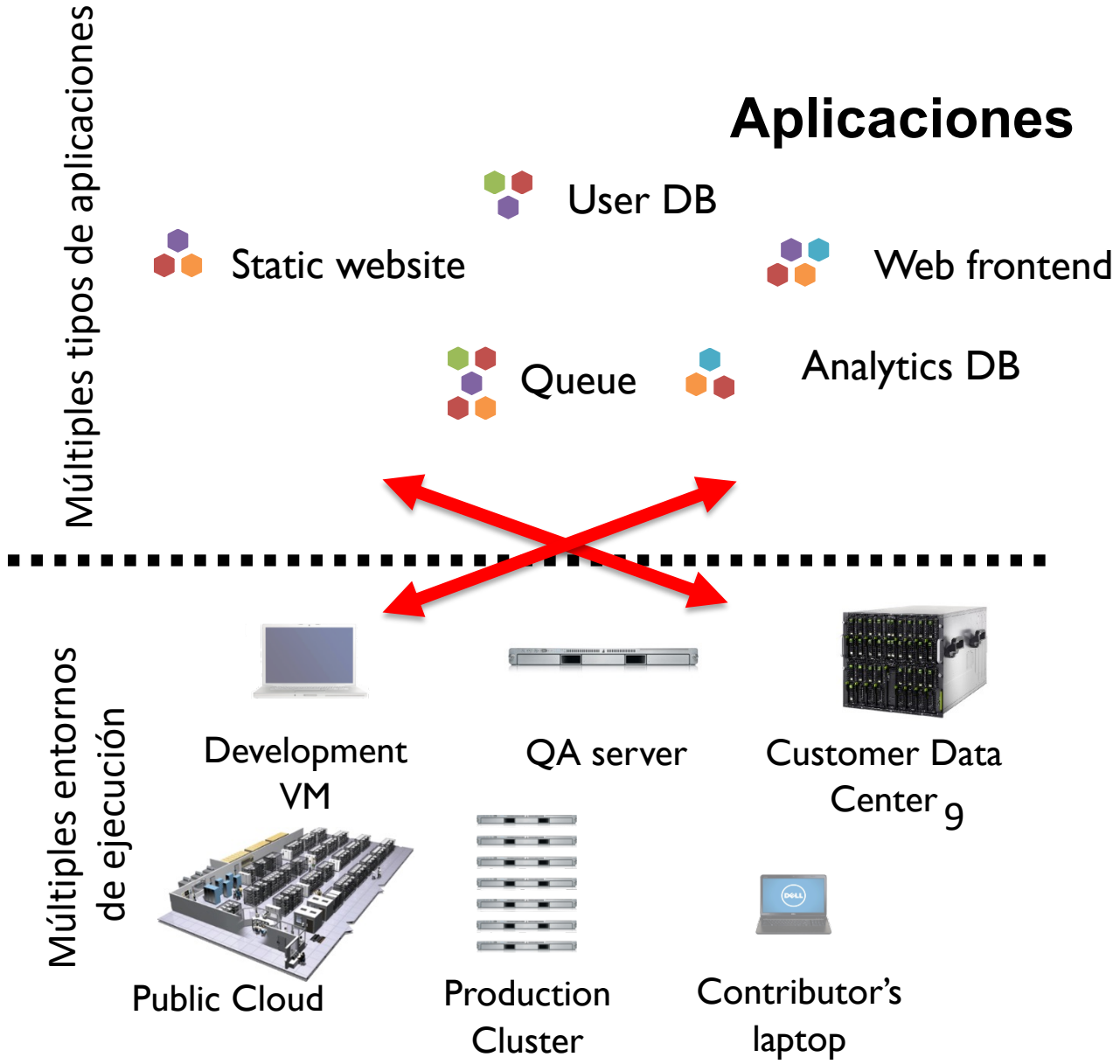
CAPITULO

Conceptos fundamentales de Docker

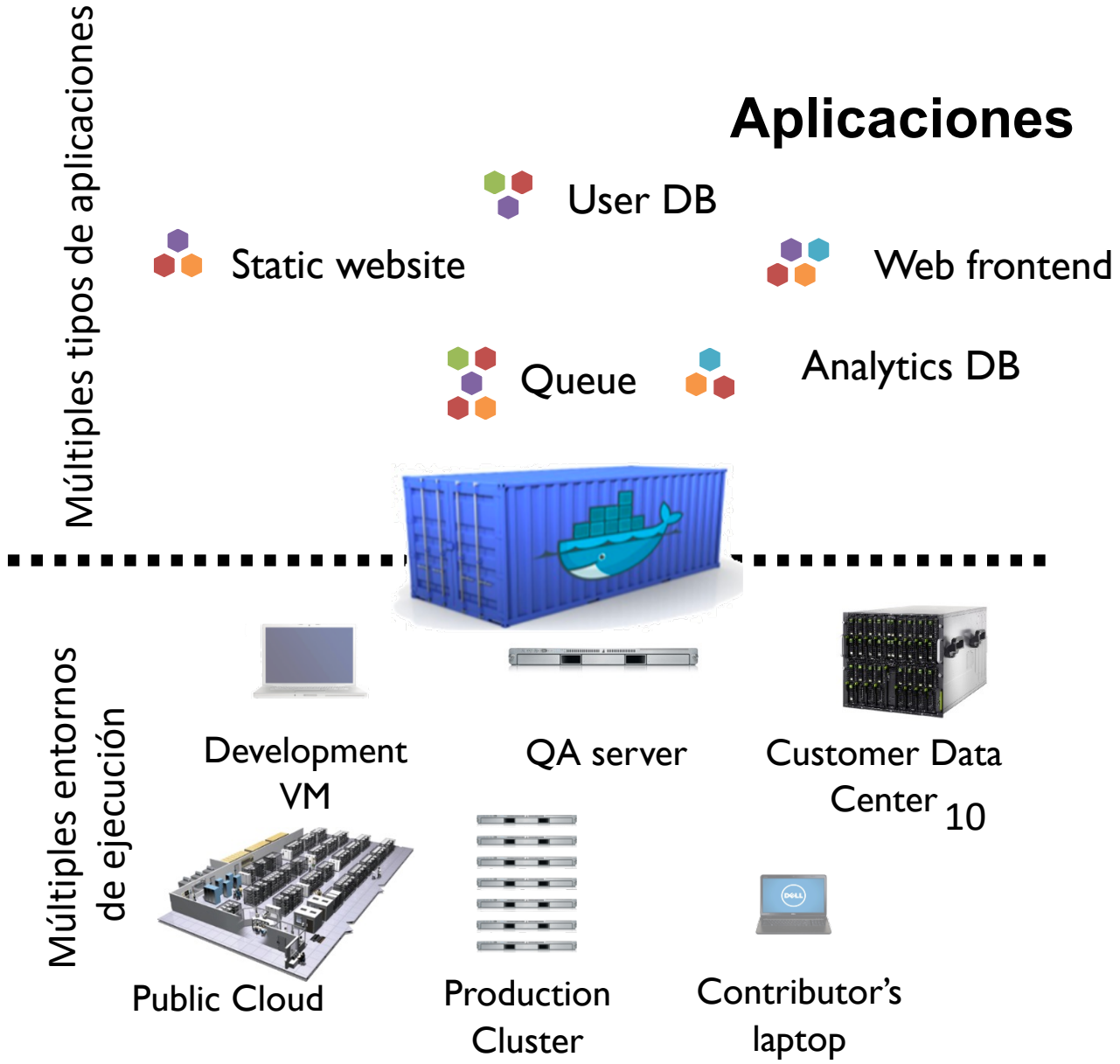
CONCEPTOS FUNDAMENTALES DE DOCKER

CONTENEDOR

CONCEPTOS FUNDAMENTALES DE DOCKER

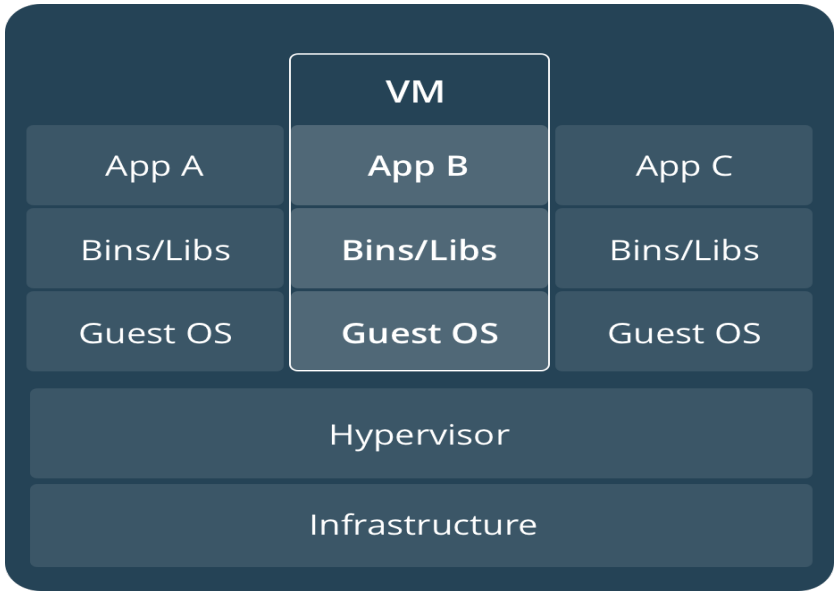


CONCEPTOS FUNDAMENTALES DE DOCKER

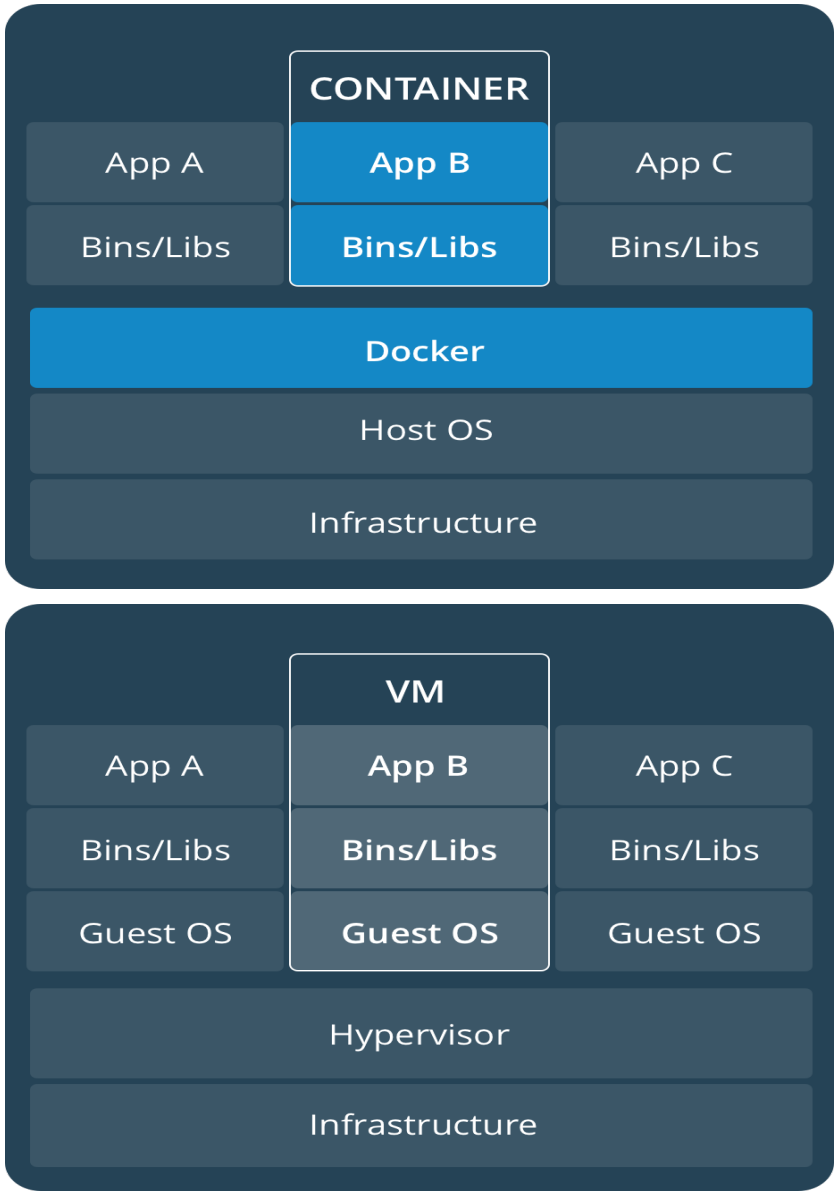


CONCEPTOS FUNDAMENTALES DE DOCKER

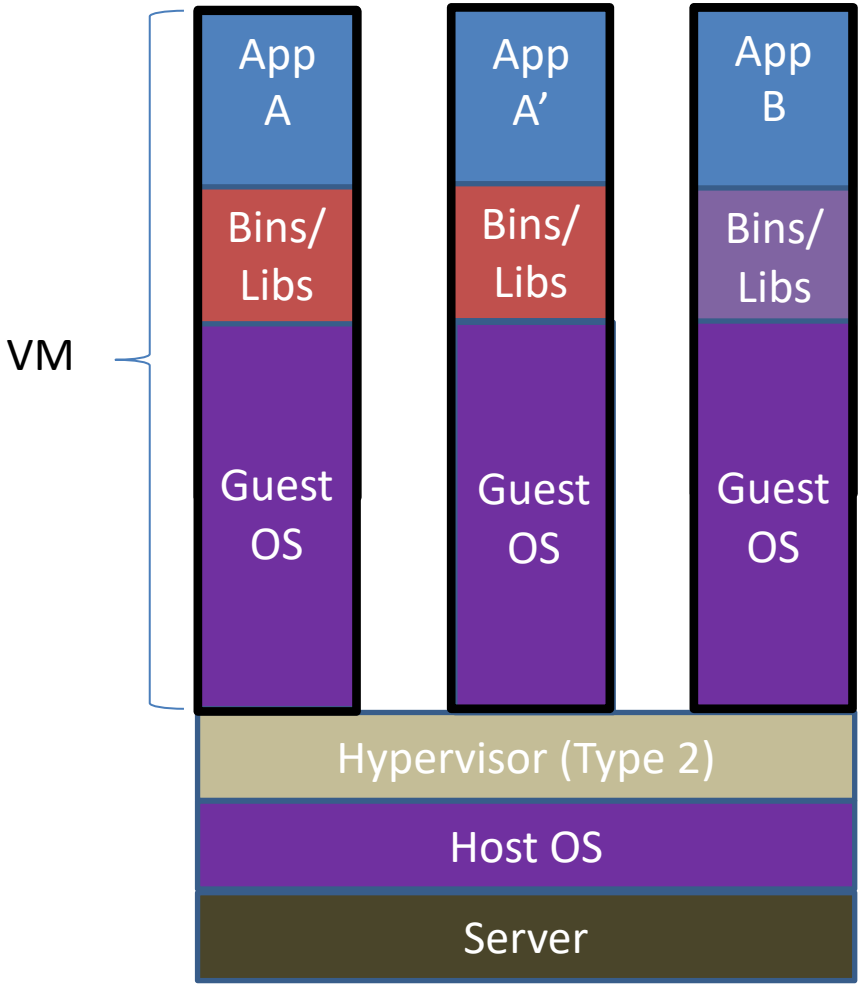
Contenedor vs Máquina virtual



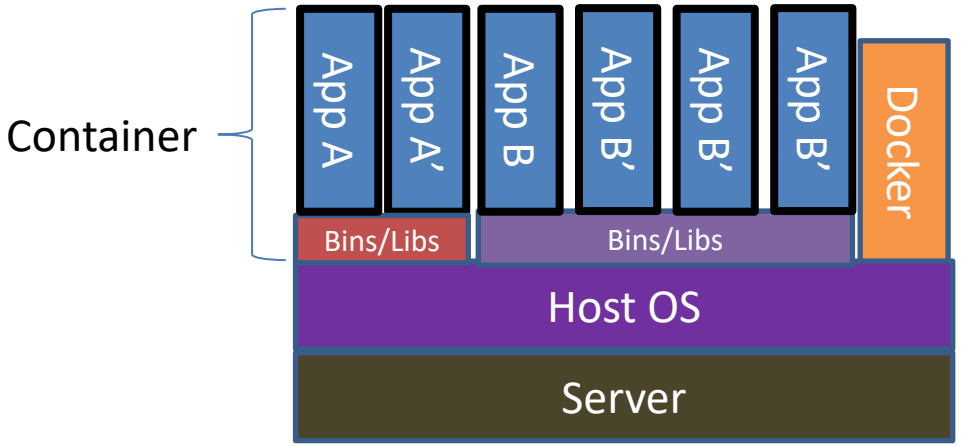
CONCEPTOS FUNDAMENTALES DE DOCKER



CONCEPTOS FUNDAMENTALES DE DOCKER

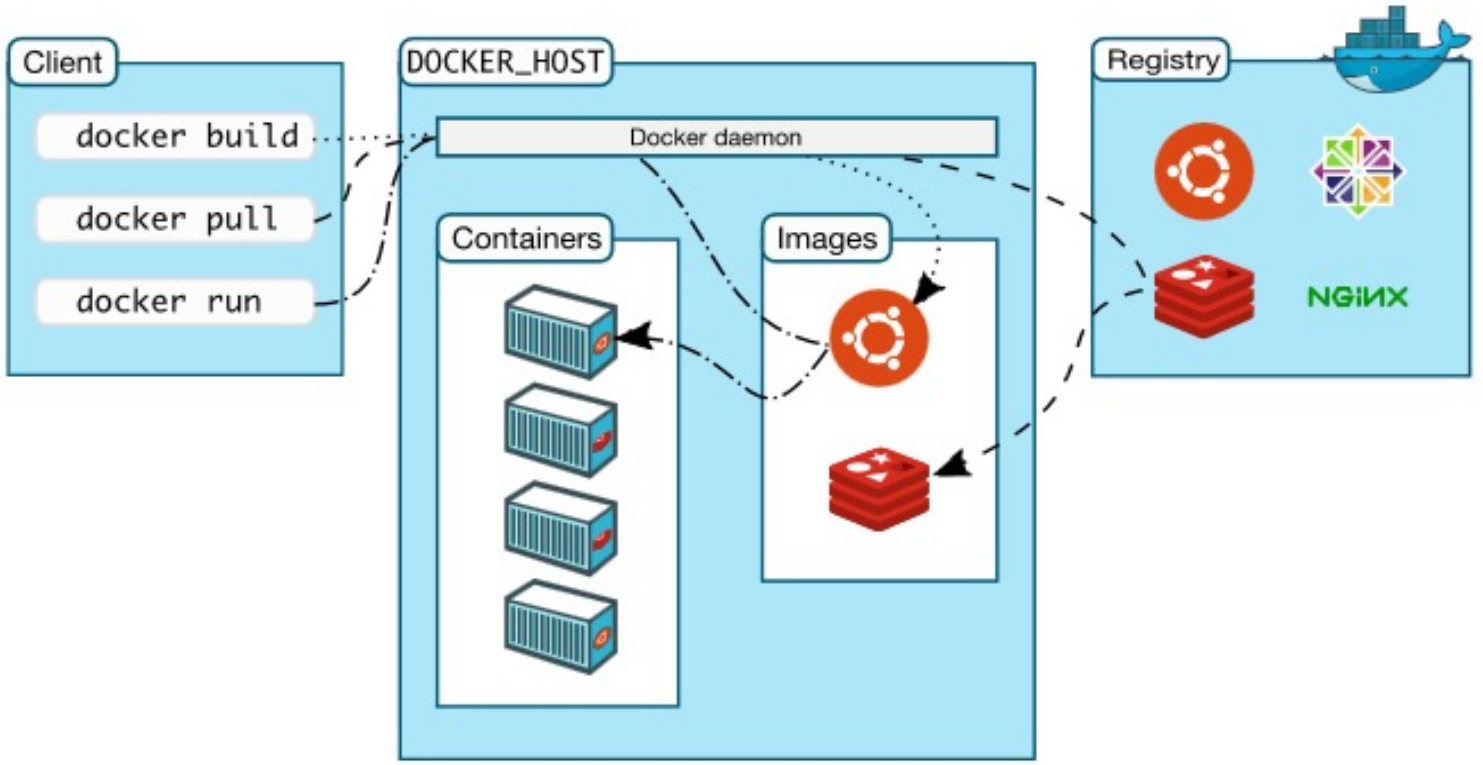


CONCEPTOS FUNDAMENTALES DE DOCKER



CONCEPTOS FUNDAMENTALES DE DOCKER

ARQUITECTURA



CONCEPTOS FUNDAMENTALES DE DOCKER

CASOS DE USO

CI/CD

DEVOPS

BIG DATA

OPTIMIZACION DE INFRAESTRUCTURA

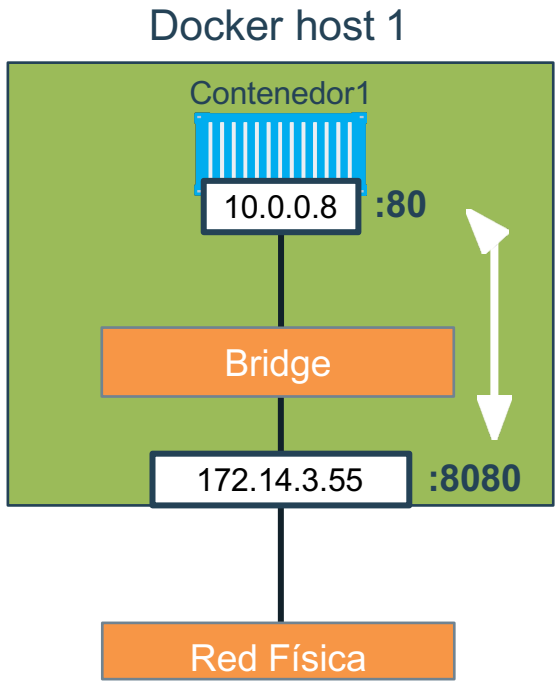
CONCEPTOS FUNDAMENTALES DE DOCKER

Versiones

Docker CE :

Docker EE :

MAPEO DE PUERTOS



Host port

Container port

```
$ docker container run -p 8080:80 ...
```

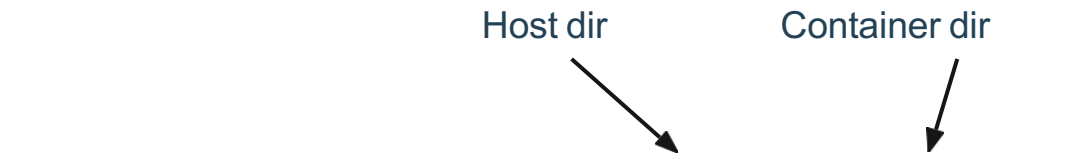
ALMACENAMIENTO

Volúmenes

Montar un directorio del host en un punto concreto del contenedor

Compartir datos entre contenedores

Datos persistentes tras apagar el contenedor




```
$ docker container run -v $(pwd):/work ...
```

IMÁGENES

DOCKERFILE

Instrucciones para crar una imagen

Muy parecido a los commandos “nativos”

IMÁGENES

DOCKERFILE

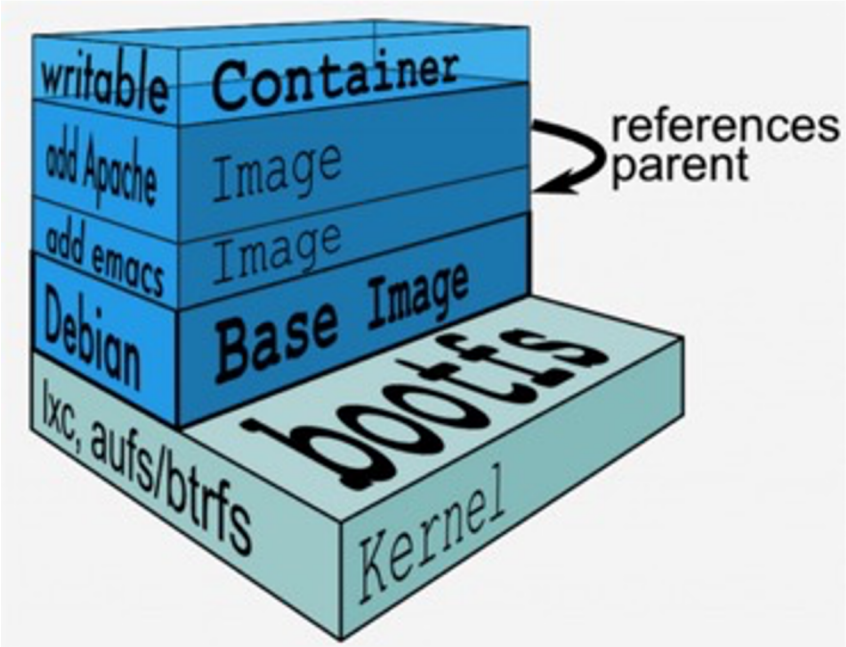
Dockerfile

```
# Create image based on the official Node 6 image from dockerhub
FROM node:latest

# Create a directory where our app will be placed
RUN mkdir -p /usr/src/app

# Change directory so that our commands run inside this new directory
WORKDIR /usr/src/app

# Copy dependency definitions
COPY package.json /usr/src/app

# Install dependecies
RUN npm install

# Get all the code needed to run the app
COPY . /usr/src/app

# Expose the port the app runs in
EXPOSE 4200

# Serve the app
CMD ["npm", "start"]
```

IMÁGENES

SISTEMA DE ARCHIVOS EN PILA

Importante optimizer el DOCKERFILE



IMÁGENES

COMANDOS: CONTENEDORES

```
$ docker container run –d –p 5000:5000 –-name
node node:latest

$ docker container ps


$ docker container stop node(or <container id>)

$ docker container rm node (or <container id>)
```

IMÁGENES

COMANDOS: IMÁGENES

```
$ docker image pull node:latest

$ docker image ls

$ docker image rmi (or <image id>)

$ docker build –t node:2.0 .

$ docker image push node:2.0
```

CONTAINER REGISTRY

¿QUÉ ES UN REGISTRO?

Un lugar centralizado para almacenar y distribuir imágenes

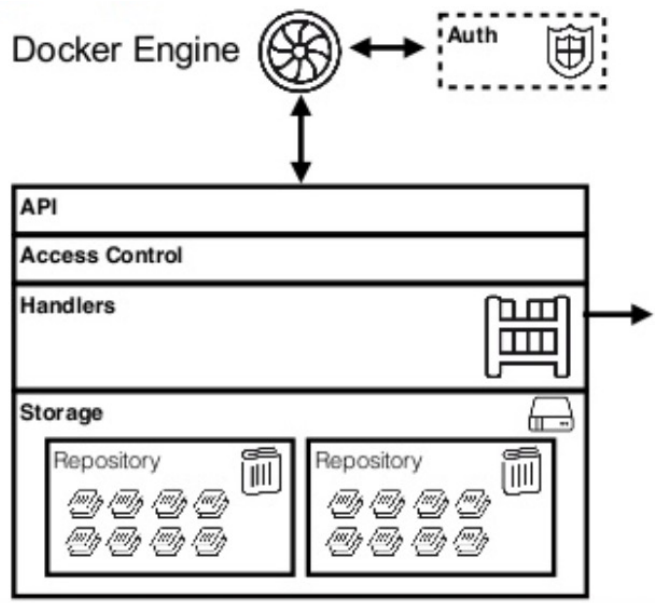
Almacena las capas y las instrucciones de cómo componer ls imágenes

Implementa un API común a todos los clients de Docker

CONTAINER REGISTRY

DOCKER REGISTRI API V2

Todo el contenido está empaquetado en Repositorios

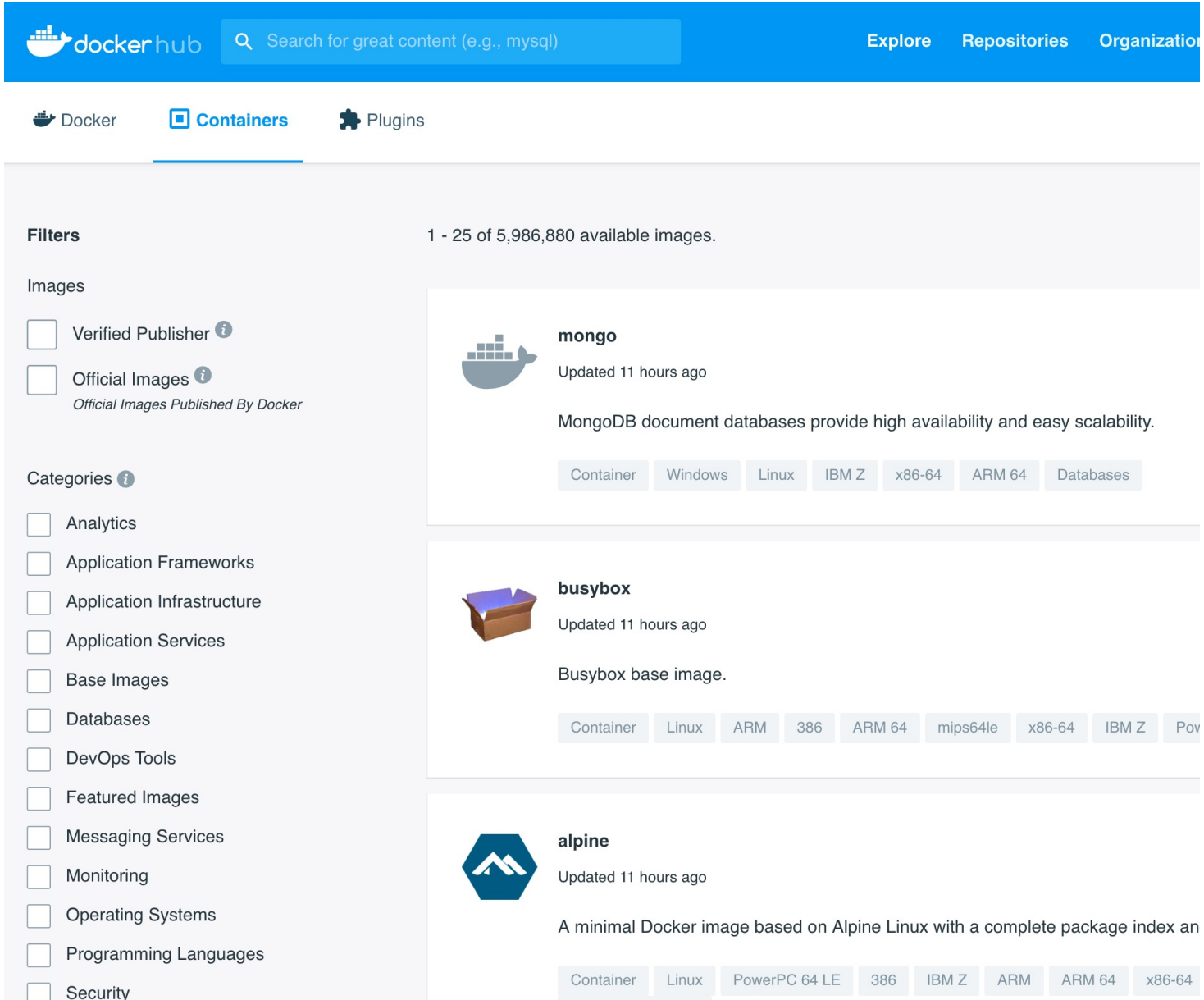


CONTAINER REGISTRY

DOCKER-HUB

CONTAINER REGISTRY

CONTAINER REGISTRY



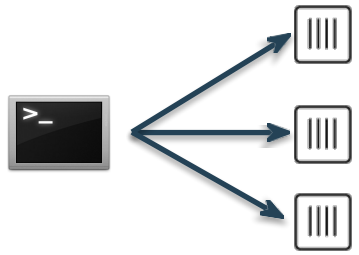
DOCKER COMPOSE

APLICACIONES MULTICONTENEDOR

Construir y ejecutar cada uno de los contenedores

Manualmente conectar los contenedores

Tener cuidado con las dependencias y el orden de arranque



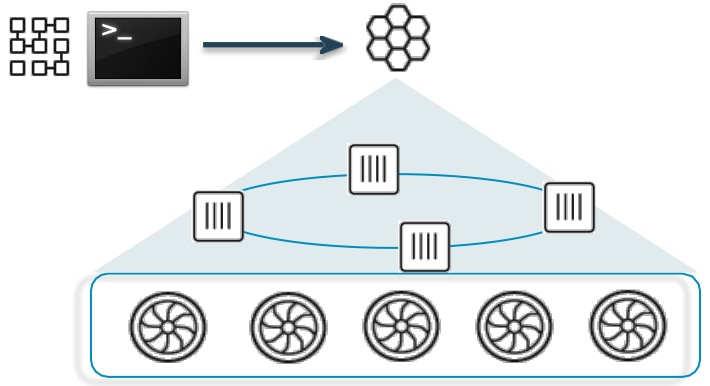
DOCKER COMPOSE

APLICACIONES MULTICONTENEDOR

Definir la aplicacion multicontenedor en un fichero: **compose.yaml**

Un solo commando para desplegar la aplicación

Gestiona todas las dependencias

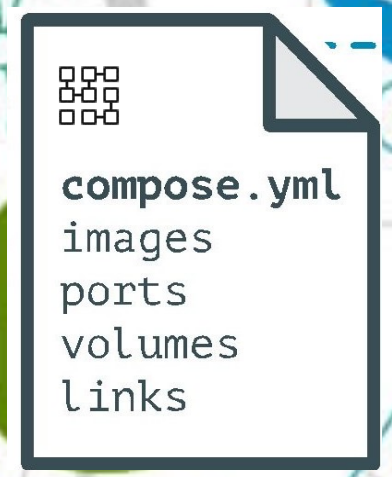


DOCKER COMPOSE

DOCKER COMPOSE

COMPOSE.YML

```
version: '2' # specify docker-compose version

# Define the services/containers to be run
services:
angular: # name of the first service
build: client # specify the directory of the Dockerfile
ports:
- "4200:4200" # specify port forewarding

express: #name of the second service
build: api # specify the directory of the Dockerfile
ports:
- "3977:3977" #specify ports forewarding

database: # name of the third service
image: mongo # specify image to build container from
ports:
- "27017:27017" # specify port forewarding
```



DOCKER CON UN MODELO DE AI

CASOS DE USO

CI/CD

DEVOPS

BIG DATA

OPTIMIZACION DE INFRAESTRUCTURA

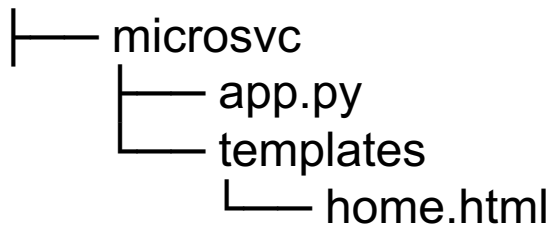
DOCKER CON UN MODELO DE AI

CASOS DE USO

Iris Versicolor **Iris Setosa** **Iris Virginica**

DOCKER CON UN MODELO DE AI

ESTRUCTURA DE LA APLICACIÓN



sudo apt install -y python3-pip

DOCKER CON UN MODELO DE AI

CREAR UN DOCKERFILE

```
# We always start from an existing container
FROM python:3.9.2

# Copy dependency lists into container
COPY requirements.txt .

# Install dependencies
RUN pip install -r requirements.txt

# Copy our code into the container
COPY app.py .
COPY iris_trained_model.pkl .

# Copy the HTML page templates directory
COPY templates templates

# Our code runs on port 5000, so allow access
EXPOSE 5000

# Set the FLASK_APP environment variable
ENV FLASK_APP app.py

# This is the command that is executed when the container starts
# Note we've added --host=0.0.0.0 as by default only local users would
# be able to access the application.
CMD [ "flask", "run", "--host=0.0.0.0" ]
```

DOCKER CON UN MODELO DE AI

CONSTRUIR LA IMAGEN

$ docker build -t IMAGE_NAME:TAG .

ó

$ docker build -t USERNAME/IMAGE_NAME:TAG .

Y probarla:

$ docker run -p 5000:5000 USERNAEM/IMAGE_NAME:TAG

DOCKER CON UN MODELO DE AI

$ docker build -t IMAGE_NAME:TAG .

ó

$ docker build -t USERNAME/IMAGE_NAME:TAG .

Y probarla:

$ docker run -p 5000:5000 USERNAEM/IMAGE_NAME:TAG

UNIDAD TEMATICA

ORQUESTCIÓN DE CONTENEDORES: KUBERNETES

KUBERNETES

Orquestador de contenedores entre múltiples servidores

Orquestación de Contenedores

KUBERNETES

Orquestación de conjuntos de contenedores desplegados en diferentes servidores

- Colocación automática, red, despliegue, escalado, roll-out/-back, A/B testing

Declarativo – not procedimental

- Declarar el estado deseado, reconciliar con él
- Se auto repara

Portabilidad de las cargas de trabajo

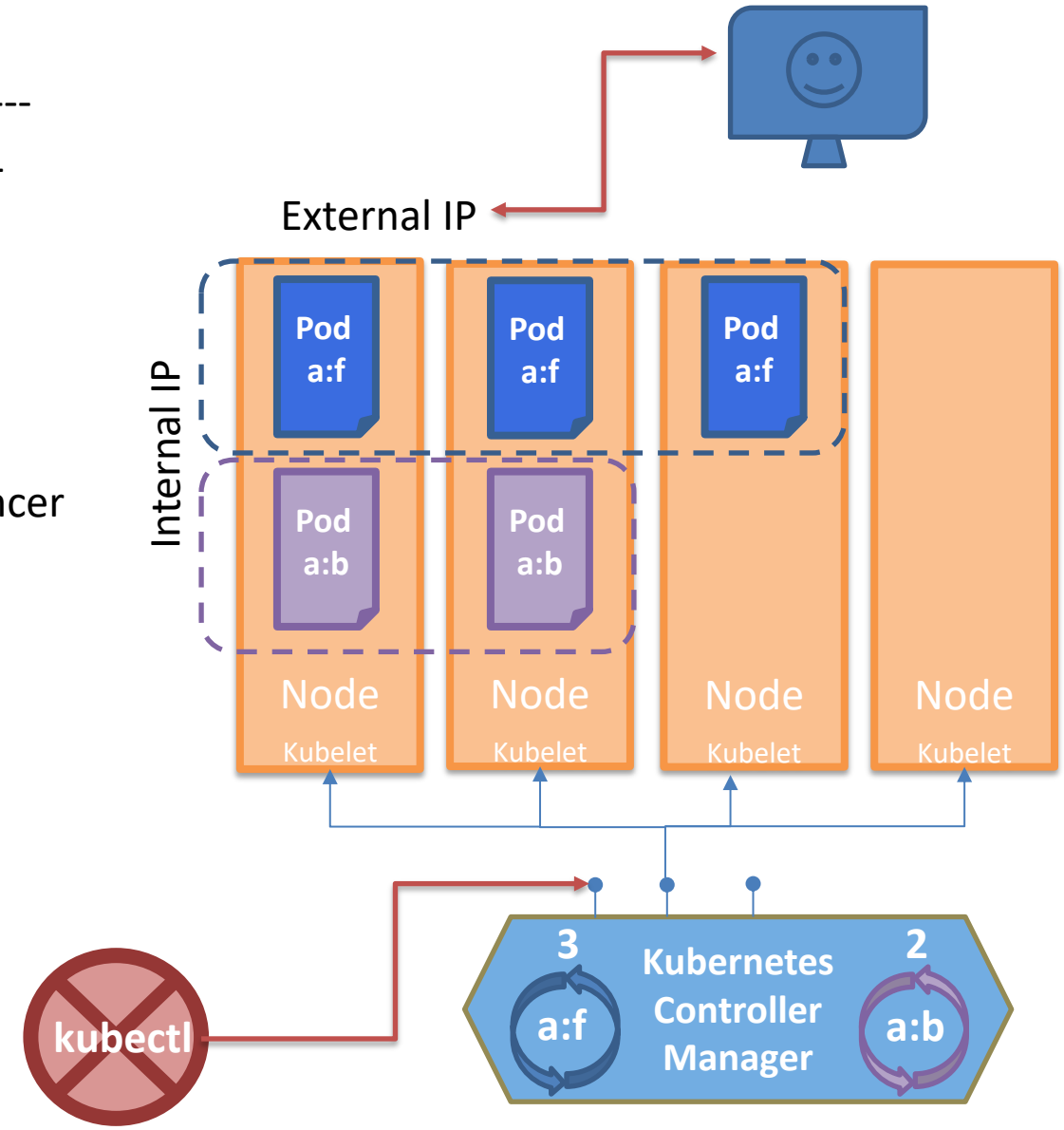
- Abstracción de las peculiaridades de los proveedores de cloud
- Multiples entoros de ejecución para los contenedores

CAPITULO

Conceptos básicos de Kubernetes

```
kind: Deployment
selector: {a:f}
replicas: 3
template:
    ---kind: Pod----
       image: f:v1
       labels:
                  a:f
- - - -
kind: Service
selector: {a:f}
type: LoadBalancer
```

KUBERNETES



CAPITULO

Pods

KUBERNETES

POD

Un Pod es una colección de contenedores que se ejecuta en un servidor

Representa una unidad lógica para agrupar aplicaciones

Los VOLUMENES pueden compartirse entre aplicaciones del mismo Pod

Los Pods son efímeros, y no pueden moverse a otro servidor

KUBERNETES

SERVICIO

Un servicio es una abstracción que define un conjunto lógico de Pods y la política de acceso a los mismos

El conjunto de Pods involucrados en un servicio se determina mediante un selector de etiquetas

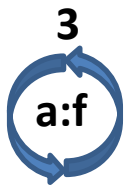
Un servicio reserve un pueto TCP o UDP

Permite a diferentes aplicaciones Kubernetes conectarse sin que necesiten conocer las direcciones IP

CAPITULO

Replication Controllers y Replica Sets

KUBERNETES



CONTROLADOR DE REPLICA

Un Controlador de replicas se asegura de que el número establecido de “copias” de un pod está en ejecución en cada momento

Utilizan Pod Templates para crear los Pods

Utilizan las etiquetas para vigilar y mantener el número correcto de réplicas

CAPITULO

Despliegues declarativos en Kubernetes

KUBERNETES

DEPLOYMENT

Un Controlador de Deployment proporciona actulizaciones declarativas para Pods y ReplicaSets

NO SON COMANDOS:
El controladod comprueba el estado actual y lo cambia por el estado deseado de forma controlada

Las modificaciones en los Pods y Replicas siempre deberían hacerse mediante Deployments

CAPITULO

Creación y uso de Namespaces

KUBERNETES

ESPACIOS DE NOMBRES

Los Namespaces en Kubernetes ayudan a que diferentes proyectos, equipos o clients compartan un mismo cluster

Proporciona un mecanismo para asignar diferentes políticas de autorización a secciones del cluster

CAPITULO

Configuración mediante ConfigMaps

CONFIG MAPS

Un ConfigMap es un almacén de propiedades de tipo clave-valor.

Los Pods pueden acceder a la información como variables de entorno, argumentos de línea de commandos, o ficheros de configuración en un volumen

No proporcionan encriptación

Su objetivo es separar la configuración del Código de las aplicaciones

CAPITULO

Secretos en Kubernetes

KUBERNETES

SECRETS

Los secretos permiten almacenar información sensible como passwords, tokens OAuth o claves ssh

Los Pods pueden acceder a la información como variables de entorno, argumentos de línea de commandos, o ficheros de configuración en un volumen

Proporcionan encriptación, pero para tener seguridad complete necesitan encriptación de los dispositivos

CAPITULO

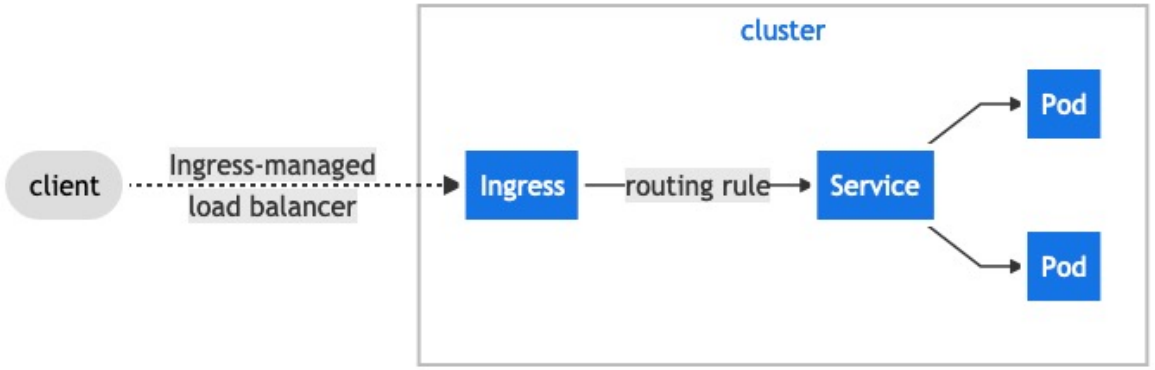
Ingress

INGRESS

Un Ingress gestiona el acceso externo a los servicios de un cluster

Típicamente utiliza protocol HTTP

Ingres puede proporcionar balanceo de carga, gestion de SSL y “virtual hosting” en base a nombres



CAPITULO

Dependencias

INIT CONTAINERS

El propósito es separar los Pods de su lógica de inicialización

Un **Init container** es un contenedor que se inicia y ejecuta *antes* del resto de los contenedores, en el mismo pod

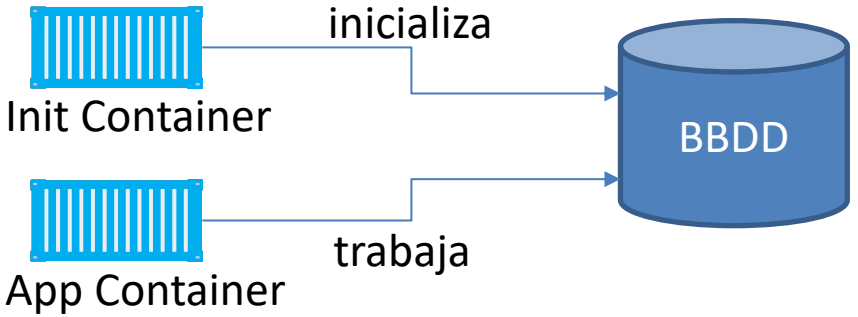
Si el Init container falla, el pod complete es reinicializado

KUBERNETES: INIT CONTAINERS

INICIALIZAR UNA DDBB

El init container se conecta al motor de base de datos y crea las tablas y datos iniciales

La aplicación se inicia y trabaja contra una base de datos adecuada

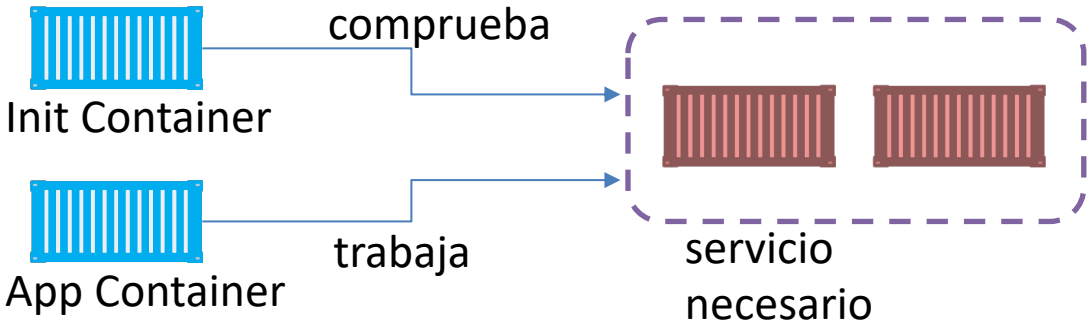


KUBERNETES: INIT CONTAINERS

Esperar a otro servicio

El init container comprueba periodicamente la existencia de otro servicio hasta que lo encuentra

La aplicación se inicia y trabaja contra una base de datos adecuada



CAPITULO

Healthchecks: Liveness, Readiness y Startup Probes

HEALTHCHECKS

Probes son procesos que comprueban el estado de los contenedores para permitir tomar decisions:

Kubelet utiliza **liveness probes** paravdecidir si debe reiniciar un contenedor

Kubelet utiliza **readiness probes** paravdecidir si un contenedor está listo para recibir peticiones

Kubelet utiliza **startup probes** para saber si un contenedor ya ha arrancado y se pueden utilizer en el las otras *probes*

CAPITULO

Helm

KUBERNETES

HELM

Es conocido como el gestor de paquetes de Kubernetes

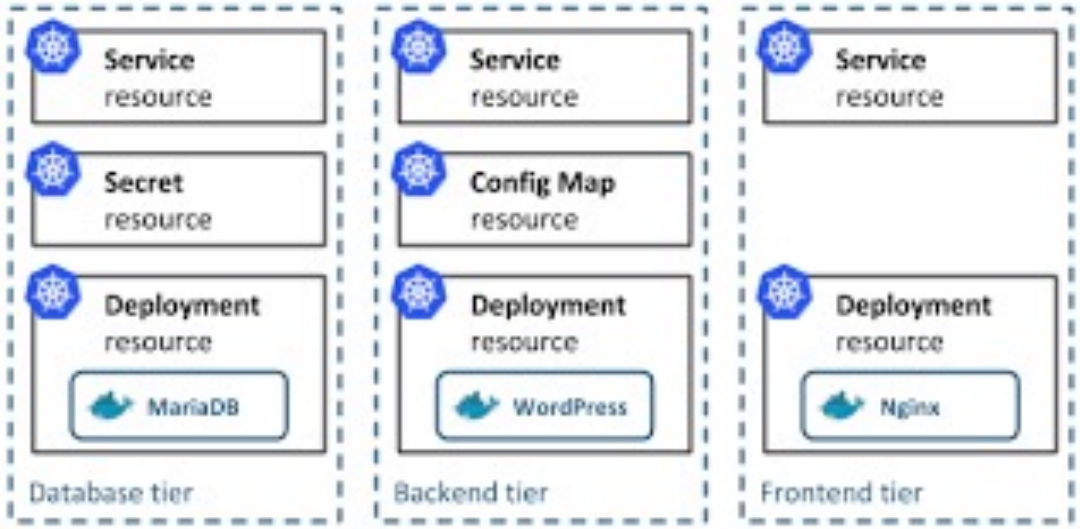
Utiliza **Helm Charts** para agrupar conjuntos de especificaciones (YAML) de kubernetes

Un Chart puede depender de otros, y Helm instala todo el arbol de dependencias autométicamente con un solo comando

KUBERNETES

HELM

Es conocido como el gestor de paquetes de Kubernetes



CAPITULO

Prometheus y Grafana

KUBERNETES

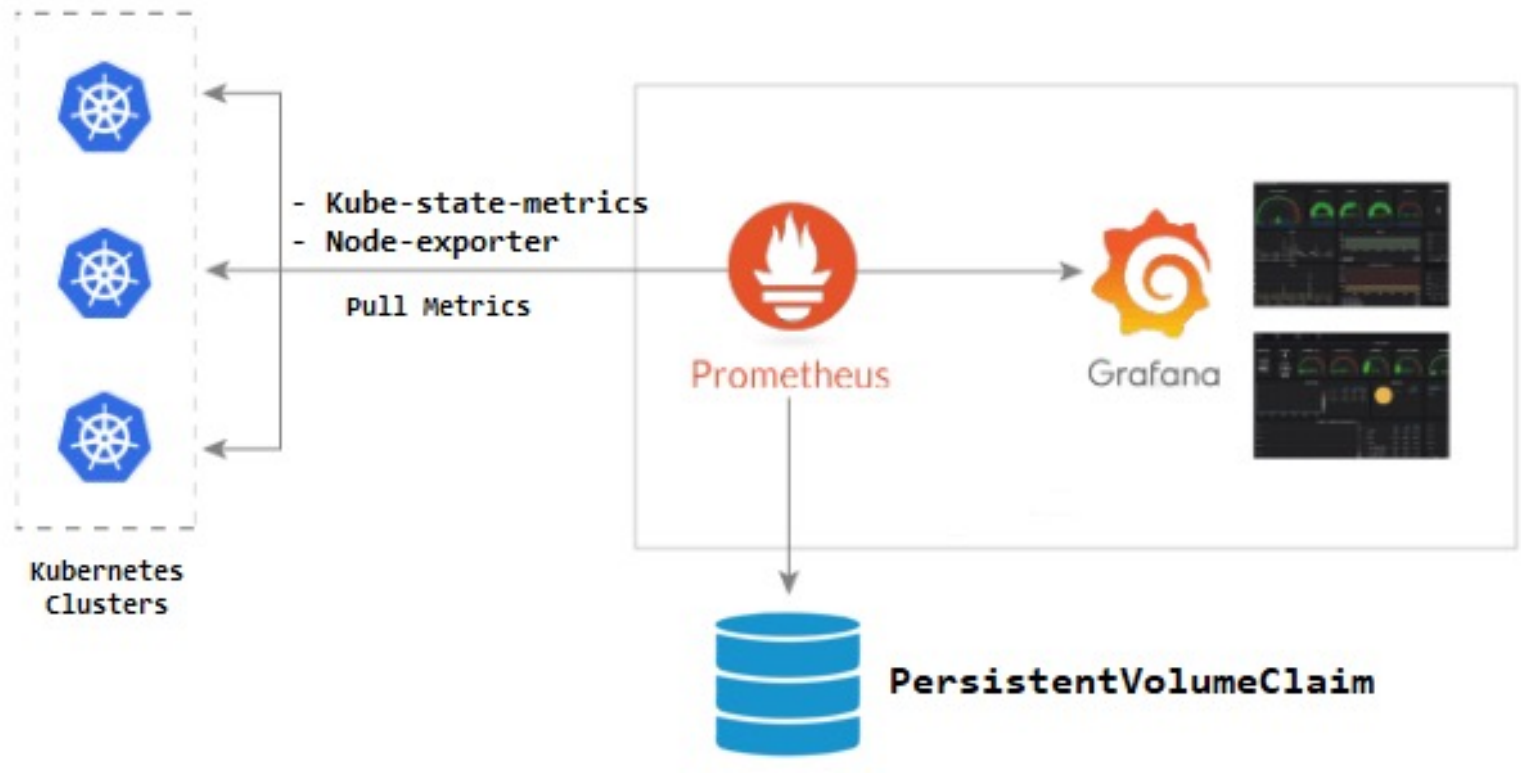
Monitorización

Prometheus hace peticiones a los diferentes elementos y guarda métricas de funcionamiento

Grafana crea dashboards visuals con la información recogida

KUBERNETES

Monitorización



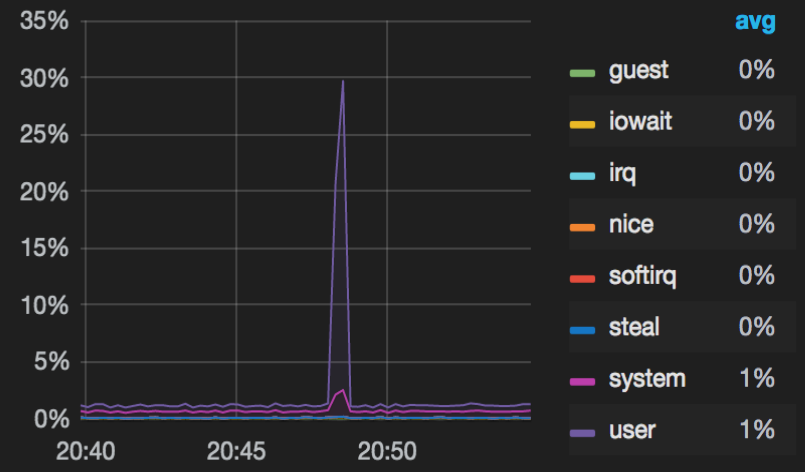
Node

Zoom Out | Last 15 minutes | Refresh every 30s

Environment: prod | Server Type: All | Node: rz-docker-registry

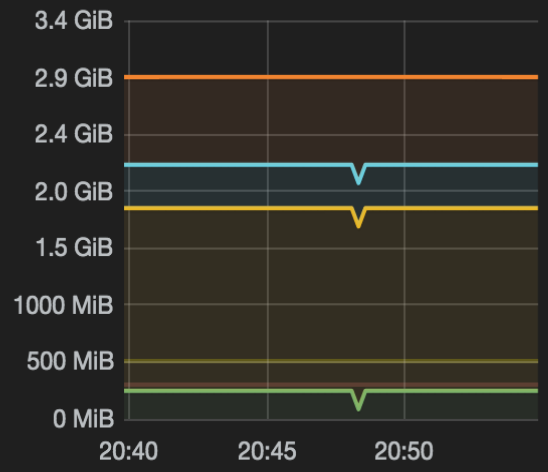
RZ-DOCKER-REGISTRY

CPU



	avg
guest	0%
iowait	0%
irq	0%
nice	0%
softirq	0%
steal	0%
system	1%
user	1%

Memory



- Free — Current: 246 MiB
- Cached — Current: 1.568 GiB
- Buffered — Current: 381 MiB
- Used — Current: 770 MiB

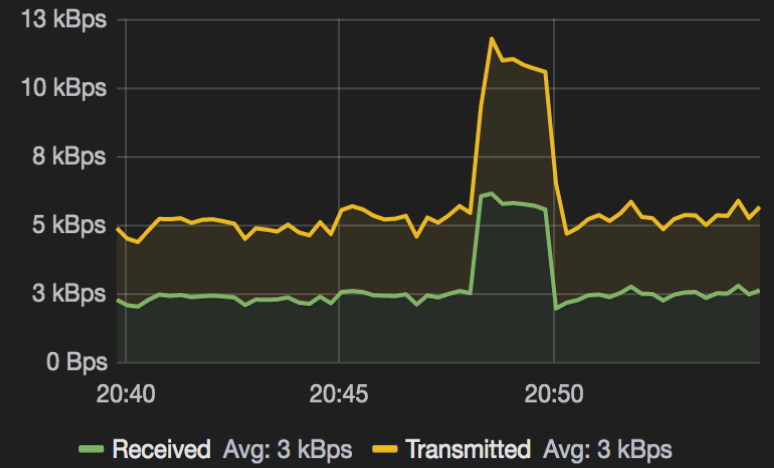
Remaining Disk Space

45%

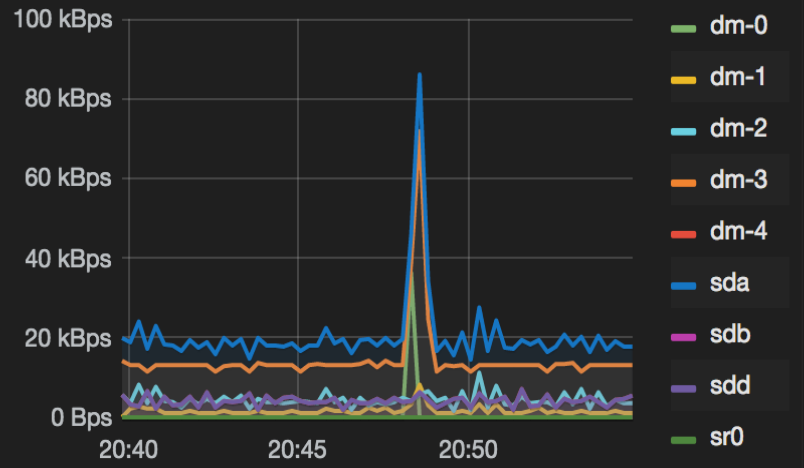
Swap

152 MiB

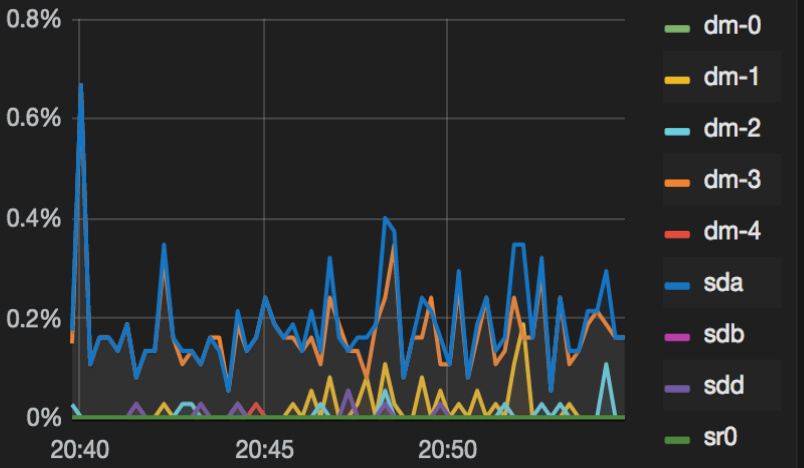
Network I/O

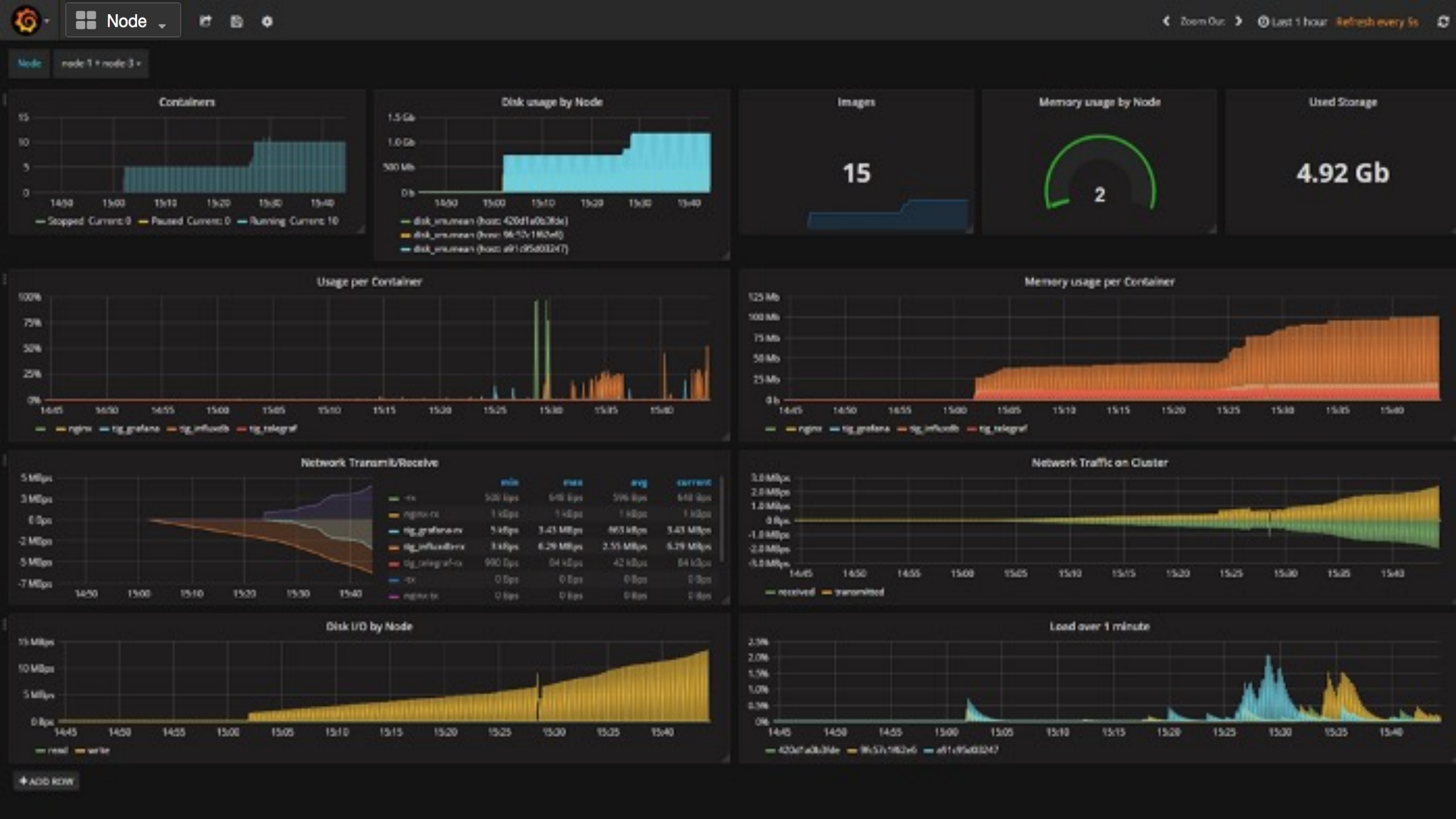


Disk Throughput



Disk Utilization




Node
Zoom Out
Last 1 hour
Refresh every 5s
Node
node 1 + node 3
Containers
Stopped Current: 0
Paused Current: 0
Running Current: 10
Disk usage by Node
Images
15
Memory usage by Node
2
Used Storage
4.92 Gb
Usage per Container
Memory usage per Container
Network Transmit/Receive
Network Traffic on Cluster
received
transmitted
Disk I/O by Node
read
write
Load over 1 minute
ADD ROW

App.Metrics.Sandbox - InfluxDB - OAuth2 Generic Web

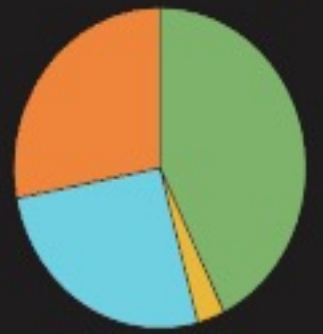
Zoom Out | Last 5 minutes | Refresh every 5s

client_id: client-1 | route: All | datasource: AppMetricsSandbox | summarize: 5s | application: App.Metrics.Sandbox | environment: stage

Throughput

Error Rate

Errors



percentage
client-1 401 51.46%
client-1 403 0.00%
client-1 404 25.73%
client-1 500 25.73%

Post Request Size

Put Request Size

CAPITULO

Open Shift

KUBERNETES

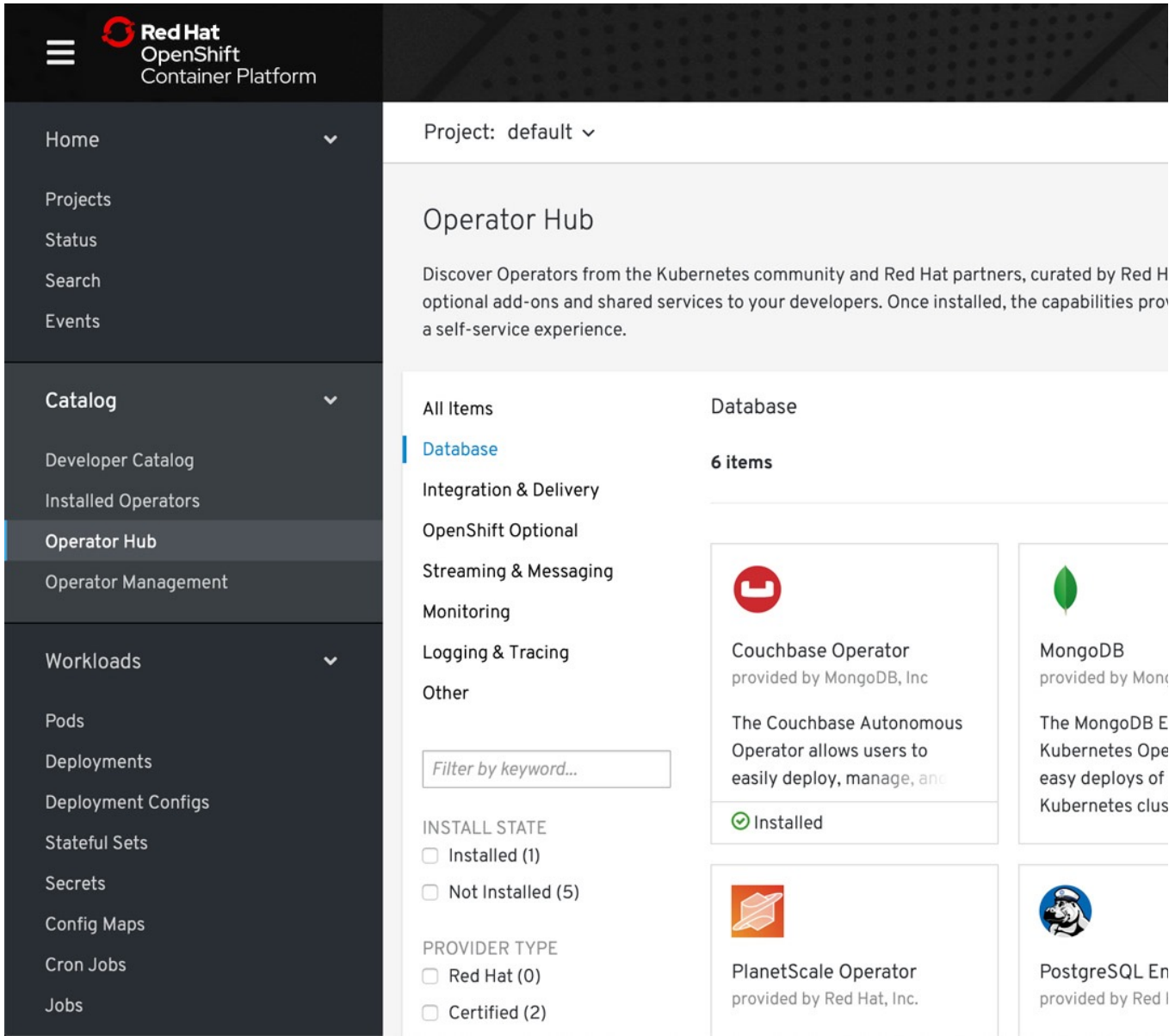
RED HAT OPEN SHIFT

Red Hat OpenShift es una plataforma de contenedores de Kubernetes empresarial

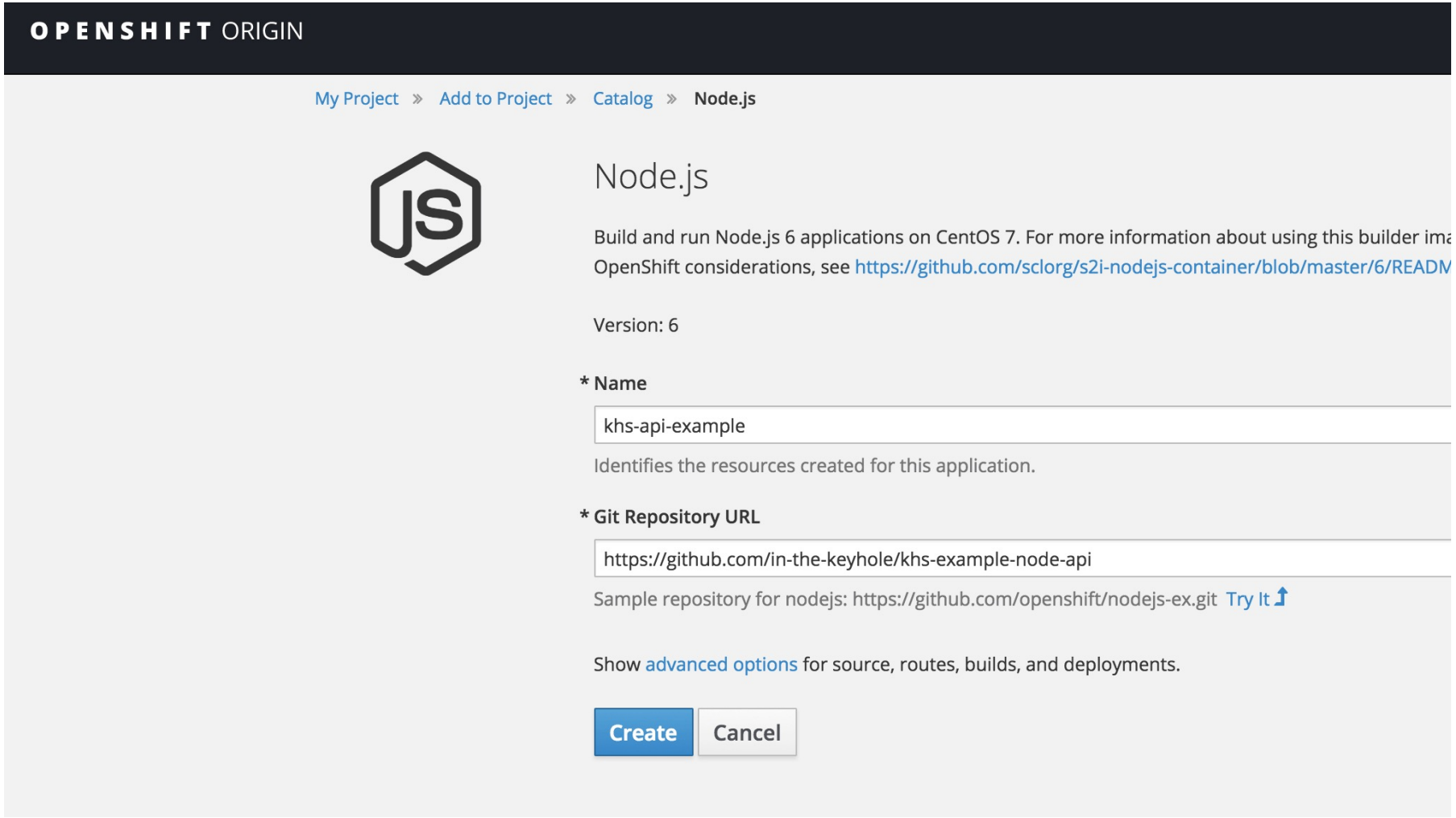
Centrada e la seguridad

Con operaciones automatizadas integrales que permite gestionar implementaciones de nube híbrida

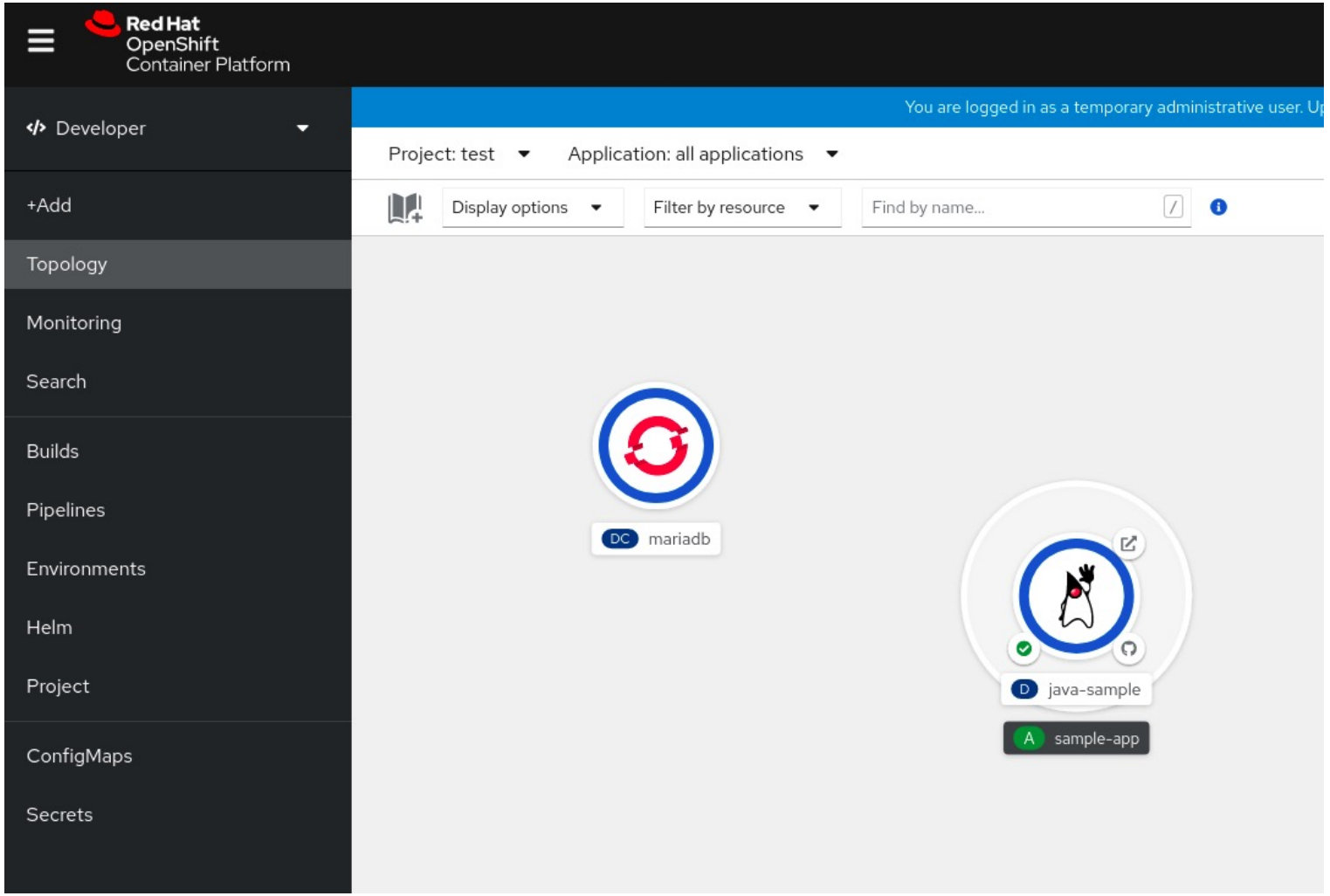
RED HAT OPEN SHIFT



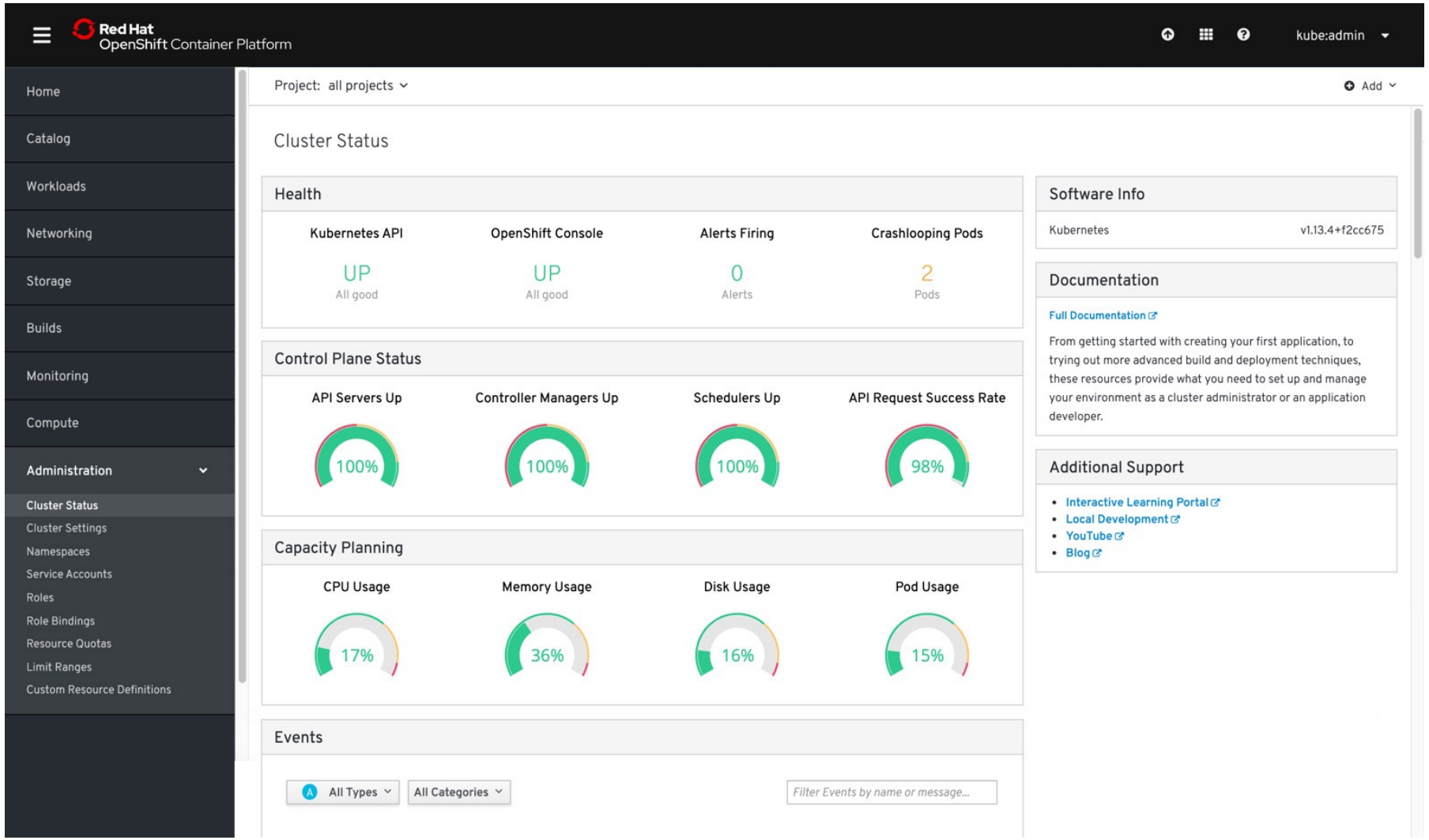
RED HAT OPEN SHIFT



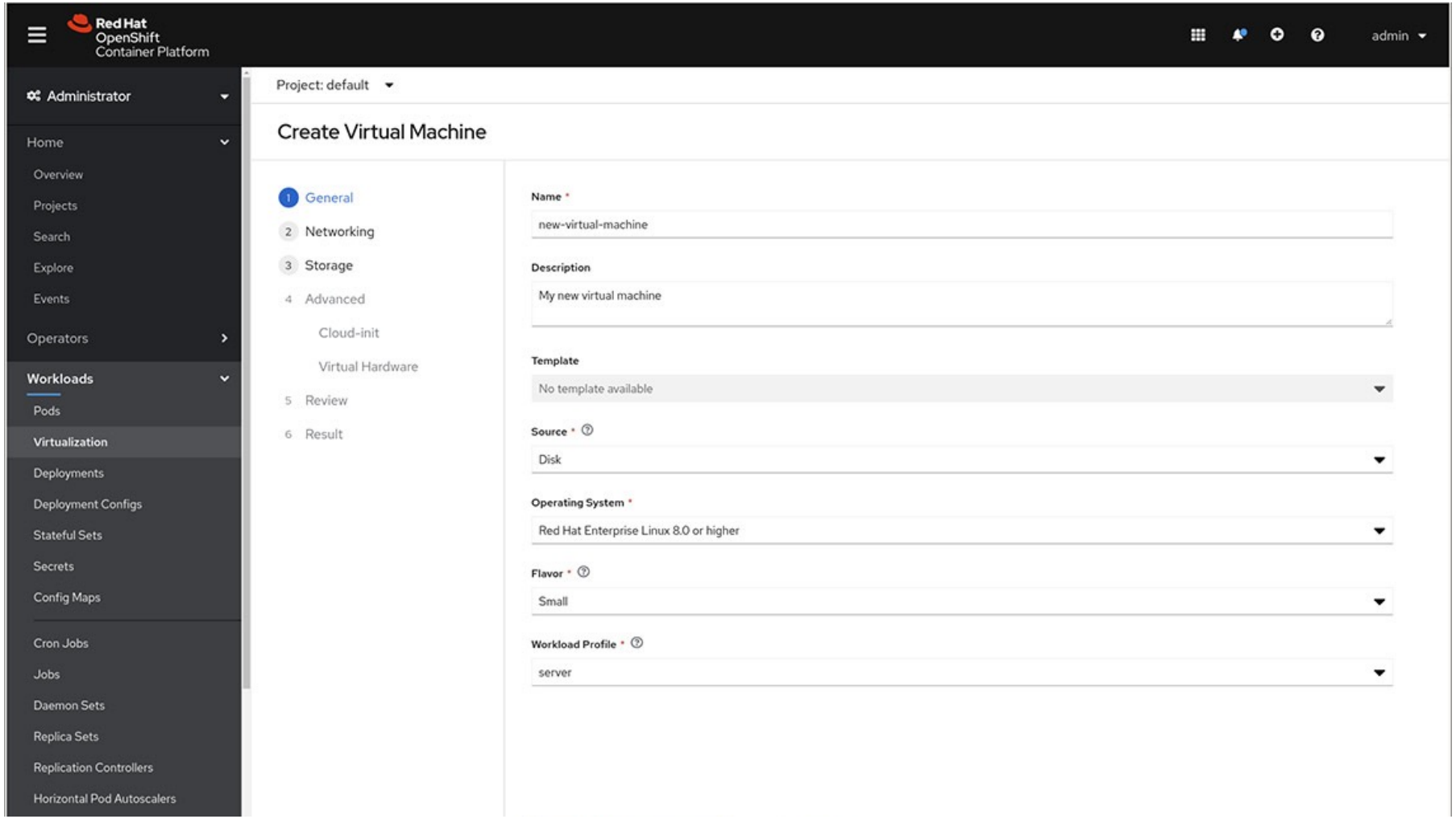
RED HAT OPEN SHIFT



RED HAT OPEN SHIFT



RED HAT OPEN SHIFT



www.ceste.es