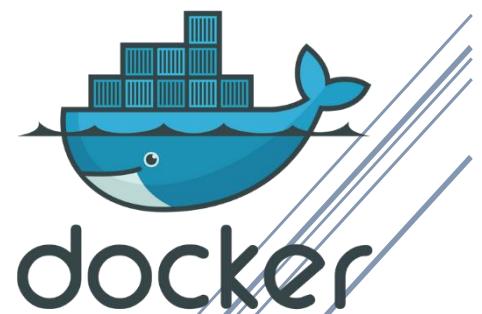
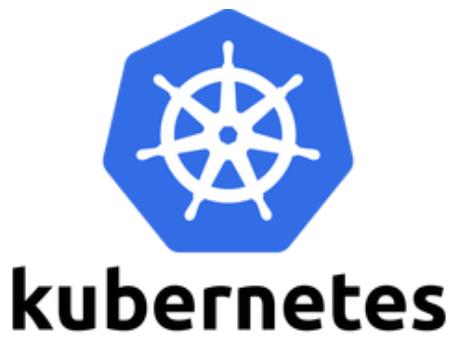


COMPARATIVA DE DESPLIEGUE Y AUTOESCALADO DE CONTENEDORES DOCKER

Proyecto Integrado de Administración de Sistemas
Informáticos en Red (PASIR)



José Antonio Díaz Gómez
I.E.S. Politécnico Jesús Marín

Contenido

1.	Introducción	4
2.	Microservicios.	5
2.1.	¿Qué son los microservicios?	5
2.2.	Despliegue automático de microservicios.	5
2.3.	Características del software de microservicios	6
2.4.	Ventajas.....	6
2.5.	Inconvenientes	7
3.	Kubernetes vs Docker Swarm. ¿Cuál es la mejor opción?	7
3.1.	¿Qué es Kubernetes?	7
3.2.	¿Qué es Docker Swarm?	9
4.	Ansible.	12
4.1.	¿Qué es ansible?	12
4.2.	Compatibilidad de Ansible.....	12
4.3.	¿Para qué sirve?.....	13
4.4.	Arquitectura de Ansible.....	13
4.5.	Instalación.....	14
4.6.	Componentes	14
4.7.	Principios básicos.....	15
5.	Docker-Swarm.....	16
5.1.	Instalación de docker en MV.....	16
5.2.	Creación de clúster	17
5.2.1.	Configuración del nodo administrador	17
5.2.2.	Configurar nodos de trabajo	17
5.2.3.	Verificar el grupo Swarm.....	18
5.3.	Dockerfile	19
5.3.1.	¿Qué es un Dockerfile?	19
5.3.2.	¿Cómo se utiliza?	19
5.3.3.	Contenido de dockerfile.....	19
5.3.4.	Ejemplo de dockerfile	20
5.4.	Docker Compose.....	21
5.4.1.	¿Qué es docker-compose?	21
5.4.2.	Instalación de Docker Compose	21

5.4.3. Contenido de Docker-Compose	22
5.4.4. Ejemplo Docker-Compose.....	22
5.5. Implementación de servicio en Clúster Swarm	23
5.5.1. Dockerfiles.....	23
5.5.2. Construcción de las imágenes	24
5.5.3. Subida de imágenes a Docker Hub.	24
5.5.4. Creación de docker compose	26
5.5.5. Creación volúmenes persistentes.....	28
5.5.6. Despliegue del entorno (stack).....	31
5.5.7. Recuperación automática ante errores	32
5.5.8. Escalado de la aplicación.....	33
5.5.9. Apagado de la aplicación y del Swarm.....	35
5.6. Portainer.io.....	36
5.6.1. ¿Qué es Portainer.io?	36
5.6.2. Instalación	36
5.6.3. Exploración de la herramienta.....	37
5.6.4. Creación de contenedor a golpe de click	49
5.6.5. Creación de stack.....	50
6. Kubernetes.....	54
6.1. Componentes a utilizar.	54
6.2. Comandos básicos.	55
6.3. Instalación de kubernetes en local	57
6.3.1. Escenario a montar en kubernetes.....	57
6.3.2. Instalación de Docker.....	57
6.3.3. Instalación de kubeadm	57
6.3.4. Plano de control de aislamiento del nodo	62
6.3.5. Unión de nodos	63
6.3.6. Limpiar	64
6.4. Despliegue de aplicaciones mediante archivos YAML.....	65
6.4.1. Despliegue de un Pod	65
6.4.2. Despliegue de un Deployment	66
6.4.3. Despliegue de un Service	67
7. Service Mesh.....	69

7.1.	¿Qué es la Service Mesh?	69
7.2.	¿Cómo funciona una Service Mesh?.....	69
7.3.	Estudio de una Service Mesh. Ventajas y Desventajas.....	69
8.	Istio.....	70
8.1.	¿Qué es Istio?	70
8.2.	Componentes de Istio	71
8.3.	Objetivos	72
8.4.	Funcionalidades	72
8.5.	Instalación de Istio en Kubernetes	72
9.	Rancher.....	74
9.1.	¿Qué es Rancher?	74
9.2.	Características de Rancher.....	74
9.3.	Instalación de Rancher.....	75
9.4.	Creación de Swarm en Rancher.....	77
9.4.1.	Exploración de la herramienta.	82
9.5.	Creación de clúster kubernetes.....	86
9.5.1.	Despliegue de aplicación.....	93
10.	Esquema Cloud.....	97
10.1.	Amazon Elastic Kubernetes Service (EKS).....	97
10.1.1.	¿Qué es Amazon EKS?	97
10.1.2.	Creación de clúster kubernetes con Ansible.....	97
10.1.2.1.	Instalación de AWS CLI.	100
10.1.3.	Despliegue de aplicación.	101
11.	Conclusiones.	103



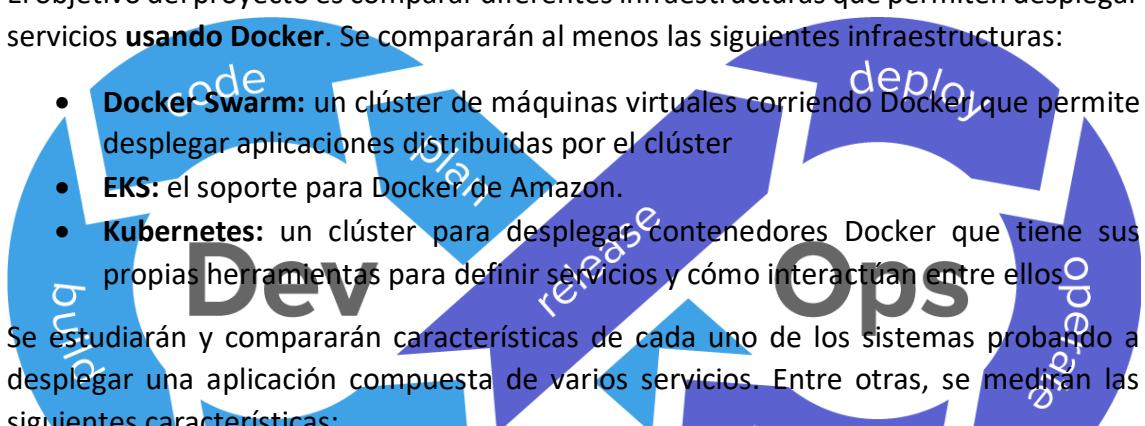
El diagrama circular de DevOps se divide en seis secciones principales: build, test, release, deploy, monitor y operate. Cada sección tiene un color y un efecto de sombra que la hacen sobresalir. El centro del círculo contiene la palabra 'Ops' en un efecto de sombra. Los pasos 'build' y 'test' están en azul, 'release' y 'deploy' en rojo, 'monitor' en verde y 'operate' en amarillo.

1. Introducción

Docker permite empaquetar y ejecutar aplicaciones en un entorno aislado, permitiendo incluso limitar los recursos que dichas aplicaciones utilizan del SO anfitrión en el que se ejecutan. Es una tecnología similar en funcionalidades a la virtualización, pero no hay una máquina virtual, sino que todas las características de aislamiento las proporciona el propio sistema operativo. A este tipo de tecnología se la denomina genéricamente contenedores y Docker es una de las implementaciones más exitosas y conocidas de este concepto (existen otras como rkt, lxc, OpenVZ, ...). El concepto de contenedor se ha extendido hasta el punto que los principales proveedores cloud proporcionan soporte para Docker de forma nativa (como Amazon AWS y Azure).

Se está hablando mucho de Docker para desplegar arquitecturas como Microservicios. Pero, ¿qué complicaciones implica desplegar un sistema compuesto de múltiples aplicaciones usando Docker?

El objetivo del proyecto es comparar diferentes infraestructuras que permiten desplegar servicios **usando Docker**. Se compararán al menos las siguientes infraestructuras:



Se estudiarán y compararán características de cada uno de los sistemas probando a desplegar una aplicación compuesta de varios servicios. Entre otras, se medirán las siguientes características:

- Facilidad de puesta en marcha.
- Compatibilidad con Docker Engine, Docker Compose y Docker Swarm.
- Facilidad de despliegue de contenedores con puertos expuestos.
- Facilidad de despliegue de contenedores linkados.
- Facilidad de actualización de contenedores en runtime.
- Soporte de grupos de autoescalado.

2. Microservicios.

2.1. ¿Qué son los microservicios?

Los **microservicios** son tanto un estilo de arquitectura como un modo de programar software. Con los microservicios, las aplicaciones se dividen en sus elementos más pequeños e independientes entre sí. A diferencia del enfoque tradicional y monolítico de las aplicaciones, en el que todo se compila en una sola pieza, **los microservicios son elementos independientes que funcionan en conjunto para llevar a cabo las mismas tareas**. Cada uno de esos elementos o procesos es un microservicio. Este enfoque de desarrollo de software valora el nivel de detalle, la sencillez y la capacidad para compartir un proceso similar en varias aplicaciones. Es un elemento fundamental de la optimización del desarrollo de **aplicaciones hacia un modelo nativo de la nube**.

En pocas palabras, el objetivo es **distribuir sistemas de software de calidad con mayor rapidez**, lo cual es posible gracias a los microservicios, pero también se deben considerar otros aspectos. Dividir las aplicaciones en microservicios no es suficiente; es necesario administrarlos, coordinarlos y gestionar los datos que crean y modifican.

2.2. Despliegue automático de microservicios.

No existe un estándar o definición formal para los microservicios, hay ciertas características que nos ayudan a identificarlos.

Básicamente, el desarrollo de un proyecto que se base en este método conforma una aplicación o herramienta mediante la conjunción de **diversos servicios independientes que se despliegan según se vayan necesitando**.

Por tanto, tendremos una aplicación modular a base de “pequeñas piezas”, que podremos ir ampliando o reduciendo a medida que se requiera.

A la hora de utilizar microservicios, hay que tener en cuenta que modelo de despliegue vamos a utilizar, es decir, al modo en que vamos a organizar y gestionar los despliegues de los microservicios, así como a las tecnologías que podemos usar para tal fin.

Existen convencionalmente dos tendencias en este sentido a la hora de **encapsular microservicios**:

- Máquinas virtuales.
- Contenedores.

2.3. Características del software de microservicios

Se puede descomponer en varias partes funcionales independientes.

Siendo así, cada uno de estos servicios podrá ser **desplegado, modificado y redesplegado sin comprometer los otros aspectos funcionales de la aplicación** y como resultado en caso de necesitarlo, **sólo tendremos que modificar un par de servicios** en lugar de redesplegar toda la aplicación al completo nuevamente.

La forma en la que se organizan los microservicios suele ser en torno a las necesidades, capacidades y prioridades del cliente o negocio en el que se implantará.

A diferencia de un entorno monolítico donde **cada equipo de trabajo tiene un enfoque específico sobre un apartado de la aplicación**, en la arquitectura de microservicios se utilizan **módulos multifuncionales**, adaptando así un módulo común a todos para que ofrezca un servicio determinado.

El **ahorro en tiempo de desarrollo es inmenso**, por no hablar de la comodidad a la hora de programar tareas de mantenimiento, donde podemos revisar un módulo mientras el resto del equipo de trabajo no ve interrumpida su jornada.



La arquitectura de microservicios mantiene un **sistema similar a un gobierno descentralizado**, donde cada módulo contará por ejemplo con su propia base de datos, en lugar de acudir todos a la misma sobrecargándola así de solicitudes y arriesgándose a que si falla ésta, todas las aplicaciones caigan.

Cuando **varios servicios están comunicados entre sí**, por lo general contarán con un **sistema de aviso y actuación** si alguno de estos servicios llega a fallar filtrando adecuadamente la información destinada a este módulo y favoreciendo la correcta gestión de los recursos entre los módulos funcionales restantes.

2.4. Ventajas

- Otorga a los desarrolladores **libertad de desarrollar y desplegar servicios** de forma independiente.
- Un microservicio se puede desarrollar con un **equipo de trabajo mínimo**.
- Se pueden usar **diferentes lenguajes de programación** en diferentes módulos.
- **Fácil integración y despliegue**.
- **Fácil de entender y modificar**, por lo que la integración de nuevos miembros al equipo de desarrollo será muy rápida.
- Los desarrolladores podrán hacer **uso de las tecnologías más actuales**.
- El uso de contenedores hará el **desarrollo y despliegue** de la app mucho más rápido.
- **Funcionalidad modular**, con lo que la modificación de un módulo no afectará al funcionamiento del resto.
- **Fácil de escalar e integrar** con aplicaciones de terceros.

2.5. Inconvenientes

- Las **pruebas o testeos pueden resultar complicados** debido al despliegue distribuido.
- Un gran número de servicios puede dar lugar a grandes bloques de información que gestionar.
- Será labor de los desarrolladores lidiar con aspectos como la latencia de la red, tolerancia a fallos, balanceo de carga, cantidad de formatos admitidos, etc...
- Sistema distribuido **puede llegar a significar doble trabajo**.
- Si se cuenta con un gran número de servicios, integrarlos y gestionarlos **puede resultar muy complejo**.
- Esta tecnología suele incurrir en un **alto consumo de memoria**.
- Fragmentar una aplicación en diferentes microservicios puede llevar **muchas horas de planificación**.

3. Kubernetes vs Docker Swarm. ¿Cuál es la mejor opción?

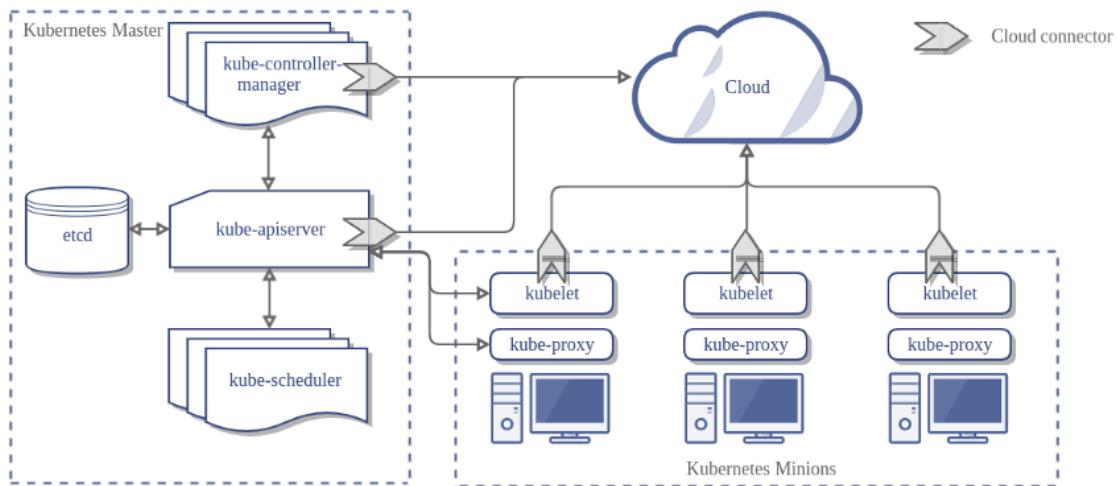


En este apartado se intentará hacer una comparación lineal de las diferencias y capacidades de estos sistemas de manejo de contenedores. Por un lado a kubernetes y por otro docker Swarm también conocido como Docker Engine.

Empezaremos por el principio:

3.1. ¿Qué es Kubernetes?

Kubernetes es una **herramienta extensible y de código abierto** para gestionar cargas de trabajo y servicios en contenedores, que facilita tanto la configuración declarativa como la automatización. **Tiene un ecosistema grande y de rápido crecimiento**. Los servicios, el soporte y las herramientas de kubernetes están ampliamente disponibles. Y se puede diagramar su base de trabajo de la siguiente manera:



Como se observa en el diagrama anterior, existen una serie de componentes asociados al clúster de Kubernetes. El nodo Master, **ubica las cargas de trabajo del contenedor en grupos de usuarios en nodos de trabajadores o en sí mismo**.

Los demás componentes son:

- **Etcdb**: Almacena los datos de configuración a los que puede acceder el Servidor API de Kubernetes Master utilizando http simple o API JSON.
- **Kube-Apiserver**: es el centro de gestión para el nodo Master, facilita la comunicación entre los diversos componentes manteniendo con ello la salud del clúster.
- **Kube-Controller-Manager**: es el encargado de asegurar la coincidencia entre el estado deseado del clúster y el estado actual, esto lo consigue escalando cargas de trabajo hacia arriba y hacia abajo
- **Kube-Scheduler**: coloca la carga de trabajo en el nodo que corresponde; en este diagrama particular, todas las cargas de trabajo se ubican localmente en su host.
- **Kubelet**: recibe las especificaciones del pod del servidor API y administra los pods que se ejecutan en el host.

Para estar en consonancia con los términos utilizados comúnmente cuando se habla de kubernetes, manejaremos la siguiente lista:

- **Pods**: Kubernetes implementa y programa contenedores en grupos llamados pods. Los contenedores en un pod se ejecutan en el mismo nodo y comparten recursos como sistemas de archivos, espacio de nombres en el kernel y una dirección IP.
- **Deployments**: estos bloques de construcción se pueden usar para crear y administrar un grupo de pods. Los despliegues se pueden usar a nivel de servicio para escalar horizontalmente garantizando disponibilidad.
- **Services**: son los puntos finales que se pueden direccionar por nombre y se pueden conectar a los pods utilizando los selectores de etiquetas. El servicio automáticamente enviará solicitudes por turnos entre pods. Kubernetes configurará un servidor DNS para el clúster que busca nuevos servicios y les

permite ser direccionados por su nombre. Los servicios son la parte frontal de las cargas de trabajo de su contenedor.

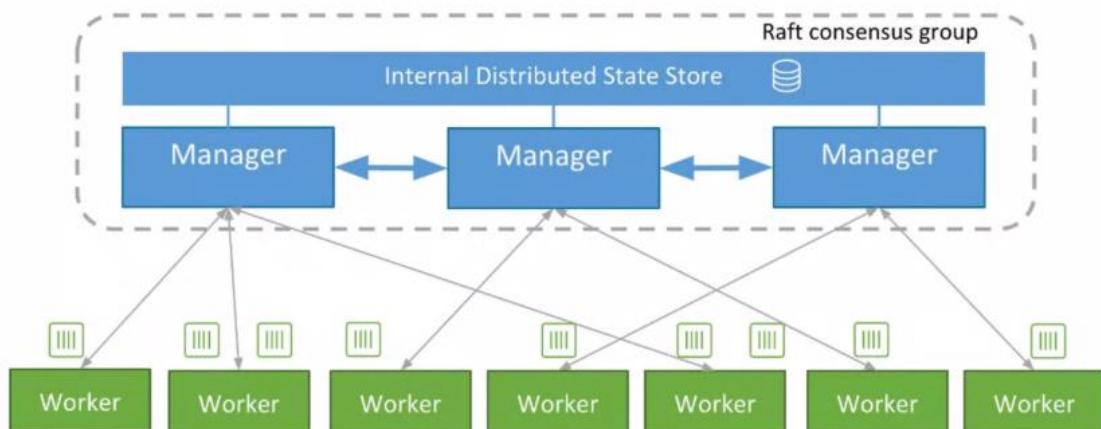
- **Labels:** son pares e clave-valor unidos a objetos y se pueden usar para buscar y actualizar múltiples objetos como un conjunto.

3.2. ¿Qué es Docker Swarm?

Es una herramienta que permite a los desarrolladores implementar contenedores en modo Swarm. Un clúster Swarm consiste en Docker Engine implementado en múltiples nodos. Los nodos de administración realizan la orquestación y la administración del clúster. Los nodos de trabajo reciben y ejecutan tareas desde los nodos de administración.

Un servicio consiste en tareas que puedes ejecutarse en nodos de Swarm. Los servicios se pueden replicar para ejecutarse en múltiples nodos. En el modelo de servicio replicados, el equilibrio de carga de ingreso y el DNS internos se pueden usar para proporcionar puntos finales de servicio altamente disponibles.

Swarm Architecture



Como se aprecia en el diagrama anterior, **la arquitectura Docker Swarm consta de Manager y Worker**. El usuario puede especificar declarativamente el estado de varios servicios para ejecutar en el clúster utilizando para ello ficheros de tipo YAML. Para esta arquitectura también listemos algunos términos comúnmente mencionados.

- **Nodo:** no es más que una instancia de un Swarm. Los nodos se pueden distribuir en las instalaciones o en nubes públicas.
- **Swarm:** lo conforman un grupo de nodos (Docker Engine). En este modelo se orquestan servicios en lugar de ejecutar comandos de contenedores.
- **Nodos managers:** reciben las definiciones de servicio del usuario y distribuyen el trabajo a los nodos de trabajadores. Los nodos de administración también pueden realizar las tareas de los nodos de trabajador.
- **Nodos workers:** recopilan y ejecutan tareas desde los nodos de administración.

- **Tarea:** es una unidad atómica de un servicio programado en un nodo trabajador.

Ya conociendo las arquitecturas y componentes de Kubernetes y Docker Swarm, pasaremos a realizar una comparativa de ambos ecosistemas.

Características		
Definición de la aplicación	<p>Las aplicaciones se pueden implementar usando una combinación de pods, implementaciones y servicios (o "microservicios"). Una implementación puede tener réplicas en múltiples nodos.</p>	<p>Las aplicaciones se pueden implementar como servicios (o "microservicios") en un clúster Swarm. Las aplicaciones de varios contenedores pueden especificarse utilizando archivos YAML. Docker Compose puede implementar la aplicación. Las tareas se pueden distribuir a través de centros de datos usando etiquetas.</p>
Construcciones escalabilidad aplicaciones	<p>Cada nivel de aplicación se define como un pod y se puede escalar cuando se administra mediante una implementación, que se especifica en YAML. La escala puede ser manual o automatizada.</p> <p>Los Pods se pueden usar para ejecutar pilas de aplicaciones integradas verticalmente, aplicaciones compartidas y coadministradas.</p>	<p>Los servicios se pueden escalar utilizando plantillas Docker Compose YAML. Los servicios pueden ser globales o replicados.</p> <p>Los servicios globales se ejecutan en todos los nodos; los servicios replicados ejecutan réplicas de los servicios en los nodos. Las tareas se pueden ampliar o reducir, y desplegar en paralelo o en secuencia.</p>
Alta disponibilidad	<p>Las implementaciones permiten que los pods se distribuyan entre los nodos para proporcionar HA, tolerando así las fallas de la aplicación.</p> <p>Los servicios de carga equilibrada detectan pods no saludables y los eliminan.</p> <p>Se admite alta disponibilidad de Kubernetes.</p> <p>Se pueden cargar múltiples nodos maestros y nodos de trabajo para solicitudes de kubectl y clientes. Etcd se pueden agrupar en clústeres y los servidores API se pueden replicar.</p>	<p>Los servicios se pueden replicar entre los nodos de Swarm.</p> <p>Los administradores de Swarm son responsables de todo el clúster y administran los recursos de los nodos de trabajadores.</p> <p>Los gerentes utilizan el equilibrio de carga de entrada para exponer los servicios externamente.</p> <p>Los administradores de Swarm usan el algoritmo de consenso de Raft para garantizar que tengan información de estado consistente.</p> <p>Se recomienda un número impar de gerentes, y la mayoría de los gerentes debe estar disponible para un clúster Swarm en funcionamiento.</p>

Balanceo de carga	Los pods se exponen a través de un servicio, que se puede usar como un equilibrador de carga dentro del clúster. Por lo general, una entrada se usa para equilibrar la carga.	El modo Swarm tiene un componente DNS que se puede usar para distribuir solicitudes entrantes a un nombre de servicio. Los servicios pueden ejecutarse en los puertos especificados por el usuario o pueden asignarse automáticamente.
Escalado automático para la aplicación	El escalado automático utilizando un objetivo simple de número de pods, se define de manera declarativa mediante implementación. El objetivo de utilización de CPU por pod está disponible.	No disponible directamente. Para cada servicio, puede declarar el número de tareas que desea ejecutar. Cuando escala manualmente hacia arriba o hacia abajo, el administrador Swarm se adapta automáticamente al agregar o eliminar tareas
Actualizaciones de aplicaciones continuas y reversión	El controlador de implementación es compatible con estrategias de actualización gradual y recreación. Las actualizaciones continuas pueden especificar el número máximo de pods no disponibles o el número máximo que se ejecuta durante el proceso.	En el momento del despliegue, puede aplicar actualizaciones continuas a los servicios. El administrador de Swarm le permite controlar la demora entre la implementación del servicio a diferentes conjuntos de nodos, con lo cual solo se actualizan 1 tarea a la vez.
Rendimiento y escalabilidad	A partir de 1.6, Kubernetes escala a clúster de 5.000 nodos. La escalabilidad de Kubernetes se compara con los siguientes objetivos de nivel de servicio: <ul style="list-style-type: none"> • Capacidad de respuesta de la API: el 99% de todas las llamadas API devuelven menos de 1s. • Tiempo de inicio del Pod: el 99% de los pods y sus contenedores (con imágenes pre-tiradas) comienzan en 5 segundos. 	Docker Swarm ha sido escalado y probado en hasta 30,000 contenedores y 1,000 nodos con 1 administrador de Swarm.

En cuanto a la popularidad y uso actual, **Kubernetes** es líder en todas las métricas cuando se compara con **Docker Swarm**, tiene más del 80% del interés en artículos de noticias, popularidad en herramientas como Github y búsquedas en la web. Por otra parte no es menos cierto la diferencia en la complejidad implicada en implementar Kubernetes en comparación con Docker Swarm, pero Kubernetes ha tratado de mitigar este inconveniente incorporando una variedad de opciones de implementación como Minikube y Kubeadm.

4. Ansible.



4.1. ¿Qué es ansible?

Ansible es un software que **automatiza el aprovisionamiento de software**, la gestión de configuraciones y el despliegue de aplicaciones. Está categorizado como una herramienta de **orquestación**, muy útil para los **administradores de sistema y DevOps**.

En otras palabras, Ansible permite a los DevOps **gestionar sus servidores**, configuraciones y aplicaciones de forma **sencilla, robusta y paralela**.

Ansible gestiona sus diferentes nodos **a través de SSH** y únicamente requiere **Python en el servidor** remoto en el que se vaya a ejecutar para poder utilizarlo. Usa **YAML** para describir acciones a realizar y las configuraciones que se deben propagar a los diferentes nodos.

Es **gratuita**, ya que utiliza una licencia **GNU GPL v3**, sigue con la filosofía del código abierto y permite automatizar la mayoría de los elementos de una infraestructura. Esto es, **desde servidores hasta dispositivos de red**. Puede funcionar, con el rol de servidor, en sistemas **GNU/Linux** y tipo **UNIX**, como por ejemplo **AIX, Solaris o BSD**.

También **permite trabajar con los proveedores de la nube**, como **AWS, Azure o Google Cloud Platform**. Gestionando componentes como redes, grupos de seguridad, direcciones IP o claves públicas.

Hace unos años Ansible fue adquirida por la compañía **Red Hat**.

4.2. Compatibilidad de Ansible.

Ansible se distribuye en Fedora, Red Hat enterprise Linux, CentOS y Scientific Linux mediante los paquetes EPEL, **además está disponible para diferentes distribuciones Linux** aparte de las anteriores mencionadas puedes verlo en este enlace y descargar la que necesites.

También está disponible para MAC, pero no para Windows, aunque podemos usarlo en máquinas virtuales.

4.3. ¿Para qué sirve?

Con él podemos **instalar aplicaciones, orquestar servicios y tareas más avanzas**. También se puede utilizar para la **estandarización de sistema operativo** y la **administración de servicios centralizados**.

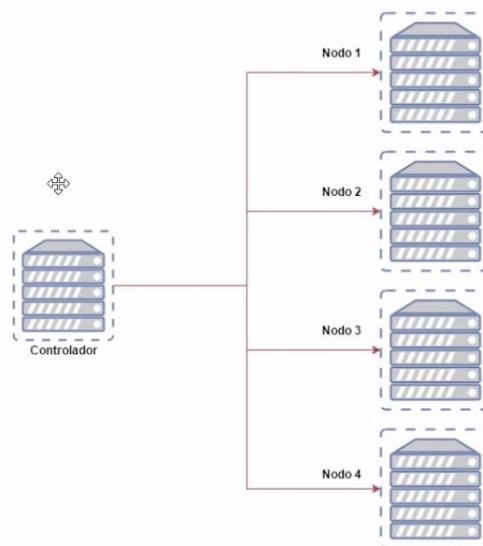
Si tenemos infraestructura tipo **IaaS**, esto es, **infraestructura como servicio**, tiene soporte para plataformas como **VMware** y **OpenStack**.

Veamos las ventajas que tiene Ansible, frente a otros productos:

- Su instalación es **muy sencilla**.
- **Gran compatibilidad** con la mayoría de los elementos de nuestra infraestructura.
- Soporta la mayoría de las distribuciones.
- Una **curva de aprendizaje muy corta**, ya que utiliza una sintaxis simple y no se necesitan excesivos conocimientos de programación.
- Una de las principales ventajas, frente a otros productos similares, es que **no necesita tener un agente en los clientes** que se gestionan. Primando de esta manera la seguridad al utilizar conexiones **SSH o WinRM**.
- Para configurar tareas complejas utiliza lenguaje **YAML**.

En Ansible existen dos tipos de servidores:

- **Controlador**: La máquina desde la que comienza la orquestación.
- **Nodo**: Es manejador por el controlador a través de SSH.



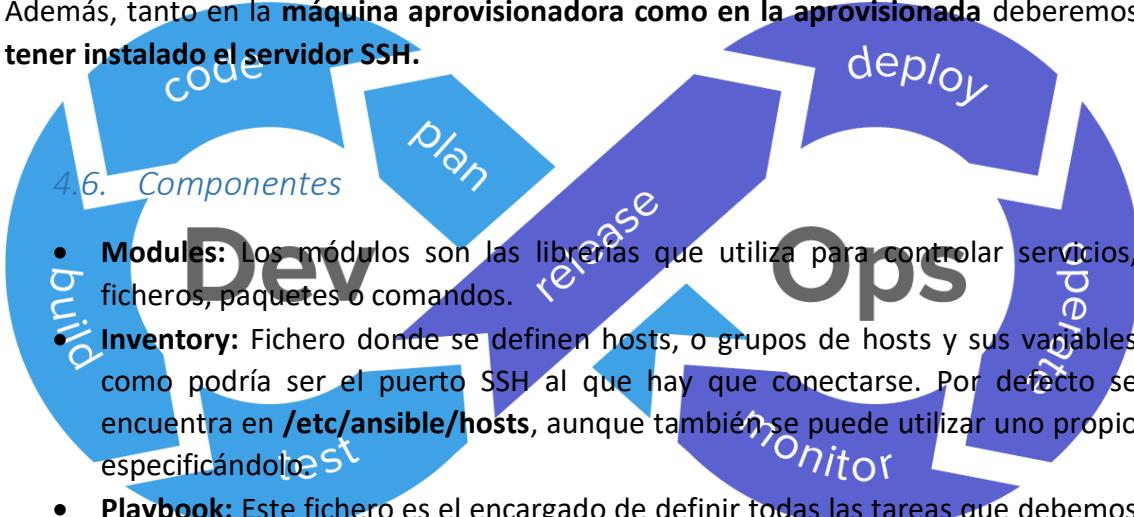
4.5. Instalación

La siguiente instalación se ha realizado una distribución **Debian** que en este caso es Debian 9.9, por lo que en otras versiones **puede variar la instalación**.

En la máquina que vamos a utilizar para aprovisionar, debemos instalar el paquete de ansible, además de Python, que se instalará automáticamente al instalar ansible.

```
root@debian:~# apt install ansible
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  ieee-data python-cffi-backend python-crypto python-cryptography python-enun
  python-ipaddress python-jinja2 python-kerberos python-markupsafe python-net
  python-pkg-resources python-pyasn1 python-selinux python-setuptools python-
Paquetes sugeridos:
  cowsay sshpass python-crypto-dbg python-crypto-doc python-cryptography-doc
  python-enum34-doc python-jinja2-doc ipython python-netaddr-docs python-gssap
Paquetes recomendados:
  python-vim
```

Además, tanto en la **máquina aprovisionadora como en la aprovisionada** deberemos tener instalado el servidor SSH.



- **Modules:** Los módulos son las librerías que utiliza para controlar servicios, ficheros, paquetes o comandos.
- **Inventory:** Fichero donde se definen hosts, o grupos de hosts y sus variables como podría ser el puerto SSH al que hay que conectarse. Por defecto se encuentra en **/etc/ansible/hosts**, aunque también se puede utilizar uno propio especificándolo.
- **Playbook:** Este fichero es el encargado de definir todas las tareas que debemos realizar sobre un conjunto de hosts.
- **Roles:** Los roles son grupos de ficheros y tareas parecidos sobre un determinado grupo de hosts.
- **Files:** Son los ficheros que se deben copiar en los hosts que pertenecen a ese role.
- **Tasks:** Definiremos las tareas que se deben ejecutar en los hosts que pertenecen a ese role.

4.7. Principios básicos

Crear par de llaves SSH

Como se ha dicho, Ansible funciona **mediante SSH**, por lo que vamos a generar un **par de llaves** para comunicar entre nuestro equipo controlador y el o los nodos que vamos a gestionar.

```
root@debian:~# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:VFA86mVjisc4Q6hgW3Vm0RK5M36A5QwqlxC4ljShTabE root@debian
The key's randomart image is:
+---[RSA 2048]---+
| +== o...B+
| +oBoo.o =.+
| o=E.oooo.o +
| ...=o=o+o *
| ...B+.SB .
| .o. . o
+---[SHA256]---+
```

Y llevamos nuestra **clave pública** al nodo o nodos **gestionados**.

```
root@debian:~# ssh-copy-id -i ~/.ssh/id_rsa.pub 192.168.15.16
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"
The authenticity of host '192.168.15.16 (192.168.15.16)' can't be established.
ECDSA key fingerprint is SHA256:Ty6jKXCQPkoxRvgCxlRtp5eJeRMjbW/dbs77i7TCwY.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@192.168.15.16's password:
Number of key(s) added: 1

Now try logging into the machine, with: "ssh '192.168.15.16'"
and check to make sure that only the key(s) you wanted were added.
```

Comprobar conexión test

En Ansible podremos **crear un inventario** como se ha explicado en apartados anteriores o modificar el fichero hosts de ansible, donde se definen hosts, o grupos de hosts y sus variables como podría ser el puerto SSH al que hay que conectarse. Por defecto **/etc/ansible/hosts**.

```
root@debian:~# nano /etc/ansible/hosts
```

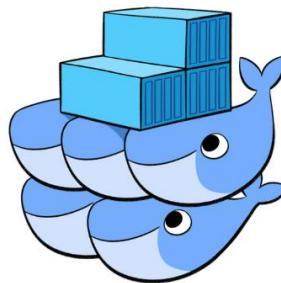
```
[prueba]
192.168.15.16
```

Y utilizaremos el módulo **ping** para probar si la conexión se realiza correctamente (aunque sea ping, se realiza la comprobación mediante SSH).

```
root@debian:~# ansible all -m ping -u root
192.168.15.16 | SUCCESS => {
    "changed": false,
    "ping": "pong"
```

En los siguientes apartados se explicará la utilización de playbook de ansible.

5. Docker-Swarm



5.1. Instalación de docker en MV.

En el apartado anterior [Que es Swarm](#) se ha explicado en que consiste docker-Swarm por lo tanto, se procederá a la instalación de **docker Engine**.

Para poder hacer el clúster con docker-Swarm deberemos de instalar antes docker Engine en las 3 máquinas virtuales que van a formar parte del clúster.

Para ello se ha seguido la guía de la página oficial de [Docker](#).

Primero descargamos los paquetes necesarios para permitir la descarga de paquetes.

```
root@swarm1:~# apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common
```

Descargamos y añadimos la clave GPG de Docker.

```
root@swarm1:~# curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add -
```

Y comprobamos que sea la correcta.

```
root@swarm1:~# apt-key fingerprint 0EBFCD88
pub    rsa4096 2017-02-22 [SCEA]
      9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88
uid            [ unknown] Docker Release (CE deb) <docker@docker.com>
sub    rsa4096 2017-02-22 [S]
```

Añadiremos el siguiente repositorio de docker para Ubuntu.

```
root@swarm1:~# add-apt-repository \
>   "deb [arch=amd64] https://download.docker.com/linux/debian \
>   $(lsb_release -cs) \
>   stable"
```

Actualizaremos nuestros repositorios con el que acabamos de añadir.

```
root@swarm1:~# apt update
Obj:1 http://security.debian.org/debian-security stretch/updates InRelease
Ign:2 http://ftp.es.debian.org/debian stretch InRelease
Des:3 http://ftp.es.debian.org/debian stretch-updates InRelease [91,0 kB]
```

E instalamos la última versión de docker.

```
root@swarm1:~# apt install docker-ce docker-ce-cli containerd.io
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
```

Finalmente comprobamos que lo tenemos instalado y verificamos que tenemos la última versión de docker.

```
root@swarm1:~# docker --version
Docker version 19.03.8, build afacb8b7f0
```

5.2. Creación de clúster

En este apartado se hará la realización del clúster con 3 máquinas virtuales las cuales serán, Swarm1, Swarm2 y Swarm3. La cual será el **nodo administrador**, el cual podrá realizar labores de administración en nuestro clúster, como crearlo, añadir miembros, etc.

5.2.1. Configuración del nodo administrador

A continuación crearemos el grupo Swarm de nuestros nodos. Para crear el clúster Swarm, necesitamos inicializar el modo Swarm en el nodo 'Swarm1' y luego unir el nodo 'Swarm2' y 'Swarm3' al clúster.

Inicializaremos el modo Docker Swarm ejecutando el siguiente comando docker en el nodo 'Swarm1'.



```
root@swarm1:~# docker swarm init --advertise-addr 192.168.15.16
Swarm initialized: current node (jj0qh4kfqlzodi60iru7x0y3) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-27kmx1sgb6b1c3xpqrbbw9fapsrmlq24dv7q9em0wmmm6tl1ck-48ff4q35gtu
qn16kaccgpyrr7 192.168.15.16:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

'Swarm1' ha generado el token de unión, que será necesario para unir los nodos de trabajo al administrador del clúster.

5.2.2. Configurar nodos de trabajo

Ahora, para unir los **nodos de trabajo** al clúster Swarm, ejecutaremos el comando join en todos los nodos de trabajo que recibimos en el paso de inicialización de Swarm.

```
root@swarm2:~# docker swarm join --token SWMTKN-1-27kmx1sgb6b1c3xpqrbbw9fapsrmlq24dv7q9em0wmmm6tl1ck-
48ff4q35gtuqn16kaccgpyrr7 192.168.15.16:2377
This node joined a swarm as a worker.
```

```
root@swarm3:~# docker swarm join --token SWMTKN-1-27kmx1sgb6b1c3xpqrbbw9fapsrmlq24dv7q9em0wmmm6tl1ck-
48ff4q35gtuqn16kaccgpyrr7 192.168.15.16:2377
This node joined a swarm as a worker.
```

5.2.3. Verificar el grupo Swarm

Para ver el estado del nodo, de modo que podamos determinar si los nodos están activos/disponibles, etc., desde el nodo administrador, enumeraremos todos los nodos en el Swarm.

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
jj0qh4kfqvlzod160iru7x0y3 *	swarm1	Ready	Active	Leader	19.03.8
sa36be3mopsm8lils0hnxxzy6	swarm2	Ready	Active		19.03.8
oq9fjrbadebllv703ew4k1v6g	swarm3	Ready	Active		19.03.8

Si en algún momento perdemos el token de unión, podemos recuperarlo ejecutando el siguiente ejecutando el siguiente comando en el nodo administrador para el token de administrador.

Este token sirve para poder añadir más nodos administradores a nuestro clúster Swarm.

```
root@swarm1:~# docker swarm join-token manager -q
SWMTKN-1-27kmx1sgb6b1c3xpqrbbw9fapsrmlq24dv7q9em0wmm6tllck-escf4opy5e9utfdl72z5lf4sq
```

La misma forma de recuperar el token de trabajo ejecutaremos el siguiente comando en el nodo administrador.

```
root@swarm1:~# docker swarm join-token worker -q
SWMTKN-1-27kmx1sgb6b1c3xpqrbbw9fapsrmlq24dv7q9em0wmm6tllck-48ff4q35gtuqn16kaccgpyrr7
```



5.3. Dockerfile

Antes de desplegar contenedores con diferentes servicios, tendremos que **generar una serie de dockerfiles** con la instalación de cada uno de ellos.

5.3.1. ¿Qué es un Dockerfile?

Es un fichero de texto que **contiene las instrucciones para automatizar la creación de una imagen** para posteriormente crear un contenedor docker.

Es decir, viene a simplificar la **creación de una imagen personalizada de un contenedor**, en el que en lugar de tener que crearlo, instalar el software necesario y tener que realizar finalmente un **commit** para obtener la imagen resultante, en el fichero Dockerfile le **indicamos la configuración final de nuestra imagen**.

5.3.2. ¿Cómo se utiliza?

Una vez tengamos creado a nuestro gusto nuestro fichero dockerfile, utilizaremos el comando **docker build** para construir la imagen con la configuración de este dockerfile.

Las opciones más habituales suelen ser las siguientes:

FROM: deberemos indicar la imagen original que utilizaremos para obtener nuestra imagen resultante.

LABEL: podemos indicar el administrador de esta imagen.

EXPOSE: indicamos el puerto que exponemos al exterior.

RUN: Escribiremos los distintos comandos a ejecutar que configurarán la imagen que obtendremos finalmente.

COPY: Podemos copiar ficheros de la máquina anfitrión al contenedor docker.

ADD: Agregar ficheros de nuestro equipo al contenedor, sin embargo también se pueden copiar archivos de una ubicación remota especificándola con una URL.

WORKDIR: Especificaremos el directorio de trabajo.

CMD: Comando predeterminado que se ejecute cuando iniciemos el contenedor que utilice esta imagen.

5.3.4. Ejemplo de dockerfile

Vamos a crear una imagen personalizando utilizando el siguiente **dockerfile**.

En él partimos de una imagen **php: 7.1-apache** e instalaremos una serie de paquetes como redis y dependencias de docker.

Exponemos interiormente el puerto 80 y crearemos un punto de montaje en el contenedor que será **/var/www/html**.

```
FROM php:7.1-apache
RUN pecl install redis \
    && docker-php-ext-enable redis
VOLUME /var/www/html
EXPOSE 80
```

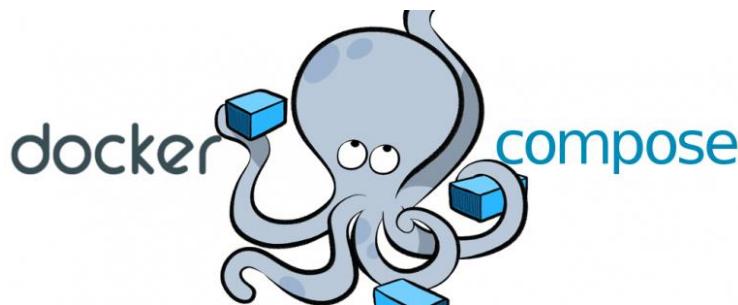
Y ejecutamos el **docker build**.

```
root@swarm1:~/docker-php-redis# docker build -t joseadiaz/web-php docker/web/...
Sending build context to Docker daemon 2.048kB
Step 1/4 : FROM php:7.1-apache
--> b9858ffdd4d2
Step 2/4 : RUN pecl install redis      && docker-php-ext-enable redis
--> Running in 8f26eb52d552
downloading redis-5.2.2.tgz ...
Starting to download redis-5.2.2.tgz (251,629 bytes)
.....done: 251,629 bytes
```

Una vez finalice nuestro build, obtendremos la imagen final.

```
root@swarm1:~/docker-php-redis# docker image ls
REPOSITORY          TAG           IMAGE ID
joseadiaz/web-php   latest        c5291fce73b1
```

5.4. Docker Compose



En este apartado se explicará la **herramienta docker-compose** porque se utilizará en apartados más adelante.

5.4.1. ¿Qué es docker-compose?

Compose es una herramienta para **definir y ejecutar aplicaciones** Docker de contenedores múltiples. Con Compose, utiliza un archivo YAML para configurar los servicios de su aplicación. Luego, con un solo comando, crearemos e iniciaremos los servicios desde su configuración.

Compose funciona en todos los entornos: **producción, puesta en escena, desarrollo, pruebas, así como flujos de trabajo de CI.**

5.4.2. Instalación de Docker Compose
Ejecutaremos este comando para descargar la versión estable actual de [Docker Compose](#).

```
root@swarm1:~# curl -L "https://github.com/docker/compose/releases/download/1.25.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Aplicaremos permisos ejecutables al binario.

```
root@swarm1:~# chmod +x /usr/local/bin/docker-compose  
root@swarm1:~#
```

Comprobaremos que la versión que se ha instalado, es la última versión.

```
root@swarm1:~# docker-compose --version  
docker-compose version 1.25.5, build 8alc60f6
```

5.4.3. Contenido de Docker-Compose

Docker-Compose es la similitud de ejecutar un **docker run**.

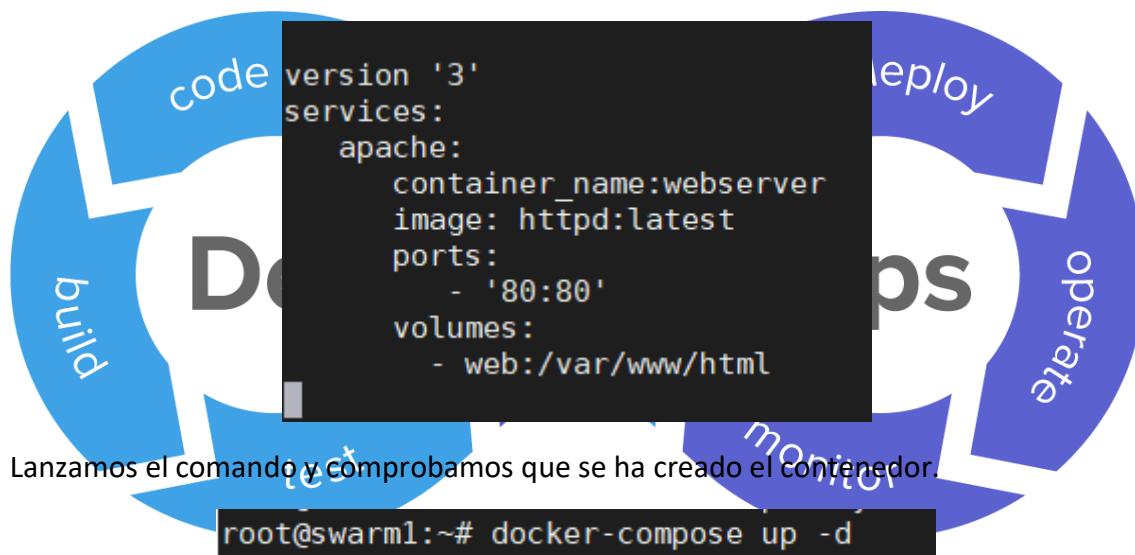
En nuestro fichero **docker-compose.yml** tendremos: el nombre del contenedor, imagen que utilizaremos, puertos que se exponen, volúmenes a utilizar, red utilizada, etc... al igual que tendríamos utilizando **docker run**.

5.4.4. Ejemplo Docker-Compose

Nos situamos en el directorio en el que tengamos nuestro fichero **docker-compose.yml** y lo ejecutamos con **docker-compose up**. Adicionalmente podemos acompañarlo con otras opciones, como por ejemplo **-d** para que se ejecute en segundo plano.

En este caso vamos a desplegar automáticamente la imagen de apache.

Con este docker-compose desplegaremos un contenedor llamado **webserver** que utilice la imagen **httpd:latest** y exponemos el puerto 80 en el exterior y el 80 en el interior.



CONTAINER ID	IMAGE NAMES	COMMAND	CREATED	STATUS	PORTS
0bcda968a002	httpd	"Httpd-foreground"	13 seconds ago	Up 11 seconds	0.0.0

5.5. Implementación de servicio en Clúster Swarm

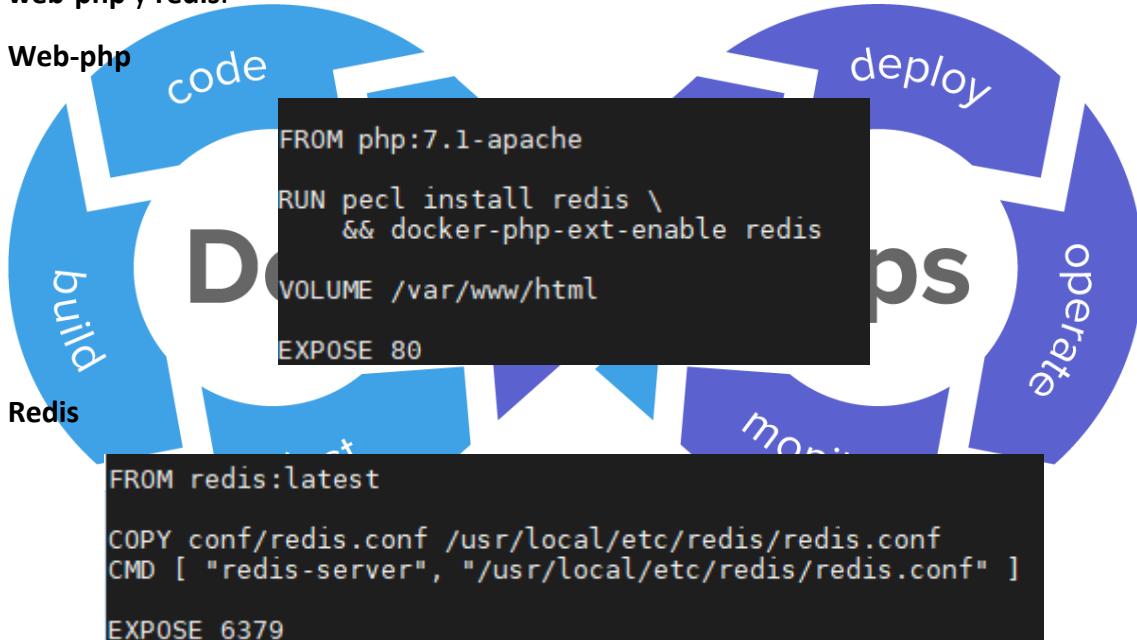
En esta apartado se creará una serie de servicios con la herramienta de **docker-Compose** que serán añadidos al clúster Swarm. Además de utilizar docker-compose para la creación de los servicios del clúster, también utilizaremos **Ansible** que se ha explicado anteriormente.

Los servicios que se van a crear se crearán mediante un **stack** que son un conjunto de servicios. El stack que se implementará en este caso será un **servidor web** y un **servidor redis** para aumentar la respuesta del servidor.

Las imágenes de los servicios se crearán mediante **dockerfile** fichero que se ha explicado en apartados anteriores.

5.5.1. Dockerfiles

A continuación se describirán los dockerfiles que se han creado para las imágenes de **web-php** y **redis**.



5.5.2. Construcción de las imágenes

Con el comando **docker build** construiremos las imágenes. Con **-t** definiremos una etiqueta o nombre de la imagen.

```
root@swarm1:~/docker-php-redis# docker build -t joseadiaz/web-php docker/web/.  
Sending build context to Docker daemon 2.048kB  
Step 1/4 : FROM php:7.1-apache  
--> b9858ffdd4d2  
Step 2/4 : RUN pecl install redis && docker-php-ext-enable redis  
--> Running in 8f26eb52d552  
downloading redis-5.2.2.tgz ...  
Starting to download redis-5.2.2.tgz (251,629 bytes)  
.....done: 251,629 bytes  
  
root@swarm1:~/docker-php-redis# docker build -t joseadiaz/redis docker/cache/.  
Sending build context to Docker daemon 50.18kB  
Step 1/4 : FROM redis:latest  
--> f9b990972689  
Step 2/4 : COPY conf/redis.conf /usr/local/etc/redis/redis.conf  
--> 1cec0207ba12  
Step 3/4 : CMD [ "redis-server", "/usr/local/etc/redis/redis.conf" ]  
--> Running in 5f3b822eb778  
Removing intermediate container 5f3b822eb778  
--> a4b9df3396be  
Step 4/4 : EXPOSE 6379  
--> Running in f832126f1fd5  
Removing intermediate container f832126f1fd5  
--> 2d6778546de0  
Successfully built 2d6778546de0  
Successfully tagged joseadiaz/redis:latest
```

Ahora listaremos las imágenes que acabamos de crear.

```
root@swarm1:~/docker-php-redis# docker image ls  
REPOSITORY          TAG           IMAGE ID            CREATED             SIZE  
joseadiaz/redis     latest         2d6778546de0   2 hours ago        104MB  
joseadiaz/web-php   latest         c5291fce73b1   2 hours ago        406MB
```

Para que los demás nodos tengan a su disposición las imágenes que han sido creadas en el nodo manager lo que haremos será subir nuestras imágenes a nuestro docker Hub.

Por eso a continuación se describirá que es docker Hub y su utilidad:

¿Qué es Docker Hub?

Docker Hub es un repositorio público en la nube, similar a Github, para distribuir los contenidos. Está mantenido por la propia Docker y hay multitud de imágenes, de carácter gratuito, que se pueden descargar y así no tener que hacer el trabajo desde cero al poder aprovechar “plantillas”.

Registro en Docker Hub

Nos registramos mediante la siguiente dirección: <https://hub.docker.com/signup>



Una vez explicado docker hub y su utilidad lo que haremos será loguearnos con nuestro usuario de docker hub en nuestra máquina para poder subir nuestras imágenes.

A continuación subiremos nuestras imágenes a nuestra cuenta.

```
root@swarm1:~/docker-php-redis# docker push joseadiaz/redis
The push refers to repository [docker.io/joseadiaz/redis]
3dc0d11099d7: Pushed
98b4a6242af2: Mounted from library/redis
4d0c19633152: Mounted from library/redis
d00fd460effb: Mounted from library/redis
379ef5d5cb40: Pushed
744315296a49: Mounted from library/redis
c2adabaecedb: Mounted from library/redis
latest: digest: sha256:1ff35ded4c79bfff52ab1427cd2c68d7e7ff86d5148575349c5f8b7e7a302bf56 size: 1781
```

```
joseadiaz@swarm1:~/docker-php-redis# docker push joseadiaz/web-php
The push refers to repository [docker.io/joseadiaz/web-php]
3380ef50c70b: Pushed
0817436a8f49: Mounted from library/php
3385a426f542: Mounted from library/php
35c986c7de74: Mounted from library/php
53bab0663330: Mounted from library/php
606c36b65880: Mounted from library/php
ab99fcc1a184: Mounted from library/php
9691e5d7a4c7: Mounted from library/php
6a4d393f0795: Mounted from library/php
e38834ac7561: Mounted from library/php
ec64f555d498: Mounted from library/php
840f3f414cf6: Mounted from library/php
17fce12edef0: Mounted from library/php
831c5620387f: Mounted from library/php
latest: digest: sha256:b9c1c6cd3ade5cd723a1a50e251baeb610d7eefbf76ae168997a2eb33143d214 size: 3246
```

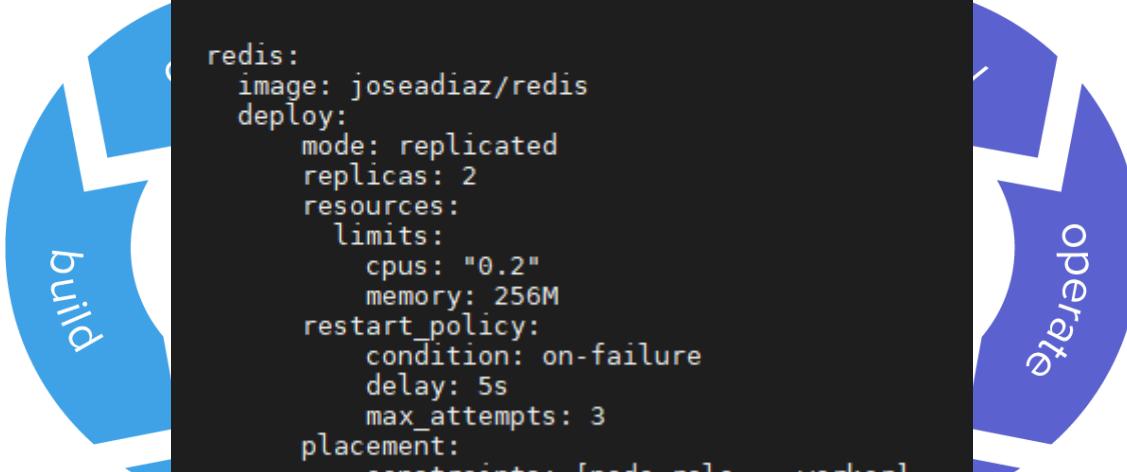
Además comprobaremos que se han subido correctamente a nuestra cuenta.



En el archivo **docker-compose.yml** definiremos cada uno de los servicios de nuestra aplicación, número de réplicas de los servicios y límites de recursos (CPU, RAM) asignados a cada contenedor.

En este caso definiremos dos servicios: **web-php** y **redis**.

- Las **imágenes** serán las que habremos creado en apartados anteriores.
- **Replicas:** nº de réplicas que tendrá el servicio definido.
- **Limits:** Límites de recursos, que en este caso es un 20% CPU y 256MB.
- **Placement:** se define el rol de nodo donde quieras correr ese servicio, en este caso se redifinirá solo en los nodos worker.
- **Restart policy:** reinicio en caso de fallar, con un retraso de 5 segundos y con 3 intentos.
- **Ports:** puertos a exponer en el servicio.
- **Volume:** se define un espacio para compartir archivos del contenedor al exterior o al contrario.
- **Network:** red que tendrán los servicios desplegados.



```
version: "3"

services:
  web-php:
    image: joseadiaz/web-php
    deploy:
      mode: replicated
      replicas: 2
      resources:
        limits:
          cpus: "0.2"
          memory: 256M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
    ports:
      - "8080:80"
    volumes:
      - /html:/var/www/html
    networks:
      - clusnet

  redis:
    image: joseadiaz/redis
    deploy:
      mode: replicated
      replicas: 2
      resources:
        limits:
          cpus: "0.2"
          memory: 256M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
      placement:
        constraints: [node.role == worker]
    ports:
      - "6379:6379"
    volumes:
      - /data:/data
    networks:
      - clusnet

networks:
  clusnet:
```

5.5.5. Creación volúmenes persistentes.

Dado que nuestro servidor web y nuestro servidor redis necesitan de volúmenes persistentes, tendremos que compartir una serie de carpetas para así tener los mismos archivos en los distintos nodos del clúster.

A continuación crearemos la estructura de carpetas que vamos a compartir en los distintos nodos.

```
root@debian:~# nano /etc/exports
root@debian:~# mkdir -p /srv/redis_web/data
root@debian:~# mkdir /srv/redis_web/html
root@debian:~# chmod 755 /srv/redis_web
root@debian:~# chown nobody:nogroup /srv/redis_web
```

Instalaremos el servidor NFS con los siguientes paquetes.

```
root@debian:~# apt install nfs-kernel-server nfs-common
```

Ahora debemos modificar **/etc/exports** donde exportaremos nuestros recursos compartidos. Especificamos el directorio **/srv/redis_web** como recurso compartido.

Le pondremos los siguientes permisos a dicha carpeta.

```
root@debian:~# nano /etc/exports
/srv/redis_web 192.168.15.0/24(rw,sync,no_subtree_check,no_root_squash,crossmnt)
```

Para aplicar los cambios en el fichero **/etc/exports**, reiniciamos el servidor kernel nfs.

```
root@debian:~# systemctl restart nfs-kernel-server.service
```

Una vez hecho la parte del servidor lo que nos toca hacer será configurar la parte del cliente, para ello se ha creado un yml de **ansible** para configurar los 3 nodos a la vez.

```
- hosts: all
  vars:
    nfs_version: "4"
    nfs_server: "192.168.15.19"
    nfs_directory: "/srv/redis_web"
  roles:
    - role: ome.nfs_mount
      nfs_share_mounts:
        - path: /data
          location: "{{nfs_server}}:{{nfs_directory}}/data"
        - path: /html
          location: "{{nfs_server}}:{{nfs_directory}}/html"
```

En este yml se han definido una serie de variables que son la **versión de nfs**, el **servidor nfs** y el **directorio que se va a compartir**.

Se ha utilizado un rol para instalar el **cliente nfs** además de configurar y montar las carpetas en el lado cliente.

Para ello se ha descargado el rol con el comando **ansible-galaxy**. Si quiere saber más de este rol visite este [enlace](#).

```
root@debian:~/nfs-client# ansible-galaxy install ome.nfs_mount
- downloading role 'nfs_mount', owned by ome
- downloading role from https://github.com/ome/ansible-role-nfs-mount/archive/1.3.0.tar.gz
- extracting ome.nfs_mount to /etc/ansible/roles/ome.nfs_mount
- ome.nfs_mount was installed successfully
```

La etiqueta **path** sirve para crear un directorio en el lado del cliente.

Y la etiqueta **location** sirve para establecer conexión con un servidor que en este caso es el de NFS.

Lo interesante de este rol es que nos monta las carpetas en el lado del cliente además de añadir una línea en el fichero **/etc/fstab** para que aunque el nodo se apague por algún motivo pueda montar la carpeta otra vez sin problemas.

Para lanzar nuestro yml ejecutaremos el comando **ansible-playbook**.

```
root@debian:~/nfs-client# ansible-playbook -i nfs-client nfs-client.yml -vvvvvvvvv
Using /etc/ansible/ansible.cfg as config file
Loading callback plugin default of type stdout, v2.0 from /usr/lib/python2.7/dist-packages/ansible/plugins/callback/_init_.pyc

PLAYBOOK: nfs-client.yml ****
1 plays in nfs-client.yml

PLAY [all] ****
TASK [setup] ****
Using module file /usr/lib/python2.7/dist-packages/ansible/modules/core/system/setup.py
Using module file /usr/lib/python2.7/dist-packages/ansible/modules/core/system/setup.py
<192.168.15.18> ESTABLISH SSH CONNECTION FOR USER: root
<192.168.15.17> ESTABLISH SSH CONNECTION FOR USER: root
<192.168.15.17> SSH: ansible.cfg set ssh_args: (-C)(-o)(ControlMaster=auto)(-o)(ControlPersist=60s)
<192.168.15.18> SSH: ansible.cfg set ssh_args: (-C)(-o)(ControlMaster=auto)(-o)(ControlPersist=60s)
<192.168.15.18> SSH: ansible_password/ansible_ssh_pass not set: (-o)(KbdInteractiveAuthentication=no)(-o)(PreferredAuthentications=gssapi-with-mic,gssapi-keyex,hostbased,publickey)(-o)(PasswordAuthentication=no)
<192.168.15.18> SSH: ANSIBLE REMOTE USER/remote_user/ansible_user/user/-u set: (-o)(User=root)
```

La opción **-i** es para definir un inventario que en este caso se ha definido el siguiente.

```
[nfs-client]
swarm1 ansible_host=192.168.15.16 ansible_user=root
swarm2 ansible_host=192.168.15.17 ansible_user=root
swarm3 ansible_host=192.168.15.18 ansible_user=root
```

Se ha establecido el **nombre del host**, su **ip** y el **usuario que va a aprovisionar** dicho hosts.

```

changed: [swarm2] => (item={'path': '/html', 'location': '192.168.15.19:/srv/redis_web/html'}) => {
    "changed": true,
    "dump": "0",
    "fstab": "/etc/fstab",
    "fstype": "nfs",
    "invocation": {
        "module_args": {
            "boot": "yes",
            "dump": null,
            "fstab": "/etc/fstab",
            "fstype": "nfs",
            "name": "/html",
            "opts": "vers=4,rsize=8192,wsize=8192,timeo=14,intr",
            "passno": null,
            "src": "192.168.15.19:/srv/redis_web/html",
            "state": "mounted"
        },
        "module_name": "mount"
    },
    "item": {
        "location": "192.168.15.19:/srv/redis_web/html",
        "path": "/html"
    },
    "name": "/html",
    "opts": "vers=4,rsize=8192,wsize=8192,timeo=14,intr",
    "passno": "0",
    "src": "192.168.15.19:/srv/redis_web/html"
}

PLAY RECAP ****
swarm1 : ok=3    changed=1    unreachable=0    failed=0
swarm2 : ok=3    changed=1    unreachable=0    failed=0
swarm3 : ok=3    changed=1    unreachable=0    failed=0

```

Se comprobará en los nodos clientes de NFS que se han creado dichas carpetas y que además se ha compartido su contenido, que en este caso son dos ficheros en la carpeta **/html** para nuestro servidor web.



Con **session.save_path** estableceremos la ruta de almacenamiento de la sesión actual que esta se guardará en redis.

```

<?php
ini_set('session.save_handler', 'redis');
ini_set('session.save_path', 'tcp://my_app_redis:6379,tcp://my_app_redis:6379');

session_name('FOOBAR');
session_start();
echo nl2br('<pre>' . session_save_path() . '</pre>' . PHP_EOL);

echo nl2br('Running PHP version: ' . phpversion() . PHP_EOL);

if (!array_key_exists('visit', $_SESSION)) {
    $_SESSION['visit'] = 0;
}
$_SESSION['visit']++;

echo nl2br('You have been here ' . $_SESSION['visit'] . ' times.');

```

Info.php: archivo para mostrar la información de php, con este archivo por ejemplo podremos comprobar las librerías que tiene activadas para php.

```
<?php phpinfo();
```

5.5.6. Despliegue del entorno (stack).

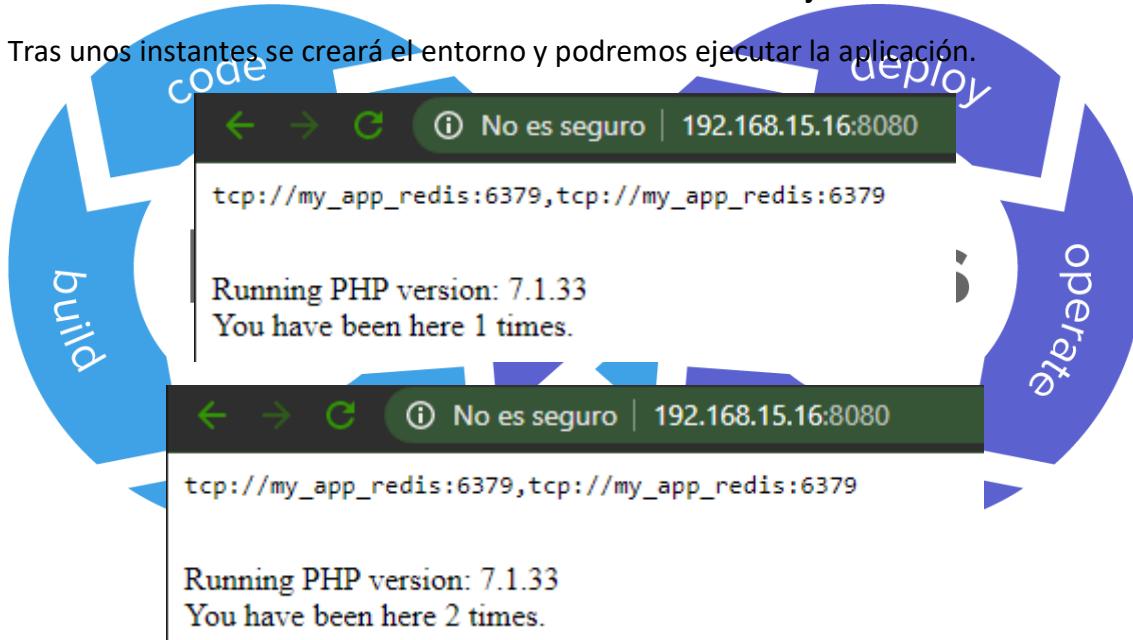
Una vez explicado nuestro fichero **docker-compose** y la **compartición de carpetas necesarias a los nodos** del clúster lo que haremos será desplegar nuestra aplicación.

Para ello ejecutaremos el siguiente comando.

```
root@swarm1:~/docker-php-redis# docker stack deploy -c docker-compose.yml my_app
Creating network my_app_clusnet
Creating service my_app_web-php
Creating service my_app_redis
```

El parámetro **-c** es opcional y especifica el **archivo compose**. **My_app** es el nombre que le damos al stack creado. Pensemos en un **stack** como un **conjunto de servicios**.

Tras unos instantes se creará el entorno y podremos ejecutar la aplicación.



Una vez comprobado que nuestra aplicación funciona correctamente vamos a ver varios comandos para conocer el estado del stack creado

Con **docker stack** podemos gestionar stacks. Por ejemplo, con **docker stack ls** vemos los stack creados con la cantidad de servicios que incluye cada uno.

```
root@swarm1:~/docker-php-redis# docker stack ls
NAME          SERVICES      ORCHESTRATOR
my_app        2             Swarm
```

Con **docker service ls** vemos los distintos servicios y la cantidad y estado de sus réplicas.

```
root@swarm1:~/docker-php-redis# docker service ls
ID          NAME      MODE      REPLICAS      IMAGE
8f5p32158tz5   my_app_redis  replicated  2/2  joseadiaz/redis:latest
b7kkopw6kdke   my_app_web-php  replicated  2/2  joseadiaz/web-php:latest
                                         PORTS
                                         *:6379->6379/tcp
                                         *:8080->80/tcp
```

Con **docker stack my_app** vemos el estado de cada una de las tareas (contenedores) del stack, ver en qué nodo este cada contenedor, posibles errores que haya pasado, etc.

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR
9mq7ehvt5jg9	my_app_redis.1	joseadiaz/redis:latest	swarm2	Running	Running 37 minutes ago	
jfyk49rwv4sa	my_app_web-php.1	joseadiaz/web-php:latest	swarm3	Running	Running 37 minutes ago	
szd5dg1trop7	_ my_app_web-php.1	joseadiaz/web-php:latest	swarm2	Shutdown	Rejected 2 hours ago	"No such image: joseadiaz/web..."
h44iu5nt0017	my_app_redis.1	joseadiaz/redis:latest	swarm2	Shutdown	Rejected 2 hours ago	"No such image: joseadiaz/redis..."
wbfdrvzjmo4w	_ my_app_redis.1	joseadiaz/redis:latest	swarm2	Shutdown	Rejected 2 hours ago	"No such image: joseadiaz/redis..."
olpfm0t774fq	my_app_web-php.1	joseadiaz/web-php:latest	swarm2	Shutdown	Rejected 2 hours ago	"No such image: joseadiaz/web..."
w9j92xu5kbos	my_app_redis.1	joseadiaz/redis:latest	swarm2	Shutdown	Rejected 2 hours ago	"No such image: joseadiaz/redis..."
p6q13h7wd8nw	my_app_web-php.1	joseadiaz/web-php:latest	swarm2	Shutdown	Rejected 2 hours ago	"No such image: joseadiaz/web..."
s8bxuyj7mgil	my_app_redis.2	joseadiaz/redis:latest	swarm3	Running	Running 37 minutes ago	
xceutnpz9jub	my_app_web-php.2	joseadiaz/web-php:latest	swarm1	Running	Running 37 minutes ago	

5.5.7. Recuperación automática ante errores

Veamos como al realizar una operación **kill** sobre uno de los contenedores, tras unos instantes vuelve a crearse un nuevo contenedor en su puesto, garantizando el número de réplicas especificado.

Primero mostraremos los contenedores actuales.

En **Swarm1** que es el nodo manager hay un contenedor web-php.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
4bc5a368def8	joseadiaz/web-php:latest	"docker-php-entrypoi..."	43 minutes ago	Up 43 minutes	80/tcp	my_app_web-php.2

En **Swarm2** worker hay un contenedor redis.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
7c7556002d4	joseadiaz/redis:latest	"docker-entrypoint.s..."	43 minutes ago	Up 43 minutes	6379/tcp	my_app_redis.1.9mq7ehvt5jg9

Y en **Swarm3** hay 2 contenedores uno redis y otro web-php.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
83d9ea9e5c83	joseadiaz/redis:latest	"docker-entrypoint.s..."	43 minutes ago	Up 43 minutes	6379/tcp	my_app_redis.2.s8
jfyk49rwv4sa3w324cih3yu4c	joseadiaz/web-php:latest	"docker-php-entrypoi..."	43 minutes ago	Up 43 minutes	80/tcp	my_app_web-php.1

Ahora lanzamos un **kill** por ejemplo sobre el contenedor **redis** de **Swarm3**.

```
root@swarm3:~# docker kill 83d9ea9e5c83
83d9ea9e5c83
```

Tras unos instantes habrá un nuevo contenedor en su puesto.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
c3ff876beb5	joseadiaz/redis:latest	"docker-entrypoint.s..."	2 minutes ago	Up About a minute	6379/tcp	my_app_redis.2.1.7
jfyk49rwv4sa3w324cih3yu4c	joseadiaz/web-php:latest	"docker-php-entrypoi..."	50 minutes ago	Up 50 minutes	80/tcp	my_app_web-php.1

5.5.8. Escalado de la aplicación

En Docker Swarm **podemos aumentar o disminuir el número de réplicas** de un servicio mediante **comandos** o **volviendo a desplegar el stack** modificando el número de réplicas.

5.5.8.1. Escalado mediante comandos

La sintaxis del comando es la siguiente:

```
#docker service scale <SERVICE-ID>=<NUMBER-OF-TASKS>
```

Por ejemplo, para que el número de réplicas del servicio **web-php** del stack **my_app** sean 4 ejecutaríamos el siguiente comando.

```
root@swarm1:~# docker service scale my_app_web-php=4
my_app_web-php scaled to 4
overall progress: 4 out of 4 tasks
1/4: running [=====]
2/4: running [=====]
3/4: running [=====]
4/4: running [=====]
verify: Service converged
```

Comprobaremos que han aumentado los contenedores de **web-php**.

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
8f5p32158tz5	my_app_redis	replicated	2/2	joseadiaz/redis:latest	*:6379->6379/tcp
b7kkopw6kdkc	my_app_web-php	replicated	4/4	joseadiaz/web-php:latest	*:8080->80/tcp

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR
jfyk49rwv4sa	my_app_web-php.1	joseadiaz/web-php:latest	swarm3	Running	Running about an hour ago	
szd5dgitrop7	my_app_web-php.1	joseadiaz/web-php:latest	swarm2	Shutdown	Rejected 2 hours ago	"No such i
olpfm0t774fq	my_app_web-php.1	joseadiaz/web-php:latest	swarm2	Shutdown	Rejected 2 hours ago	"No such i
p6013h7wd8nw	my_app_web-php.1	joseadiaz/web-php:latest	swarm2	Shutdown	Rejected 2 hours ago	"No such i
xceutnppz9jub	my_app_web-php.2	joseadiaz/web-php:latest	swarm1	Running	Running about an hour ago	
glsbje0mporm	my_app_web-php.2	joseadiaz/web-php:latest	swarm1	Shutdown	Shutdown about an hour ago	
o79tr9yix8xq	my_app_web-php.3	joseadiaz/web-php:latest	swarm2	Running	Running 3 minutes ago	
w6t61wnit9eo	my_app_web-php.4	joseadiaz/web-php:latest	swarm1	Running	Running 3 minutes ago	

5.5.8.2. Escalado volviendo a desplegar el stack

Para escalar con la técnica de *redespliegue*, tendremos que editar el archivo **docker-compose.yml** con el nuevo número de réplicas y volver a hacer el despliegue

Por ejemplo, probaremos a reducir a 3 el número de réplicas. Con **docker stack ps my_app** podemos ver los cambios, así como con **docker ps**.

```
version: "3"

services:
  web-php:
    image: joseadiaz/web-php
    deploy:
      mode: replicated
      replicas: 3
      resources:
        limits:
          cpus: "0.2"
          memory: 256M
```

Redesplegaremos el **stack** con el siguiente comando:

```
root@swarm1:~/docker-php-redis# docker stack deploy -c docker-compose.yml my_app
Updating service my_app_web-php (id: b7kkopw6kdke3irloamsqna)
Updating service my_app_redis (id: 8f5p32158tz5gloli6w9164vi)
```

Comprobaremos que se ha establecido correctamente las réplicas.

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
8f5p32158tz5	my_app_redis	replicated	2/2	joseadiaz/redis:latest	*:6379->6379/tcp
b7kkopw6kdke	my_app_web-php	replicated	3/3	joseadiaz/web-php:latest	*:8080->80/tcp

Haremos **docker stack ps my_app** y buscaremos la línea de Shutdown donde dice que el contenedor se ha apagado, pero eso ha sido que el sistema ha apagado ese contenedor porque hemos reducido las réplicas.

gl3sbje0mporm	my_app_web-php.2	joseadiaz/web-php:latest	swarm1	Shutdown	Shutdown 2 hours ago
o79tr9yix8xq	my_app_web-php.3	joseadiaz/web-php:latest	swarm2	Running	Running 33 minutes ago

NOTA: Esta operación de actualización del despliegue es la que también se usa para añadir nuevos servicios a un *stack*. Basta con añadir los nuevos servicios a **docker-compose.yml** y *redesplegar* el stack.

5.5.9. Apagado de la aplicación y del Swarm

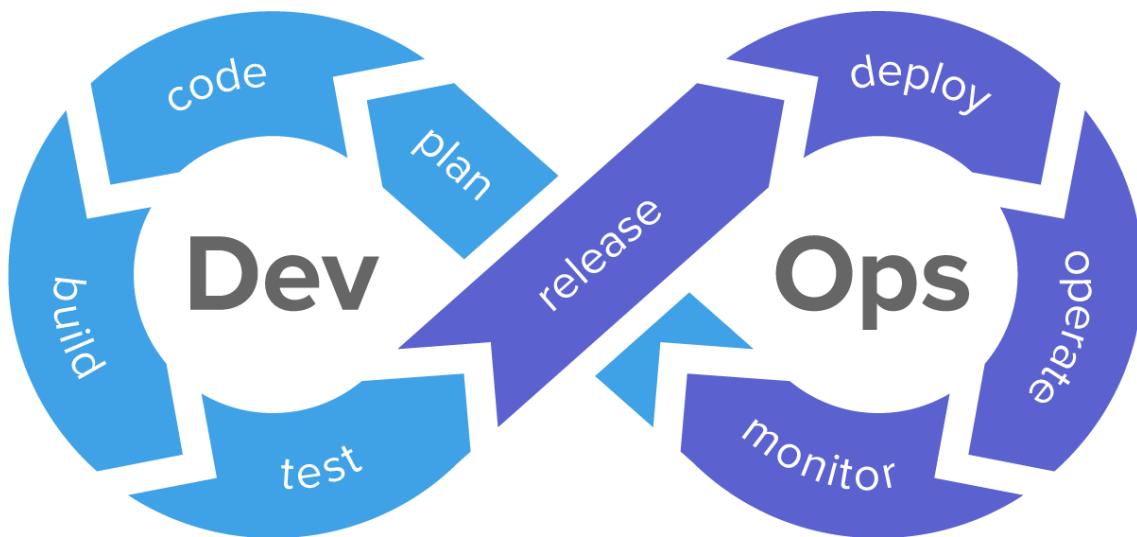
Para eliminar el stack de dos servicios solo tendremos que ejecutar el siguiente comando.

```
root@swarm1:~/docker-php-redis# docker stack rm my_app
Removing service my_app_redis
Removing service my_app_web-php
Removing network my_app_clusnet
```

Esta operación detendrá todos los contenedores asociados al stack y además borrará la red creada para esta aplicación.

Para que un nodo ya sea worker o manager deje el Swarm ejecutaremos el siguiente comando.

```
root@swarm1:~/docker-php-redis# docker swarm leave --force
```



5.6. Portainer.io



5.6.1. ¿Qué es Portainer.io?

Portainer es una **herramienta web open-source** la cual se ejecuta ella misma como un contenedor, por tanto deberemos tener Docker instalado. Esta aplicación nos va a permitir gestionar de forma **muy fácil e intuitiva** nuestros contenedores Docker a través de una interfaz gráfica.

Con esta herramienta podemos administrar las pilas de Docker, los contenedores, imágenes, volúmenes y redes. Es compatible con **Docker CE, Docker EE y Docker Swarm**.

Para la instalación de Portainer se ha seguido este [enlace](#).

Implementar Portainer para administrar un clúster es fácil. Podemos implementar directamente Portainer como **un servicio en el clúster de Docker**. Tengamos en cuenta que este método desplegará automáticamente una única instancia en el servidor Portainer y desplegará el agente Portainer como un servicio global en cada nodo de su clúster.

```
root@swarm1:~/portainer# curl -L https://downloads.portainer.io/portainer-agent-stack.yml -o portainer-agent-stack.yml
% Total    % Received % Xferd  Average Speed   Time   Time     Time Current
          Dload  Upload   Total Spent  Left Speed
100  754 100  754    0     0  294      0  0:00:02  0:00:02    --:-- 294
```

Ahora lo siguiente que haremos será lanzar el stack de Portainer.

```
root@swarm1:~/portainer# docker stack deploy -c portainer-agent-stack.yml portainer
Creating network portainer_agent_network
Creating service portainer_portainer
Creating service portainer_agent
```

Comprobaremos que se han creado **dos contenedores en el nodo manager**, uno es el **agente de Portainer** y otro es el contenedor que tiene el **servidor de Portainer**. Además que en los **nodos worker** se ha creado el **contenedor agente de Portainer**.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
NAMES					
238063ef592	portainer/portainer:latest	"/portainer -H tcp://.."	53 seconds ago	Up 52 seconds	9000/tcp
portainer_portainer.1.fgjwu3j3y9azxbabwbqq9w5bk					
2d1463dbe74b	portainer/agent:latest	"/.agent"	About a minute ago	Up 59 seconds	
portainer_agent.jj0qh4kfqvLzodi60iru7x0y3.252af9n3mcxzjh0yjkm3kf27c					

Podremos acceder al panel web mediante un navegador la siguiente URL, se ha de decir que el puerto por defecto de Portainer es el **9000**.

http://IP_DOCKER:9000

Al acceder vía web por primera vez nos pedirá **configurar la contraseña del usuario administrador**, tal y como se muestra en la captura.

Please create the initial administrator user.

Username: admin

Password:*

Confirm password:* | ✓

✓ The password must be at least 8 characters long

Create user

La **contraseña** debe tener, por lo menos **ocho caracteres**.

5.6.3. Exploración de la herramienta.

Una vez creado el usuario administrador con una **contraseña válida** para la política de password, pasaremos a la siguiente ventana donde ya podremos ver la interfaz de Portainer propiamente dicha.

portainer.io

Home

SETTINGS

Extensions

Users

Endpoints

Registries

Settings

Endpoints

primary 2020-05-17 20:01:29

1 stack 2 services 8 containers - 4 4 / 0 0 0 0 1 volume

15 images

No tags

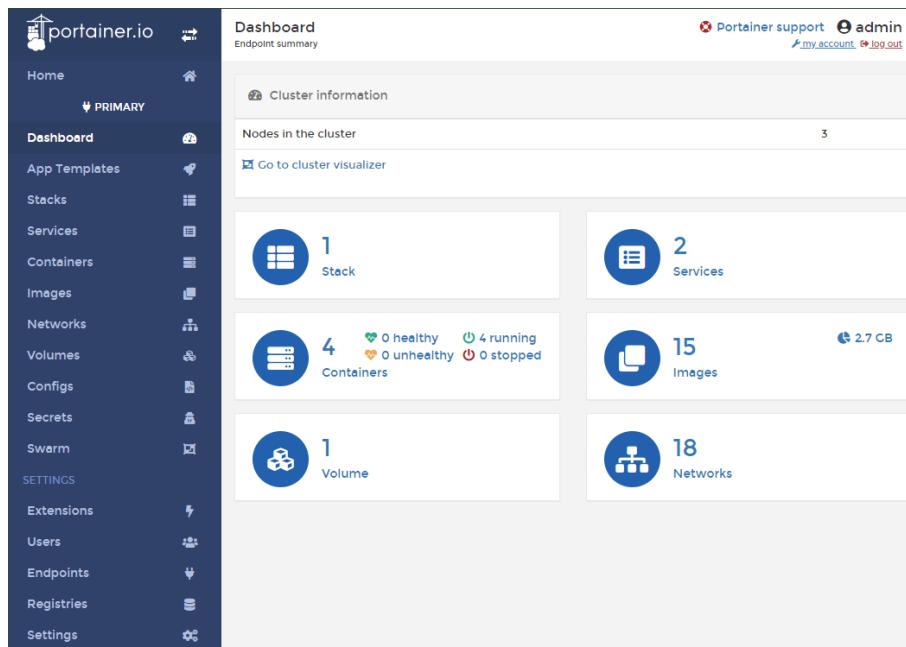
Group: Unassigned

Swarm 19.03.8 + Agent

Items per page 10

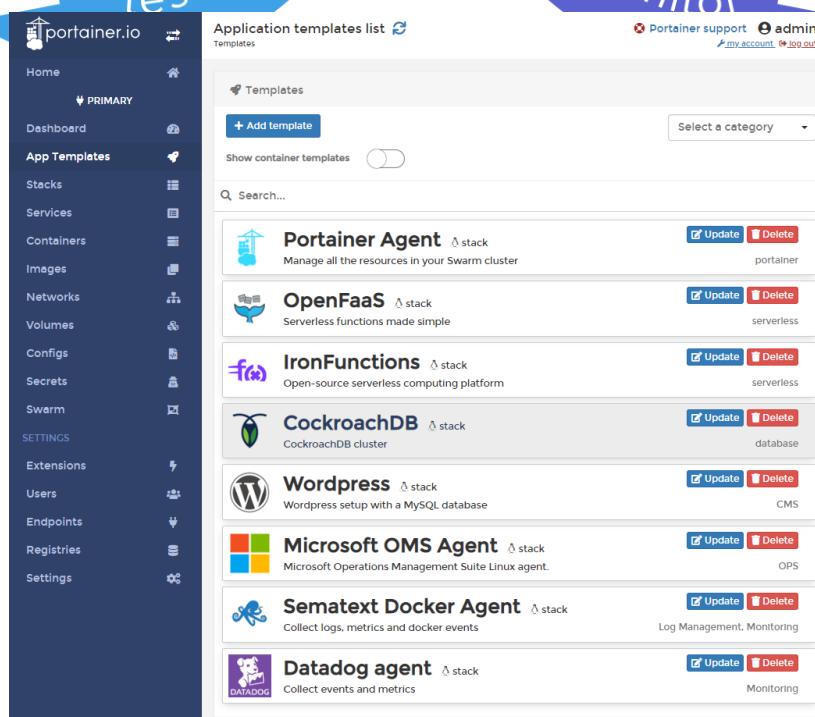
En esta primera ventana nos muestra un resumen muy breve de que hay corriendo o creado en nuestro clúster docker, así como stack, servicios, contendores, volúmenes e imágenes.

Si pinchamos sobre el nodo primario se nos mostrará el siguiente panel de control:



En esta ventana se muestra un resumen de manera visual de nuestro sistema de Docker: número de stack, servicios, contenedores, imágenes volúmenes o redes. **Esta sección es de gran utilidad para mostrarnos de manera global cual es el estado actual de nuestro clúster.**

En la pestaña están ubicadas las plantillas de stack y contenedores que trae por defecto Portainer. En esta ventana se pueden añadir nuevas plantillas tanto de contenedores como de stack.



En la pestaña anterior veímos los stack que trae por defecto, para poder ver los **contenedores** que trae por defecto tendremos que pinchar en esta opción.

En esta pestaña podremos ver los stack que están activos en nuestro clúster.

Name	Type	Control	Ownership
portainer	Swarm	Limited	administrators

Además podremos añadir nuevos stacks pinchando en el botón **Add stack**. En esta opción podremos definir nuestro stack definiéndolo **directamente desde Portainer**.

O por otra podemos descargarnos directamente un **docker-compose** de un repositorio github que hayamos definido.

Create stack

Name: e.g. mystack

This stack will be deployed using the equivalent of the `docker stack deploy` command.

Build method:

- Web editor
- Upload
- git Repository

Git repository

You can use the URL of a git repository.

Repository URL: <https://github.com/portainer/portainer-compose>

Specify a reference of the repository using the following syntax: branches with `refs/heads/branch_name` or tags with `refs/tags/tag_name`. If not specified, will use the default `HEAD` reference normally the `master` branch.

Repository reference: refs/heads/master

Indicate the path to the Compose file from the root of your repository.

Compose path: docker-compose.yml

Authentication:

Environment

Environment variables:

Access control:

Enable access control:

Administrators: I want to restrict the management of this resource to administrators only

Restricted: I want to restrict the management of this resource to a set of users and/or teams

Actions: Deploy the stack

Además de poder añadir stack personalizados por nosotros, pinchando en el stack podremos ver en detalle los servicios y las réplicas que tiene cada servicio.

Stack details

Stack: portainer

Information

This stack was created outside of Portainer. Control over this stack is limited.

Stack details

portainer

Services

<input type="button" value="Update"/> <input type="button" value="Remove"/>	Name	Image	Scheduling Mode	Published Ports	Last Update
<input type="checkbox"/>	portainer_agent	portainer/agent:latest	global 3 / 3	-	2020-05-17 18:54:13
<input type="checkbox"/>	portainer_portainer	portainer/portainer:latest	replicated 1 / 1	Scale 8000:8000 9000:9000	2020-05-17 18:54:10

Items per page: 10

5.6.3.3. Services

En esta pestaña podremos ver los **servicios activos** en nuestro clúster, aquí se podrá escalar las réplicas que tendrá cada servicio.

Name	Stack	Image	Scheduling Mode	Published Ports	Last Updated
portainer_agent	portainer	portainer/agent:latest	global 3 / 3	-	2020-05
portainer_portainer	portainer	portainer/portainer:latest	replicated 1 / 1 Scale	8000:8000 9000:9000	2020-05

5.6.3.4. Container

Podremos ver los contenedores que tenemos en docker además podremos iniciarlos, páralos, eliminarlos y muchas opciones más. Además podremos añadir un contenedor.

Name	State	Quick actions	Stack	Image	Created	IP Address	Host	Published Ports	Ownership
portainer_portainer.1.mv98lsu...	running		portainer	portainer/portainer:latest	2020-05-01 20:56:57	10.0.0.9	HLCserver	-	administrators
portainer_agent.lvf5nvn1uk7h...	running		portainer	portainer/agent:latest	2020-03-03 20:26:40	10.0.2.3	HLCserver	-	administrators
WordPress	running		-	wordpress:latest	2020-05-02 22:05:31	192.168.1.3	HLCserver	8080:80	administrators
mysql	running		-	mysql:5.7	2020-03-02 19:55:09	192.168.1.2	HLCserver	3306:3306	administrators

Además podremos hacer más opciones con los contenedores como ver los logs, hacer un inspect, ver los recursos que consumen, y entrar a la terminal.

Name	State	Quick actions	Stack	Image	Created	IP Address	Host	Published Ports	Ownership
WordPress	running		-	wordpress:latest	2020-05-02 22:03:31	192.168.1.3	HLCserver	8080:80	administrators

Logs

Podremos ver los logs que ha tenido ese contenedor podemos filtrar por días, líneas etc.

```

rome@80-0-3987-102:~$ tail -f /var/log/nginx/error.log
192.168.13.27 - - [03/Jan/2020:20:33:09 +0000] "GET /wp-includes/js/wp-auth-check.min.js?v=ver=5.3.2 HTTP/1.1" 200 1122 "http://192.168.15.17:8080/wp-admin/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.122 Safari/537.36"
192.168.13.27 - - [03/Jan/2020:20:33:09 +0000] "GET /wp-includes/js/swipe.min.js?v=ver=5.3.2 HTTP/1.1" 200 4196 "http://192.168.15.17:8080/wp-admin/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.122 Safari/537.36"
192.168.13.27 - - [03/Jan/2020:20:33:09 +0000] "GET /wp-includes/js/jquery/ui/position.min.js?v=ver=1.11.4 HTTP/1.1" 200 2858 "http://192.168.15.17:8080/wp-admin/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.122 Safari/537.36"
192.168.13.27 - - [03/Jan/2020:20:33:09 +0000] "GET /wp-includes/js/jquery/ui/menu.min.js?v=ver=1.11.4 HTTP/1.1" 200 3169 "http://192.168.15.17:8080/wp-admin/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.122 Safari/537.36"
192.168.13.27 - - [03/Jan/2020:20:33:09 +0000] "GET /wp-includes/js/jquery/ui/autocomplete.min.js?v=ver=1.11.4 HTTP/1.1" 200 3131 "http://192.168.15.17:8080/wp-admin/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.122 Safari/537.36"
192.168.13.27 - - [03/Jan/2020:20:33:09 +0000] "GET /wp-admin/load-styles.php?c=8&d=rtrLoenN5chun_0NS0dashicons,admin-bar,common,forms,admin-menu,dashboard,list-tables,edit-revisions,media,themes,about,new-menus,wpt-pointer,widgets&lreak8kRun_INSPN-site-icon,10m_buttons,wp-admin-checker=v3.3.2 HTTP/1.1" 200 8667 "http://192.168.15.17:8080/wp-admin/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.122 Safari/537.36"
192.168.13.27 - - [03/Jan/2020:20:33:09 +0000] "GET /wp-includes/js/chickenbox/loadingNotification.min.js?v=ver=1.1.1 HTTP/1.1" 200 1550 "http://192.168.15.17:8080/wp-admin/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.122 Safari/537.36"
192.168.13.27 - - [03/Jan/2020:20:33:09 +0000] "GET /favicon.ico HTTP/1.1" 200 180 "http://192.168.15.17:8080/wp-admin/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.122 Safari/537.36"
192.168.13.27 - - [03/Jan/2020:20:33:09 +0000] "POST /wp-admin/admin-ajax.php HTTP/1.1" 200 180 "http://192.168.15.17:8080/wp-admin/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.122 Safari/537.36"
192.168.13.27 - - [03/Jan/2020:20:33:09 +0000] "POST /wp-admin/admin-ajax.php?action=dashboard_widgets&widget=dashboard_primary&genous=dashboard HTTP/1.1" 200 900 "http://192.168.15.17:8080/wp-admin/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.122 Safari/537.36"
192.168.13.27 - - [03/Jan/2020:20:34:09 +0000] "GET /wp-admin/admin-ajax.php HTTP/1.1" 200 512 "http://192.168.15.17:8080/wp-admin/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.122 Safari/537.36"
192.168.13.27 - - [03/Jan/2020:20:36:09 +0000] "POST /wp-admin/admin-ajax.php HTTP/1.1" 200 512 "http://192.168.15.17:8080/wp-admin/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.122 Safari/537.36"

```

Inspect

En esta opción podremos ver con detalle la configuración del contenedor.

```

Container inspect
Containers > WordPress > Inspect

Inspect

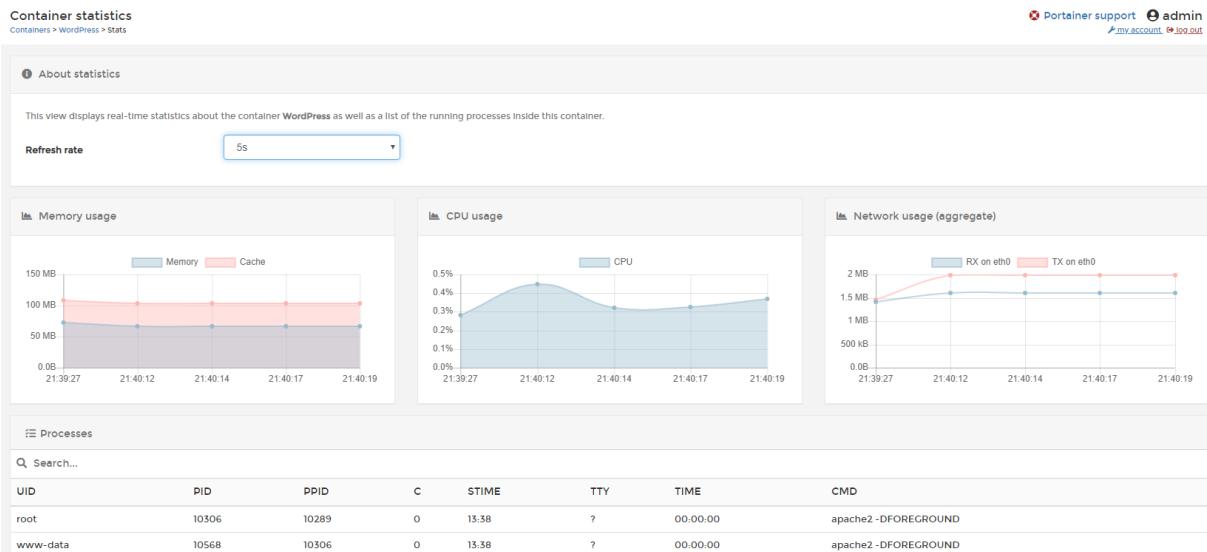
d46fce9820321950e5bf6d04763celff8246250a55436ddac8eb63f6a5a89f9:
  AppArmorProfile:
  Args: [ apache2-foreground ]
  Config: { AttachStderr: false, AttachStdin: false, AttachStdout: false, Cmd: apache2-foreground, Domainname: , Entrypoint: Created: 2020-03-02T21:03:31.711465051Z
  Driver: overlay2
  ExecIDs:
  GraphDriver: { Data: [object Object], Name: overlay2 }
  HostConfig: { AutoRemove: false, Binds: /mysql/var/www/html:/rw, BikioDeviceReadBps: null, BikioDeviceReadIops: null, Bl HostnamePath: /var/lib/docker/containers/d46fce9820321950e5bf6d04763celff8246250a55436ddac8eb63f6a5a89f9/hostname
  HostPath: /var/lib/docker/containers/d46fce9820321950e5bf6d04763celff8246250a55436ddac8eb63f6a5a89f9/hosts
  Id: d46fce9820321950e5bf6d04763celff8246250a55436ddac8eb63f6a5a89f9
  Image: sha256:126aa0dec0cd46076be62251la8bb2a4b08550d703a183052923466c5e6c8a
  LogPath: /var/lib/docker/containers/d46fce9820321950e5bf6d04763celff8246250a35436ddac8eb63f6a5a89f9/d46fce9820321950e5bf6d04763celff8246250a35436ddac8eb63f6a5a89f9.json.log
  MountLabel:
  Mounts: [ [object Object] ]
  Name: /WordPress
  NetworkSettings: { Bridge: , EndpointID: dtfb27df41ba48b55b8ca3a8f4ead13e86e8c2372d253a95579c41f5b0b6c, Gateway: 192.1
  Path: docker-entropy-point.sh
  Platform: linux
  ProcessLabel:
  ResolvConfPath: /var/lib/docker/containers/d46fce9820321950e5bf6d04763celff8246250a35436ddac8eb63f6a5a89f9/resolv.conf
  RestartCount: 0
  State: { Dead: false, Error: , ExitCode: 0, FinishedAt: 0001-01-01T00:00:00Z, OOMKilled: false, Paused: false, Pid: 10306,

```

Statistics

Podremos ver información como la memoria usada por ese contenedor, CPU, y red, además de que usuarios están haciendo uso de ese contenedor.

También podremos poner un tiempo de refresco para que se actualice las estadísticas.



Terminal

En esta opción podremos acceder por terminal a la máquina. Además podremos elegir el comando y el usuario.

Container console
Containers > WordPress > Console

code

Execute

Command: /bin/bash

Use custom command:

User: root

Connect

Container console
Containers > WordPress > Console

deploy

Execute

Exec into container as default user using command bash **Disconnect**

```
root@d46fce98203:/var/www/html#
```

root

5.6.3.5. Images

Podremos ver las imágenes que tenemos en nuestro clúster incluyendo en cada nodo del clúster, además podremos exportar, importar imágenes y descargarnos imágenes de dockerHub.

Image ID	Tag	Size	Created	Host
sha256:d4e60c8eb27a1e2a9370e6d0120008...	Unused httpd:latest	165.5 MB	2020-05-15 21:13:45	swarm3
sha256:b2c2ab6dcf2e526597d0a5fc506f12...	Unused httpd:latest	165.5 MB	2020-04-23 05:07:33	swarm1
sha256:252a97fead692b67b562ea6064e4ea...	Unused joseadiaz/redis:<none>	104.1 MB	2020-05-13 12:54:31	swarm2
sha256:252a97fead692b67b562ea6064e4ea...	Unused joseadiaz/redis:<none>	104.1 MB	2020-05-13 12:54:31	swarm3
sha256:2d6778546de0717cb82b3a0d8e581c...	Unused joseadiaz/redis:latest	104.1 MB	2020-05-16 15:04:31	swarm1
sha256:ec4d20b76f64b7df1ade1852db458b...	Unused joseadiaz/web-php:<none>	405.6 MB	2020-05-13 13:35:11	swarm2
sha256:ec4d20b76f64b7df1ade1852db458b...	Unused joseadiaz/web-php:<none>	405.6 MB	2020-05-13 13:35:11	swarm3
sha256:ec4d20b76f64b7df1ade1852db458b...	Unused joseadiaz/web-php:<none>	405.6 MB	2020-05-13 13:35:11	swarm1
sha256:c5291fce73b16d6ef799d69456643a...	Unused joseadiaz/web-php:latest	405.6 MB	2020-05-16 15:01:57	swarm1
sha256:b84d68d0a7db79194091fae58b71af...	mysql:<none>	448.1 MB	2020-05-15 22:11:03	swarm2

5.6.3.6. Network

Tendremos listadas las distintas redes de nuestros contenedores, pero además de **listar** también podremos **crear y borrar** redes a nuestro gusto.

Cuando creamos una nueva red le podremos decir en qué nodo del clúster queremos que se cree dicha red.

Name	Stack	Scope	Driver	Attachable	Internal	IPAM Driver	IPAM Subnet	IPAM Gateway	Host	
none	System	-	local	null	false	false	default	-	-	swarm
none	System	-	local	null	false	false	default	-	-	swarm
none	System	-	local	null	false	false	default	-	-	swarm
portainer_agent_network	portainer	swarm	overlay	true	false	false	default	10.0.12.0/24	10.0.12.1	swarm
Wordpress_default	Wordpress	swarm	overlay	false	false	false	default	10.0.13.0/24	10.0.13.1	swarm

5.6.3.7. Volumes

En esta pestaña podremos crear, borrar o listar los distintos volúmenes de los distintos nodos de nuestro clúster.

Como con la creación de redes podremos definir para que nodo queremos crear un volumen.

Name	Stack	Driver	Mount point
13e74402ff45387972d02f87a5084052d0703...	-	local	/var/lib/docker/volumes/1[...]945985d
545fcfc28ddf5dfa3bacdd616d112ba46678d...	-	local	/var/lib/docker/volumes/5[...]00474f0
dfe7306c084554268ef957c0b893e09041c76...	-	local	/var/lib/docker/volumes/d[...]f12e140f
portainer_portainer_data	portainer	local	/var/lib/docker/volumes/p[...]iner_po
Wordpress_db_data	Wordpress	local	/var/lib/docker/volumes/Wordpress_

En esta pestaña nos encontraremos con la **información del clúster**, cuantos nodos tiene en que nodo esta cada tarea (contenedor), el total de CPU, memoria, versión API de docker.

Name	Role	CPU	Memory	Engine	IP Address	Status	Availability
swarm1	manager	1	1 GB	19.03.8	192.168.15.16	ready	active
swarm2	worker	1	1 GB	19.03.8	192.168.15.17	ready	active
swarm3	worker	1	1 GB	19.03.8	192.168.15.18	ready	active

Si pinchamos en alguno de los nodos del clúster tendremos las opciones de pausarlo, sacarlo del clúster y activarlo, además de detalles como de dicho nodo.

The screenshot shows the Docker Swarm UI interface. On the left is a sidebar with various options: Dashboard, App Templates, Stacks, Services, Containers, Images, Networks, Volumes, Configs, Secrets, Swarm, Extensions, Users, Endpoints, Registries, and Settings. The 'Swarm' option is selected. The main area is titled 'Host Details' and contains the following information:

Hostname	swarm1
OS Information	linux x86_64
Total CPU	1
Total memory	1 GB

Below this is the 'Engine Details' section:

Version	19.03.8
Volume Plugins	local
Network Plugins	bridge, host, ipvlan, macvlan, null, overlay

Finally, the 'Node Details' section shows the node's role as 'manager (192.168.15.16:2377)'. It includes a dropdown menu for 'Availability' with the following options: Active (selected), Active, Pause, and Drain. A button labeled 'Apply changes' is at the bottom.

Y para mí la opción más interesante de esta pestaña sería **Go to cluster visualizer**.

The screenshot shows the 'Cluster status' section of the Docker Swarm UI. It displays the following metrics:

Nodes	3
Docker API version	1.40
Total CPU	3
Total memory	3.08 GB

At the bottom of this section is a link labeled 'Go to cluster visualizer', which is highlighted with a red box.

En esta opción podemos ver de forma gráfica y casi a tiempo real los contenedores que están en cada nodo, además de su estado, memoria.

Cluster information Hide

Nodes	3
Services	4
Tasks	11

Options

Only display running tasks

Display node labels

[Refresh](#)

Rate ▾

Cluster visualizer

swarm1 ⌚

manager
CPU: 1
Memory: 1.03 GB
ready

portainer_agent Image: portainer/agent:latest Status: running Update: 2020-05-17 18:54:44
portainer_portainer Image: portainer/portainer:latest Status: running Update: 2020-05-17 20:01:35
portainer_portainer Image: portainer/portainer:latest Status: failed Update: 2020-05-17 19:46:06

swarm2 ⌚

worker
CPU: 1
Memory: 1.03 GB
ready

portainer_agent Image: portainer/agent:latest Status: running Update: 2020-05-17 18:54:52
Wordpress_db Image: mysql:5.7 Status: running Update: 2020-05-17 22:05:29
Wordpress_wordpress Image: wordpress:latest Status: running Update: 2020-05-17 22:21:24

swarm3 ⌚

worker
CPU: 1
Memory: 1.03 GB
ready

portainer_agent Image: portainer/agent:latest Status: running Update: 2020-05-17 18:55:10
Wordpress_wordpress Image: wordpress:latest Status: running Update: 2020-05-17 22:03:46

5.6.4. Creación de contenedor a golpe de click

Para poder crear un contenedor de una forma rápida y fácil deberemos de irnos a la pestaña de **App templates**, donde buscaremos una plantilla de un contenedor que va a ser utilizada para crear uno nuevo.

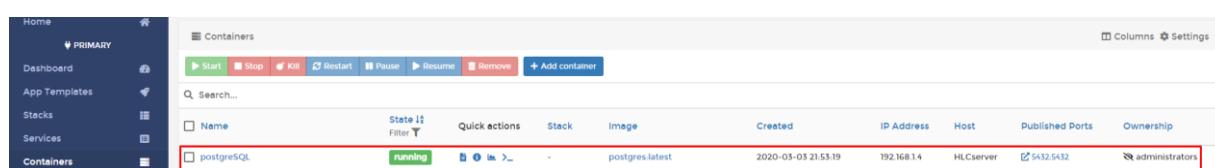
Pincharemos en la plantilla que más nos convenga, es este caso se va a utilizar **PostgreSQL**.



Una vez dentro podremos definir el nombre del contenedor, la red a la que va a pertenecer.

Mapeo de puertos, el volumen que va a ser creado y de que ruta de carpetas, además también podremos poner el hostname del contenedor. Una vez lista la configuración pincharemos en **Deploy the container**.

Una vez hecho esto iremos a la pestaña de Contenedores y comprobaremos que se ha creado el contenedor.



5.6.5. Creación de stack

En este apartado se creará un **stack de wordpress** para ello nos iremos a la opción **App Templates** a la sección de stack donde podremos ver la plantilla de wordpress.

The screenshot shows the Portainer interface with the sidebar menu open. The 'App Templates' option is selected and highlighted with a red box. In the main content area, there is a list of available templates. One template, 'Wordpress', is also highlighted with a red box. Other templates listed include 'Portainer Agent', 'OpenFaaS', 'IronFunctions', 'CockroachDB', and 'Microsoft OMS Agent'. Each template entry includes a 'Update' and a 'Delete' button.

Aquí lo que podremos hacer es definir el **nombre de nuestro stack** además de definir la **contraseña de la base de datos**.

This screenshot shows the configuration page for the 'Wordpress' application template. The left sidebar shows the 'App Templates' section selected. The main panel displays the 'Wordpress' template details. Under the 'Information' section, it says 'Deploys a Wordpress instance connected to a MySQL database.' In the 'Configuration' section, the 'Name' field is set to 'Wordpress' and the 'Database root password' field is set to 'bolson'. Below these fields is an 'Access control' section with a toggle switch for 'Enable access control'. There are two options: 'Administrators' (selected) which says 'I want to restrict the management of this resource to administrators only', and 'Restricted' which says 'I want to restrict the management of this resource to a set of users and/or teams'. At the bottom of the panel is a 'Deploy the stack' button.

Una vez creado el stack podremos ver que se ha creado en la pestaña **stacks**.

This screenshot shows the 'Stacks' section of the Portainer interface. The left sidebar has the 'Stacks' option selected and highlighted with a red box. The main table lists existing stacks. The first stack, 'portainer', has a 'Control' level of 'Limited'. The second stack, 'Wordpress', has a 'Control' level of 'Total'. Both rows have a red box around them. The table includes columns for Name, Type, Control, and Ownership. At the top of the table, there are buttons for 'Remove' and '+ Add stack'. A search bar is also present above the table.

Si seleccionamos sobre **Wordpress** o nos vamos a la pestaña **servicios** podremos ver los servicios que tiene a cargo, que son una base de datos y un contenedor wordpress que tendrá un servidor web y php.

Además podremos saber en qué nodo del clúster esta cada contenedor de cada servicio.

The screenshot shows the Docker Swarm UI interface. On the left, a sidebar lists various management options like Home, Dashboard, App Templates, Stacks, Services, Containers, Images, Networks, Volumes, Configs, Secrets, Swarm, and Settings. The 'Stacks' section is currently selected. The main area is titled 'Stack details' for the 'Wordpress' stack. It includes a 'Delete this stack' button and a 'Stack duplication / migration' section with fields for 'Stack name (optional for migration)' and 'Select an endpoint'. Below these are 'Migrate' and 'Duplicate' buttons. A large table titled 'Services' lists the components of the Wordpress stack. The table has columns for Name, Image, Scheduling Mode, Published Ports, and Last Update. Two entries are shown: 'Wordpress_db' using 'mysql:5.7' and 'replicated' mode, and 'Wordpress_wordpress' using 'wordpress:latest' and also 'replicated' mode. Both entries show they are 'running' on specific tasks. At the bottom right of the table, there's a dropdown for 'Items per page' set to 10.

Name	Image	Scheduling Mode	Published Ports	Last Update
Wordpress_db	mysql:5.7	replicated 1 / 1 Scale	-	2020-05-17 21:59:35
Wordpress_wordpress	wordpress:latest	replicated 1 / 1 Scale	30000:80	2020-05-17 21:59:32

Desde la pestaña **stacks** dentro de wordpress podremos ver la plantilla con la que ha sido creado este stack.

```

version: '3'
services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_DATABASE_PASSWORD}
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
  wordpress:
    image: wordpress:latest
    ports:
      - 80
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
volumes:
  db_data:

```

Como comprobamos la plantilla con la que ha sido creado este stack es bastante similar a la que nosotros hemos hecho en apartados anteriores con redis y el servidor web.

Portainer trabaja con archivos yml como también podemos hacerlo nosotros de forma manual.

Para poder visitar nuestro wordpress deberemos de ingresar la siguiente URL en nuestro navegador



5.6.5.1. Escalabilidad de servicios

Para poder escalar un servicio tendremos que ir a la pestaña de **Services**, donde comprobaremos que tenemos una opción donde pone **replicated**.

Name	Stack	Image	Scheduling Mode	Published Ports	Last Update
portainer_agent	portainer	portainer/agent:latest	global 3 / 3	-	2020-05-19
portainer_portainer	portainer	portainer/portainer:latest	replicated 1 / 1 Scale	8000-8000 9000-9000	2020-05-19
Wordpress_db	Wordpress	mysql:5.7	replicated 1 / 1 Scale	-	2020-05-19
Wordpress_wordpress	Wordpress	wordpress:latest	replicated 1 / 1 Scale	30000:80	2020-05-19

En esta opción podremos escalar nuestro servicio cuanto queramos.

Name	Stack	Image	Scheduling Mode	Published Ports	Last Update
portainer_agent	portainer	portainer/agent:latest	global 3 / 3	-	2020-05-19
portainer_portainer	portainer	portainer/portainer:latest	replicated 1 / 1 Scale	8000-8000 9000-9000	2020-05-19
Wordpress_db	Wordpress	mysql:5.7	replicated 1 / 1 Scale	-	2020-05-19
Wordpress_wordpress	Wordpress	wordpress:latest	replicated 1 / 2 Scale	30000:80	2020-05-19

Tras unos segundos podremos comprobar que sea creado un nuevo contenedor en el nodo **Swarm2** para este servicio.

Name	Stack	Image	Scheduling Mode	Published Ports	Last Update
portainer_agent	portainer	portainer/agent:latest	global 3 / 3	-	2020-05-19
portainer_portainer	portainer	portainer/portainer:latest	replicated 1 / 1 Scale	8000-8000 9000-9000	2020-05-19
Wordpress_db	Wordpress	mysql:5.7	replicated 1 / 1 Scale	-	2020-05-19
Wordpress_wordpress	Wordpress	wordpress:latest	replicated 1 / 2 Scale	30000:80	2020-05-19

Status	Filter	Task	Actions	Slot	Node	Last Updated
preparing		0s3kv5dqbpsdmf9upprxr4kc		2	swarm2	2020-05-19
running		dq2t74mfb77ok818fnngtg2pz8		1	swarm3	2020-05-19

Name	Stack	Image	Scheduling Mode	Published Ports	Last Update
portainer_agent	portainer	portainer/agent:latest	global 3 / 3	-	2020-05-19
portainer_portainer	portainer	portainer/portainer:latest	replicated 1 / 1 Scale	8000-8000 9000-9000	2020-05-19
Wordpress_db	Wordpress	mysql:5.7	replicated 1 / 1 Scale	-	2020-05-19
Wordpress_wordpress	Wordpress	wordpress:latest	replicated 2 / 2 Scale	30000:80	2020-05-19

Status	Filter	Task	Actions	Slot	Node	Last Updated
running		0s3kv5dqbpsdmf9upprxr4kc		2	swarm2	2020-05-19
running		dq2t74mfb77ok818fnngtg2pz8		1	swarm3	2020-05-19

6. Kubernetes



kubernetes

6.1. Componentes a utilizar.

Previamente levantado el clúster y habiendo relacionado nuestro comando kubectl a nuestro nodos, procederemos a la implementación de nuestra infraestructura.

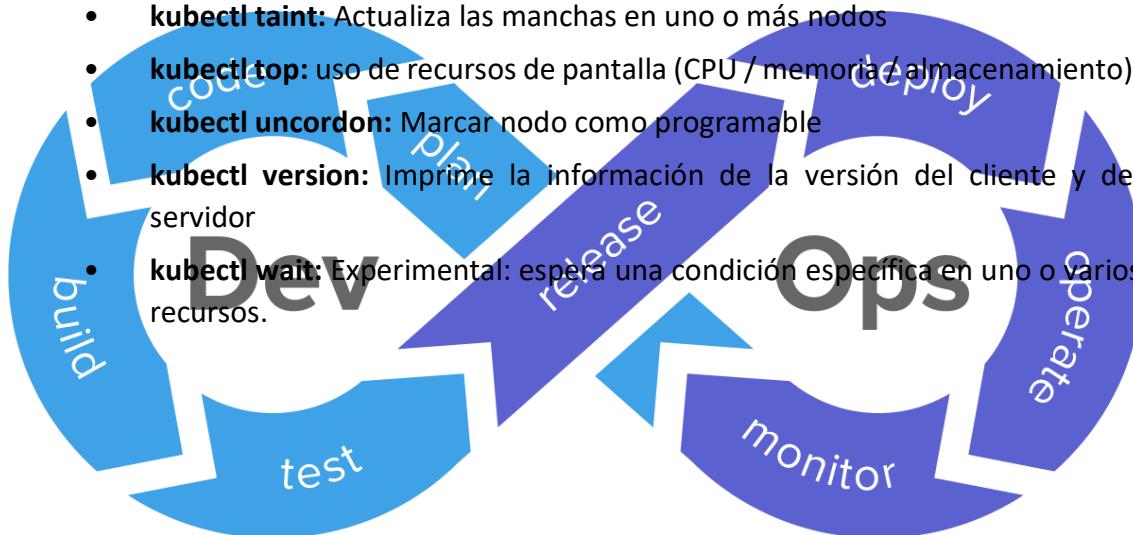
Para implementar nuestra infraestructura lo haremos creando diferentes **pods**, **deployments** y **services**, estos los generaremos mediante ficheros de tipo yaml con el comando kubectl.

- **Pod** es la unidad fundamental de despliegue en Kubernetes. 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.
- **Replicaset** asegura que se esté ejecutando un **número específico de réplicas de pod** en un momento dado. Sin embargo, una implementación es un concepto de nivel superior que administra ReplicaSets y proporciona actualizaciones declarativas a Pods junto con muchas otras características útiles.
- **Deployment** añade la **capacidad de poder actualizar la aplicación definida** en, por ejemplo, un pod sin pérdida de servicio, mediante actualización continua. Es decir, **si definimos un deployment generará un pod**(o más según lo especificado) y en caso de que caiga por cualquier motivo lo volverá a levantar. En caso de definir solo el pod, al caerse no se levantará.
- **Service**, según la definición que nos da Kubernetes, es una forma abstracta de **exponer una aplicación que se ejecuta en un conjunto de pods como un servicio de red**. Definiremos un service para poder **exponer el pod dentro y/o fuera de nuestro clúster**. Puede ser de tres tipos
 - **ClusterIP**: expone el servicio en una IP interna de clúster. La elección de este valor hace que el servicio solo sea accesible desde dentro del clúster.
 - **NodePort**: Expone el servicio en la IP de cada Nodo en un puerto estático (el NodePort). Se crea automáticamente un servicio ClusterIP, al que se enrutarán el servicio NodePort. Podrá contactar al servicio NodePort, desde fuera del clúster.
 - **LoadBalancer**: expone el servicio externamente utilizando el equilibrador de carga de un proveedor en la nube. Estos serán los diferentes recursos que usaremos para definir nuestra estructura en Kubernetes.

6.2. Comandos básicos.

- **kubectl annotate:** Actualiza las anotaciones en un recurso
- **kubectl api-resources:** Imprime los recursos API admitidos en el servidor
- **kubectl api-versions:** Imprima las versiones de API compatibles en el servidor, en forma de "grupo / versión"
- **kubectl apply :** aplica una configuración a un recurso por nombre de archivo o stdin
- **kubectl attach:** Adjuntar a un contenedor en ejecución
- **kubectl auth:** Inspeccionar autorización
- **kubectl autoscale:** Escala automáticamente un despliegue, ReplicaSet o ReplicationController
- **kubectl certificate:** Modifique los recursos del certificado.
- **kubectl cluster-info:** Muestra información del clúster
- **kubectl completion:** Código de finalización del shell de salida para el shell especificado (bash o zsh)
- **kubectl config:** Modificar archivos kubeconfig
- **kubectl convert:** Convierte archivos de configuración entre diferentes versiones de API
- **kubectl cordon:** Marcar nodo como no programable
- **kubectl cp:** Copia archivos y directorios desde y hacia contenedores.
- **kubectl create:** Crea un recurso desde un archivo o desde stdin.
- **kubectl delete:** Elimina recursos por nombres de archivo, stdin, recursos y nombres, o por recursos y selector de etiquetas
- **kubectl describe:** Muestra detalles de un recurso específico o grupo de recursos
- **kubectl diff :** Diferir la versión en vivo contra la posible versión aplicada
- **kubectl drain:** Nodo de drenaje en preparación para el mantenimiento
- **kubectl edit :** Edita un recurso en el servidor
- **kubectl exec:** Ejecuta un comando en un contenedor
- **kubectl explain:** Documentación de recursos
- **kubectl expose:** Tome un controlador de replicación, servicio, implementación o pod y exponga como un nuevo servicio Kubernetes
- **kubectl get :** Muestra uno o varios recursos
- **kubectl kustomize:** Construye un objetivo de personalización desde un directorio o una url remota.
- **kubectl label:** Actualiza las etiquetas en un recurso
- **kubectl logs:** Imprime los registros de un contenedor en un pod

- **kubectl options:** Imprime la lista de banderas heredadas por todos los comandos
- **kubectl patch:** Actualiza los campos de un recurso usando parche de fusión estratégica
- **kubectl plugin:** Proporciona utilidades para interactuar con complementos.
- **kubectl port-forward :** Reenvía uno o más puertos locales a un pod
- **kubectl proxy:** Ejecute un proxy para el servidor API de Kubernetes
- **kubectl replace:** Reemplaza un recurso por nombre de archivo o stdin
- **kubectl rollout:** Administre el lanzamiento de un recurso
- **kubectl run:** Ejecuta una imagen en particular en el clúster
- **kubectl scale:** Establezca un nuevo tamaño para un Controlador de implementación, ReplicaSet o Replication
- **kubectl set:** Establece características específicas en objetos
- **kubectl taint:** Actualiza las manchas en uno o más nodos
- **kubectl top:** uso de recursos de pantalla (CPU / memoria / almacenamiento).
- **kubectl uncordon:** Marcar nodo como programable
- **kubectl version:** Imprime la información de la versión del cliente y del servidor
- **kubectl wait:** Experimental: espera una condición específica en uno o varios recursos.



6.3. Instalación de kubernetes en local

6.3.1. Escenario a montar en kubernetes

En nuestro caso, vamos a montar un clúster de kubernetes con kubeadm. Hemos elegido este método, por su fácil instalación, que se llevará acabo en las siguientes máquinas:

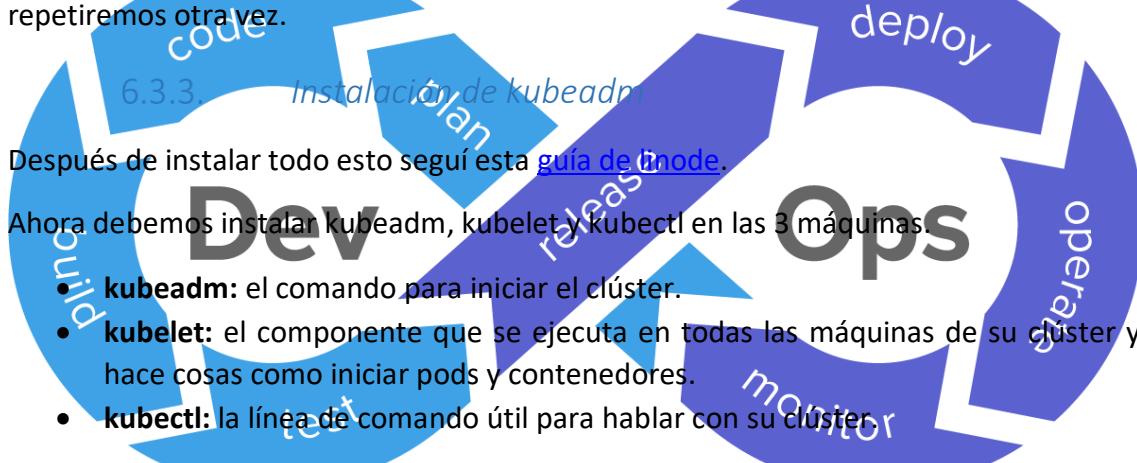
- **Jupiter.** Maquina Debian de 2GB de RAM. Sera el Master.
- **Saturno.** Maquina Debian de 2GB de RAM. Worker.
- **Urano.** Maquina Debian de 2GB de RAM. Worker.

La instalación se llevará acabo de la siguiente manera:

6.3.2. Instalación de Docker

Lo primero que haremos será instalar Docker en nuestras tres máquinas, al ser las tres máquinas Debian, la instalación puede ser diferente a otras distribuciones.

Como en este documento ya se ha descrito los pasos para [instalar docker](#) no los repetiremos otra vez.



Para ello, seguimos los pasos que aparecen en la [documentación de kubernetes](#), que se basa en añadir la clave GPG de kubernetes.

```
root@jupiter:~# curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
OK
```

Añadiremos su repositorio.

```
root@jupiter:~# cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
> deb https://apt.kubernetes.io/ kubernetes-xenial main
> EOF
deb https://apt.kubernetes.io/ kubernetes-xenial main
```

Y actualizaremos los repositorios e instalaremos los paquetes necesarios

```
root@jupiter:~# apt-get update
```

```
root@jupiter:~# apt-get install -y kubelet kubeadm kubectl
```

6.3.3.1. Configurar controlador cgroup utilizado por kubelet.

NOTA: cgroups es una característica del kernel de Linux que limita, explica y aísla el uso de recursos (CPU, memoria, E / S de disco, red, etc.) de una colección de procesos.

A continuación agregaremos la configuración de Kubelet para que coincida con controlador de cgroup de docker.

```
root@jupiter:~# apt-mark hold kubelet kubeadm kubectl
kubelet fijado como retenido.
kubeadm fijado como retenido.
kubectl fijado como retenido.
```

Kubelet / Kubernetes desde la versión 1.8 no funciona “**no soporta**” el uso o habilitación de la memoria de intercambio o Swap los equipos”, por ello es necesario desactivar la memoria swap. Desactivaremos la memoria swap.

```
root@jupiter:~# swapoff -a
```

Añadiremos la siguiente configuración al demonio de docker para que el **cgroup** coincida con el sistema que en este caso es **systemd**.

```
root@jupiter:~# cat > /etc/docker/daemon.json <<EOF
> {
>   "exec-opts": ["native.cgroupdriver=systemd"],
>   "log-driver": "json-file",
>   "log-opt": {
>     "max-size": "100m"
>   },
>   "storage-driver": "overlay2"
> }
> EOF
```

Una vez hecho esto deberemos de reiniciar el demonio de docker además de reiniciar el servicio de docker.

```
root@jupiter:~# mkdir -p /etc/systemd/system/docker.service.d
root@jupiter:~# systemctl daemon-reload
root@jupiter:~# systemctl restart docker
```

Además tendremos que añadir estos prerrequisitos para que funcione la creación del nodo master del clúster.

```
root@jupiter:~# modprobe overlay
root@jupiter:~# modprobe br_netfilter
root@jupiter:~# cat > /etc/sysctl.d/99-kubernetes-cri.conf <<EOF
> net.bridge.bridge-nf-call-iptables = 1
> net.ipv4.ip_forward = 1
> net.bridge.bridge-nf-call-ip6tables = 1
> EOF
root@jupiter:~# sysctl --system
* Applying /etc/sysctl.d/99-kubernetes-cri.conf ...
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
* Applying /etc/sysctl.d/99-sysctl.conf ...
* Applying /etc/sysctl.conf ...
```

A continuación, instalar CRI-O como tiempo de ejecución CRI.

NOTA: CRI-O es una implementación de Kubernetes CRI (Container Runtime Interface) para **permitir el uso de tiempos de ejecución compatibles con OCI** (Open Container Initiative). Es una alternativa liviana al uso de Docker como el tiempo de ejecución para kubernetes.

Añadiremos los repositorios para poder instalar cri-o.

```
root@jupiter:~# echo 'deb http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/Debian_10/
/' > /etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list
root@jupiter:~# wget -nv https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable/Debian_10/Release.key -O -
2020-06-10 08:47:24 URL:https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable/Debian_10/Release.key [1094/1094] -> "-" [1]
OK
```

Ejecutaremos el siguiente comando para poder instalar cri-o.

```
root@jupiter:~# apt-get install cri-o-1.17
```

Una vez instalado deberemos de recargar el demonio cri-o además de iniciar el servicio.

```
root@jupiter:~# systemctl daemon-reload
root@jupiter:~# systemctl start crio
```

A continuación deberemos de definir el socket de cri-o además de cgroup para kubernetes.

```
root@jupiter:~# nano /etc/default/kubelet
```

```
KUBELET_EXTRA_ARGS="--cgroup-driver=systemd --container-runtime=remote --container-runtime-endpoint="unix:///var/run/cri$
```

Para utilizar el systemdcontrolador cgroup, establecido **plugins.cri.systemd_cgroup = true** en /etc/containerd/config.toml. Cuando use kubeadm, configure manualmente el controlador cgroup para kubelet

```
root@jupiter:~# nano /etc/containerd/config.toml
```

```
plugins.cri.systemd_cgroup = true
```

Al usar Docker, kubeadm detectará automáticamente el controlador cgroup para el kubelet y lo configurará en el **/var/lib/kubelet/config.yaml** archivo durante el tiempo de ejecución.

Si está utilizando un CRI diferente, debe modificar el archivo con su cgroupDriver valor, así:

```
mkdir /var/lib/kubelet  
nano /var/lib/kubelet/config.yaml
```

```
apiVersion: kubelet.config.k8s.io/v1beta1  
kind: KubeletConfiguration  
cgroupDriver: systemd
```

Una vez hecho estos pasos deberemos de recargar el demonio además de reiniciar kubelet.

```
root@jupiter:~# systemctl daemon-reload  
root@jupiter:~# systemctl restart kubelet  
root@jupiter:~# systemctl status kubelet  
● kubelet.service - kubelet: The Kubernetes Node Agent  
  Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: enabled)  
  Drop-In: /etc/systemd/system/kubelet.service.d  
            └─10-kubeadm.conf  
    Active: active (running) since Wed 2020-06-10 13:51:01 CEST; 6s ago  
      Docs: https://kubernetes.io/docs/home/  
   Main PID: 16495 (kubelet)  
     Tasks: 15 (limit: 2330)  
    Memory: 30.3M  
   CGroup: /system.slice/kubelet.service  
           └─16495 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.co
```

La detección automática del controlador cgroup para otros tiempos de ejecución de contenedor como CRI-O y containerd está en progreso.

Después de todo esto ha funcionado reseteando la configuración con el cri-socket de cri-o.

```
root@jupiter:~# kubeadm reset --cri-socket=/var/run/crio/crio.sock  
[reset] Reading configuration from the cluster...  
[reset] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'  
W0610 13:59:25.834566 17102 reset.go:99] [reset] Unable to fetch the kubeadm-config ConfigMap from cluster: failed to get config map: configmaps "kubeadm-config" not found  
[reset] WARNING: Changes made to this host by 'kubeadm init' or 'kubeadm join' will be reverted.  
[reset] Are you sure you want to proceed? [y/N]: y  
[preflight] Running pre-flight checks  
W0610 13:59:28.554443 17102 removeetcdmember.go:79] [reset] No kubeadm config, using etcd pod spec to get data directory  
[reset] Stopping the kubelet service  
[reset] Unmounting mounted directories in "/var/lib/kubelet"  
[reset] Deleting contents of config directories: [/etc/kubernetes/manifests /etc/kubernetes/pki]  
[reset] Deleting files: [/etc/kubernetes/admin.conf /etc/kubernetes/kubelet.conf /etc/kubernetes/bootstrap-kubelet.conf /etc/kubernetes/controller-manager.conf /etc/kubernetes/scheduler.conf]  
[reset] Deleting contents of stateful directories: [/var/lib/etcd /var/lib/kubelet /var/lib/docker /var/run/containers /var/run/cni]
```

SOLO EN EL MASTER (Jupiter)

A continuación iniciaremos el nodo master. Este comando se hará en el nodo que se quiera poner como master.

```
root@jupiter:~# kubeadm init --pod-network-cidr 192.168.0.0/16 --ignore-preflight-errors=Swap --v=5 --cri-socket=/var/run/crio/crio.sock

[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:
kubeadm join 192.168.1.10:6443 --token h473c2.1pp9r23lyvs66n37 \
--discovery-token-ca-cert-hash sha256:70f638e662dc34d11c87586c7447d4f471ea4a440fbf1139967abe521cd71bf0
```

Para hacer que kubectl funcione para su usuario no root, ejecute estos comandos, que también son parte de la kubeadm init salida:

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Alternativamente, si somos root usuario, podemos ejecutar:

```
root@jupiter:~# export KUBECONFIG=/etc/kubernetes/admin.conf
```

Debemos de desplegar una Interfaz de red de contenedores, Complemento de red Pod basado en (CNI) para que nuestros Pods puedan comunicarse entre sí.

El DNS del clúster (CoreDNS) no se iniciará antes de instalar una red.

Varios proyectos externos proporcionan redes Kubernetes Pod utilizando CNI, algunos de los cuales también son compatibles con la Política de red.

Podremos consultar la lista de [complementos de redes y políticas de red](#) disponibles.

Podemos instalar un complemento de red Pod con el siguiente comando en el nodo del plano de control o un nodo que tenga las credenciales de kubeconfig:

```
kubectl apply -f <add-on.yaml>
```

Solo podremos instalar una red Pod por clúster. La red Pod que se va a instalar en este momento será Calicó.

NOTA: Calico es un proveedor de redes y políticas de red. Calico admite un conjunto flexible de opciones de red para que pueda elegir la opción más eficiente para su situación, incluidas las redes sin superposición y superpuestas, con o sin BGP. Calico usa el mismo motor para aplicar la política de red para hosts, pods y aplicaciones (si usa **Istio**

& Envoy) en la capa de malla de servicio. Calico trabaja en varias arquitecturas, incluyendo amd64, arm64y ppc64le.

Calico detectará automáticamente qué rango de direcciones IP usar para las IP de pod en función del valor proporcionado a través de la **--pod-network-cidr** bandera o la configuración de kubeadm.

```
root@jupiter:~# kubectl apply -f https://docs.projectcalico.org/v3.14/manifests/calico.yaml
```

Una vez que se ha instalado una red Pod, se puede confirmar que está funcionando verificando que CoreDNS este Running en la salida de **kubectl get pods --all-namespaces**. Y una vez que CoreDNS esté funcionando, puede continuar uniéndose a sus nodos.

```
root@jupiter:~# kubectl get pods --all-namespaces
NAMESPACE      NAME                               READY   STATUS    RESTARTS   AGE
kube-system    calico-kube-controllers-76d4774d89-tlgvj   1/1     Running   0          13m
kube-system    calico-node-4r8pf                  1/1     Running   0          12m
kube-system    calico-node-krvnf                 1/1     Running   0          13m
kube-system    calico-node-vdlk2                  1/1     Running   0          12m
kube-system    coredns-66bff467f8-8pkdl            1/1     Running   0          17m
kube-system    coredns-66bff467f8-fmn6x            1/1     Running   0          17m
kube-system    etcd-jupiter                      1/1     Running   0          17m
kube-system    kube-apiserver-jupiter             1/1     Running   0          17m
kube-system    kube-controller-manager-jupiter       1/1     Running   0          17m
kube-system    kube-proxy-2rrjl                  1/1     Running   0          12m
kube-system    kube-proxy-c8kvtt                1/1     Running   0          17m
kube-system    kube-proxy-k2kwh                  1/1     Running   0          12m
kube-system    kube-scheduler-jupiter             1/1     Running   0          17m
```

6.3.4. Plan de control de despliegue del nodo

Por defecto, su clúster no programará Pods en el nodo del plano de control por razones de seguridad. Si desea poder programar Pods en el nodo del plano de control, por ejemplo, para un clúster Kubernetes de una sola máquina para el desarrollo, ejecute:

```
root@jupiter:~# kubectl taint nodes --all node-role.kubernetes.io/master-
node/jupiter untainted
taint "node-role.kubernetes.io/master" not found
taint "node-role.kubernetes.io/master" not found
```

Esto eliminará la **node-role.kubernetes.io/master** contaminación de cualquier nodo que lo tenga, incluido el nodo del plano de control, lo que significa que el planificador podrá programar Pods en todas partes.

6.3.5. Unión de nodos

SOLO EN LOS WORKER (Saturno y Urano)

Los nodos son donde se ejecutan sus cargas de trabajo (contenedores y Pods, etc.). Para agregar nuevos nodos a su clúster, haga lo siguiente para cada máquina:

```
root@saturno:~# kubeadm join 192.168.1.10:6443 --token h473c2.lpp9r23lyvs66n37 --discovery-token-ca-cert-hash sha256:70f638e662dc34d11c87586c7447d4f47lea4a440fbff139967abe521cd71bf0 --v=5 --cri-socket=/var/run/crio/crio.sock
```

Una vez haya salido esto los nodos se han unido a nuestro clúster.

```
This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

Una vez hecho esto podremos irnos al nodo maestro y comprobaremos los nodos que se han unido a nuestro clúster.

NAME	STATUS	ROLES	AGE	VERSION
jupiter	NotReady	master	118m	v1.18.3
saturno	Ready	<none>	10m	v1.18.3
urano	Ready	<none>	10m	v1.18.3

Si no tenemos el token, podemos obtenerlo ejecutando el siguiente comando en el nodo del plano de control.

```
root@jupiter:~# kubeadm token list
TOKEN          TTL           EXPIRES          USAGES          DESCRIPTION
h473c2.lpp9r23lyvs66n37  20h  2020-06-11T14:02:14+02:00 authentication,signing  The default bootstrap token generated by 'kubeadm init'.
```

Por defecto, los token caducan después de 24 horas. Si está uniendo un nodo al clúster después de que el token actual haya expirado, puede crear un nuevo token ejecutando el siguiente comando en el nodo del plano de control.

```
root@jupiter:~# kubeadm token create
[WARNING: kubeadm cannot validate component configs for API groups [kubebundle.config.k8s.io kubeproxy.config.k8s.io]
8d3g9c49bs76mwa9c8420u
```

Si no tiene el valor de **--discovery-token-ca-cert-hash**, puede obtenerlo ejecutando la siguiente cadena de comandos en el nodo del plano del control.

```
openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der >/dev/null | \
openssl dgst -sha256 -hex | sed 's/^.* //'
```

6.3.6. Limpiear

Si utilizó servidores desecharables para su clúster, para las pruebas, puede apagarlos y no realizar más limpiezas. Puede usar **kubectl config delete-cluster** para eliminar sus referencias locales al clúster.

Sin embargo, si desea desaprovisionar su clúster de manera más limpia, primero debe drenar el nodo y asegurarse de que el nodo esté vacío, luego desconfigurar el nodo.

6.3.6.1. Eliminar nodo

Estando en el nodo manager ejecutaremos los siguientes comandos.

```
root@jupiter:~# kubectl drain saturno --delete-local-data --force --ignore-daemonsets
root@jupiter:~# kubectl delete node saturno
node "saturno" deleted
root@jupiter:~# kubectl get nodes
NAME      STATUS   ROLES     AGE      VERSION
jupiter   Ready    master    3h6m    v1.18.3
urano     Ready    <none>   77m    v1.18.3
```

Luego, en el nodo que se está eliminando, restablezca todo el kubeadm estado instalado:

```
root@saturno:~# kubeadm reset --cri-socket=/var/run/crio/crio.sock
```



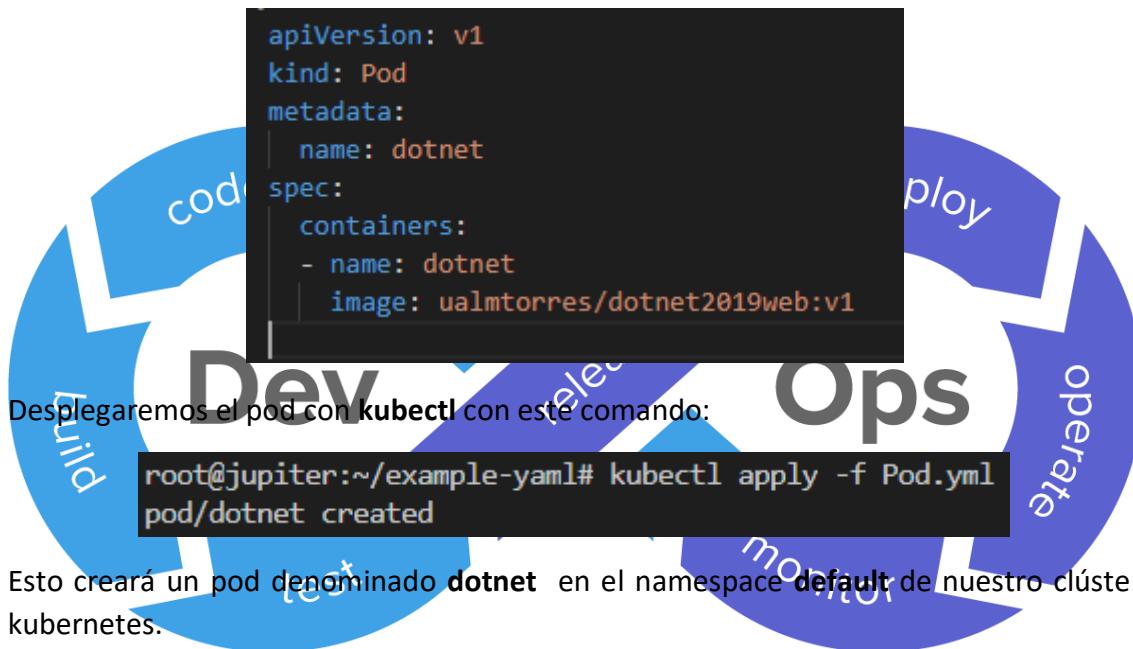
6.4. Despliegue de aplicaciones mediante archivos YAML.

La forma de uso de kubernetes está más orientada a la creación de las tareas puntuales. En cambio, cuando se trata de operaciones que queremos que sean repetibles, la forma de operar consiste en **crear archivos YAML especificando el objeto que se quiere crear** en Kubernetes (espacio de nombres, despliegue, servicio,...). Una vez creados estos archivos, se usará **kubectl** para cargarlos/desplegarlos en Kubernetes.

6.4.1. Despliegue de un Pod

Vamos a crear un archivo de manifiesto para desplegar un pod mediante un archivo de manifiesto YAML. El ejemplo despliega una web de contenido estático.

Archivo **Pod.yml**.



Para ver la aplicación de forma provisional haremos un port forward entre el pod y nuestro equipo local con:

```
root@jupiter:~/example-yaml# kubectl port-forward dotnet 83:80
```

Ahora lo que haremos será hacer un curl a **http://localhost:83** ya que nuestra máquina no tiene interfaz gráfica.

```
root@jupiter:~/example-yaml# curl http://localhost:83
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="description" content="">
    <meta name="author" content="Mark Otto, Jacob Thornton, and Bootstrap contributors">
    <meta name="generator" content="Jekyll v3.8.5">
    <title>DotNet Almería 2019</title>

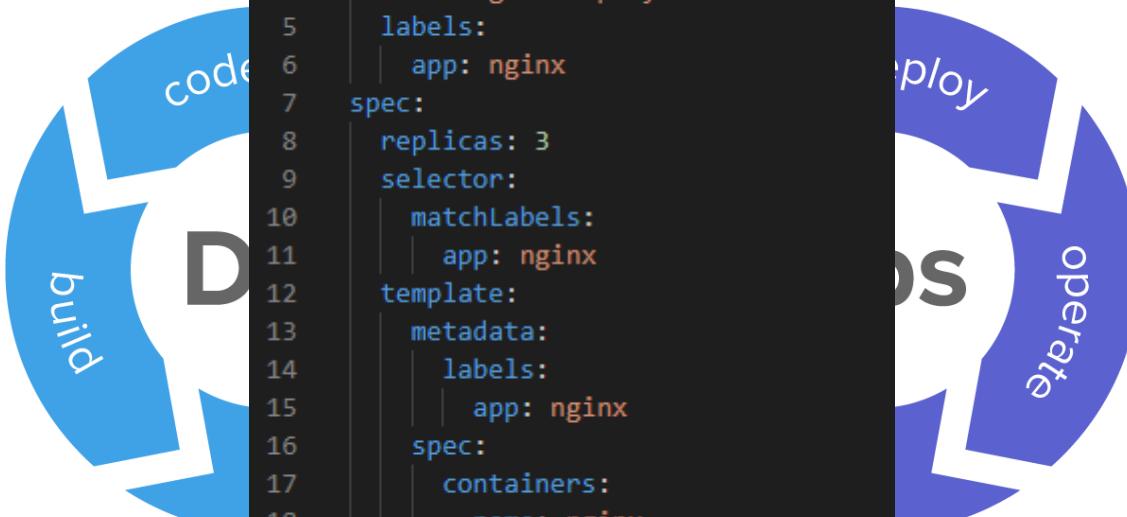
    <link rel="canonical" href="https://getbootstrap.com/docs/4.3/examples/jumbotron/">
```

6.4.2. Despliegue de un Deployment

Normalmente no desplegamos pods. En su lugar desplegaremos Deployments. En ellos podremos incluir contenedores con imágenes diferentes para que puedan trabajar de forma coordinada.

Este archivo contiene entre otras cosas, el nombre del despliegue, el selector que usa el despliegue para seleccionar los pods que forman parte del mismo Deployment, número de réplicas y la imagen usada para crear el contenedor de cada pod.

En este ejemplo se creara un deployment de in servidor web nginx y de este mismo 3 réplicas.



```
! Deployment.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx-deployment
5    labels:
6      app: nginx
7  spec:
8    replicas: 3
9    selector:
10      matchLabels:
11        app: nginx
12    template:
13      metadata:
14        labels:
15          app: nginx
16      spec:
17        containers:
18          - name: nginx
19            image: nginx:1.14.2
20            ports:
21              - containerPort: 80
```

El despliegue se realiza con **kubectl** con el siguiente comando:

```
root@jupiter:~/example-yaml# kubectl apply -f Deployment.yaml
deployment.apps/nginx-deployment created
```

Al crear el despliegue, se procederá a descargar la imagen y se pasarán a crear los pods indicados para este despliegue.

```
root@jupiter:~/example-yaml# kubectl get deployment
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment  3/3     3           3           10m
```

También podemos ver los ReplicaSets creados por los despliegues.

```
root@jupiter:~/example-yaml# kubectl get rs
NAME           DESIRED   CURRENT   READY   AGE
nginx-deployment-6b474476c4   3         3         3      10m
```

Los pods los podemos ver junto con sus etiquetas con el parámetro **--show-labels**:

```
root@jupiter:~/example-yaml# kubectl get pods --show-labels
NAME          READY   STATUS    RESTARTS   AGE   LABELS
dotnet        1/1     Running   0          3h7m  <none>
nginx-deployment-6b474476c4-9gdpj   1/1     Running   0      18m   app=nginx,pod-template-hash
=6b474476c4
nginx-deployment-6b474476c4-qmj7l   1/1     Running   0      18m   app=nginx,pod-template-hash
=6b474476c4
nginx-deployment-6b474476c4-ztk42   1/1     Running   0      18m   app=nginx,pod-template-hash
=6b474476c4
```

Ahora podríamos ver a cualquiera de los pods de nginx haciendo port forward a nuestro equipo.

```
root@jupiter:~/example-yaml# kubectl port-forward nginx-deployment-6b474476c4-9gdpj 83:80
```

Este sería el resultado en local.



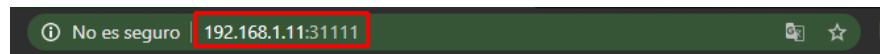
Vamos a crear un archivo de Servicio denominado **Service.yml**. Este archivo básicamente contiene entre otros el nombre de servicio, el tipo del servicio (ClusterIP, NodePort,...), el puerto de acceso a los pods del despliegue y el selector que identifica al despliegue con el que se corresponde el servicio creado.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      nodePort: 31111
      targetPort: 80
```

El despliegue se realiza con **kubectl** con el siguiente comando.

```
root@jupiter:~/example-yaml# kubectl apply -f Servicie.yaml
service/my-service created
```

El despliegue nos permitirá acceder a la aplicación en el puerto **31111**.



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

También lo que podremos hacer es meter nuestro servicio y nuestro deployment en el mismo yaml.



Y para desplegar este yaml tendríamos que ejecutar el comando **kubectl** con el nombre de este yaml.

7. Service Mesh

7.1. ¿Qué es un Service Mesh?

Una service mesh es una **capa de infraestructura que controla la comunicación entre los servicios de una red**. Esta tiene el **control del envío y entrega de solicitudes entre los servicios**. Entre las características comunes que proporciona un service mesh, destacan: el descubrimiento de servicios, el balanceo de carga, el cifrado, la tolerancia a fallos y la resiliencia. En resumen, el service mesh hace que la comunicación entre servicios sea **rápida, confiable y segura**.

7.2. ¿Cómo funciona una Service Mesh?

Una Service Mesh, normalmente usa un tipo de proxy conocido como **Sidecar**. Un “Proxy Sidecar”, se **vincula a un servicio para ampliar o añadir funcionalidades**.

En el caso que estamos desarrollando, y al ser una estructura de microservicios en Kubernetes, cada proxy sidecar se implantará en cada pod.

Una vez implementado el proxy, se podrán distinguir dos “capas” en la Service Mesh: el **data plane**(o plano de datos) y el **control plane**(o plano de control).

El **data plane** está formado por las instancias, los sidecars y sus interacciones; mientras que, por otro lado, el **control plane**, es el encargado de la **administración de tareas, la monitorización, y la implementación de políticas**.

7.3. Estudio de una Service Mesh. Ventajas y Desventajas

La implementación de un Service Mesh resuelve algunos problemas clave a la hora de montar una estructura de microservicios, pero no todos. Entre las ventajas que destacan a la hora de montar una Service Mesh destacan:

- **Simplifica la comunicación entre los microservicios.**
- **Facilidad a la hora de encontrar** fallos en la comunicación, ya que esta se encuentra “empaquetada” en su propia capa de infraestructura.
- **Permite el cifrado, la autenticación y la autorización.**
- **Permite un desarrollo, prueba y despliegue de las aplicaciones.**
- **Mejora en la administración de las redes** de un clúster de contenedores.

Por otro lado, también existen desventajas relacionadas con la implantación de una Service Mesh, entre ellas:

- El tiempo de ejecución de las instancias aumenta exponencialmente.
- Añade un paso más en la comunicación, ya que la comunicación debe pasar a través del proxy.

8. Istio



8.1. ¿Qué es Istio?

Istio, es una plataforma abierta para **gestionar, conectar y securizar microservicios**, o dicho de otra manera, es una forma de **implementar “service mesh” en nuestra estructura de microservicios**.

Entre las ventajas que nos da Istio (y una vez estudiado que es el “Service Mesh”), podemos enumerar, las opciones que nos provee:

- Balanceo de carga y enrutador inteligente.
- Tolerancia a fallos.
- Aplicación de políticas en todos los servicios.
- Telemetría y generación de informes y reportes en profundidad.

El mayor punto a favor de Istio reside en la **facilidad de crear una red de servicios desplegados con balanceo de carga, autenticación, monitorización, etc sin realizar ningún cambio en el código de cada servicio, ya que el soporte es agregado a los servicios mediante la implementación de un proxy de tipo sidecar**, nombrado en Istio como “Envoy”.

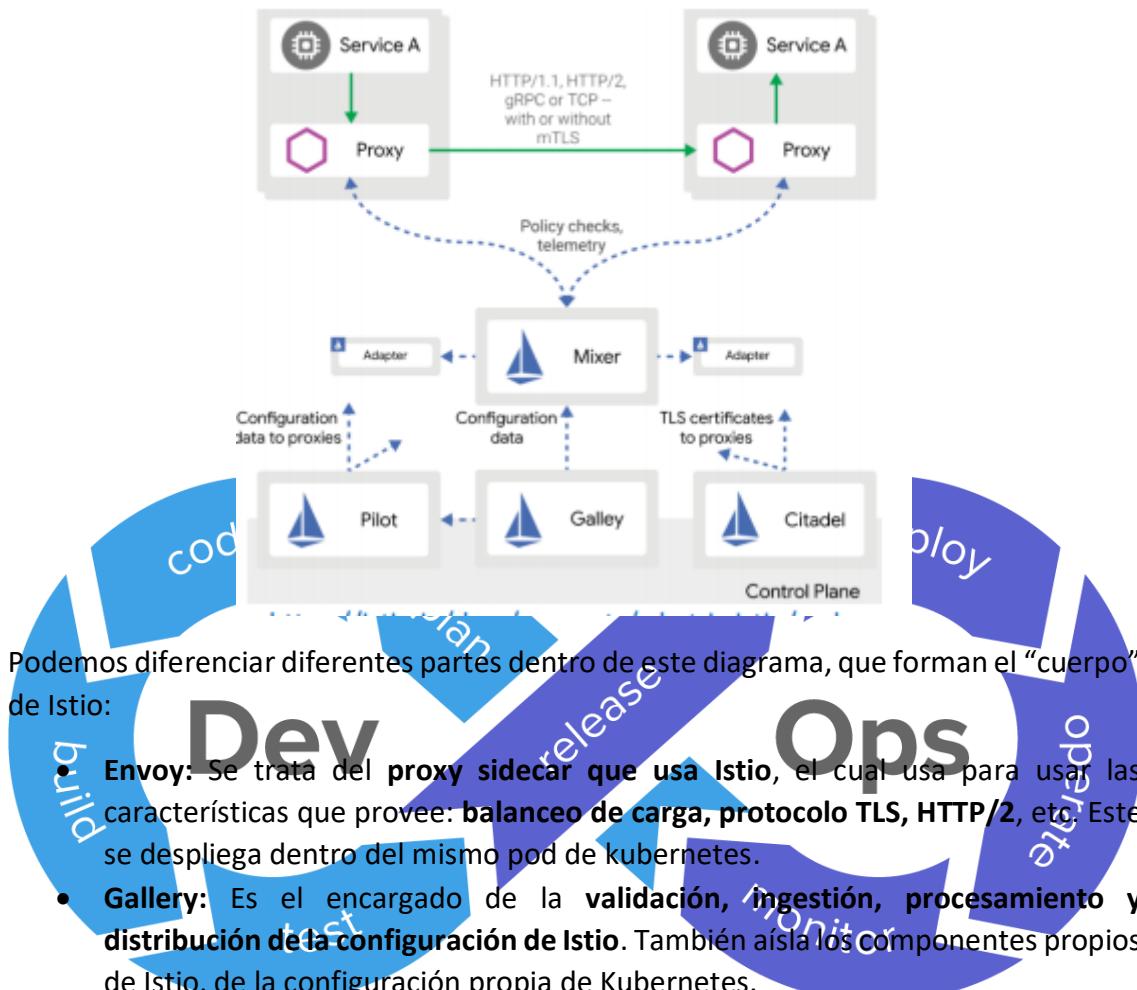
Actualmente sólo permite despliegues sobre Kubernetes, aunque se tiene previsto que se añada su soporte para otros entornos en el futuro.

Como vimos a la hora de ver que es un Service Mesh, la arquitectura de Istio se divide en:

- **Plano de datos**, formado por los Envoy, que se encargan de controlar el tráfico de red de todas las comunicaciones entre microservicios.
- **Plano de control**, es responsable de gestionar y configurar los proxies y de gestionar las políticas.

8.2. Componentes de Istio

El siguiente diagrama, que nos proporciona Istio a través de su web, nos muestra los componentes que forman Istio, y el plano en el que trabajan:



- **Envoy:** Se trata del proxy sidecar que usa Istio, el cual usa para usar las características que provee: **balanceo de carga, protocolo TLS, HTTP/2, etc.** Este se despliega dentro del mismo pod de Kubernetes.
- **Gallery:** Es el encargado de la **validación, ingestión, procesamiento y distribución de la configuración de Istio**. También aísla los componentes propios de Istio, de la configuración propia de Kubernetes.
- **Mixer:** La principal función de este componente es **aplicar políticas y Access control** a partir de la información obtenida de Envoy.
- **Pilot:** Provee descubrimientos de servicios para Envoy, así como, **enrutado inteligente** (aplicación de tests) como también **resiliencia** (reintentos, tiempo de espera, etc).
- **Citadel:** Por último, Citadel, se encarga tanto de la **comprobación de credenciales a nivel de servicio como a nivel de usuario**.

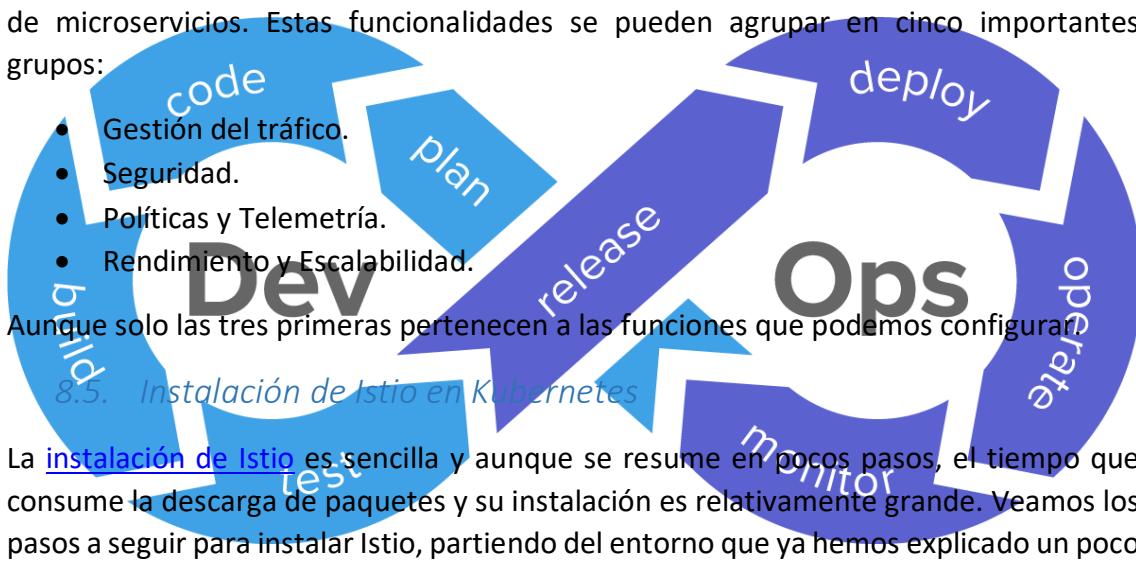
8.3. *Objetivos*

Istio se ajusta a una serie de objetivos en su diseño, para maximizar su rendimiento y facilitar su uso de cara al usuario. Entre estos objetivos destacan:

- **Máxima transparencia:** La principal idea de Istio, es que se requiera el mínimo esfuerzo si se quiere adoptar este.
- **Incrementabilidad:** El sistema está planteado para que crecer según las necesidades del entorno, dado que se creará una dependencia a Istio.
- **Portabilidad:** Istio está pensado para ser usado en cualquier plataforma con el coste mínimo de esfuerzos y recursos.
- **Uniformidad de política:** Permite aplicar políticas a todo lo que controla, este expuesto al exterior o no.

8.4. *Funcionalidades*

En este apartado, vamos a ver las funcionalidades que nos aporta Istio en nuestra red de microservicios. Estas funcionalidades se pueden agrupar en cinco importantes grupos:



8.5. *Instalación de Istio en Kubernetes*

La [instalación de Istio](#) es sencilla y aunque se resume en pocos pasos, el tiempo que consume la descarga de paquetes y su instalación es relativamente grande. Veamos los pasos a seguir para instalar Istio, partiendo del entorno que ya hemos explicado un poco antes.

Lo primero que haremos será descargarnos los ficheros de instalación de Istio. Para ello ejecutaremos:

```
root@jupiter:~# curl -L https://istio.io/downloadIstio | sh -
```

Lo siguiente que haremos será movernos al directorio que hemos descargado.

```
root@jupiter:~# cd istio-1.6.1/
root@jupiter:~/istio-1.6.1#
```

Agregaremos istioctl cliente a nuestra ruta.

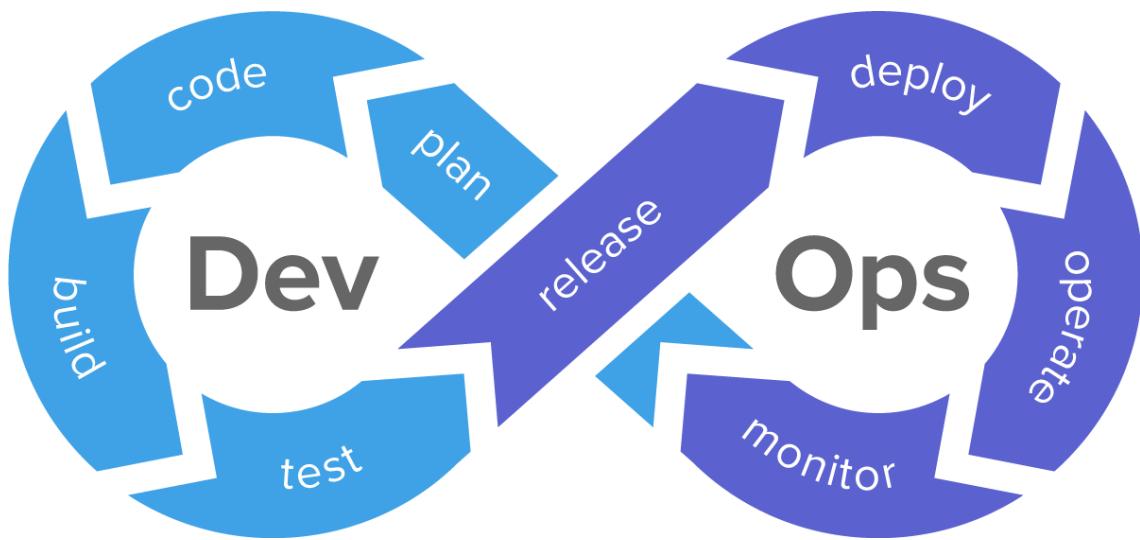
```
root@jupiter:~/istio-1.6.1# export PATH=$PWD/bin:$PATH
```

Para esta instalación, utilizaremos el demo [perfil de configuración](#). Se selecciona para tener un buen conjunto d valores predeterminados para las pruebas, pero hay otros perfiles para las pruebas de producción o rendimiento.

```
root@jupiter:~/istio-1.6.2# istioctl install --set profile=demo
Detected that your cluster does not support third party JWT authentication. Falling back to less
/security/#configure-third-party-service-account-tokens for details.
✓ Istio core installed
  - istio-system
```

Así se instalaría Istio en un cluster de k8s en local, en este caso istio tuvo fallos por lo que no se ha procedido más profundidad en este.

Dado que Istio ha ocasionado problemas en local se procederá su instalación y prueba en cloud dando importancia a esta herramienta.



9. Rancher



9.1. ¿Qué es Rancher?

Rancher es una pila completa de software para equipos que **adoptan contenedores**. Aborda los desafíos operativos y de **seguridad de administrar múltiples clústeres en cualquier infraestructura**, al tiempo que proporciona a los equipos de DevOps **herramientas integradas para ejecutar cargas de trabajo** en contenedores.

9.2. Características de Rancher.

- Gestión unificada de múltiples clústeres

Rancher une los grupos de kubernetes con autenticación centralizada y control de acceso, seguridad empresarial, auditoria, copias de seguridad, actualizaciones, observabilidad y alertas. Implemente y asegure clústeres de manera consistente en minutos cualquier lugar utilizando nuestra interfaz de usuario intuitiva o nuestra potente CLI.



Este diagrama circular ilustra el flujo continuo de DevOps. Se divide en tres secciones principales: "Dev" (Desarrollo) en la parte izquierda, "Ops" (Operaciones) en la parte derecha y "CI/CD" (Integración Continua / Despliegue Continuo) en la parte superior. Los pasos secuenciales son: "code" (código), "build" (compilar), "test" (pruebas), "release" (releases), "deploy" (despliegue) y "monitor" (monitoreo). Los términos "code", "build", "test" y "release" están escritos en azul, mientras que "deploy" y "monitor" están escritos en gris.

- *Catálogo de aplicaciones centralizado*

Aproveche Helm para la implementación llave de mano de múltiples clústeres de herramientas o aplicaciones de código abierto populares del ecosistema de socios en Rancher.

- *Soporte híbrido y multi-nube*

Administre clústeres locales y aquellos alojados en servicios en la nube como AKS, EKS y GKE desde un solo panel de vidrio. Configure centralmente la política de seguridad, audite los registros y monitoree el rendimiento.

Implemente **aplicaciones de múltiples clústeres** de manera consistente desde nuestro catálogo de aplicaciones. Controle el acceso conectándolos a su proveedor de identidad interno **como Active Directory, LDAP u Okta**.

- Acelerar la adopción de DevOps

Rancher admite herramientas que los equipos DevOps ya adoran, como **Jenkins, Gitlab o Travis** para construir tuberías de CI / CD, **Prometheus y Grafana** para la observabilidad, **Fluentd** para el registro e **Istio** para la malla de servicio.

- **Política de seguridad consistente y cumplimiento**

Rancher permite la **habilitación directa del proveedor de cifrado** y la configuración del **registro de auditoría y la limitación de velocidad**. Escanee y evalúe las configuraciones de clúster contra las mejores prácticas de referencia de CIS. Implemente constantemente clústeres aprovisionados de forma segura utilizando plantillas de clúster en múltiples sustratos. Incluye soporte para entornos con espacios de aire.

9.3. Instalación de Rancher

En este caso Rancher será **instalado en uno de los nodos del clúster Swarm** pero perfectamente **puede ser instalado en un equipo aislado del clúster**.

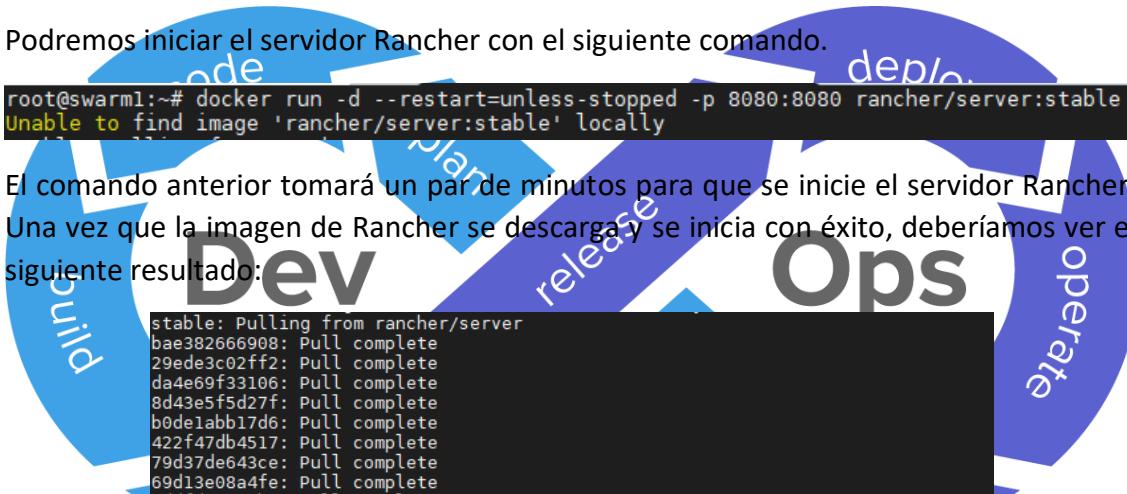
Lo primero que tenemos que tener para instalar Rancher será tener instalado Docker en el equipo donde queremos instalar Rancher. Por lo tanto seguiremos los pasos de los apartados anteriores de [instalación de docker](#).

Una vez docker ya este funcionando, es hora de instalar Rancher.

Podremos iniciar el servidor Rancher con el siguiente comando.

```
root@swarm1:~# docker run -d --restart=unless-stopped -p 8080:8080 rancher/server:stable  
Unable to find image 'rancher/server:stable' locally
```

El comando anterior tomará un par de minutos para que se inicie el servidor Rancher. Una vez que la imagen de Rancher se descarga y se inicia con éxito, deberíamos ver el siguiente resultado:



```
stable: Pulling from rancher/server  
bae382666908: Pull complete  
29ede3c02ff2: Pull complete  
da4e69f33106: Pull complete  
8d43e5f5d27f: Pull complete  
b0de1abb17d6: Pull complete  
422f47db4517: Pull complete  
79d37de643ce: Pull complete  
69d13e08a4fe: Pull complete  
2ddfd3c6a2b7: Pull complete  
bc433fed3823: Pull complete  
b82e188df556: Pull complete  
dae2802428a4: Pull complete  
07bf18e8eec0: Pull complete  
339e24088f91: Pull complete  
9372455de0b8: Pull complete  
5a33b348bf45: Pull complete  
3286997d8874: Pull complete  
bd79bfb954de: Pull complete  
ba7c19991a31: Pull complete  
0c19aca4f8a1: Pull complete  
e03fc76c8997: Pull complete  
Digest: sha256:95b55603122c28baea4e8d94663aa34ad770bbc624a9ed6ef986fb3ea5224d91  
Status: Downloaded newer image for rancher/server:stable  
5d8a0a5d2bef45d061d154628b0779e02123c78cccd4f7bfccel4e9ce23ceb254
```

Ahora abriremos nuestro navegador y escribiremos la siguiente URL http://SERVER_IP:8080, seremos redirigidos a la página de administración de Rancher como se muestra a continuación.

The screenshot shows the Rancher web interface at the URL 192.168.15.16:8080/env/1a5/apps/stacks. The top navigation bar includes links for Default, STACKS, CATALOG, INFRASTRUCTURE, ADMIN (with a warning icon), and API. A prominent warning message states: "Before adding your first service or launching a container, you'll need to add a Linux host with a supported version of Docker. Add a host". Below this, the "User Stacks" section is visible with an "Add Stack" button. A central callout box is titled "Adding your first Stack" and contains text explaining what a service is and how it can be added from the Catalog. At the bottom of this box are two buttons: "Define a Service" and "Browse Catalog".

Rancher no configura el control de acceso por defecto. Para configurar el control de acceso, por eso nos iremos a la pestaña **administrador** y haremos clic en **Control de acceso**.

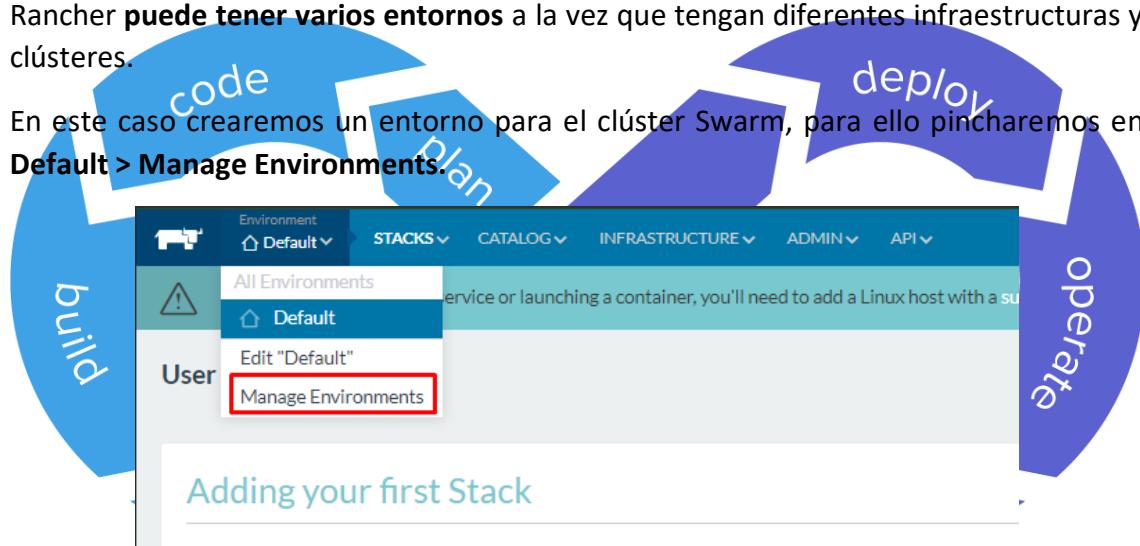
The screenshot shows the Rancher web interface at the URL 192.168.15.16:8080/admin/access-control. The top navigation bar includes links for Default, STACKS, CATALOG, INFRASTRUCTURE, ADMIN (with a warning icon), and API. A warning message at the top says: "Before adding your first service or launching a container, you'll need to add a Linux host with a supported version of Docker. Add a host". The main content area is titled "Access Control" and features a grid of icons for different authentication providers: Active Directory, Azure AD, GITHUB, LOCAL (which has a checkmark), OpenLDAP, and SHIBBOLETH. Below this, a callout box states: "Local Authentication is not configured" and provides a note: "Rancher can be configured to restrict access to a set of accounts defined in the Rancher database. This is not currently set up, so anybody that reaches this page (or the API) has full control over the system." At the bottom of the page, there are two sections: "1. Setup an Admin user" and "2. Enable Access Control". The "1. Setup an Admin user" section contains fields for "Login Username" (jose), "Full Name" (jose), "Password" (*****), and "Confirm Password" (*****). The "2. Enable Access Control" section contains a button labeled "Enable Local Auth".

Aquí haremos clic en **local** proporcionaremos nuestro nombre de usuario y contraseña de administrador, luego haremos clic en **Habilitar autenticación local**.

9.4. Creación de Swarm en Rancher.

Para poder crear un clúster mediante Rancher deberemos de crear primero un entorno, Rancher **puede tener varios entornos** a la vez que tengan diferentes infraestructuras y clústeres.

En este caso crearemos un entorno para el clúster Swarm, para ello pincharemos en **Default > Manage Environments**.



Una vez dentro tendremos que crear un entorno, por defecto estará creado el de por defecto. Para poder crear el entorno deberemos de pinchar en **Add Environment**.

State	Name	Description	Template	Orchestration	Default
⚠️ Unhealthy	Default	No description	Cattle	Cattle	<input checked="" type="checkbox"/>

En la siguiente ventana definiremos el **nombre del entorno, descripción, plantilla de entorno** que hay varias como Cattle, Kubernetes la más famosa, Mesos y por ultimo Swarm y Windows que son experimentales.

Además de las opciones descritas también se podrá definir un grupo o usuario para el control de acceso a dicho entorno.

Add Environment

Name: Swarm-local

Description: e.g. Environment for developer experimentation

Environment Template

- Cattle
- Kubernetes
- Mesos
- Swarm** (selected, highlighted with a red box)
- EXPERIMENTAL Windows

Orchestration: SwarmKit
 Framework: Network Services, Scheduler, Healthcheck Service
 Management: portainer
 Networking: Rancher IPsec

Access Control

Name	Type	Role
jose	Local User	Owner

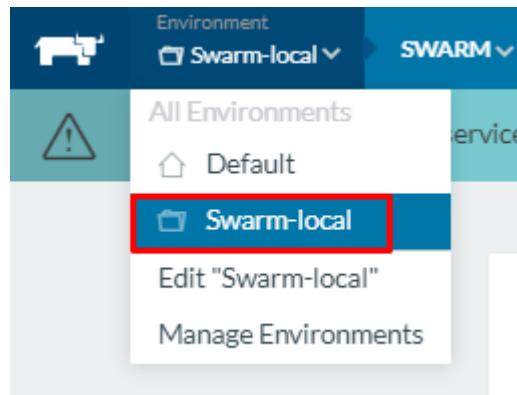
Create Cancel

Una vez escritas las opciones le daremos a **Create** y comprobaremos que el entorno ha sido creado.

State	Name	Description	Template	Orchestration	Default
Unhealthy	Default	No description	Cattle	Cattle	✓
Active	Swarm-local	No description	Swarm	Swarm	-

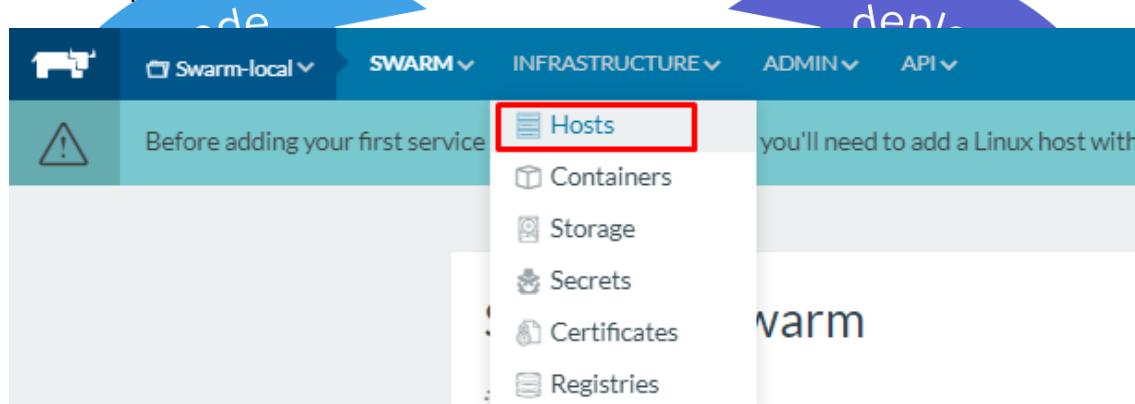
Environment Templates Add Template

Para poder seleccionar el entorno creado para nuestro clúster Swarm deberemos de pinchar en **Environment > Swarm-local**.

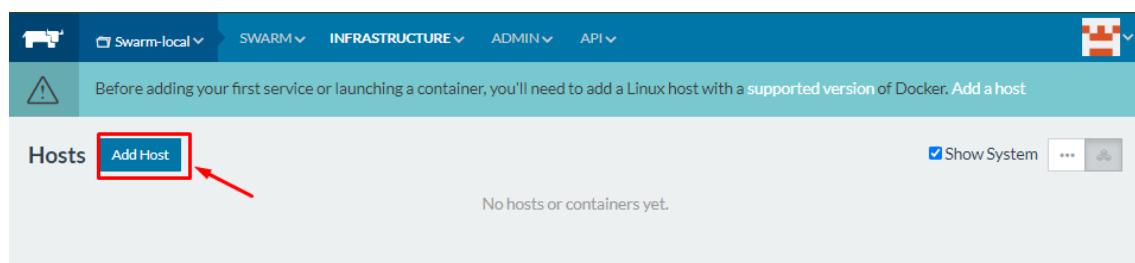


Una estamos en el entorno que hemos creado deberemos de añadir los hosts que participaran en nuestro clúster, en este caso los nodos serán **Swarm1, Swarm2 y Swarm3**.

Para ello pincharemos en **Infrastructure > hosts**.



Para añadir un host deberemos de pinchar en **Add Host**.



En la siguiente pantalla podremos definir la dirección a la que nuestros equipos se van a conectar, ya sea la **dirección del equipo donde está instalado Rancher, dirección de un proxy** que nos redirige a Rancher, hay muchas posibilidades posibles.

En este caso la dirección a la que se van a conectar será la ip del equipo donde está instalado Rancher.

Host Registration URL

What base URL should hosts use to connect to the Rancher API?

This site's address: `http://192.168.15.16:8080`

Something else: e.g. `http://example.com:8080`

Don't include `/v1` or any other path, but if you are doing SSL termination in front of Rancher, be sure to use `https://`.

i Are you sure all the hosts you will create will be able to reach `http://192.168.15.16:8080`? It looks like a private IP or local network.

Save

Una vez le daremos a siguiente podremos seleccionar el equipo que queremos añadir a nuestro entorno. Hay diferentes posibilidades por defecto más las que puedes añadir.

Por defecto vienen **Custom, Amazon EC2, Azure, DigitalOcean** y **Packet**.

En este caso se ha elegido la opción **custom** porque añadiremos un **equipo local**. Una vez elegido el tipo de host deberemos copiar el comando resaltado en la captura y ejecutarlo en cada uno de los nodos que formaran parte de nuestro cluster.

Hosts: Add Host

Custom

AMAZON EC2

Azure

DigitalOcean

packet

Manage available machine drivers

- Start up a Linux machine somewhere and install a supported version of Docker on it.
- Make sure any security groups or firewalls allow traffic:
 - From and To all other hosts on **UDP** ports **500** and **4500** (for IPsec networking)
- Optional: Add labels to be applied to the host.

Add Label
- Specify the public IP that should be registered for this host. If left empty, Rancher will auto-detect the IP to use. This generally works for machines with unique public IPs, but will not work if the machine is behind a firewall/NAT or if it is the same machine that is running the **rancher/server** container.

e.g. 12.3.4
- Copy, paste, and run the command below to register the host with Rancher:


```
sudo docker run --rm --privileged -v /var/run/docker.sock:/var/run/docker.sock -v /var/lib/rancher:/var/lib/rancher rancher/agent:v1.2.11 http://192.168.15.16:8080/v1/scripts/59441A4B4CE87E651484:1577758400000:tK4Z4F8tuC5CdnP9gkRd5H2Dw
```
- Click close below. The new host should pop up on the Hosts screen within a minute.

Close

Como se ha descrito anteriormente debemos de copiar este comando en todos los nodos que formaran parte de nuestro clúster.

```
root@swarm1:~# docker run --rm --privileged -v /var/run/docker.sock:/var/run/docker.sock -v /var/lib/rancher:/var/lib/rancher rancher/agent:v1.2.11 http://192.168.15.16:8080/v1/scripts/59441A4B4CE07E651484:1577750400000:t3K4Z4F8tuc5CdnP9gkRdShZDw

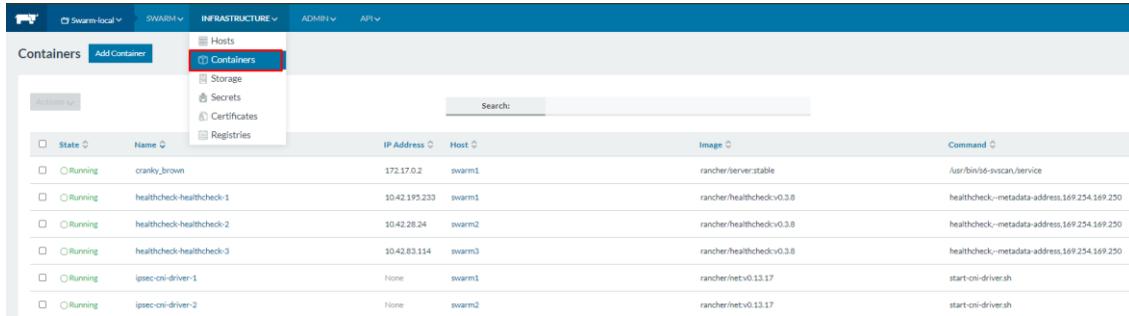
INFO: Running Agent Registration Process, CATTLE_URL=http://192.168.15.16:8080/v1
INFO: Attempting to connect to: http://192.168.15.16:8080/v1
INFO: http://192.168.15.16:8080/v1 is accessible
INFO: Configured Host Registration URL info: CATTLE_URL=http://192.168.15.16:8080/v1 ENV_URL=http://192.168.15.16:8080/v1
INFO: Inspecting host capabilities
INFO: Boot2Docker: false
INFO: Host writable: true
INFO: Token: xxxxxxxx
INFO: Running registration
INFO: Printing Environment
INFO: ENV: CATTLE_ACCESS_KEY=DCC2BB3BEABC9636959
INFO: ENV: CATTLE_HOME=/var/lib/cattle
INFO: ENV: CATTLE_REGISTRATION_ACCESS_KEY=registrationToken
INFO: ENV: CATTLE_REGISTRATION_SECRET_KEY=xxxxxxxx
INFO: ENV: CATTLE_SECRET_KEY=xxxxxxxx
INFO: ENV: CATTLE_URL=http://192.168.15.16:8080/v1
INFO: ENV: DETECTED_CATTLE_AGENT_IP=172.17.0.1
INFO: ENV: RANCHER_AGENT_IMAGE=rancher/agent:v1.2.11
INFO: Launched Rancher Agent: 21c6c13f2ca6337ad3567fddba838d38564b529d6ff09584f577f1887115726b
```

Una vez copiados todos los comandos en todos los equipos comprobaremos que irán apareciendo en la pestaña hosts. Por defecto en el clúster Swarm hecho por Rancher todos los equipos serán manager.

Host	Stack	Container	IP Address
swarm1	Stack: healthcheck	healthcheck-1	10.42.64.105
	Stack: ipsec	cni-driver-1	None
		ipsec-1	10.42.252.180
swarm2	Stack: healthcheck	healthcheck-2	10.42.150.213
	Stack: ipsec	cni-driver-2	None
		ipsec-2	10.42.199.215
swarm3	Stack: healthcheck	healthcheck-3	10.42.37.252
	Stack: ipsec	cni-driver-3	None
		ipsec-3	10.42.61.200

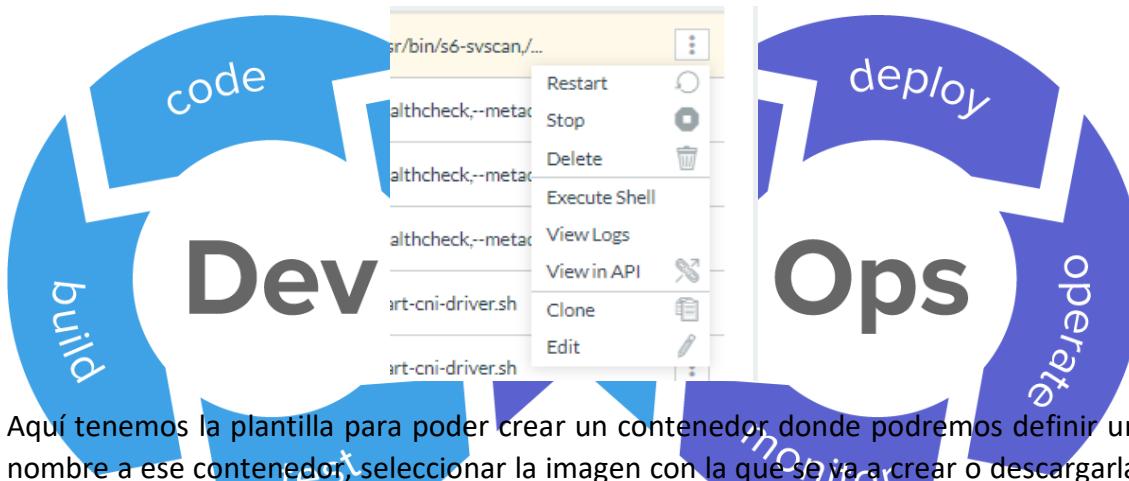
9.4.1. Exploración de la herramienta.

En la pestaña de **Infrastructure** también nos podremos encontrar la sección de **Containers**. Esta pestaña tendrá todos los contenedores de nuestros hosts del entorno en el que estemos. También podremos crear un nuevo contenedor.



State	Name	IP Address	Host	Image	Command
Running	cranky_brown	172.17.0.2	swarm1	rancher/server:stable	/usr/bin/s6-svscan/service
Running	healthcheck-healthcheck-1	10.42.195.233	swarm1	rancher/healthcheck:v0.3.8	healthcheck--metadata-address,169.254.169.250
Running	healthcheck-healthcheck-2	10.42.20.24	swarm2	rancher/healthcheck:v0.3.8	healthcheck--metadata-address,169.254.169.250
Running	healthcheck-healthcheck-3	10.42.83.114	swarm3	rancher/healthcheck:v0.3.8	healthcheck--metadata-address,169.254.169.250
Running	ipsec-cni-driver-1	None	swarm1	rancher/netv0.13.17	start-oni-driver.sh
Running	ipsec-cni-driver-2	None	swarm2	rancher/netv0.13.17	start-oni-driver.sh

Además de crear contenedores también podremos hacer las siguientes opciones que se plantean en la siguiente captura.

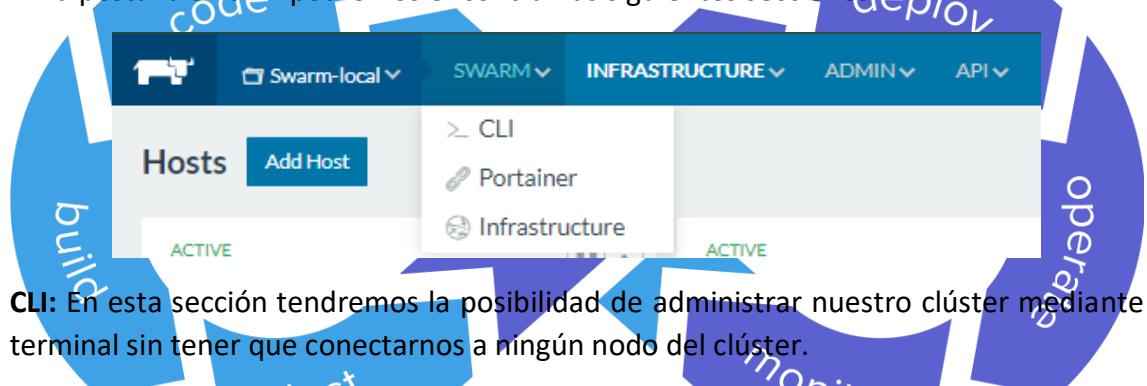


Aquí tenemos la plantilla para poder crear un contenedor donde podremos definir un nombre a ese contenedor, seleccionar la imagen con la que se va a crear o descargarla de Docker Hub, también podremos elegir el mapeo de puertos además de comandos, volúmenes, redes, networking, etc.

Add Container

Name <input type="text" value="e.g. myapp"/>	Description <input type="text" value="e.g. My Application"/>
Select Image* <input type="text" value="ubuntu:14.04.3"/>	<input type="checkbox"/> Always pull image before creating
(+) Port Map	
Command Volumes Networking Security/Host Secrets Health Check Labels Scheduling	
Command <input type="text" value="e.g. /usr/sbin/httpd -f httpd.conf"/>	
Entry Point <input type="text" value="e.g. ./bin/sh"/>	
Working Dir <input type="text" value="e.g. /myapp"/>	User <input type="text" value="e.g. apache"/>
Console <input checked="" type="radio"/> Interactive & TTY (-t-t) <input type="radio"/> TTY(-t)	<input type="radio"/> Interactive (-i) <input type="radio"/> None
Auto Restart <input checked="" type="radio"/> Never <input type="radio"/> Always	<input type="radio"/> On failure (non-0 exit code), forever <input type="radio"/> On failure, up to <input type="text" value="5"/> times
Environment (+) Add Environment Variable	

En la pestaña **Swarm** podremos encontrar las siguientes secciones:



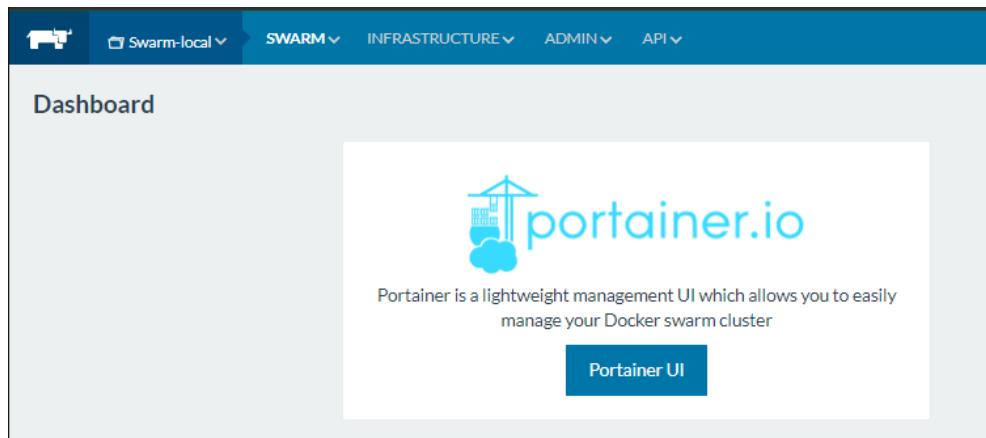
CLI: En esta sección tendremos la posibilidad de administrar nuestro clúster mediante terminal sin tener que conectarnos a ningún nodo del clúster.

Swarm CLI

```
>_ Shell: swarm-swarmkit-mon-1
# Run docker commands inside here
# e.g. docker service ls
> [REDACTED]
```

Connected

Portainer: Esta sección se nos da la posibilidad de abrir Portainer para poder administrar nuestro clúster Swarm con una herramienta más potente.



Aquí podremos comprobar que en nuestro navegador se nos ha abierto el panel web de Portainer.

A detailed screenshot of the Portainer Home dashboard. On the left, a sidebar menu lists various sections: ACTIVE ENDPOINT (swarm1), ENDPOINT ACTIONS (Dashboard, App Templates, Services, Containers, Images, Networks, Volumes, Secrets, Swarm), and PORTAINER SETTINGS (Endpoints, Registries, Settings). The main content area is titled 'Home Dashboard'. It contains two sections: 'Node info' (Name: swarm1, Docker version: 19.03.8, CPU: 1, Memory: 1GB) and 'Swarm info' (This node is part of a Swarm cluster, Node role: Manager, Nodes in the cluster: 3). Below these are four summary cards: 'Containers' (11 running, 0 stopped), 'Images' (11, 3.3 GB), 'Volumes' (8), and 'Networks' (5).

Infraestructure: en esta sección podremos encontrarnos los stack creados en nuestro clúster, además de poder crearlos a nuestro antojo o desde el catalogo que trae ya Rancher.

Stack	Status	Services	Containers	Manage
healthcheck	Up to date	1 Services	3 Containers	[Manage]
ipsec	Up to date	2 Services	12 Containers	[Manage]
network-services	Up to date	2 Services	9 Containers	[Manage]
portainer	Up to date	1 Services	2 Containers	[Manage]
scheduler	Up to date	1 Service	1 Container	[Manage]
swarm	Up to date	1 Services	3 Containers	[Manage]

Podremos meter más plantillas de stack en Rancher pinchando en **Add from Catalog**.

Ya dentro le daremos al botón **Manage**.

Catalog Item	Description	Manage
Alibaba Cloud DNS	Rancher External DNS service powered by Alibaba Cloud	[View Details] [Manage]
AutoSpotting	Replaces AWS On-Demand instances with cheaper Spot-Instances where possible	[View Details] [Manage]
AWS Spot Instance Helper	Automatically evicts spot instances that are marked for termination	[View Details] [Manage]

Aquí dentro pondremos añadir una dirección de github donde tenga unos catálogos de Rancher.

Manage Catalogs

You can define your own custom catalog sources here, which will be available only to users of this environment. Global catalogs are available to all environments and can only be modified in Admin Settings by an administrator. Each catalog needs a unique name and a URL that `git clone` can handle (see [docs](#) for more info).

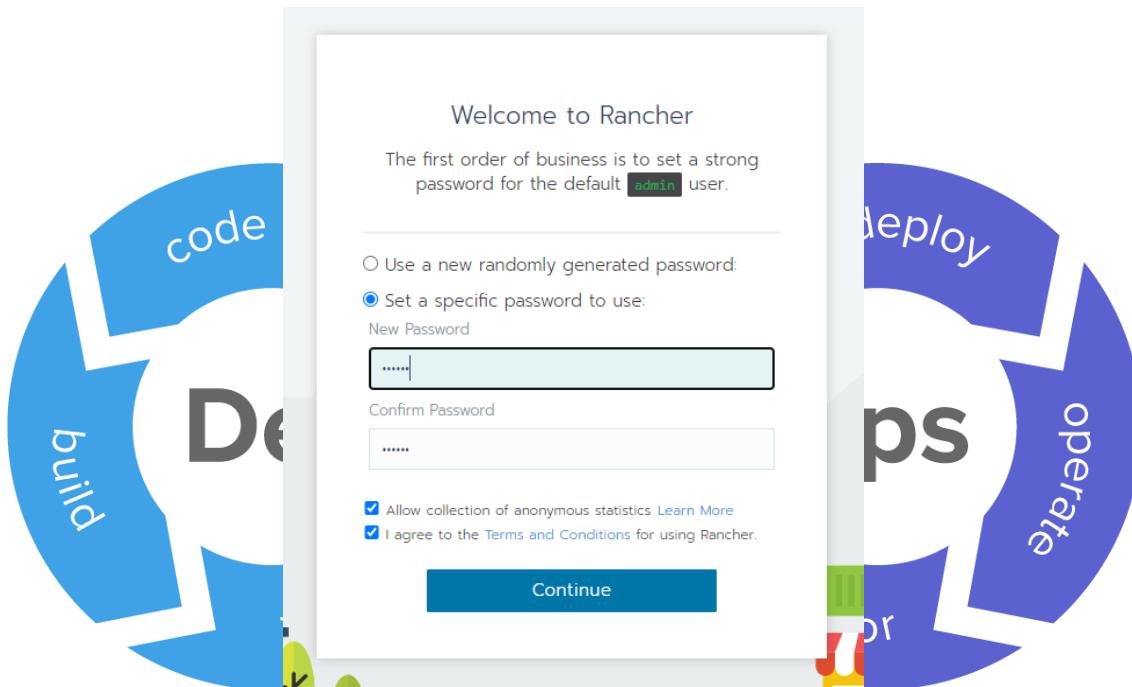
Name	URL	Kind	Branch
library	https://git.rancher.io/rancher-catalog.git	Global	v1.6-release
community	https://git.rancher.io/community-catalog.git	Global	master
<input type="text" value="e.g. https://github.com/mycompany/mycatalog.git"/>		Environment	master

9.5. Creación de clúster kubernetes.

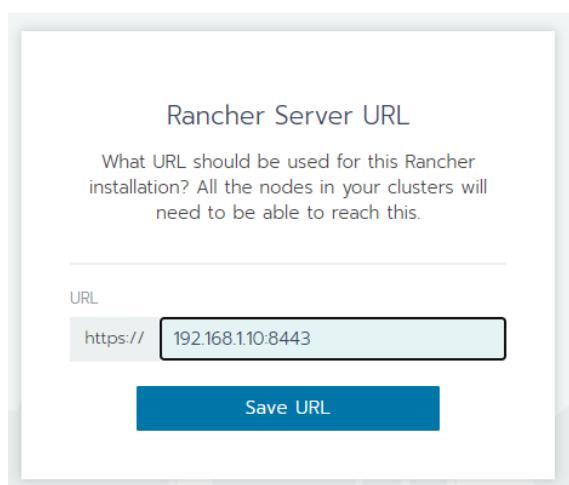
Para poder instalar Rancher hay muchas maneras y formas de hacerlo entre otras en HA dentro de un clúster pero en este caso se ha instalado en el nodo **jupiter** el cual será el equipo que tenga **etcd** y **control plane**, los nodos **Saturno** y **urano** serán **Worker**.

Rancher ha sido instalado de la misma forma que en el punto anterior por lo que no se va a repetir la instalación. La única diferencia que ha habido ha sido que se ha instalado la versión **Rancher/Rancher:latest**.

Una vez instalado ingresamos la Ip de nuestro nodo además del puerto **8080** o **8443** si es seguro. Lo primero que nos pedirá será la contraseña del usuario administrador por lo tanto la especificaremos.



A continuación nos preguntara la URL de del equipo donde se encuentra **Rancher server** instalado que en este caso es el mismo nodo.



Lo próximo que nos sale será esta pantalla donde podremos añadir un clúster pinchando en el botón **Add Cluster**.

The screenshot shows the Rancher web interface. At the top, there's a navigation bar with links for Global, Clusters, Apps, Settings, Security, and Tools. Below the navigation is a search bar and a table header with columns for State, Cluster Name, Provider, Nodes, CPU, and RAM. A message at the bottom says "There are no clusters defined." The "Add Cluster" button in the top right corner is highlighted with a red box.

A continuación podremos elegir un clúster ya existente o podemos crear uno con **RKE (Rancher Kubernetes Engine)**. En este caso se creara un nuevo clúster.

This screenshot shows the "Add Cluster - Select Cluster Type" page. It has two main sections: "From existing nodes (Custom)" and "Import an existing cluster". The "From existing nodes (Custom)" section is highlighted with a red box. Below these are two sections: "With RKE and new nodes in an infrastructure provider" (listing Amazon EC2, Azure, DigitalOcean, Linode, and vSphere) and "With a hosted Kubernetes provider" (listing Amazon EKS, Azure AKS, and Google GKE). The background features large, semi-transparent circles labeled "building", "testing", "operating", and "monitoring" around the central form.

Una vez elegido esto nos saldrá la siguiente pantalla donde podremos definir el nombre del clúster que será visible en Rancher, los miembros que tendrán acceso y los que podrán administrarlo.

This screenshot shows the "Add Cluster - Custom" configuration form. It includes fields for the cluster name ("Nombre del Clúster") which is set to "clusterlocal", and a "Roles de los miembros" section where "Default Admin (admin)" is listed with the role "Cluster Owner". There are also sections for "Etiquetas y anotaciones" and "Opciones del Clúster", and a "Editar como YAML" button at the bottom right. The background features large, semi-transparent circles labeled "building", "testing", "operating", and "monitoring" around the central form.

En esta misma pantalla más para abajo nos saldrán las opciones de kubernetes, versión, tipo de red que vamos a instalar en el clúster para que se comuniquen los pods, que en este caso será **Flannel**. Además podremos seleccionar en que sistema va a estar instalado nuestro clúster.

Cluster Options

Edit as YAML

Kubernetes Options
Customize the kubernetes cluster options

Expand All

Kubernetes Version
v1.17.6-rancher2-1

Network Provider
Flannel

Windows Support
 Enabled
 Disabled

Project Network Isolation
 Enabled
 Disabled

Cloud Provider ⓘ
If your cloud provider is not listed, please use the **Custom** option.

None
 Amazon
 Azure
 Custom
 External

Private Registry
Configure a default private registry for this cluster. When enabled, all images required for cluster provisioning and system add-ons startup will be pulled from this registry.

Disabled
 Enabled



La siguiente opción que nos podemos encontrar será deshabilitar el registro privado en nuestro clúster que en este caso lo vamos a dejar por defecto.

En la sección **Avanced Options** podremos encontrarnos las siguientes opciones entre otras cosas:

The diagram shows the 'Advanced Options' section of a cluster configuration interface. It includes settings for Nginx Ingress, Node Port Range, Metrics Server Monitoring, Pod Security Policy Support, Docker version on nodes, Docker Root Directory, etcd Snapshot Backup Target, Recurring etcd Snapshot, Scheduled CIS Scan, Maximum Worker Nodes Unavailable, and Drain nodes.

- 1- Habilitar proxy nginx ingress.
- 2- Definir puertos Node Port.
- 3- Habilitar las métricas de monitoreo.
- 4- Soportar versiones de docker en los nodos.
- 5- Donde se van a guardar las snapshot de **etcd**.
- 6- Nodos máximos de Worker no disponibles (%).

En la siguiente pantalla podremos elegir los roles de los nodos que se van a unir a nuestro clúster, una vez se elijan los roles se creara un comando el cual tendremos que ejecutar en los nodos que tendrán ese rol.

Add Cluster - Custom

Cluster Options

Customize Node Run Command
Editing node options will update the command you will run on your existing machines

Node Options
Choose what roles the node will have in the cluster

Node Role

etcd Control Plane Worker
[Show advanced options](#)

Run this command on one or more existing machines already running a supported version of Docker.

```
sudo docker run -d --privileged --restart=unless-stopped --net=host -v /etc/kubernetes:/etc/kubernetes -v /var/run:/var/run rancher/rancher-agent:v2.4.4 --server https://192.168.1.10:8443 --token g2t5dmxfjzxhupmwf6jkw4z879x1qgpmvc1ztzpf8jfnh6zvwm --ca-checksum 917c7533cccd1dbf874ceba7fa3cf65b373c009431fb0c3a2ba29ac4dac28c3 --etcd --controlplane
```

Add Cluster - Custom

Cluster Options

Customize Node Run Command
Editing node options will update the command you will run on your existing machines

Node Options
Choose what roles the node will have in the cluster

Node Role

etcd Control Plane Worker
[Show advanced options](#)

Run this command on one or more existing machines already running a supported version of Docker.

```
sudo docker run -d --privileged --restart=unless-stopped --net=host -v /etc/kubernetes:/etc/kubernetes -v /var/run:/var/run rancher/rancher-agent:v2.4.4 --server https://192.168.1.10:8443 --token g2t5dmxfjzxhupmwf6jkw4z879x1qgpmvc1ztzpf8jfnh6zvwm --ca-checksum 917c7533cccd1dbf874ceba7fa3cf65b373c009431fb0c3a2ba29ac4dac28c3
```

El comando de los roles de **etcd** y **control plane** será ejecutado en el nodo **jupiter**.

```
root@jupiter:~# docker run -d --privileged --restart=unless-stopped --net=host -v /etc/kubernetes:/etc/kubernetes
-v /var/run:/var/run rancher/rancher-agent:v2.4.4 --server https://192.168.1.10:8443 --token 8rbxhg4t4n8lmsmhhlvnv
hvxbgrpr7k6ckksdrp2hmg4d9f44ww2hr --ca-checksum 917c7533cccd1dbf874cbea7fa3c8f65b373c009431fb0c3a2ba29ac4dac28c3 -
-etcd --controlPlane
ba29d0411d120a2e20e3bd1bf05cdb101458a157826da95021470d793490ee0c
```

Y el comando del rol de **Worker** será ejecutado en los nodos **Saturno** y **Urano**.

```
root@saturno:~# docker run -d --privileged --restart=unless-stopped --net=host -v /etc/kubernetes:/etc/kubernetes
-v /var/run:/var/run rancher/rancher-agent:v2.4.4 --server https://192.168.1.10:8443 --token 8rbxhg4t4n8lmsmhhlvnv
hvxbgrpr7k6ckksdrp2hmg4d9f44ww2hr --ca-checksum 917c7533cccd1dbf874cbea7fa3c8f65b373c009431fb0c3a2ba29ac4dac28c3 -
-worker
30ff45027f5f9f657e96d4187625ela92625d0640032a49c92cd7704ab255c2e2
```

Una vez se hayan ejecutado los comandos en los distintos nodos saldrá el siguiente mensaje en la pantalla anterior.

Ya unidos los nodos le daremos al botón **Done** para finalizar la configuración del clúster.

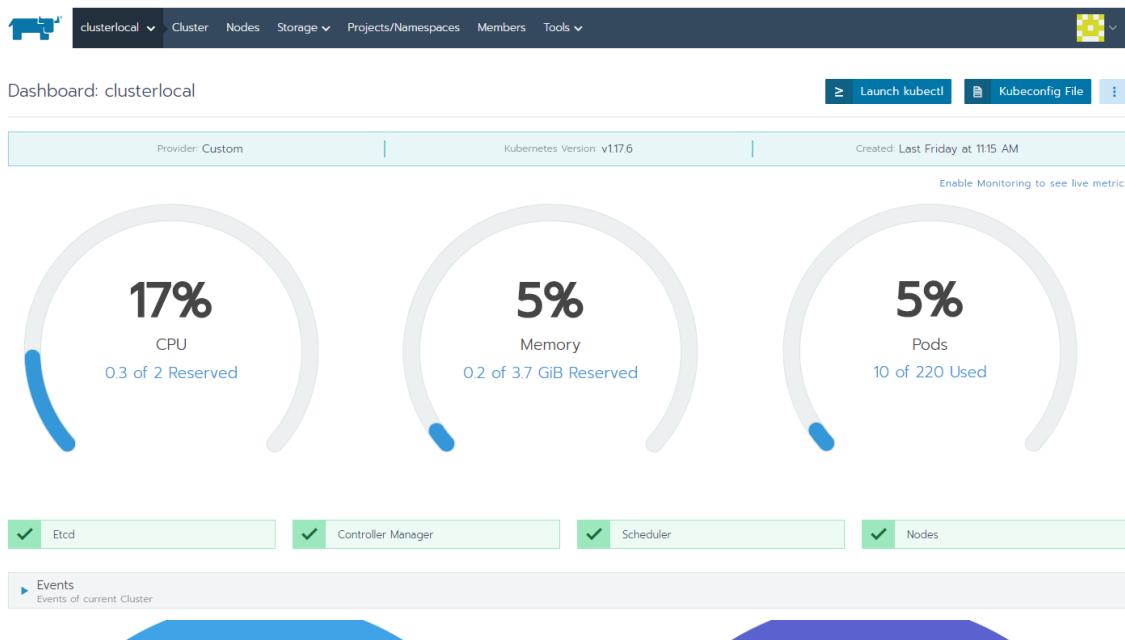


En la pestaña de clúster encontraremos el estado de **provisioning** el cual quiere decir que se está aprovisionando el clúster.

En la pestaña **Nodes** podremos ver los nodos que han sido añadidos al clúster.

	Name	Roles	Version	CPU	RAM	Pods
<input type="checkbox"/>	jupiter 192.168.1.10	etcd, Control Plane	n/a	n/a	n/a	n/a
<input type="checkbox"/>	saturno 192.168.1.11	Worker	n/a	n/a	n/a	n/a
<input type="checkbox"/>	urano 192.168.1.12	Worker	n/a	n/a	n/a	n/a

Una vez provisionado podremos ver el **dashboard** de nuestro clúster.

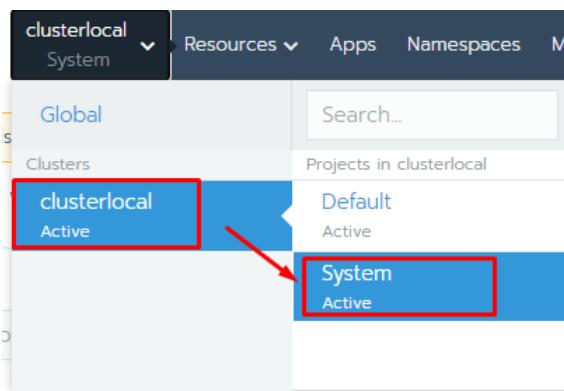


En este clúster tenemos creados dos proyectos, **default** y **System**. Estos proyectos pueden tener varios namespaces los cuales podemos crear nosotros.

Normalmente se desplegarán los pods en el proyecto **Default** a no ser que sean del sistema que será en **System**.

The screenshot shows the 'Projects/Namespace' section of the Cattle interface. The top navigation bar includes 'clusterlocal', 'Cluster', 'Nodes', 'Storage', 'Projects/Namespace' (which is highlighted with a red box), 'Members', and 'Tools'. Below the navigation is a search bar and a 'Created' filter. A large blue decorative graphic with the words 'code', 'play', 'deploy', 'release', and 'run' is overlaid on the interface. The main table lists two projects: 'Project: Default' (Default project created for the cluster) and 'Project: System' (System project created for the cluster). The 'System' project contains several namespaces: 'cattle-system', 'ingress-nginx', 'kube-node-lease', 'kube-public', 'kube-system', and 'security-scan'. Each namespace entry includes an 'Active' status indicator, a creation date ('Last Friday at 11:36 AM' or '11:33 AM'), and a three-dot menu icon.

Para ver lo que contiene el proyecto **system** podremos ver todos los namespaces, pods, deployment, loadbalancer dirigidos al sistema.



En el proyecto **system** de nuestro clúster se encontraran los namespaces, deployment, services, volumenes relacionados al clúster ya sea dns, red, monitorización o métricas entre otras cosas.

Namespace	Nombre	Imagen	Estado	Detalles
cattle-system	cattle-cluster-agent	rancher/rancher-agent:v2.4.4	Active	1 Puerto / Creado en 2 days ago / Reinicios de Pod: 1
cattle-system	cattle-node-agent	rancher/rancher-agent:v2.4.4	Active	3 Puertos / Creado en 2 days ago / Reinicios de Pod: 3
cattle-system	kube-api-auth	rancher/kube-api-auth:v0.14	Active	1 Puerto / Creado en 2 days ago / Reinicios de Pod: 1
ingress-nginx	default-http-backend	rancher/nginx-ingress-controller-defaultbackend15-rancher1	Active	1 Puerto / Creado en 2 days ago / Reinicios de Pod: 1
ingress-nginx	nginx-ingress-controller	rancher/nginx-ingress-controllernginx-0-320-rancher1	Active	443/tcp, 443/tcp, 80/tcp, 80/tcp / Creado en 2 days ago / Reinicios de Pod: 2
kube-system	coredns	rancher/coredns-coredns:1.6.5	Inactive	ReplicaSet "coredns-7c5566588d" has timed out: progressing.
kube-system	coredns-autoscaler	rancher/cluster-proportional-autoscaler:1.1	Active	1 Puerto / Creado en 2 days ago / Reinicios de Pod: 0
kube-system	kube-flannel	rancher/coreos-flannel:v0.11.0-rancher1-1	Active	3 Puertos / Creado en 2 days ago / Reinicios de Pod: 5
kube-system	metrics-server	rancher/metrics-server:v0.3.6	Active	1 Puerto / Creado en 2 days ago / Reinicios de Pod: 10

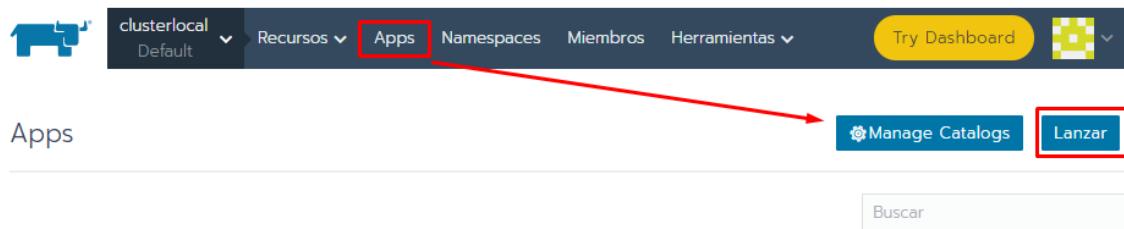
En el proyecto **Default** nos encontraremos los objetos que habremos creado nosotros por defecto.

There are no workloads deployed.

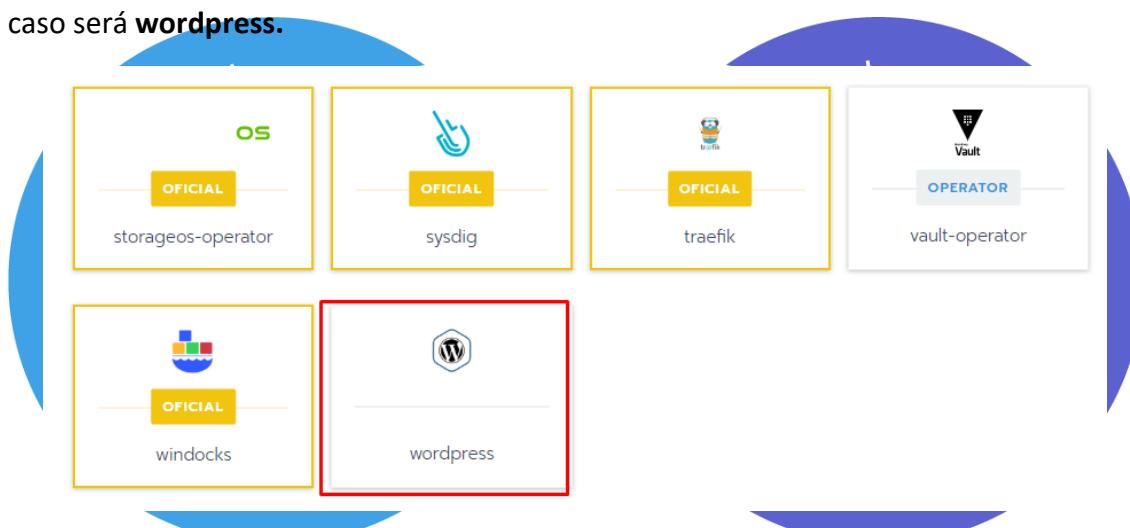
9.5.1. Despliegue de aplicación.

Para poder desplegar una aplicación hay varias formas en Rancher pero en este caso de explicarán 2. Crear una aplicación a partir de un YAML ya creado o desplegar una aplicación del catálogo que ya trae Rancher.

En este caso desplegaremos una aplicación con el catálogo que trae por defecto Rancher. Para ello nos iremos al apartado de **Apps** y le daremos al botón de **Lanzar**.



Una vez dentro podremos ejecutar la aplicación que veamos conveniente, que en este caso será **wordpress**.



Dentro de la plantilla de wordpress podremos definir varios parámetros como el nombre que va a tener esta aplicación, si queremos hacer la instalación con Helm.

Catálogo: wordpress

The screenshot shows the Helm Catalog interface for the WordPress chart. At the top is the WordPress logo. Below it are sections for 'Descripción (predeterminado)' (Description) and 'Opciones de Configuración' (Configuration Options). The 'Nombre' (Name) is set to 'wordpress' and the 'Versión de la plantilla' (Template Version) is '9.0.3'. A link 'Seleccionar una versión de la plantilla para desplegar' (Select a template version to deploy) is present. The 'ESPACIO DE NOMBRES' (Namespace) section shows 'Namespaces *' with 'wordpress' selected. Under 'UTILIZAR UN NAMESPACE EXISTENTE' (Use existing namespace), there is a link 'Utilizar un namespace existente'. The 'HELM OPTIONS' section includes 'Helm Wait' (radio buttons for Verdadero or Falso, with Falso selected) and 'Helm Timeout' (set to 300 seconds). A note says 'A value in seconds to wait for Kubernetes commands to complete.' At the bottom right is a blue button labeled 'Editar como YAML' (Edit as YAML).

En esta misma pantalla podremos definir varios valores, como la imagen con la que se van a crear los pods, si queremos que sea la original o otra a nuestro gusto, los valores de usuario de wordpress, mysql y contraseñas de ambos.

Así como en la sección de **Services and Load Balancing**, podremos definir si queremos un servicio **load balancer** o **NodePort** así como **ClusterIP**, dependiendo de nuestro caso elegiremos uno u otro.

En este caso se ha elegido loadbalancer para exponer nuestra aplicación fuera mediante el **proxy ingress nginx**.

Use Default Image
 Verdadero Falso
 Use default Docker image

WORDPRESS SETTINGS

WordPress Username *	WordPress Password
<input type="text" value="frodo"/>	<input type="password" value="....."/> Generar
password will be auto-generated if not specified	
WordPress Admin Email *	WordPress Persistent Volume Enabled *
<input type="text" value="user@example.com"/>	<input type="radio"/> Verdadero <input checked="" type="radio"/> Falso
Enable persistent volume for WordPress	
Admin email	

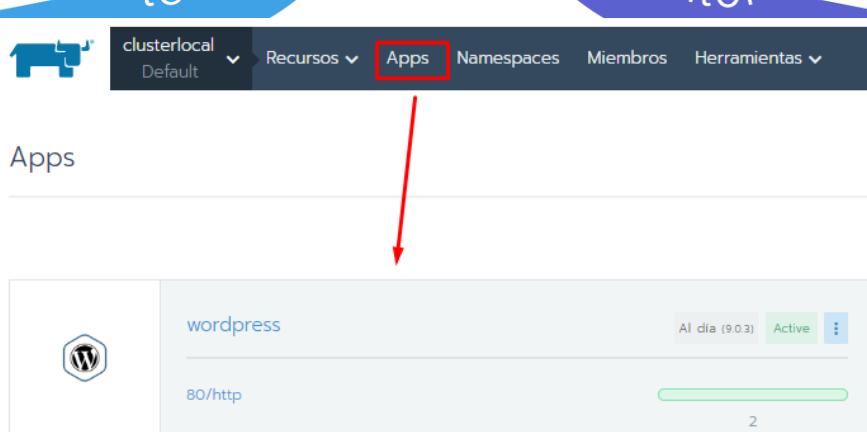
DATABASE SETTINGS

Install MariaDB *	MariaDB Database
<input checked="" type="radio"/> Verdadero <input type="radio"/> Falso	<input type="text" value="wordpress"/>
Deploy a database server as part of this deployment, or set to false and configure an external database connection.	
MariaDB User	MariaDB Password
<input type="text" value="wordpress"/>	<input type="password" value="....."/> Generar
password will be auto-generated if not specified	
MariaDB Root Password	MariaDB Persistent Volume Enabled *
<input type="password" value="....."/> Generar	<input type="radio"/> Verdadero <input checked="" type="radio"/> Falso
Enable persistent volume for MariaDB	
root user password, will be auto-generated if not specified	

SERVICES AND LOAD BALANCING

Expose app using Layer 7 Load Balancer <input checked="" type="radio"/> Verdadero <input type="radio"/> Falso	Hostname *
Expose app using Layer 7 Load Balancer - ingress	<input checked="" type="radio"/> Generar automáticamente un xip.io nombre de host
	<input type="radio"/> Especifique un nombre de host para usar
	Hostname to your WordPress installation

En la pestaña **Apps** encontraremos nuestra aplicación que está activa.



Si pinchamos dentro de ella podremos ver una serie secciones donde podremos ver si nuestra aplicación ha sido levantada, si hay algún pod, servicio o loadbalancer que no funciona entre otras cosas.

App: wordpress

Versión 9.0.3 Espacio de nombres wordpress Creado 9:58 AM

Notas
Instrucciones de cómo usar esta aplicación

Estado
Estado actual de las aplicaciones

Answers
Customized answers of this application

Endpoints
Endpoints públicos de esta aplicación

Cargas de trabajo
Cargas de trabajo creadas para esta aplicación

Reglas de Entrada
Reglas de entrada creadas para esta aplicación

Servicios
Servicios creados con esta aplicación

Volúmenes
Reclamaciones de volumen persistente creados con esta aplicación

Secretos
Secretos asociados con esta aplicación

Mapas de configuración
Config Maps asociados con esta aplicación

En la pestaña **Cargas de trabajo** podremos encontrar los deployment que han sido creados con nuestra aplicación, si pinchamos en **80/http** nos redirigirá a la página de wordpress.

Cargas de trabajo Balanceadores de carga Descubrimiento del servicio Volumenes Importar YAML Desplegar

Redeploy Pausar orquestación Descargar YAML Eliminar Buscar Escalar

Estado	Nombre	Imagen	Escalar
Active	wordpress 80/http	docker.io/bitnami/wordpress:5.3-debian-10-r43	1 Puerto / Creado en 3 minutos ago / Reinicios de Pod: 1
Active	wordpress-mariadb	docker.io/bitnami/mariadb:10.3.22-debian-10-r44	1 Puerto / Creado en 3 minutos ago / Reinicios de Pod: 0

No es seguro | wordpress.wordpress-2.192.168.1.11.xip.io

User's Blog!
Just another WordPress site

UNCATEGORIZED

Hello world!

By wordpress June 15, 2020 1 Comment

Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

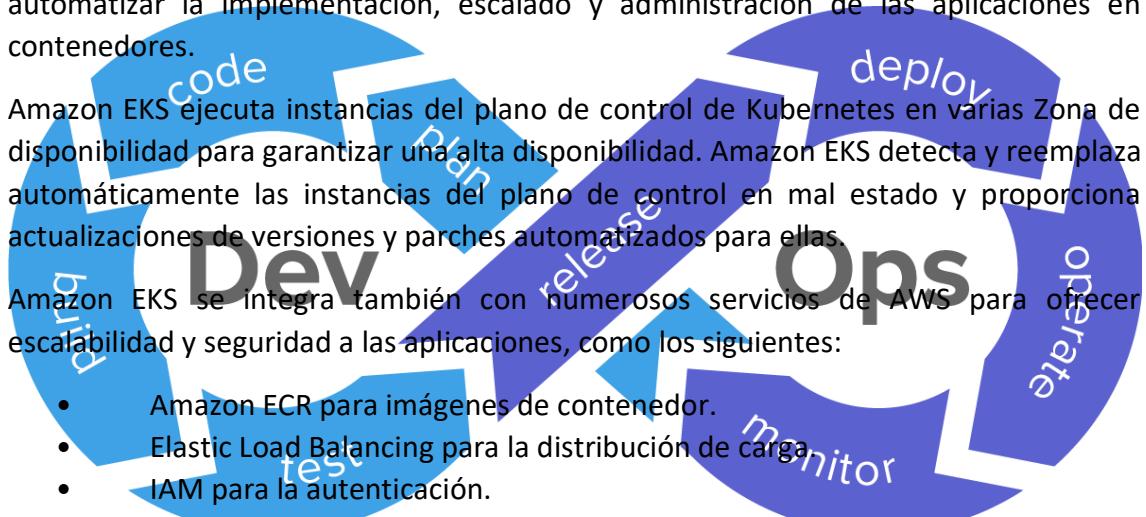
10. Esquema Cloud.

10.1. Amazon Elastic Kubernetes Service (EKS).



10.1.1. ¿Qué es Amazon EKS?

Amazon Elastic Kubernetes Service (Amazon EKS) es un servicio administrado que le permite ejecutar fácilmente Kubernetes en AWS sin necesidad de crear ni mantener su propio plano de control de Kubernetes. Kubernetes es un sistema de código abierto para automatizar la implementación, escalado y administración de las aplicaciones en contenedores.



Amazon EKS ejecuta instancias del plano de control de Kubernetes en varias Zona de disponibilidad para garantizar una alta disponibilidad. Amazon EKS detecta y reemplaza automáticamente las instancias del plano de control en mal estado y proporciona actualizaciones de versiones y parches automatizados para ellas.

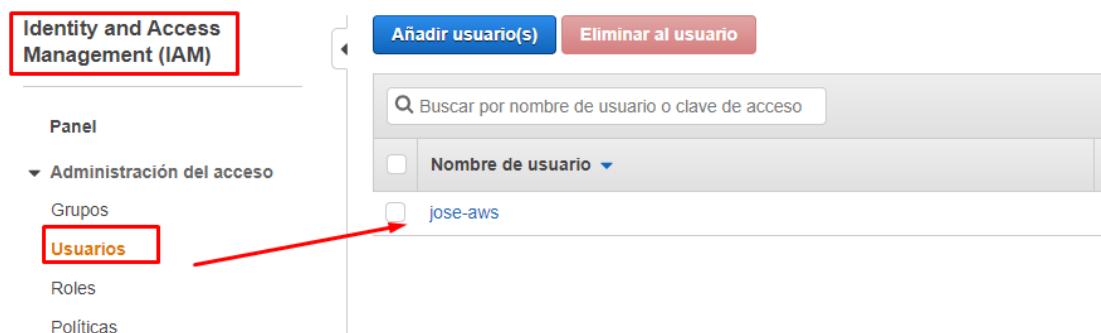
Amazon EKS se integra también con numerosos servicios de AWS para ofrecer escalabilidad y seguridad a las aplicaciones, como los siguientes:

- Amazon ECR para imágenes de contenedor.
- Elastic Load Balancing para la distribución de carga.
- IAM para la autenticación.
- Amazon VPC para el aislamiento.

10.1.2. Creación de clúster kubernetes con Ansible.

En este apartado se ha creado un clúster Amazon EKS con Ansible.

Para poder crear el cluster necesitaremos los parametros `aws_access_key` y `aws_secret_key` para poder acceder a AWS



La captura de pantalla muestra la interfaz de usuario de AWS Identity and Access Management (IAM). En la parte superior, hay un cuadro de diálogo con los botones 'Añadir usuario(s)' y 'Eliminar al usuario'. A la izquierda, un menú lateral titulado 'Panel' incluye 'Administración del acceso', 'Grupos', 'Roles' y 'Políticas'. El ítem 'Usuarios' está resaltado con un cuadro rojo. La lista principal muestra un usuario llamado 'jose-aws' con un cuadro rojo enfrente de su nombre. Una flecha roja apunta desde el cuadro rojo en el menú lateral hacia el cuadro rojo que rodea el nombre del usuario en la lista.

Para ello nos vamos al apartado IAM Management Console en AWS y creamos un nuevo usuario con permisos AdministratorAccess. Y creamos una nueva **Credencial**, que descargaremos en formato .csv

The screenshot shows the AWS IAM Management Console. On the left, there's a sidebar with 'Grupos', 'Roles', 'Políticas', 'Proveedores de identidad', 'Configuración de cuenta', 'Informes de acceso' (with 'Analizador de acceso' and 'Reglas de archivo' under it), 'Informe de credenciales', 'Actividad de la organización', and 'Políticas de control de condiciones'. The 'Usuarios' tab is selected and highlighted with a red box. On the right, the main panel shows the 'Permisos' tab selected. The 'Ruta' is shown as '/'. The 'Hora de creación' is '2020-03-06 19:46 UTC+0100'. Below the tabs, it says 'Políticas de permisos (1 política aplicada)'. There's a button 'Añadir permisos' and a dropdown 'Nombre de la política' set to 'AdministratorAccess'. A red arrow points from the bottom of the 'Nombre de la política' dropdown to the 'Credenciales de seguridad' tab. The 'Credenciales de seguridad' tab is also highlighted with a red box. Below it, there's a section 'Asociados desde el grupo' with 'AdministratorAccess' listed. At the bottom, there's a section 'Límite de permisos (no definido)'.

```
root@debiangraf:~/cluster-aws-eks# export AWS_ACCESS_KEY_ID='[REDACTED]'  
root@debiangraf:~/cluster-aws-eks# export AWS_SECRET_ACCESS_KEY='[REDACTED]'
```

Para aprovisionar una instancia EC2, deberemos descargar con pip los siguientes paquetes: **boto**.

```
root@debiangraf:~/cluster-aws-eks# pip install boto
```

Una vez explicado los yaml de instalación se procederá a provisionar la infraestructura EKS con el comando **ansible-playbook**.

```
root@debiangraf:~/cluster-aws-eks# ansible-playbook -i inventory main.yml -vvv  
Using /etc/ansible/ansible.cfg as config file  
Loading callback plugin default of type stdout, v2.0 from /usr/lib/python2.7/dist-packages/ansible/plugins/callback/_init__.pyc  
  
PLAYBOOK: main.yml ****  
1 plays in main.yml  
  
PLAY [localhost] ****  
  
TASK [Ensure VPC exists via CloudFormation.] ****  
task path: /root/cluster-aws-eks/main.yml:9  
Using module file /usr/lib/python2.7/dist-packages/ansible/modules/core/cloud/amazon/cloudformation.py  
<127.0.0.1> ESTABLISH LOCAL CONNECTION FOR USER: root  
<127.0.0.1> EXEC /bin/sh -c '( umask 77 && mkdir -p `echo ~/.ansible/tmp/ansible-tmp-1592311392.01-4507094427010` `echo ~/.ansible/tmp/ansible-tmp-1592311392.01-4507094427010` )&& rm -rf /tmp/ansible-tmp-1592311392.01-4507094427010/*'
```

Comprobaremos que ha terminado con éxito.

```
        "last_updated_time": null,  
        "logical_resource_id": "NodeInstanceRole",  
        "physical_resource_id": "eks-pasir-nodegroup-NodeInstanceRole-1R09P3JHRMGDW",  
        "resource_type": "AWS::IAM::Role",  
        "status": "CREATE_COMPLETE",  
        "status_reason": null  
    }  
]  
}  
  
PLAY RECAP ****  
127.0.0.1 : ok=5      changed=3      unreachable=0      failed=0
```

Nos iremos a Amazon EKS y comprobaremos que se ha creado el clúster de kubernetes.

The screenshot shows the AWS CloudWatch Metrics interface. A line graph tracks 'AWS Lambda' metrics over time. The Y-axis represents metrics like 'Invocations' and 'Errors', and the X-axis represents time. The 'Invocations' metric shows a sharp increase from approximately 100 to over 1000, while 'Errors' remains low. Other metrics like 'Latency' and 'Throughput' also show corresponding growth.

Comprobaremos que se han creado las redes con éxito.

The screenshot shows the AWS EKS Cluster Networking configuration page for a cluster named 'eks-pasir'. The 'Networking' tab is selected. It displays the VPC configuration (vpc-01cf77ff3c08aa53f), which includes three subnets: subnet-0adb9066b3125abd, subnet-0de23fe1649390f2d, and subnet-0a98e3e8c1660c2d3. It also shows the Cluster security group (sg-0abebefdf315eba5a) and the API server endpoint access, which is set to 'Public' with a public access source whitelist of 0.0.0.0/0 (open to all traffic).



Amazon EKS utiliza IAM para proporcionar autenticación al clúster de Kubernetes a través del **autenticador de IAM de AES para Kubernetes**.

Para la instalación haremos los siguientes pasos.

1. Descargaremos el binario **aws-iam-authenticator** ofrecido por Amazon EKS desde Amazon S3.

```
root@debiangraf:~# curl -o aws-iam-authenticator https://amazon-eks.s3.us-west-2.amazonaws.com/1.16.8/2020-04-16/bin/linux/amd64/aws-iam-authenticator
```

2. Ejecutaremos permisos de ejecución al binario que hemos descargado.

```
root@debiangraf:~# chmod +x ./aws-iam-authenticator
```

3. Copiaremos el binario en una carpeta en **\$PATH**. Se recomienda que se cree **\$HOME/bin/aws-iam-authenticator** y se asegure de que **\$HOME/bin** viene en lugar en la variable **\$PATH**.

```
root@debiangraf:~# mkdir -p $HOME/bin && cp ./aws-iam-authenticator $HOME/bin/aws-iam-authenticator && export PATH=$PATH:$HOME/bin
```

4. Agregaremos **\$HOME/bin** a la variable de entorno **PATH**.

```
root@debiangraf:~# echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc
```

5. Comprobaremos que el binario **aws-iam-authenticator** funciona con el siguiente comando.

```
root@debiangraf:~# aws-iam-authenticator help
A tool to authenticate to Kubernetes using AWS IAM credentials

Usage:
  aws-iam-authenticator [command]

Available Commands:
  help      Help about any command
  init      Pre-generate certificate, private key, and kubeconfig files for the server.
  server    Run a webhook validation server suitable that validates tokens using AWS IAM
  token     Authenticate using AWS IAM and get token for Kubernetes
  verify    Verify a token for debugging purpose
  version   Version will output the current build information
```

10.1.2.2. Instalación de kubectl.

Kubernetes utiliza la utilidad de línea de comandos **kubectl** para comunicarse con el servidor de la API del clúster.

Descargaremos el binario **kubectl** incluido en Amazon EKS.

```
root@debiangraf:~# curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.16.8/2020-04-16/bin/linux/amd64/kubectl
```

Ejecutaremos permisos de ejecución al binario.

D
root@debiangraf:~# chmod +x ./kubectl

Coparemos el binario en una carpeta en **PATH**. Se recomienda que se cree **\$HOME/bin/kubectl** y se asegure que **\$HOME/bin** es el primer elemento de **\$PATH**.

```
root@debiangraf:~# mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/bin
```

10.1.2.1. Instalación de AWS CLI.

Descargaremos el archivo comprimido de la página oficial de AWS.

```
root@debiangraf:~# curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
```

Descomprimimos el instalador.

root@debiangraf:~# unzip awscliv2.zip

Ejecutaremos el programa de instalación y una vez terminado comprobaremos que

```
root@debiangraf:~# ./aws/install
You can now run: /usr/local/bin/aws --version
root@debiangraf:~# aws --version
aws-cli/2.0.22 Python/3.7.3 Linux/4.9.0-12-amd64 botocore/2.0.0dev26
```

A continuación crearemos un kubeconfig archivo para EKS.

```
root@debiangraf:~# aws eks --region us-east-1 update-kubeconfig --name eks-pasir --kubeconfig ~/.kube/eks-pasir
Added new context arn:aws:eks:us-east-1:915508415579:cluster/eks-pasir to /root/.kube/eks-pasir
```

Indicaremos a kubectl dónde encontrar el kubeconfig.

```
root@debiangraf:~# export KUBECONFIG=~/.kube/eks-pasir
```

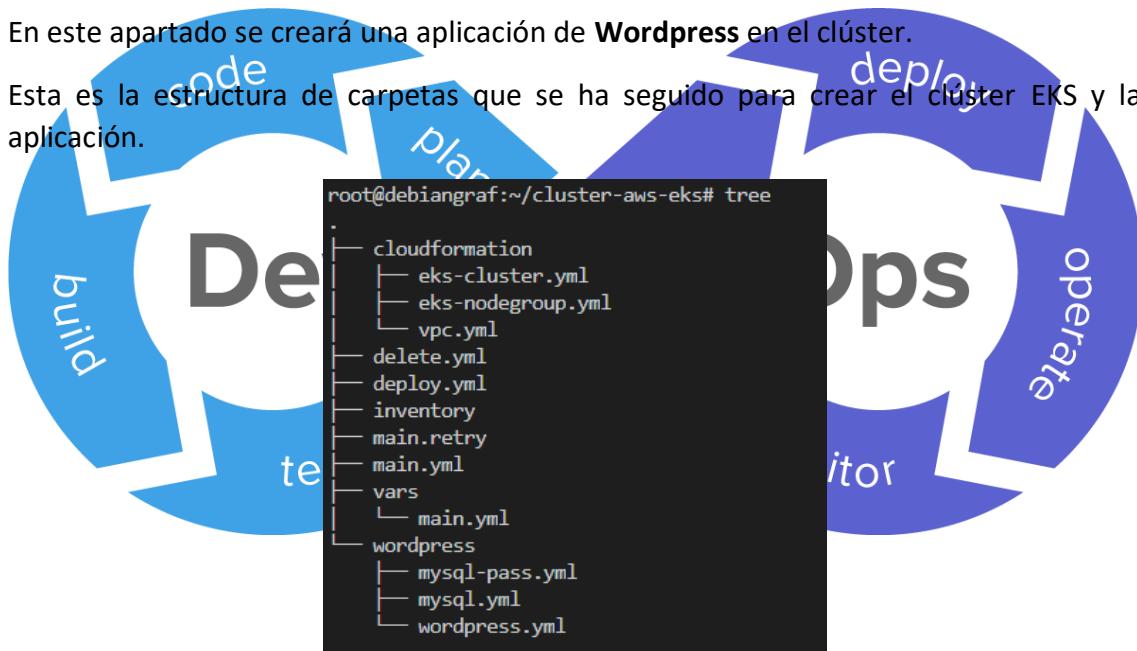
A continuación comprobamos que kubectl puede ver el clúster.

```
root@debiangraf:~# kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
kubernetes     ClusterIP  10.100.0.1  <none>        443/TCP   5h30m
```

10.1.3. Despliegue de aplicación.

En este apartado se creará una aplicación de **Wordpress** en el clúster.

Esta es la estructura de carpetas que se ha seguido para crear el clúster EKS y la aplicación.



Los scripts de ansible serán subido a mi github: <https://github.com/JoseDiazGomez>

Una vez hechos todos los scripts de la aplicación de wordpress ejecutaremos el siguiente comando.

```
root@debiangraf:~/cluster-aws-eks# ansible-playbook -i inventory deploy.yml -vvvv
ansible-playbook 2.9.9
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/root/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/dist-packages/ansible
  executable location = /usr/bin/ansible-playbook
  python version = 2.7.13 (default, Sep 26 2018, 18:42:22) [GCC 6.3.0 20170516]
Using /etc/ansible/ansible.cfg as config file
setting up inventory plugins
host_list declined parsing /root/cluster-aws-eks/inventory as it did not pass its verify_file() method
script declined parsing /root/cluster-aws-eks/inventory as it did not pass its verify_file() method
auto declined parsing /root/cluster-aws-eks/inventory as it did not pass its verify_file() method
Set default localhost to 127.0.0.1
```

Comprobaremos que ha finalizado de aprovisionar el playbook.

```

    "aaeee4e0149c6490f98c3e58e4afabbc-1938333059.us-east-1.elb.amazonaws.com."
  },
  "vpc_id": null,
  "wait": true,
  "wait_timeout": 300,
  "weight": null,
  "zone": "joseadiaz.tk"
}
}
}
META: ran handlers
META: ran handlers

PLAY RECAP ****
127.0.0.1 : ok=7   changed=1   unreachable=0   failed=0   skipped=0   rescued=0   ignored=0

```

Comprobaremos que se han creado los nodos que forman parte del clúster.

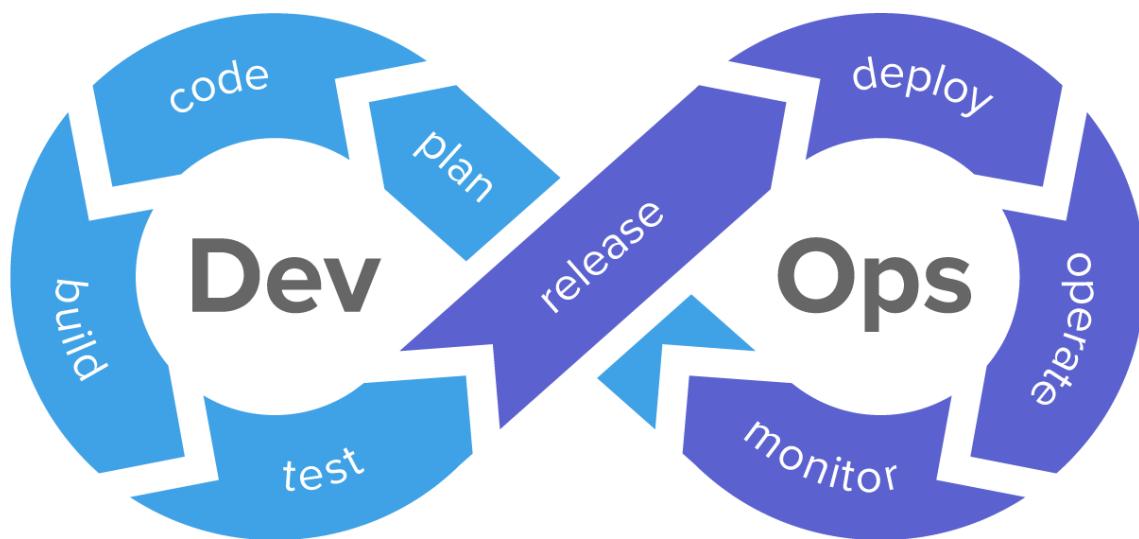
<input type="checkbox"/>	i-061b5785a54095c79	t3.medium	us-east-1a	● running	● 2/2 checks ...	None	● ec2-3-83-65-212.comp...	3.83.65.212	-	disabled	June 11
<input type="checkbox"/>	i-09add464bd678ba	t3.medium	us-east-1b	● running	● 2/2 checks ...	None	● ec2-54-91-103-225.co...	54.91.103.225	-	disabled	June 11
<input type="checkbox"/>	i-0e810eff56a156cc0	t3.medium	us-east-1c	● running	● 2/2 checks ...	None	● ec2-52-99-217-228.co...	52.99.217.228	-	disabled	June 11

Además comprobaremos que se ha creado nuestro balanceador de carga.

Comprobaremos que copiando el DNS del balanceador de carga podremos acceder a nuestro wordpress.

11. Conclusiones.

En este trabajo se ha visto las distintas herramientas para administrar Docker y hacer infraestructuras de ella. Debido a falta de tiempo no se ha profundizado más en el tema. Pero es una rama a la que me gustaría pertenecer además se seguir formándome de ella.



WEBGRAFÍA

Docker Swarm

<https://openwebinars.net/academia/aprende/docker-compose-swarm/?enroll=1>

<https://www.nubersia.com/es/blog/kubernetes-vs-docker-swarm/>

<https://www.redhat.com/es/topics/microservices>

<https://docs.docker.com/engine/install/debian/>

<https://linuxconfig.org/how-to-configure-docker-swarm-with-multiple-docker-nodes-on-ubuntu-18-04#h2-swarm-concept-in-detail>

<https://www.ionos.es/digitalguide/servidores/know-how/docker-compose-y-swarm-gestion-multicontenedor/>

<https://docs.docker.com/swarm/install-manual/>

<https://ualmtorres.github.io/SeminarioDockerPresentacion/#truecrear-un-contenedor-apache>

<https://codefresh.io/docker-tutorial/deploy-docker-compose-v3-swarm-mode-cluster/>

Ansible

Dev

<https://github.com/ome/ansible-role-nfs-mount>

https://docs.ansible.com/ansible/latest/scenario_guides/guide_aws.html

<https://github.com/geerlingguy/ansible-for-kubernetes/tree/master/cluster-aws-eks>

https://docs.aws.amazon.com/es_es/eks/latest/userguide/install-aws-iam-authenticator.html

Ops

deploy

operate

monitor

release

plan

code

Kubernetes

<https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/>

<https://kubernetes.io/docs/setup/production-environment/container-runtimes/#docker>

<https://insujang.github.io/2019-11-21/installing-kubernetes-and-crio-in-debian/>

<https://juantrucupei.wordpress.com/2018/03/25/instalacion-de-kubernetes-en-linux-con-kubeadm/>

<https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>

<https://www.linode.com/docs/kubernetes/getting-started-with-kubernetes/>

Istio:

<https://istio.io/latest/docs/setup/getting-started/#download>

<https://istio.io/latest/docs/setup/additional-setup/config-profiles/>

<https://openwebinars.net/academia/aprende/istio/>

Rancher:

<https://www.howtoforge.com/tutorial/debian-rancher-docker-container-manager/>

<https://es.slideshare.net/AdrianGarciaCasas/orquestadores-para-las-cuatro-estaciones-swarm-kubernetes-mesos-y-rancher>

<https://rancher.com/docs/rancher/v2.x/en/installation/>

<https://rancher.com/docs/rancher/v1.6/en/hosts/>

https://docs.aws.amazon.com/es_es/eks/latest/userguide/install-aws-iam-authenticator.html

