

Introducción práctica a Kubernetes



Monolitos y Microservicios

Monolitos son aplicaciones enfocadas en generar un único entregable, la mayoría de productos inician con este enfoque, pero al momento de adicionar funcionalidades...

- Presentan problemas de desarrollo y despliegue
- Demasiadas interdependencias entre funcionalidades
- Fuertemente dependientes de la red y los recursos

La solución actual, es dividir por **Microservicios** enfocados al mismo dominio del negocio, donde sea más fácil modificar unidades de código, y desplegarlas independientemente.



Contenedores y Docker



Contenedores

Permite especificar con exactitud qué partes de tu entorno necesitas y cuales quieres ejecutar. Son fáciles de empaquetar y mover, usualmente usan los recursos del computador de una forma eficiente. Desacoplan una aplicación de sus dependencias del sistema operativo.



Docker

Virtualización a nivel de sistema operativo, es primordialmente desarrollado para ambientes Linux donde usa la aislación de recursos propias del Kernel, lo que le permite la ejecución independiente de contenedores en una sola instancia de Linux , evitando la sobrecarga de una máquina virtual.

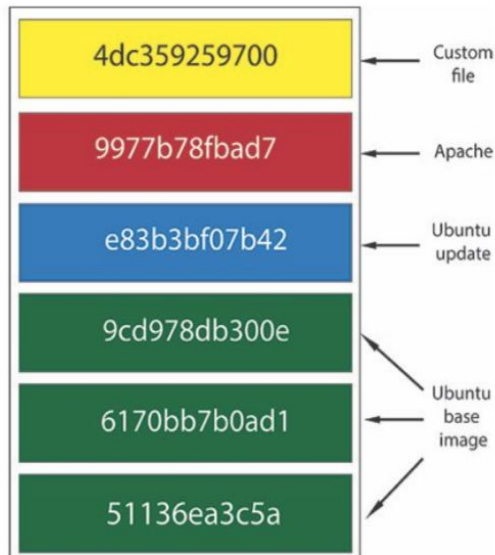


Docker: Dockerfile, Imágenes y Contenedores

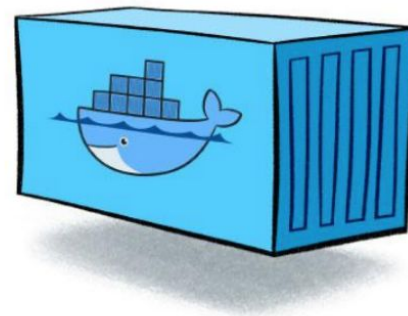
Dockerfile




Image



Container



Katacoda + Docker

**Katacoda**

More Information about KatacodaSearchBlogYour ProfileLog Out

Persisting Data Using Volumes

◀ Step 1 of 3 ▶

Step 1 - Data Volumes

Docker Volumes are created and assigned when containers are started. Data Volumes allow you to map a host directory to a container for sharing data.

This mapping is bi-directional. It allows data stored on the host to be accessed from within the container. It also means data saved by the process inside the container is persisted on the host.

Task

This example will use Redis as a way to persist data. Start a Redis container below, and create a data volume using the `-v` parameter. This specifies that any data saved inside the container to the `/data` directory should be persisted on the host in the directory `/docker/redis-data`.

```
docker run -v /docker/redis-data:/data \
  --name r1 -d redis \
  redis-server --appendonly yes ✓
```

Terminal +

Your Interactive Bash Terminal. A safe place to learn and execute commands.

```
[root@host01 ~]#
[root@host01 ~]# docker run -v /docker/redis-data:/data \
> --name r1 -d redis \
> redis-server --appendonly yes
7bcdcf8067391bb74e70b9d92b3520a49e1dc8be0e3b11e8852428107b09b2a2
[root@host01 ~]# cat data | docker exec -i r1 redis-cli --pipe
All data transferred. Waiting for the last reply...
Last reply received from server.
errors: 0, replies: 1
[root@host01 ~]# ls /docker/redis-data
appendonly.aof
[root@host01 ~]# docker run -v /docker/redis-data:/backup ubuntu ls /backup
appendonly.aof
[root@host01 ~]#
```

<https://www.katacoda.com/courses/docker/persisting-data-using-volumes>



Docker: Ej1. Contenedor Nginx

```
docker container run --rm -d -p 80:80 --name nginx nginx:1.8-alpine
```

```
[root@host01 ~]# docker container run --rm -d -p 80:80 --name nginx nginx:1.8-alpine
Unable to find image 'nginx:1.8-alpine' locally
1.8-alpine: Pulling from library/nginx
420890c9e918: Pull complete
a3ed95caeb02: Pull complete
ec7ddadc79c3: Pull complete
706e18ee5d76: Pull complete
f08749d57ceb: Pull complete
Digest: sha256:1b86fec4652825fca0345d7c5ba1370b02d2d705c3836c5fc67c27cce636e8e3
Status: Downloaded newer image for nginx:1.8-alpine
b3a18f4664e45f5fe2d823c0aeef48ce94ca5a60f7a01f2839d94a340b168cb6
[root@host01 ~]#
```

Click  luego “view port Host 1”, digitas el puerto **80** y

Display Port



Docker: Ej2. Nginx + Volume

```
mkdir website
```

```
cd website
```

```
vim index.html
```

```
<!DOCTYPE html>
<html>
<head>
  <title>Docker</title>
</head>
<body><center>
  
  <h1 style="color:blue">Baby steps with docker!</h1>
</center></body>
</html>
```

https://repl.it/@twogg_git/web0

```
docker container run --rm -p 80:80 --name nginx -v
/root/website:/usr/share/nginx/html nginx:1.8-alpine
```



Docker: Ej3. Httpd + Volume

```
docker container run --rm -d -p 8080:80 --name httpd -v /root/website/:/usr/local/apache2/htdocs/ httpd:2.4-alpine
```

```
[root@host01 ~]# docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
d73a22160ce8	httpd:2.4-alpine	"httpd-foreground"	57 seconds ago
Up 55 seconds	0.0.0.0:8080->80/tcp	httpd	
b3a18f4664e4	nginx:1.8-alpine	"nginx -g 'daemon of...'"	4 minutes ago
Up 4 minutes	0.0.0.0:80->80/tcp, 443/tcp	nginx	

Click  luego “view port Host 1”, digitas el puerto **8080** y

Display Port



Docker: Ej4. Httpd + Dockerfile

```
cd ..
```

```
vim Dockerfile
```

```
FROM httpd:2.4-alpine
```

```
ADD website/ /usr/local/apache2/htdocs/
```

```
EXPOSE 80
```

```
docker build -t httpd .
```

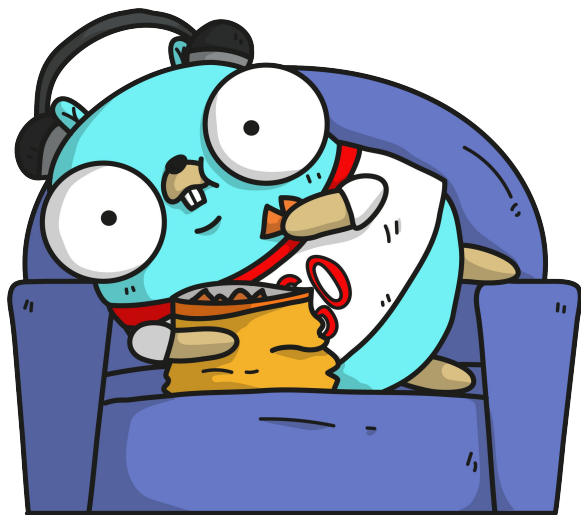
```
docker container run --rm -p 90:80 --name httpd httpd
```

Click  luego “view port Host 1”, digitas el puerto **90** y

Display Port



Golang

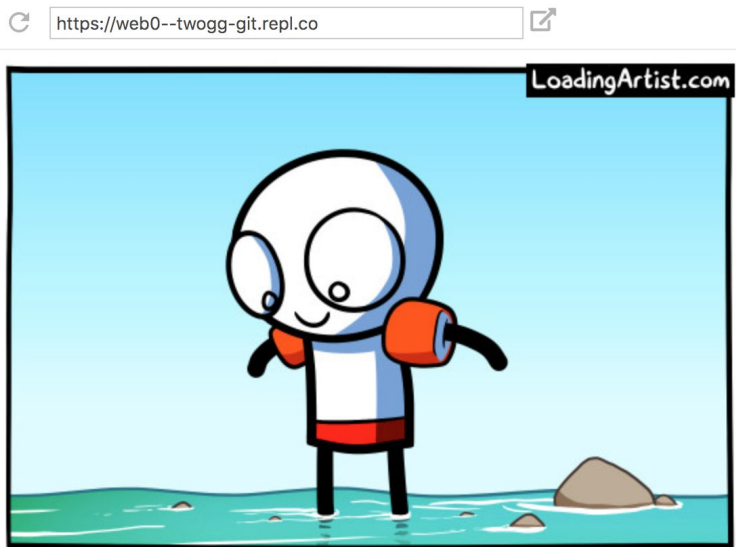


- Lenguaje Imperativo
- Fuertemente tipado
- Sintaxis similar a C, menos (), no ;
- No es necesaria una máquina virtual
- No hay clases, pero sí estructuras con métodos
- Funciones pueden retornar varios parámetros
- Diseñado para concurrencia



Golang: Ej 1. Aplicación Web

https://repl.it/@twogg_git/web0



Practicando con Kubernetes

Versión: `twogghub/web:1.0`

Golang: Ej 2. Docker + Golang

<https://github.com/twogg-git/k8s-workshop/tree/master/k8s>

```
FROM golang:1.10-alpine3.7 as builder
WORKDIR /go/src/k8s-workshop/k8s
COPY k8s.go .
RUN go get -d ./... && go build -o k8s .

FROM alpine:3.8
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=builder /go/src/k8s-workshop/k8s .

EXPOSE 8080
ENTRYPOINT ./k8s
```




GitHub



GitHub: Ej1. Docker + Httpd

<https://github.com/twogg-git/k8s-workshop/tree/master/docker>

 **twogg-git / k8s-workshop**

Unwatch ▾ 1

★ Star 0

🍴 Fork 0

<> Code

! Issues 0

🔗 Pull requests 0

📁 Projects 0

📖 Wiki

📊 Insights

⚙ Settings


Branch: master ▾ **k8s-workshop / docker /**

Create new file


Upload files

Find file


History

 **twogg-git** Refactor on starter version of files Latest commit 650b171 5 days ago

..

 **website**

Refactor on starter version of files 5 days ago

 **Dockerfile**

Adding kubernetes resources 5 days ago



GitHub: Ej2. Docker + Golang

<https://github.com/twogg-git/k8s-workshop/tree/master/k8s>

twogg-git / k8s-workshop

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Branch: master k8s-workshop / k8s / Create new file Upload files Find file History

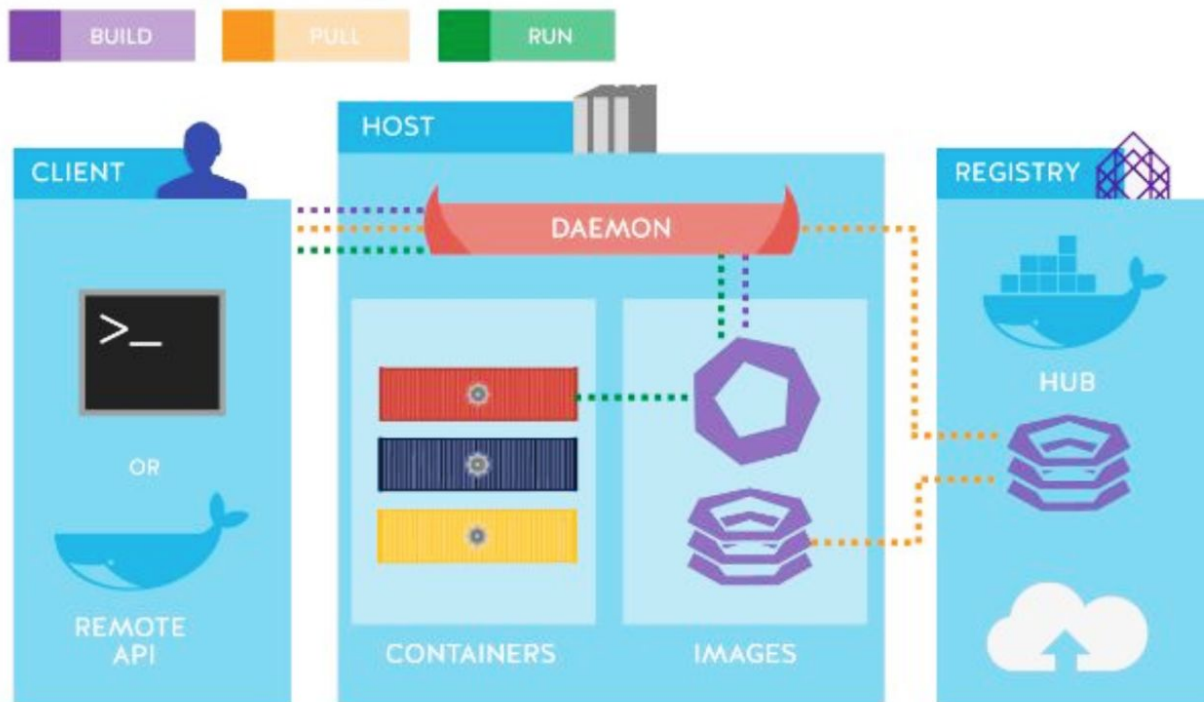
twogg-git Updating .go file to 1.3-liveness Latest commit 313fa66 4 days ago

..

Dockerfile	Adding QA version 1.1 image, fixing port to 8080	5 days ago
k8s.go	Updating .go file to 1.3-liveness	4 days ago

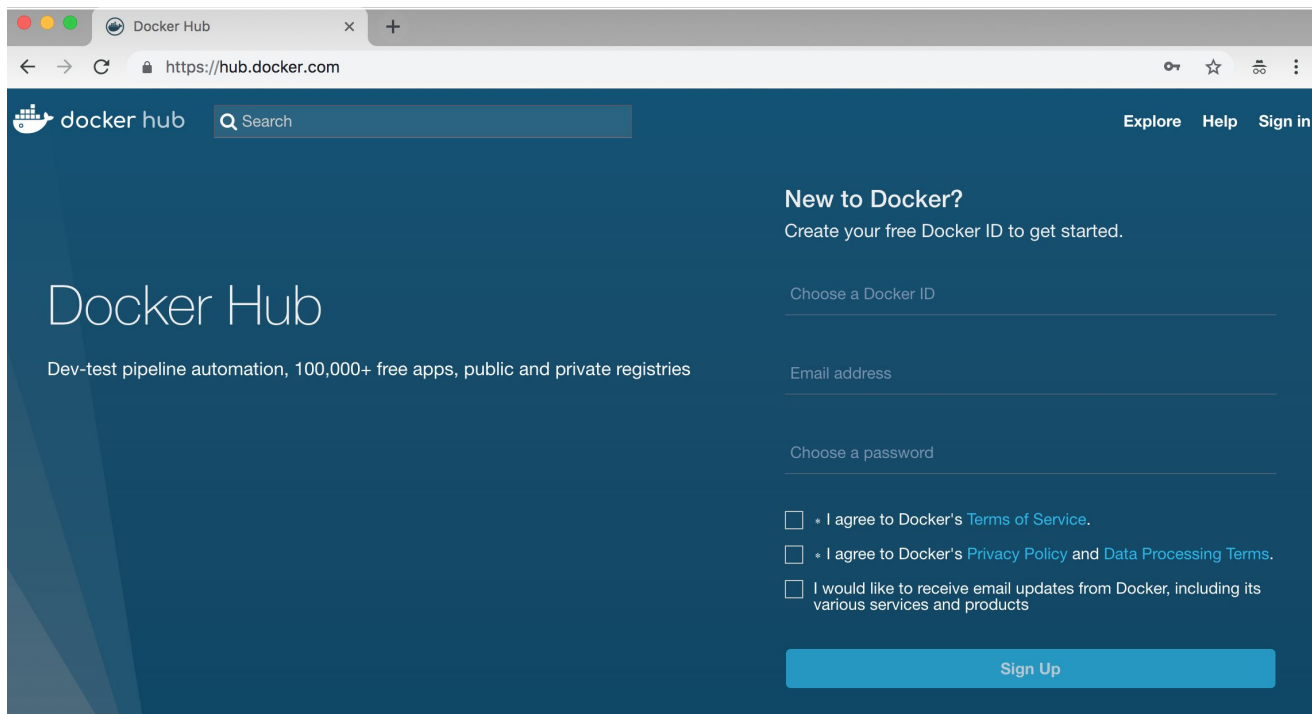


Docker Registry



Registry: Ej1. GitHub + Docker Hub

<https://hub.docker.com/u/twogghub/>



Docker Hub

Dev-test pipeline automation, 100,000+ free apps, public and private registries

New to Docker?
Create your free Docker ID to get started.

Choose a Docker ID

Email address

Choose a password

☐ • I agree to Docker's [Terms of Service](#).

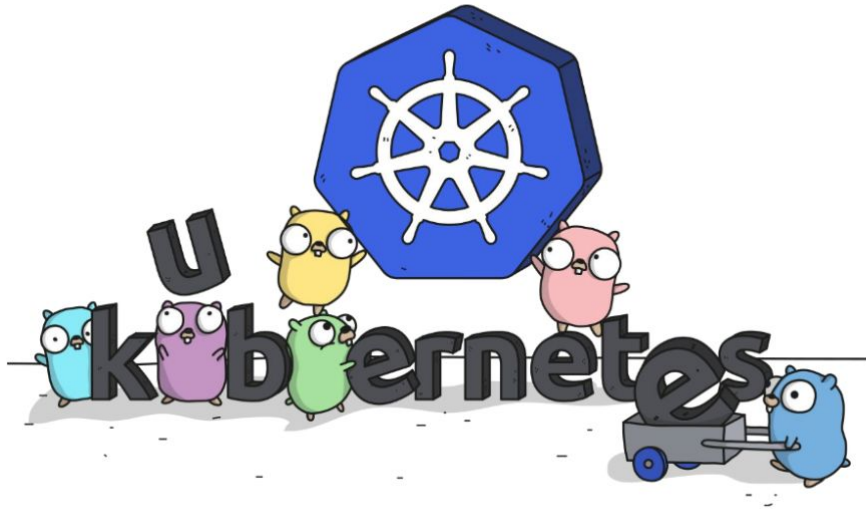
☐ • I agree to Docker's [Privacy Policy and Data Processing Terms](#).

☐ I would like to receive email updates from Docker, including its various services and products

Sign Up



Kubernetes



Sistema de orquestación de código abierto, para contenedores Docker creado por **Google**. Inició su desarrollo desde el 2003 como Omega, luego Borg.

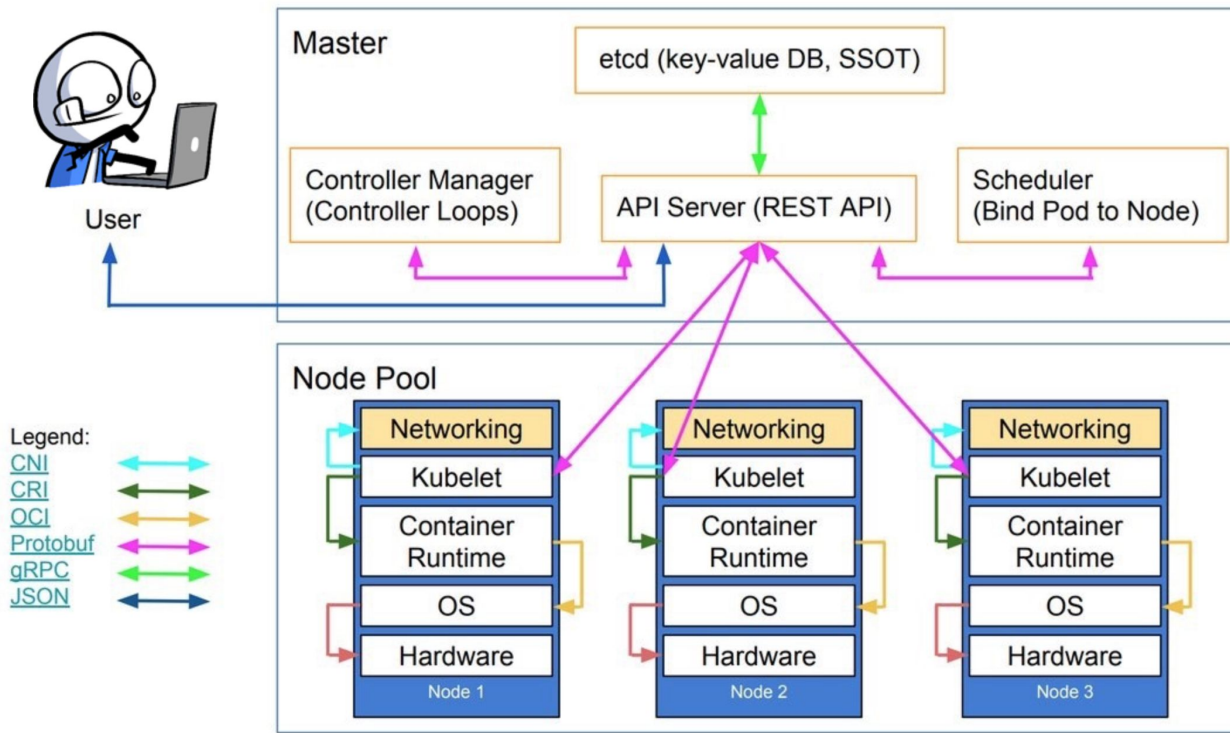
Maneja el despliegue de estos contenedores en un sistema Cluster, controlando los recursos y la red.

Simplifica las tareas de seguimiento, despliegue, escalabilidad, configuración, y versionamiento. Usa etiquetas para identificar objetos DevOps.

Ahora manejado por CNCF Cloud Native Computing Foundation. Soportado por AWS, Docker y Azure.



Arquitetura

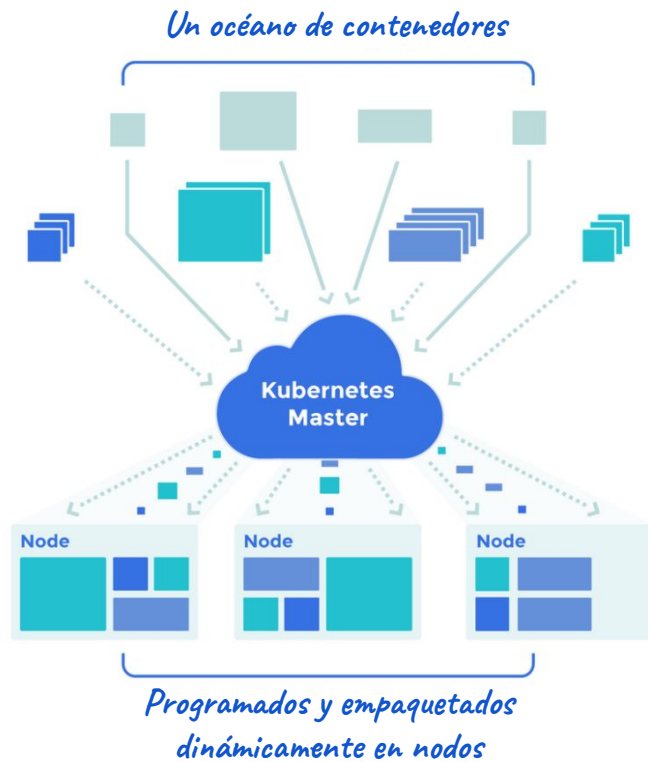


<https://speakerdeck.com/luxas/kubeadm-cluster-creation-internals-from-self-hosting-to-upgradability-and-ha>



Beneficios Destacados

- Agrupación de contenedores
- Auto-recuperación
- Auto-escalabilidad
- Administración DNS
- Balanceo de cargas
- Ejecución de actualizaciones
- Monitoreo de recursos y logs



Nodo Master



Panel de control del sistema, encargado de monitorear, hacer cambios, programar las tareas, y responder a los eventos. Compuesto por cuatro funcionalidades:

API Server: Único componente del nodo maestro que permite interacción del usuario, mediante la exposición de servicios REST y consumo de archivos JSON.

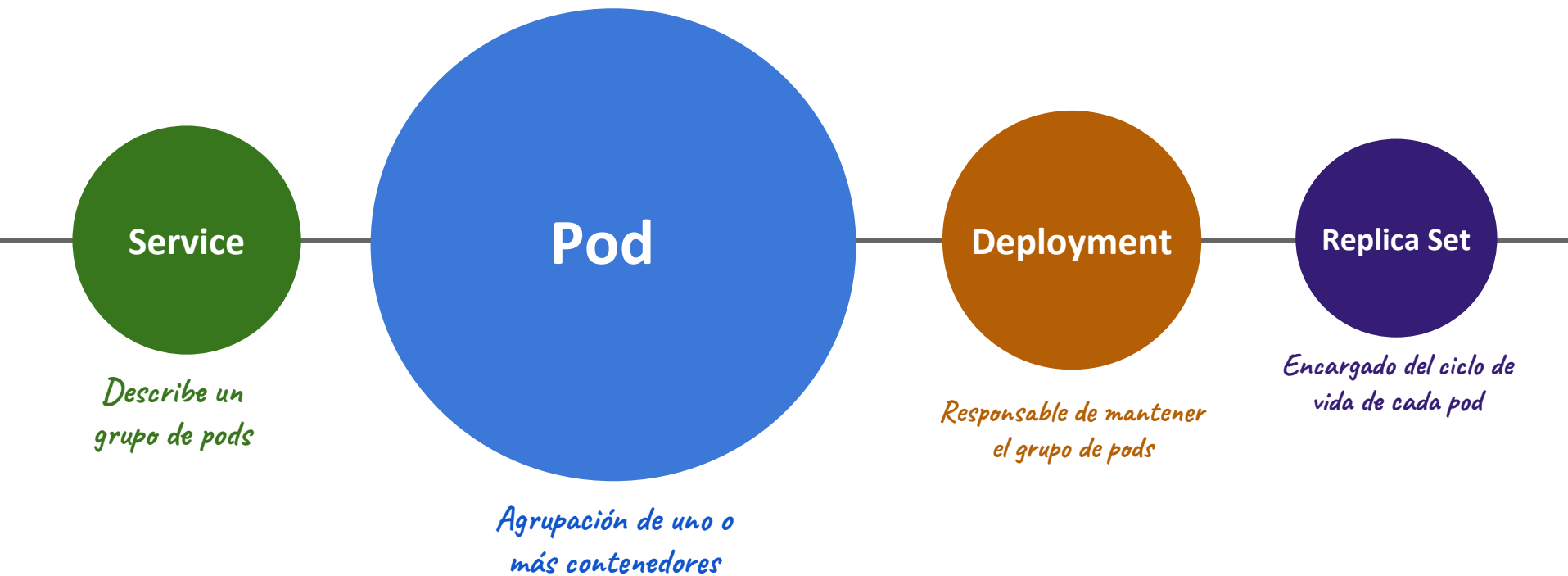
Data Store: Base de datos de tipo llave valor, tipo “etcd”, es robusta, consistente y brinda alta-disponibilidad.

Controller Manager: Administra las tareas del clúster, incluyendo los controladores de: nodos, replicación, endpoints, cuentas, y tokens.

Scheduler: Realiza el control y seguimiento de los nuevos pods (ya sea un uno o un grupo), y les asigna nodos.



Objetos Kubernetes



Kubernetes: Pod



Unidad básica de construcción de k8s, desplegados como una unidad en un nodo del cluster.

Agrupación de uno o más contenedores, los cuales comparten almacenamiento y servicios de red, como dirección IP y puerto.

Los pod son objetos con un **ciclo de vida definido**, luego de presentar una falla o ser programados para redespigie dejan de existir.

Kubernetes: Service

Abstracción que **define un grupo lógico de pods** y las reglas de acceso a los mismos.

Se encarga de ocultar múltiples pods detrás de una dirección de red.

El encargarse de la identificación de los pods, le permite realizar el balanceo de cargas.

Métodos de identificación de servicios:

- Variables de entorno (kubelet node agent)
- Servicio propio de DNS (cluster add-in)



Kubernetes: Replica Set



Es trabajo del replica set asegurarse que todos los pods requeridos están en ejecución. Pueden manejar unidades o grandes grupos de pods. Es recomendado **asignar etiquetas** de identificación por grupos y unidades.

Todo este proceso de control y despliegue está **automatizado**, y no es necesaria intervención de usuario más que para la configuración, lo cual facilita el manejo de la capacidad de escala.

Kubernetes: Deployment

Los archivos yaml puede ser creados manualmente o de forma automática al ejecutar comandos de despliegue.


Buenas practicas:

- Agregar chequeo de estados
- imagen:versión por contenedor
- Asignar etiquetas descriptivas
- Agregar validaciones de inicio
- Usar canales de comunicación



Controlador que **describe y mantiene el estado deseado** del grupo de pods y las réplicas, mediante el uso de archivos yaml.

Katacoda + Minikube

 **Katacoda**

More Information about Katacoda Search

Launch Single Node Kubernetes Cluster

◀ Step 1 of 4 ▶

Step 1 - Start Minikube

Minikube has been installed and configured in the environment. Check that it is properly installed, by running the `minikube version` command:

```
minikube version ↵
```

Start the cluster, by running the `minikube start` command:

```
minikube start ↵
```

Great! You now have a running Kubernetes cluster in your online terminal. Minikube started a virtual machine for you, and a Kubernetes cluster is now running in that VM.

CONTINUE

Terminal Dashboard +

Your Interactive Bash Terminal.

```
$  
$ minikube version  
minikube version: v0.25.0  
$ minikube start  
There is a newer version of minikube available (v0.25.2). Download it here:  
https://github.com/kubernetes/minikube/releases/tag/v0.25.2  
  
To disable this notification, run the following:  
minikube config set WantUpdateNotification false  
Starting local Kubernetes v1.9.0 cluster...  
Starting VM...  
Getting VM IP address...  
Moving files into cluster...  
Setting up certs...  
Connecting to cluster...  
Setting up kubeconfig...  
Starting cluster components...  
Kubectl is now configured to use the cluster.  
Loading cached images from config file.  
$
```

<https://www.katacoda.com/courses/kubernetes/launch-single-node-cluster>



Kubernetes: Ej1. Comandos Básicos

```
$ kubectl run k8s --image=twogghub/k8s-workshop:1.1-rolling
```

```
$ kubectl get deployment k8s --output wide
```

```
kubectl expose deployment k8s --port=8080 --external-ip=$(minikube ip)  
--type=LoadBalancer
```

En Katacoda click en + y “Select port to view on Host1”, para ver en un nuevo tab el puerto **8080**

```
$ kubectl describe service k8s
```

```
$ kubectl get pods,services,deployments --output wide
```



Kubernetes: Ej2. Archivos YAML

```
$ kubectl set image deployment k8s k8s=twogghub/k8s-workshop:1.2-yaml
```

```
$ kubectl scale --replicas=3 deployment k8s
```

La bandera **--watch** permite el seguimiento continuo de los eventos de un grupo de pods.

```
$ kubectl get pods --output wide --watch
```

```
kubectl get pods --output wide
```

```
$ kubectl delete pod <pod-name>
```



Kubernetes: Ej3. Estrategias de Actualización

<https://github.com/twogg-git/k8s-workshop/tree/master/yamls>

```
$ kubectl create -f https://url_repo/despliegues/yamls/k8s...
```

```
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 1
    maxUnavailable: 1
```

```
strategy:
  type: Recreate
```

```
kubectl delete deployments,services,pods,replicasets --selector="env=qa"
```



Kubernetes: Ej4. Validación de Estado

```
$ kubectl create -f https://url_repo/despliegues/yamls/k8sdp.yaml
```

```
$ kubectl scale --replicas=3 deployment k8sdp
```

La bandera **--watch** permite el seguimiento continuo de los eventos de un grupo de pods.

```
$ kubectl get pods --output wide --watch --selector="env=k8sdp"
```

```
$ kubectl set image deployment k8sdp k8sdp=twogghub/k8s-workshop:1.2-yaml
```

```
$ kubectl rollout undo deployment/k8sdp
```



Links y Referencias

- Introducción comandos Kubectl
<https://github.com/twogg-git/k8s-intro>
- Documentación oficial
<https://kubernetes.io/docs/home/>
- Kubernetes con ejemplos
<http://kubernetesbyexample.com/>
- Cursos Katacoda
<https://www.katacoda.com/learn>
- Dzone Introduccion a Kubernetes
<https://dzone.com/...kubernetes-in-10-min>
- Taller comandos Kubectl
https://repl.it/@twogg_git/
<https://github.com/twogg-git/k8s-workshop>



Catherin Cruz
@UserTwoGG

