DIGITALHOUSE

Arquitectura de Kubernetes

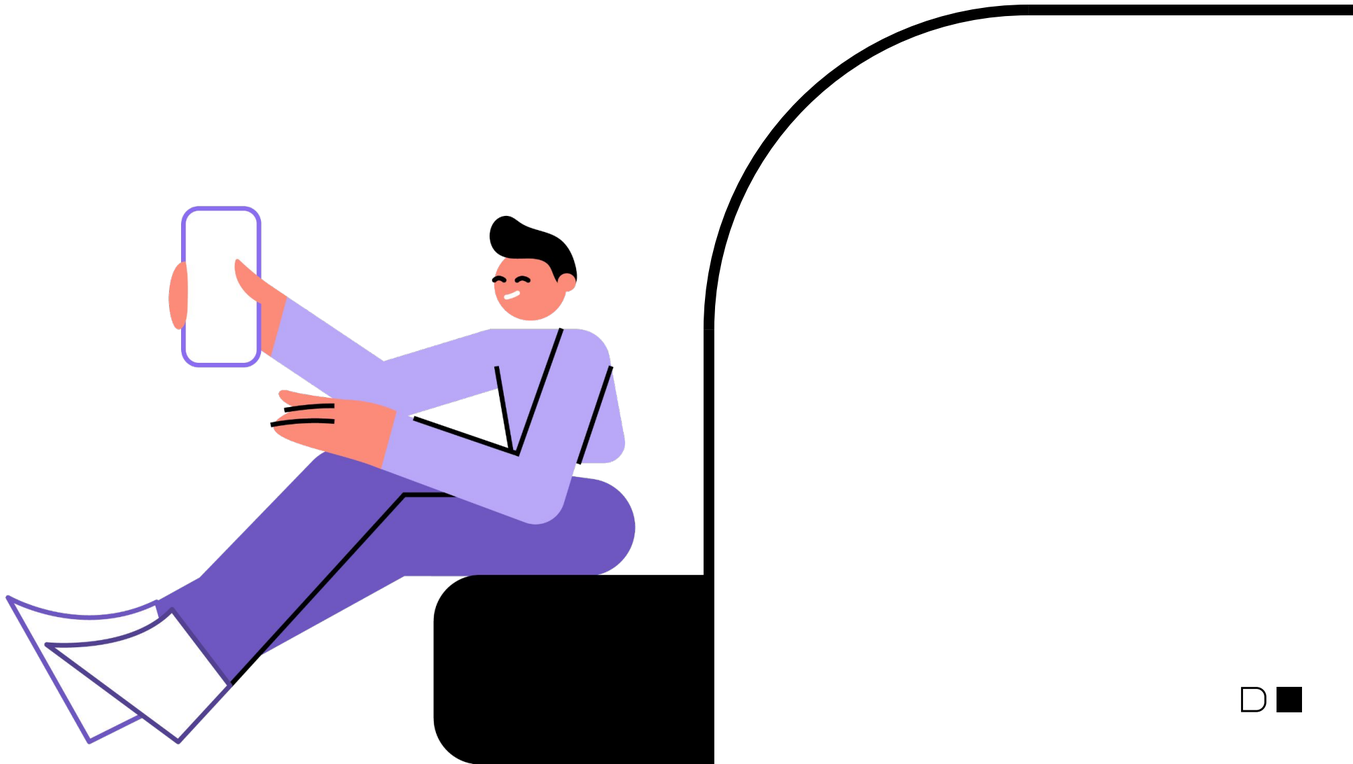
Certified Tech Developer
The Ultimate Degree

Índice

01 Introducción

02 Arquitectura

03 Objetos básicos de Kubernetes



01 Introducción

¿Qué es?

Kubernetes es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Se basa en la experiencia que tiene Google luego de haber trabajado con aplicaciones en producción a gran escala por más de una década, junto a las mejores ideas y prácticas de la comunidad.

Como definición, podríamos decir que sirve para la gestión y orquestación de contenedores, permitiendo programar su despliegue, escalado, monitorización y otras funcionalidades más.

¿Qué significa Kubernetes?

El nombre **Kubernetes** proviene del griego y significa timonel o piloto, y eso es lo que hace esta herramienta: una dirección de distintos aspectos del sistema. Kubernetes, también conocido como **K8S**, abstrae la infraestructura de hardware y expone todo el centro de datos como un enorme recurso computacional, que permite desplegar y ejecutar sus componentes de software sin tener que conocer los servidores reales que se encuentran debajo.

Enlaces para leer:

Note-taking: A Research Roundup

Documentación oficial

02 Arquitectura

Elementos de Kubernetes

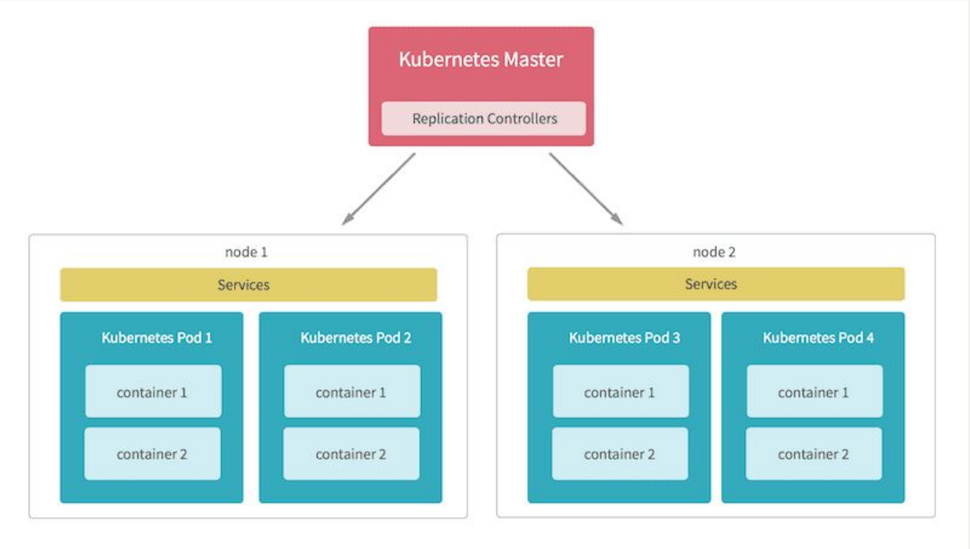
Un clúster de Kubernetes consiste en un conjunto de máquinas **worker** llamadas **nodos** que corren aplicaciones en “containers”.

Todos los clústeres tienen al menos un worker y dentro corren los **pods**, objetos más pequeños del clúster que corren las aplicaciones.

El **control plane** corre en los nodos **Masters** y es lo que maneja a los workers y a los pods en el clúster. Además, un clúster suele tener varios nodos para proveer “fault-tolerance” y “high availability”.

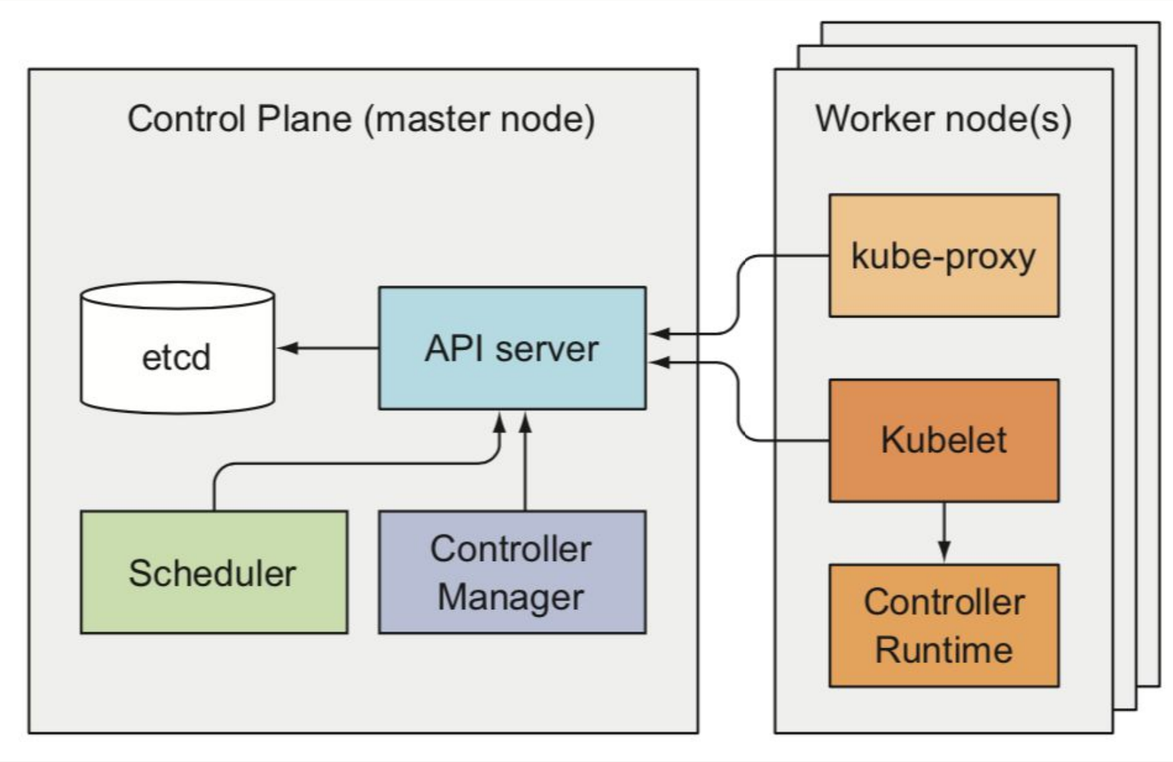
Se utilizan los objetos de la **API de Kubernetes** para describir el estado deseado del clúster: qué aplicaciones u otras cargas de trabajo se quieren ejecutar, qué imágenes de contenedores usan, el número de réplicas, qué red y qué recursos de almacenamiento deberían estar disponibles, etc.

Se especifica el estado deseado del clúster mediante la creación de objetos usando la API de Kubernetes, típicamente mediante la interfaz de línea de comandos, **kubectl**.



Componentes del plano de control (Master)

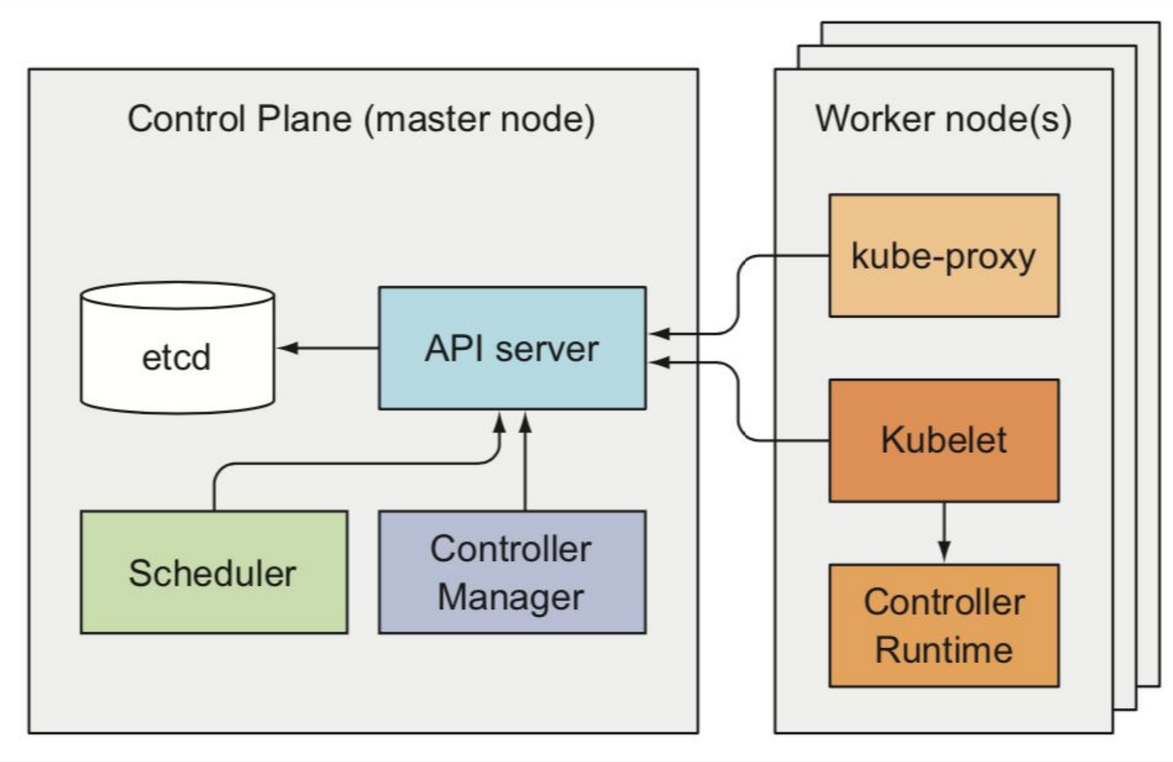
- **etcd**: es un almacén de datos persistente, consistente y distribuido de clave-valor utilizado para almacenar toda la información del clúster de Kubernetes.
- **Controller Manager:** es el responsable de detectar y responder cuándo un nodo deja de funcionar. Incluye también al controlador de replicación: el responsable de mantener el número correcto de pods para cada controlador de replicación del sistema.



Componentes del plano de control (Master)

- **Scheduler:** Componente del plano de control que está pendiente de los pods que no tienen ningún nodo asignado y selecciona uno donde ejecutarlo.

- **API server:** El servidor de la API es el componente del plano de control que expone la API de Kubernetes. Se trata del front end de Kubernetes, recibe las peticiones y actualiza acordemente el estado en **etcd**.

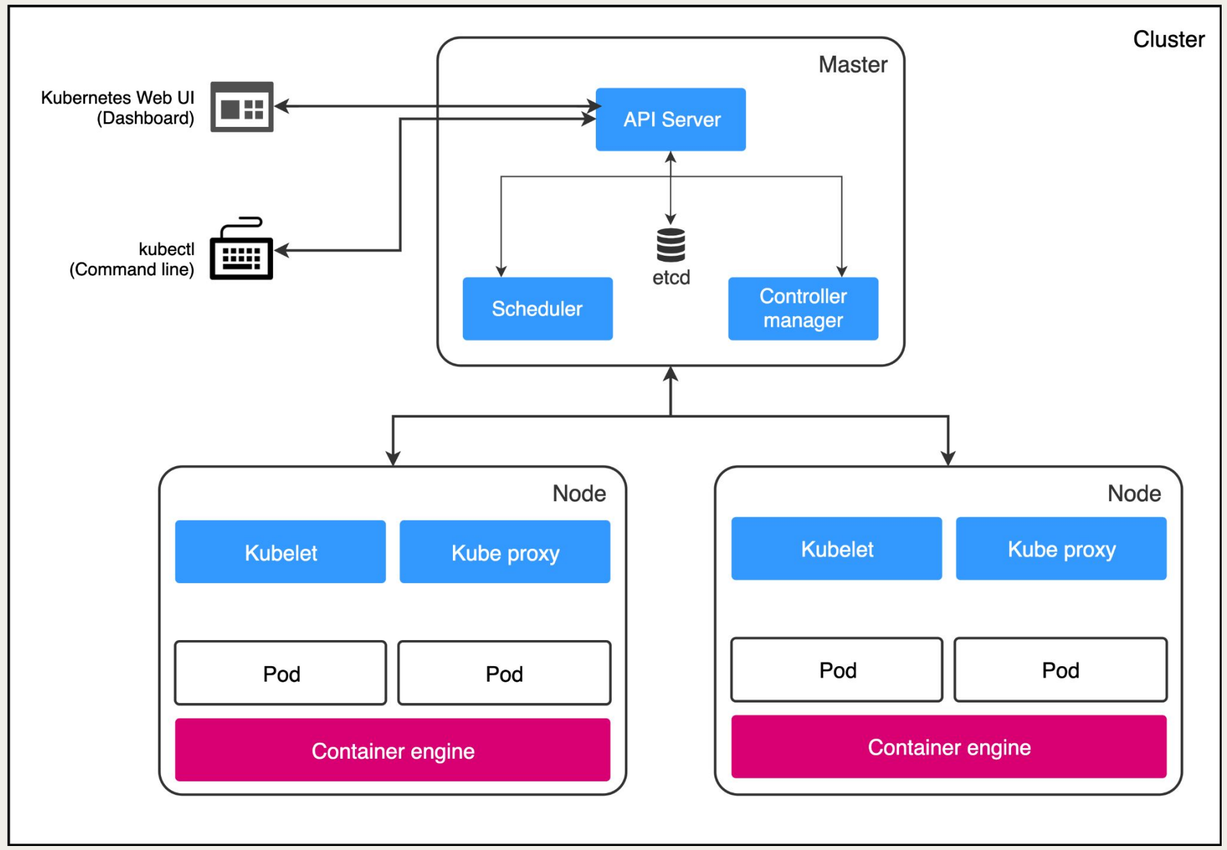


Componentes de los nodos (Worker)

- **Kubelet:** Agente que se ejecuta en cada nodo de un clúster. Se asegura de que los contenedores estén corriendo en un pod.

- **Kube-proxy:** permite abstraer un servicio en Kubernetes manteniendo las reglas de red en el anfitrión y haciendo reenvío de conexiones.

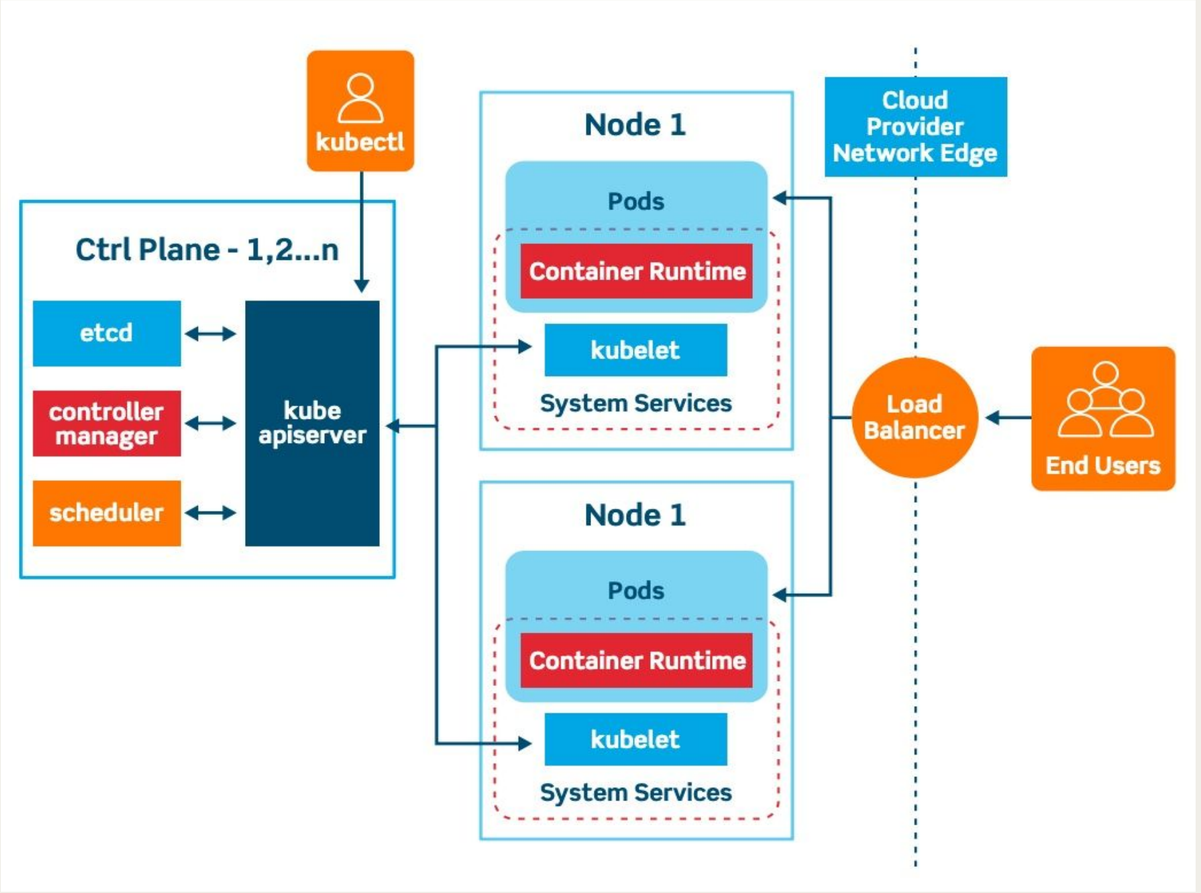
- **Runtime de los contenedores:** es el software responsable de ejecutar los contenedores. Kubernetes soporta varios de ellos: **Docker**, **containerd**, **cri-o**, **rktlet** y cualquier implementación de la interfaz de runtime de contenedores de Kubernetes o Kubernetes CRI.



Integración con la nube

- **Cloud Controller Manager:** ejecuta controladores que interactúan con proveedores de la nube. El concepto del Cloud Controller Manager (CCM) fue creado originalmente para permitir que Kubernetes y el código específico de proveedores de servicios en la nube evolucionen de forma independiente. Se ejecuta a la par con otros componentes de los nodos Masters, como Kubernetes Controller Manager, API Server, etc.

- El diseño del Cloud Controller Manager está basado en un sistema de plugins, lo que permite a nuevos proveedores de servicios integrarse de forma fácil con Kubernetes. Se está trabajando en implementar nuevos proveedores de servicios para migrar los existentes del antiguo modelo al nuevo CCM.



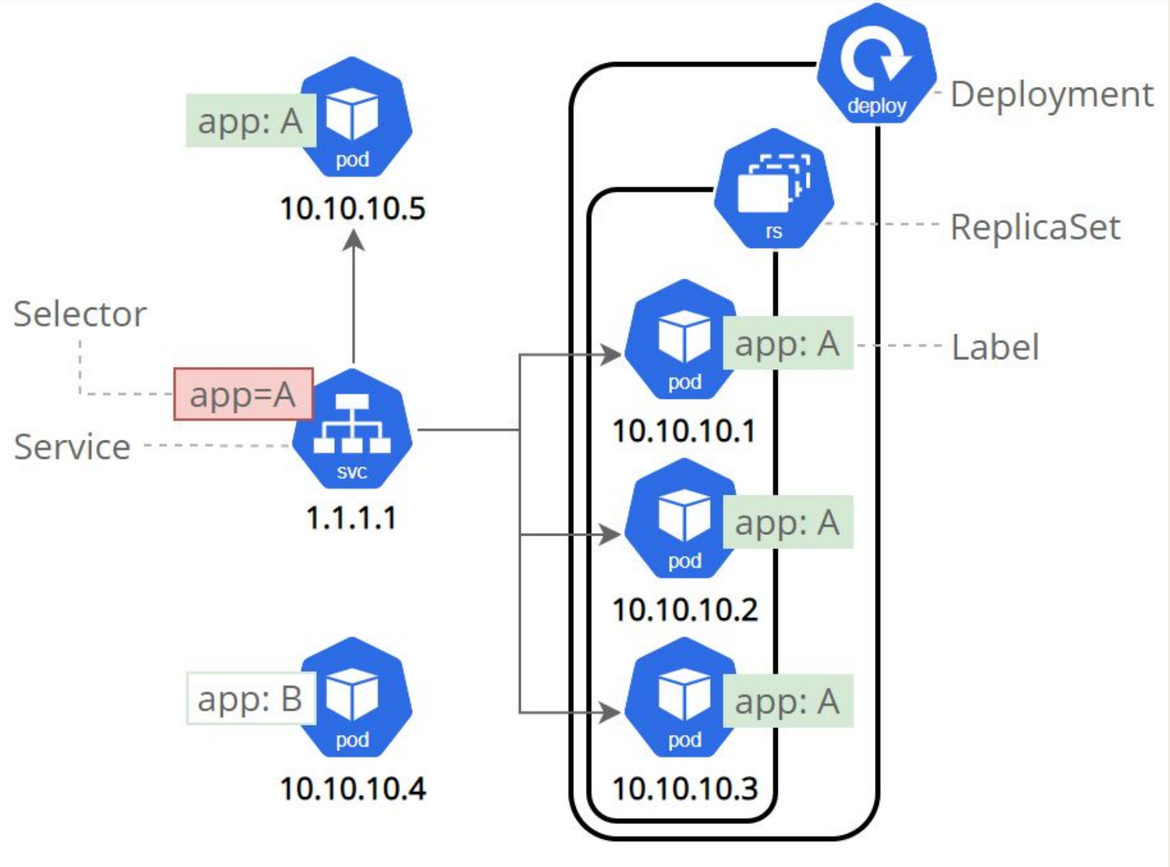
03 Objetos básicos de Kubernetes

Objetos de Kubernetes

Los objetos de Kubernetes son entidades persistentes dentro de la plataforma que Kubernetes utiliza para poder presentar el estado del clúster. Estas entidades describen:

- Qué aplicaciones corren en contenedores.
- Recursos disponibles para dichas aplicaciones.
- Políticas acerca de cómo dichas aplicaciones se comportan, como las de reinicio, actualización y tolerancia a fallos.

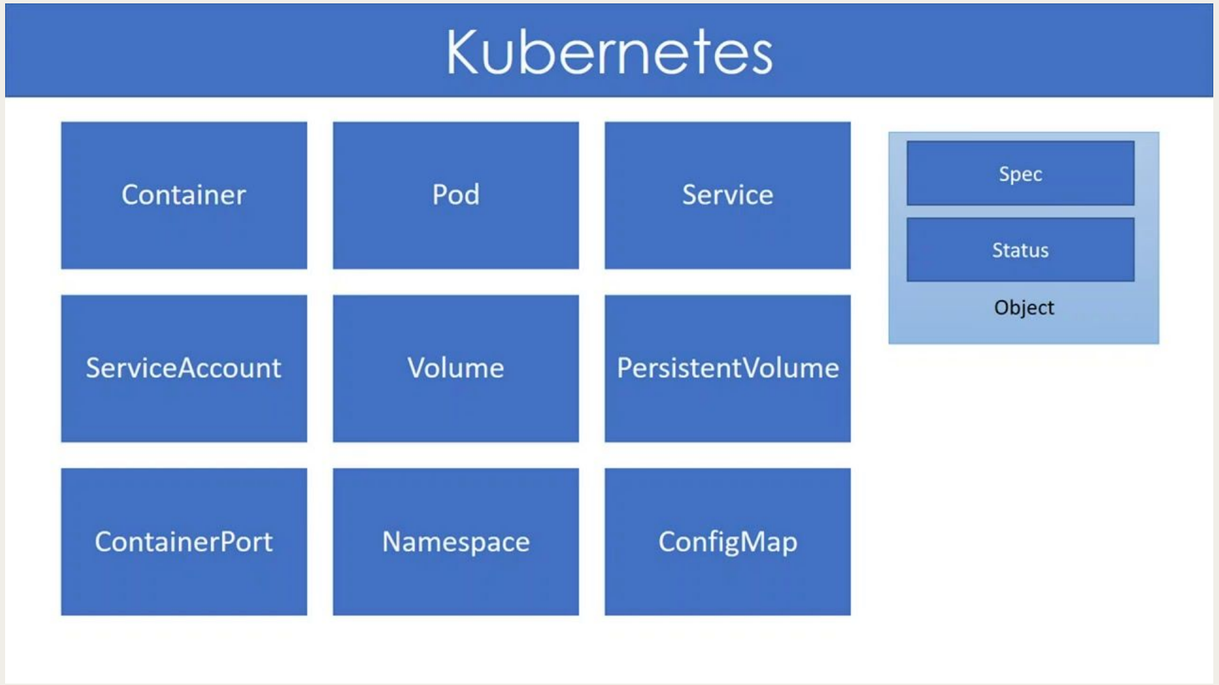
Podríamos decir que cada objeto es un “registro de intención”, es decir, con cada objeto que se crea, le estamos diciendo a Kubernetes cómo debería ser la carga de trabajo del clúster, es decir, el estado deseado.



Estado de los objetos

Con cada objeto que se define, existen dos campos anidados, el **spec** y el **status**. El campo **spec**, describe cómo es el estado deseado, y el campo **status**, que es gestionado por el sistema de Kubernetes, es el estado actual del objeto el cual será guardado. En cualquier momento, el Control Plane verifica si el estado actual coincide con el estado deseado. En caso de que esto no sea así, se tomarán medidas necesarias para tratar de llegar al estado deseado.

Por ejemplo, si tenemos un objeto **Deployment**, el cual nos servirá para desplegar una aplicación en nuestro clúster, le podemos decir en el apartado **Spec** que queremos tres réplicas. Si en cualquier momento hubiese un fallo en alguna de las instancias, el sistema de Kubernetes lo solucionaría instanciando una nueva.



Definiendo tipos de objetos en Kubernetes

{}

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.15.11
        ports:
        - containerPort: 80
```

{}

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.15.11
    ports:
    - containerPort: 80
```

Existen diferentes objetos de Kubernetes. Para definir estos objetos se puede realizar en JSON o en YAML (este es el más utilizado). Una vez que se ha definido el fichero, será enviado al API Server a través del Kubectl.

- **apiVersion:** qué versión de la API de Kubernetes vamos a usar para crear el objeto. (Campo obligatorio)
- **kind:** el tipo de objeto que estamos definiendo. (Campo obligatorio)
- **metadata:** datos para identificar al objeto. (Campo obligatorio)
- **spec:** contiene campos anidados que dependen del tipo de objeto que se defina. No será igual cuando se defina un pod que cuando se defina un deployment.

Tipos de objetos básicos

- **Namespace**: Los namespaces nos permiten aislar recursos para el uso por los distintos usuarios del clúster, para trabajar en distintos proyectos.
- **Pods**: Los pods son las unidades de computación desplegables de menor tamaño que la plataforma de Kubernetes puede crear y administrar de forma directa. Es una colección de uno o más contenedores. Los pods son efímeros y son usados con controladores que gestionen la replicación, la tolerancia a fallos, etc.
- **ReplicaSet**: Utilizando un ReplicaSet podemos escalar el número de pods que ejecutan una imagen. Este asegura que el número establecido de réplicas se están ejecutando, es decir, mantener el estado deseado. No es necesario administrar ReplicaSets y pods por separado, la mejor aproximación es usar Deployments y configurar la replicación.
- **Deployments**: El objetivo de un Deployment es proporcionar una manera declarativa de actualizar pods y ReplicaSets. Describe un estado deseado y el Deployment Controller cambia el estado actual al esperado. Se pueden definir Deployments para crear nuevos ReplicaSets, o para eliminar Deployments existentes y adoptar todos sus recursos con nuevos Deployments.

Tipos de objetos básicos

- **DaemonSet**: Es un objeto que garantiza que todos (o algunos) de los nodos ejecuten una copia de un Pod. Conforme se añade más nodos al clúster, nuevos Pods son añadidos a los mismos. Conforme se eliminan nodos del clúster, dichos Pods se destruyen. Al eliminar un DaemonSet se limpian todos los Pods que han sido creados.
- **StatefulSet**: A diferencia de los Deployments, los StatefulSet mantienen una identidad fija para cada uno de los pods y no son intercambiables entre sí, puesto que se encuentran asociados a un almacenamiento que se mantiene cada vez que el pod es reemplazado.
- **Service**: Este objeto se utiliza para agrupar un conjunto de pods y proporcionar acceso a ellos según sea necesario. Un servicio utiliza un único nombre de sistema de nombres de dominio (DNS) para el grupo de pods y dirige el tráfico de red interno a una serie de pods para administrar la carga de trabajo y proporcionar comunicación entre los pods. Un servicio se define mediante una dirección IP estática que puede adjuntarse y reconocerse mediante los pods participantes.

Tipos de objetos básicos

- **Ingress**: Se utiliza para enrutar el tráfico externo al clúster. Es posible que también se utilice para ofrecer un equilibrio de carga adicional a los servicios del clúster.
- **Persistent Volumes**: Los volúmenes persistentes son recursos que están disponibles para el clúster con el propósito de almacenamiento de recursos de sistema y datos.
- **ConfigMap**: Es un objeto de la API que permite almacenar la configuración de otros objetos utilizados. Aunque muchos objetos de Kubernetes tienen un spec, un ConfigMap tiene una sección data para almacenar items, identificados por una clave y sus valores.
- **Secret**: Aunque funciona de manera parecida al ConfigMap, el contenido de los secretos está ofuscado y permite almacenar datos sensibles a los que no puede acceder cualquiera en el clúster. Si asignáramos permisos de solo lectura a un usuario en K8s, este no podría ver el contenido de los secretos, pero sí el de los configmaps.

DIGITALHOUSE

¡Muchas gracias!