Para entender qué es y cómo funcionan Kubernetes y Kubernetes Engine, vamos a aprender qué es un contenedor:

* **Bare Metal**: Son los servidores físicos. Ademas de administrar nuestra aplicación, debemos hacernos cargo de la infraestructura *(todo lo relacionado con Hardware: redes, memoria, CPU, etc)* y el sistema operativo *(actualizaciones, parches de seguridad, etc)*.
* **Virtual Machine**: La infraestructura esta controlada por una capa virtualizada, no nos preocupamos más por la parte física de nuestros servidores.
* **Containers**: Los contenedores son el funcionamiento más optimo de nuestras aplicaciones. Nos preocupamos únicamente por administrar nuestra aplicación y sus dependencias *(lenguajes de programación, frameworks, logs, etc)*, mientras que de la infraestructura y el sistema operativo se van a encargar nuestros proveedores de nube.

**Kubernetes Engine** es la solución administrada de Google en la plataforma de Google Cloud Platform. Su función es administrar el *cluster* de máquinas, la instalación de los clientes y todos los procesos para el despliegue de nuestras aplicaciones. Google se encarga de todo.

Esta herramienta tiene las siguientes funcionalidades:

* **Formato Docker**: Nuestras máquinas y contenedores deben usar este formato para poder ser desplegadas.
* **Auto-escalado**: La performance de nuestra aplicación no se verá afectada cuando entren muchísimos usuarios al mismo tiempo. Podrás soportar toda esta carga sin necesidad de generar tus máquinas virtuales a mano.
* **Stackdriver**: La solución de monitoreo y control de Google. Nos ayuda a entender muy fácilmente qué está pasando en la infraestructura: logs, alertas, monitoreo de la CPU y la memoria, etc.
* **Cloud VPN**: Nos permite tener una nube híbrida con la cual tendremos nuestros datos y aplicaciones segregados en la nube y utilizando información de nuestros servidores locales. Todos estos procesos de forma segura.
* **Cloud IAM**: Administración de usuarios gracias al soporte de autenticación de Google. Podemos asignar permisos y roles de lectura, escritura y administración sobre nuestro cluster.

Ventajas de Kubernetes Engine:

* Actualizaciones automáticas
* Reparación automática
* Registro privado de contenedores
* Versiones uniformes y rápidas
* Soporte para GPU

Alternativas:

* Redhat Openshift
* Docker Swarm
* Amazon Elastic Container Service for Kubernetes
* IBM Cloud Kubernetes Service
* Azure Kubernetes Service (AKS)

Docker es una tecnología open source, desarrollada en el lenguaje de programación GO, utilizando la licencia Apache 2.0 *(podemos desplegar nuestra aplicación en cualquier servidor, sin que nuestro código fuente sea open source)*.

Docker utiliza las características de los sistemas operativos tipo UNIX para administrar los contenedores y el despliegue de nuestras aplicaciones en el area de memoria del sistema operativo Linux.

Ventajas de los Contenedores:

**Ventajas**

Completamente portables, te permiten ejecutar tu aplicación en local sobre cualquier sistema operativo, en cualquier servidor on-premise o incluso en la nube. Esto hace que Docker sea un complemento perfecto para los equipos ágiles y acaba con el “esto en mi ordenador funcionaba”.

Te permite tratar la infraestructura como un fichero más dentro de las fuentes de tu proyecto, con lo que puedes automatizar dentro del build la creación de la propia infraestructura, creando una imagen completa con el software base y tu aplicación instalada.

Hay miles de imágenes ya disponibles en DockerHub con casi cualquier stack tecnológico, las puedes usar directamente o modificarlas para evitar partir de cero.

La gestión de paquetes y dependencias es muy simple y transparente, facilitando la integración de los equipos de desarrollo y sistemas.

Tiene mejor rendimiento que la virtualización tradicional, ya que está basado en LXC, que corre directamente sobre el kernel de la máquina donde se aloja, evitando la capa de virtualización tradicional basada en un hipervisor que penaliza el rendimiento.

Por las mismas razones que el punto anterior, los contenedores arrancan en segundos, mucho más rápido que una máquina virtual.

COMANDOS BASICOS DE DOCKER

* docker pull: Descargar y guardar las imágenes de nuestra aplicación
* docker images: Listar las imágenes descargas en nuestra máquina
* docker ps: Listar los contenedores corriendo en nuestra máquina *(id, puerto, etc)*
* docker ps -a: Listar los contenedores apagados de nuestra máquina\_
* docker exec: Ingresar a nuestro contenedor y listar las carpetas dentro de el
* docker stop: Detener la ejecución de nuestro contenedor
* docker rm: Borrar nuestro contenedor

CLUSTER Y NODOS

Un cluster es una agrupación de máquinas que corren una cierta cantidad de servicios para que nuestra aplicación pueda funcionar sobre Kubernetes.

El endpoint o nodo maestro es una máquina dentro de nuestro cluster, es la puerta de entrada a todo el cluster. Tiene las APIs de Kubernetes, los servicios REST, el agendamiento de pods y la sincronización servicios. Tambien cuenta con integración a los servicios de Google Cloud Platform.

Un Nodo es un worker machine, osea, una maquina dentro de un cluster. Esta máquina tiene todas las herramientas para el despliegue de nuestras aplicaciones. Puede ser una maquina virtual o una máquina física con todos los servicios necesarios para correr pods.

Las direcciones, condiciones, capacidad e información de nuestros nodos se llaman estados y podemos acceder a ellos desde la terminal.

Un pod es un conjunto de contenedores que se despliegan en nuestros nodos. Más adelante aprenderemos más a detalle cómo funcionan.

DEPLOYMENTS

Los deployments son una abstracción de nuestra aplicación que nos permite crear una arquitectura e indicar la cantidad de pods que se van a necesitar. Con esto, haremos un despliegue de nuestra aplicación en nuestro cluster, tomando la cantidad de pods y replicas que indicamos previamente.

SERVICIOS

Los servicios nos permiten ingresar tráfico y conectar los diferentes servicios de nuestra aplicación. Los 3 tipos de servicios disponibles son: **Cluster IP**, **Node Port** y **Load Balancer**.

**Cluster IP**  
Se dedica a unir microservicios dentro de mi arquitectura. Utilizando para ello IP privada, por lo que no puede comunicarse con el exterior. Es decir, permite que los pods se puedan comunicar entre ellos, pero no se pueda acceder de forma externa (interno al clúster). Por ejemplo, para comunicar un backend con una base de datos.

**Node ports**  
Se genera un mapeo de puertos a nivel de nodos. Permite abrir un puerto en todos los nodos para comunicarse con ellos mediante una IP externa.

**Load Balancer**  
Permite acceder al pod mediante una IP externa, esta es única de este pod, y solamente permite acceder a él.

ARCHIVOS DESCRIPTIVOS DE INFRA

Los archivos descriptivos nos permiten hacer despliegues de nuestras aplicaciones, pero en este caso, escribiendo nosotros mismos los archivos de configuración. No vamos a usar la interfaz gráfica. Estos archivos los escribimos en formato .yaml.

En estos archivos vamos a describir toda la parte lógica de nuestra aplicación: Pods, Deployments, Services e incluso las conexión entre aplicaciones. Recuerda que en estos archivos no podemos configurar ni los clusters ni lo nodos.

LABELS

Los Labels son una metadata arbitraria, esto quiere decir que podemos poner los nombre que queramos para generar una identidad a cualquier objeto *(servicios, deployments, etc)*. Pueden ser de tipo *Queryable* *(buscables o seleccionables)*, lo que nos permite agruparlos para generar despliegues de forma más fácil y hacer segregación de nuestros servicios.

Los selectores son una herramienta que nos permite utilizar los labels para saber cuándo comenzar a utilizar unos pods u otros en nuestros despliegues.

NAMESPACES

Los Namespaces son una separación virtual dentro de nuestro cluster con el fin de hacer un aislamiento de nuestros datos y tener ambientes completamente separados. Podemos tener multiples namespaces dentro de nuestros clusters.

DEOLOYMENT AVANZADI (BLUE GREEN Y CANARY DEPLOYEMENT

**Blue-green**: Es una técnica de despliegues que nos ayuda a tener actualizaciones de nuestra aplicación con cero *down time*. Para esto necesitamos, un ambiente el tráfico *(para el entorno de producción)* y un segundo ambiente con las nuevas características de tu aplicación.

**Canary**: Nos sirve para hacer pruebas de nuevas versiones a un público pequeño, esto con el fin de averiguar cómo se comportan los usuarios frente a esta nueva versión

VOLUMENES

Los volúmenes nos ayudan a implementar un almacenamiento persistente, en el que podamos acoplar y desacoplar los nodos sin que esta información desaparezca. Pueden ser de tipo persistente, NFS o cluster, pero dependiendo de la integración con diferentes nubes se podrán implementar nuevos tipos de volúmenes.

Los volúmenes son ideales para guardar la información de nuestros usuarios, archivos, configuraciones y compartir esta información con diferentes pods.

DESPLEGAR UNA APLICACIÓN STATEFUL TEORIA

Las aplicación Stateful son aplicaciones que guardan el estado de los datos para ser consultados posteriormente. Buenos ejemplos de aplicaciones Stateful son: las Bases de Datos, Data Warehouse, Modelos predictivos de IA, Gestores documentales, entre otros.

AUTO SCALING

Auto Scaling es una practica que nos permite crecer nuestros clusters de manera dinámica.

Entre más tráfico tenga nuestra aplicación, más nodos se crearán en nuestro cluster. Así también, entre menos tráfico tenga nuestra aplicación, estos nodos van a desaparecer, dejando la cantidad de nodos suficiente para atender a los usuarios.

NODE POOLS

Los Node Pools son agrupaciones de máquinas que nos permiten generar un cluster híbrido con máquinas de diferentes tipos y tamaños.

El protocolo https es una herramienta útil a nivel de seguridad de la información debido a que te permite cifrar la información haciendo más segura la transmisión de datos. Esta herramienta también te permite generar confianza en tus usuarios por que les indica que estas preocupado por proteger su información. Por este motivo en esta lectura vamos a ver como crear un certificado https con Let’s Encrypt para que funcione en los diferentes servicios de Google Cloud Platform.

Lo primero que tienes que saber es que es Let’s Encrypt, la cual es una entidad certificadora que está dando certificados ssl de manera gratuita (normalmente estos certificados tienen un costo que supera los 50 dólares por año) con una duración de 3 meses (esto significa que hay que renovarlos antes de la fecha de vencimiento).

Ahora vamos a ver cuales son los pasos para crear el certificado:

## 0. Prerrequisitos:

a. Comprar un dominio y tener acceso al administrador de los DNS  
b. Configurar un proyecto en Google Cloud Platform  
c. Habilitar la facturación ingresando una tarjeta de crédito (te sugiero activar los créditos que ofrece la plataforma para no tener gastos en la tarjeta de crédito).  
d. Acceder a Google Cloud Shell entrando a <https://console.cloud.google.com> y haciendo click en el botón de Cloud Shell que está en la parte superior izquierda del panel

SERVICES MESH CON ISTIO

Service Mesh nos ayuda a fortalecer las políticas de comunicación en nuestros clusters de kubernetes.

Si tenemos una aplicación que se comunica con un almacén de datos en producción, no queremos que las versiones de desarrollo se comuniquen con las versiones de producción.

Para esto, normalmente necesitaríamos generar muchas máquinas aisladas por sub-redes, fortaleciendo los procesos por medio de firewalls, proxies y demás servicios de red. Todo esto enfocado hacia los bytes que se transmiten dentro de la red.

Con Service Mesh podemos definir estas reglas de comunicación a un alto nivel, a través de servicios usando un l

enguaje declarativo.

**Caracteristicas de Service Mesh:**

* Red para servicios, no para bytes
* Visibilidad
* Resiliencia y eficacia
* Control de tráfico
* Seguridad
* Fortalecimiento de políticas

Istio es un producto creado por Google e IBM que nos ayuda a reducir la complejidad del Service Mesh. Ademas, nos ayuda a tener control sobre los servicios de nuestra aplicación y entender el tráfico de cada uno de estos.

**Características de Istio:**

* Provee un service discovery
* Seguridad
* Instrumentación
* Rutas dinámicas
* Telemetría

CI/CD CON CLOUD BUILDER

**Continuous Integration** y **Continuous Delivery** son dos conceptos que nos ayudan a desplegar muy rápido las versiones de nuestra aplicación. Estaremos integrando continuamente nuestra aplicación a través de herramientas como Jenkins o Terraform, para describir la infraestructura de nuestra aplicación y estar constantemente enviando nuevas versiones de nuestras aplicaciones.

Cloud Builder nos ayuda a automatizar muchas tareas en el despliegue de aplicaciones. Gracias a esta herramienta y todas sus integraciones (Cloud Registry, Cloud Repository) vamos a generar el build de nuestra aplicación, generar el contenedor, hacer pruebas y hacer el despliegue.

KNATIVE , GKE SERVERLESS ADDON

**Serverless** es un tipo de arquitectura donde los servidores *(físicos o en la nube)* dejan de existir para el desarrollador y en cambio el código corre en ““ambientes de ejecución”” que administran proveedores como Amazon, Google, IBM, entre otros.

**Características de Serverless**:

* No se administran servidores
* Auto Escalable
* Fácil integración con otros servicios del proveedor
* Cobro por uso

**Knative** es una solución open-source que brinda servicios tipo serverless para desplegar contenedores. Esta herramienta es amigable con los desarrolladores ya que se encarga de la administración del cluster, saber cuantas replicas necesita nuestra aplicación, cuales son los pods, etc. Incluso podemos escalar nuestras aplicaciones desde 0.

**GKE Serverless Addon** es una implementación de Knative auto-administrada gracias a Google Kubernetes Engine. Esta herramienta fue anunciada en el Google Cloud Next 18 y aún esta en estado Alfa, pero muy pronto estará disponible para el público.

BUENAS PRACTICAS

* Hacer contenedores pequeños *(las versiones ALPINE son las mejores en cuanto a seguridad)*
* Organizar despliegues con namespaces
* Configurar los Health Checks
* Configurar límites en el número de peticiones
* Terminar con Gracia *(cerrar los servicios correctamente guardando antes la información de tus usuarios)*