| **🖥 Máquinas compartidas** | **💻 VMs/bare metal** | **📦 Contenedores** |
| --- | --- | --- |
| ❌ Sin Aislamiento | ✅ Aislamiento | ✅ Aislamiento |
| ❌ Librerías en común | ✅ Sin librerías en común | ✅ Sin librerías en común |
| ❌ Alto acoplamiento entre apps y OS | ❌ Caras e ineficientes | ✅ Menos overhead |
| - | ❌ Difíciles de mantener | ✅ Menos dependencia del sistema operativo huésped |

**1. Máquinas Compartidas**

Los administradores de sistemas tenian que conseguir ejecutar varios servidores físicos.

Existen **Dependencias aplicativas** como:

* Kernel (requerido)
* Otras dependencias (opcionales)
  + Runtimes del Lenguaje (Java, Node.js, etc.)
  + Módulos del Kernel
  + Herramientas del sistema
  + Librerías de sistema
  + Configuración

![[Dependencias aplicativas.png]]

La manera antigua: Instalar aplicaciones en el host

* Múltiples aplicaciones por máquina
* Dependencias compartidas  
  **Beneficios**:
* Ejecutables de tamaño pequeño
* Menos uso de almacenamiento y memoria
* Mayor utilización de recursos  
  **Problemas**
* Las dependencias se mezclaban unas con otras y con la máquina huésped
* Una aplicación puede acaparar recursos, impactando a otras
* Difícil de replicar y comúnmente existían diferencias  
  entre ambientes (¡corre bien en desarrollo!)

**La solución (temporal)**

* Dedicar máquinas huésped para las aplicaciones críticas
* Empaquetar todas las dependencias con la aplicación  
  **Problemas**
* Dedicar máquinas huésped desperdiciaba recursos
* Máquinas físicas toman tiempo en aprovisionar y configurar

**2. VMs/bare metal**

Se segmentaba una máquina en varias máquinas virtuales (Virtualización)

* Permitir que el hardware físico sea compartido entre aplicaciones  
  como máquinas virtuales (VMs)
* Soluciones como Chef/Puppet/Ansible eran comúnmente utilizadas  
  para administrar máquinas huésped y aplicativos
* VMs inmutables daban despliegues y rollbacks predecibles  
  **Problemas**
* Recursos desperdiciados (mucho *overhead*)
* Las VMs tardan mucho en iniciar

**3. Contenedores**

Mientras tanto en Google los desarrolladores añadieron capacidades al kernel de Linux para soportar el aislamiento de procesos, poniendo las bases para la contenerización:

* **Grupos de control** (cgroups) --límites de recursos.
* **Namespaces** —aislamiento de procesos.
* **Change Root** (chroot) — aislamiento del sistema de archivos (ya existía).

Esto permitió a las apps/procesos:

* Correr con CPU y memoria asignada (específica)
* Aislar de otros procesos
* Proveer acceso limitado al sistema de archivos

La nueva manera es **desplegar contenedores**

* Virtualización a nivel OS (namespaces, cgroups, …)
* Aislado, entre otros procesos y el host
  + Las imágenes de contenedor tienen su propio sistema de archivos, sus propios recursos y se ejecutan como su propio proceso
* Rápido
* Portable entre sistemas operativos y entre nubes
  + Siempre y cuando el kernel objetivo fuera el mismo que el de la máquina huésped
* Ambientes consistentes entre desarrollo y producción
  + Creando imágenes inmutables durante el proceso de construcción, en lugar de durante el proceso de despliegue

INTRIDUCCIÓN A DOCKER

* Herramienta de contenedores dominante
* Liberada como OSS en 2013 por DotCloud inc. (hosted PaaS)
* Logro crear y ejecutar imagenes de contenedores fácil y rápido
* Adopcion hyper-acelerada
* DotCloud Inc → Docker Inc (vendiendo el negocio PaaS)
* Empezó a cambiar su negocio (up the stack) en 2015 por presión de generación de ingresos
* Docker es un formato o especificación para construir imágenes de contenedores
* Imagen de contener Docker - binario empaquetado con un sistema operativo (sin el kernel y todas las dependencias)
* Docker container - proceso aislado (cgroup/chroot jail)
* Container registry (Docker Hub/GCR/Quay) - repositorio central de imágenes. Punto de acceso para la descarga de imágenes
* Las funcionalidades del kernel desarrolladas por Google han sido incorporadas con el kernel
* Nuevas funcionalidades han sido incorporadas con el paraguas de LinuxContainersorg
* La iniciativa de los contenedores abiertos ha sido formada para estandarizar los formatos de contenedores:
  + La especificación del runtime (runtime-spec)
  + La especificación de las imágenes (image-spec)
* Docker libero como OSS el motor de ejecución (containerd)
* Otros runtimes fueron desarrollados (ejemplo: rkt)

COMANDOS

//setear proyecto de trabajo

gcloud config set project <PROJECT-ID>

// verificar que docker esté instalado

docker -v

//verificar que haya alguna imagen de docker

docker images

// correr un hello world con docker, si no existe crea la imagen

docker run hello-world

// verificar procesos de docker

docker ps

// correr una imagen

docker run -p <puerto-maquina:puerto conteiner> <imagen-docker>

// mostrar procesos que has corrido

docker ps -a

// borrar imagen

docker rm <id-del-conteiner>

// correr en segundo plano (demonio)

docker run -d --name <nombre> -p <puerto-maquina:puerto conteiner> <imagen-docker>

// parar aplicacion

docker stop <nombre-app>

KUBERNETES

Una **abstracción** sobre la infraestructura. Ahora interactuamos con un plano de control para que ejecute las instrucciones que le damos de forma **declarativa** (es decir que busque el **estado ideal**).

ES UN CONTROLADOR DECLARATIVO

**Escalamiento Vertical:** Mas recursos a una carga de trabajo.  
**Escalamiento Horizontal:** Múltiples copias de la misma carga de trabajo.

-Administra clusters de contenedores

-Inspirado por los sistemas internos de Google

-Con soporte para despliegues multi-nube y sobre ambientes convencionales de virtualización y bare-metal

-Soporta múltiples runtimes de contenedores.

-100% open source, escrito en GO.

-ADMINISTRA APLICACIONES NO MÁQUINAS.

BRINDA GRANDES FUNCIONALIDADES PARA TUS CARGAS DE TRABAJO

-calendarización.

-encontrar el huésped adecuado

- monitorear salud

- escalamiento elástico bajo demanda.

- Movimiento bajo demanda

## Componentes de los nodos

#### kubelet

Un agente que se ejecuta en cada nodo del clúster. Se asegura de que los contenedores se ejecuten en un Pod.

kubelet toma un conjunto de PodSpecs que se proporcionan a través de varios mecanismos y asegura que los contenedores descritos en esas PodSpecs estén funcionando y en buen estado. El kubelet no administra contenedores que no fueron creados por Kubernetes.

#### kube-proxy

kube-proxy es un proxy de red que se ejecuta en cada nodo de su clúster, implementando parte del concepto de servicio de Kubernetes.

kube-proxy mantiene las reglas de red en los nodos. Estas reglas de red permiten la comunicación de red con sus pods desde sesiones de red dentro o fuera de su clúster.

kube-proxy usa la capa de filtrado de paquetes del sistema operativo si hay una y está disponible. De lo contrario, kube-proxy reenvía el tráfico en sí.

#### Container runtime

El tiempo de ejecución del contenedor es el software responsable de ejecutar los contenedores.

Kubernetes admite varios tiempos de ejecución de contenedores: Docker, conteiner, CRI-O y cualquier implementa

Creación de Kubernetes CRI (Container Runtime Interface).

CONTROL PLANE

-API server. El frontend del control plane

-Fuente única de la verdad(etcd)

-Revisar por nuevos pods y asignarlos a un nodo(kub-scheduler)

CONTROLADORES

-Responden a cambio de objetos

-Mover estado actual a estado deseado

TODO ES UN RECURSO(OBJETO()

-kind(tipo)

-apVersion

-metadata

-spec < representación del estado deseado

-status< representación del estado actual

**CONCEPTO CLAVE: EL POD**

Unidad atómica de kubernetes

Compuesto de uno o mas contenedores con recursos de almacenamiento y red compartidos(ej: servidor de apps y web server)

Los contenedores dentro de un pod comparten el mismo Linux namespace, pero diferentes grupos de control

Kubernetes automatiza de buena manera la configuración de namespace, cgroup, etc

Unidad de despliegue empaquetada

## ¿Qué es la Cloud Native Computing Foundation?

La Cloud Native Computing Foundation (CNCF) es un proyecto de la Fundación Linux que se fundó en 2015 para ayudar al avance de la tecnología de contenedores y alinear la industria tecnológica en torno a su evolución.

Se anunció junto con Kubernetes 1.0, un administrador de clústeres de contenedores de código abierto, que Google contribuyó a la Fundación Linux como tecnología semilla. Los miembros fundadores incluyen Google, CoreOS, Mesosphere, Red Hat, Twitter, Huawei, Intel, Cisco, IBM, Docker, Univa y VMware. Hoy, CNCF cuenta con el apoyo de más de 450 miembros.

### Graduated projects

* Kubernetes
* container d
* Helm
* Fluentd  
  por mencionar algunos…

Hablando de limitación de recursos en pod : Los más importantes son cpu y memoria:  
"Esto es particularmente importante en el caso de la memoria. Sin límites, un contenedor con un proceso de roaming puede consumir rápidamente toda la memoria que ofrece su nodo. Un escenario de poca memoria podría eliminar otros pods programados en ese nodo, ya que el administrador de memoria a nivel del sistema operativo comenzaría a matar procesos para reducir el uso de memoria.

Establecer un límite de memoria permite a Kubernetes terminar el contenedor antes de que comience a afectar otras cargas de trabajo en el clúster, y mucho menos los procesos externos. Pierde su carga de trabajo, pero el clúster global gana estabilidad."  
[https://www.tremplin-numerique.org/es/cómo-establecer-los-límites-de-recursos-del-pod-de-kubernetes](https://www.tremplin-numerique.org/es/c%C3%B3mo-establecer-los-l%C3%ADmites-de-recursos-del-pod-de-kubernetes).

Vision de Google para Aplicaciones Modernas

El futuro de las aplicaciones y la infraestructura en la que se ejecutan, se crea con microservicios en contenedores. Administrados a través de un sistema declarativo con una experiencia de control única que utiliza una malla de servicios para abarcar todas las ubicaciones de las aplicaciones

Kubernetes está construido como el sistema interno para maximizar la productividad de los desarrolladores de software. esta hecho Open Source.

¿Como se ve el rendimiento máximo?

-208 veces más de despliegues de códigos frecuentes

-106 veces más rápido el tiempo de commit a deploy

-2.604 veces más rápido el tiempo de recuperacion ante un incidente

-7 veces menos de taza de cambios con fallos.

Kubernetes era complicado, con el tiempo se dieron cuenta que era demasiado complicado… (aparición de Anthos)

Anthos

Anthos es una plataforma administrada que extiende los servicios y prácticas de ingeniería de Google Cloud en tus ambientes para que puedas modernizar la aplicación más rápido y establecer consistencia operacional entre ellas.

ARQUITECTURA DE UN DESPLIEGUE HÍBRIDO Y MULTI-NUBE

Conectando tu cluster a Google

* + - Registrar un cluster con GCP e instala el GKE Conecct Agent en tu celular.
    - No se requiere de una IP pública.
    - Traversa VPNS, Nats, Firewalls y Proxies

**Anthos on VMware High Level Architecture**

ADMIN CLUSTER

Maneja el ciclo de vida de las operaciones.

Monitorea la salud del plano de control de los clusters de usuarios.

USER CLUSTER

Ejecuta cargas de trabajo de usuario.

Puede variar en tamaño y configuración.

*ANTHOS ON VMware Arch Overview*

1)Admin Workstation

. Workstation de Linux

. Disponible como OVA (open virtualización appliance)

. Contiene herramienta para realizar la instalación, escalamiento y actualización

2) Jump host

. Puede ser la “laptop” o el “desktop” de un usuario.

. Administra el ciclo de vida del admin Workstation utilizando la herramienta “gkdadmin”(CLI)

. Soporta Linux, Windows y pronto Mac.

3)Admin CLuster

. Administra el ciclo de vida de operaciones para los cliusters de usuario: install, upgrade, etc.

. Monitorea la salud del plano de control de los clusters de usuario.

. Ejecuta addons de sistema como seguridad, logging , mointoring.

. Instancia única por Vsphere Datacenter.

4) User Cluster(s)

. Ejecuta cargas de trabajo de usuario(aplicaciones)

. Puede variar en tamaño y configuración.

. Puede ser desplegado en un vsphere datastores separado

. Cada admin cluster soporta 10 user cluster con 100 nodos

5)Load Balancer.

. Balancea cargas de trabajo y el master/api server de k8s

. soporta:

-Bunded-LB Balanceador provisto y soportado por Google (recomendado)

-f5 BIG IP (Físico y virtual)

- Bring your load balancer (BYOLB) Citrix

**ARQUITECTURA MULTI-NUBE**

**. GKE Management Service** es desplegado en la misma VPC que los cluster administra

. GKE Mangement Service hace peticiones al API de AWS/AZURE para provisionar recursos del cluster.

. El connect agent es desplegado en ele cluster para habilitar capacidades de administración y visibilidad multinube desde la consola de Google CLoud

Google Kubernetes Engine

AUTO-REPAOR

AUTO-UPGRADE

AUTO-SCALE

Conectando tu cluster a Google

. Registra un cluster con GCP e instala el GKE Conect Agent en tu cluster

. No se requiere de una IP pública.

. Traversa vpnS, nats, Firewalls y Proxies,

## Introducción a la malla de servicios (Service Mesh)

#### ¿Qué es una malla de servicio?

Una malla de servicio provee una manera transparente e independiente al lenguaje para automatizar funciones de red de forma fácil y flexible

* Separa las aplicaciones del networking de las aplicaciones
* Desacopla la operación del desarrollo
* Provee un framework consistente para administrar la multi-nube

#### ¿Qué es Istio?

Una malla de servicio. Pero aún más: es una **plataforma de servicios abiertos** para manejar las **interacciones entre servicios** entre contenedores y VMs

#### Beneficios

* Administra el flujo de tráfico de entrada, salida y dentro de tus servicios [Traffic splitting, canary, blue-green, mirroring…]
* Acceso y comunicación segura entre tus servicios usando políticas [mTLS,…]
* Visibilidad completa de todo lo que está pasando con tus servicios con instrumentación mínima

#### ¿Por qué habilitar una malla de servicio?

1. Manera consistente de monitorear rendimiento, tasas de error y la postura de seguridad.
2. Te ayuda a construir servicios resilientes con despliegues seguros y failover zonal o regional.
3. Puedes tener seguridad consistente en tus servicios, incluyendo encriptación en tránsito, redes de cero confianza y protección de PII.

#### Componentes funcionales de Istio

**Pilot**: Plano de control para configurar y mandar políticas de comunicación

**Citadel**: Servicio-a-servicio auth[n,z] usando TLS mutuo, con manejo de identidades y credenciales incorporado

#### La magia del sidecar

● Desplegado con cada carga de trabajo  
● Proxy de todo el tráfico de entrada y salida del servicio  
● Direcciona tráfico (incluyendo reglas de enrutamiento  
● Hace cumplir las políticas  
● Reporta la telemetría  
● Todo sin tener que embeber librerías de cliente

**Beneficios de ASM sobre OSS lstio**

* Mantenida y soportada
* Tableros integrados ( SLOs y Tablero de seguridad)
* Funcionalidad de Enterprise

*Única restricción*

*Para instalar una malla* de servicio (Service Mesh) al cluster, hay una restricción: **los nodos deben tener mínimo 4 vCPUs**

**Beneficios de la malla de servicios.**

**Observabilidad unificada**  
-Flujo consistente de logs y métricas  
-Monitorear comportamiento que implica al usuario  
-Entender dependencia de servicios  
-Monitorear SLOs de salud y presupuestos de error  
-Peticiones de tracing y flujos de respuesta

**Afilidad operacional**  
-Bifurcación de tráfico  
-Conducción del tráfico  
-Control de egreso  
-Circuit breaking  
-Inyección de fallas

**Seguridad dirigida por políticas**  
-Encripción del tráfico  
-Auth de servicios  
-Controles de auditoria  
-Políticas de acceso

**INTRODUCCIÓN A Istio Ingress**

**TIPOS DE SERVICIOS:**

**Cluster IP**  
-Asigna una IP interna al cluster para el servicio  
-El servicio no es expuesto a recursos fuera del cluster

**NodePort**  
-Genera un mapeo de cada nodo del cluster a un servicio  
-Las peticiones que llegan a un nodo alcanzan los pods en el cluster por reglas de iptables (config. por kube-proxy)  
-Las peticiones pueden ser enrutadas a cualquier pod  
-Utilizado para exponer servicios dentro de una VPC

**Load Balancer**  
-La manera estándar de exponer tu servicio al internet  
-En GKE, genera un balanceador de carga que te da una IP única (dirigir trafico al servicio)  
-Cada vez que expones un servicio, se generará una IP por servicio (se vuelve caro)

**Ingress**  
-Actua como router inteligente que sirve como la entrada al cluster  
-En GKE, el controlador de Ingress genera un balanceador de carga HTTP(S)  
-Es la manera más útil de exponer múltiples servicios usando una IP única  
-Más funcionalidades de caja (SSL, Routing, Auth, etc)