Universidad Nacional Autónoma de México

Facultad de Ingeniería

Sistemas Operativos

Orquestadores de contenedores

Integrantes del equipo:

- **Garcia Riba Emilio**
- **Rivas Gil Maria Lucia**

Semestre 2026-1

ÍNDICE

Conceptos basicos

Para entender el documento es necesario saber los siguientes terminos.

- Maquina host / máquina padre:en este contexto se utilizan de forma intercambiable. Es la máquina en donde se va a ejecutar la virtualización.
- Virtualización: Se puede definir como la separación de los recursos físicos de una máquina padre a entornos virtuales.
- Maquinas virtuales: Forma de virtualizar una maquina a través de la particionar el hardware de la máquina host.
- Contenedores: Un contenedor es un proceso aislado que se ejecuta en un sistema operativo host que contiene todo lo necesario para correr una aplicación.
- CLI: es la línea de comandos con cual se puede interactuar con un software
- Daemon: programa que se encarga de la administración en segundo plano de otros programas.
- Nodos: la unidad de los orquestadores donde se hace un trabajo, como una aplicación
- Pod: la unidad más pequeña cuando se habla de orquestadores. Sería un proceso, podría ser un contenedor.
- Cluster: es la combinación de múltiples nodos.

Introducción

En general las aplicaciones se buscan que hoy en dia sean resilientes, escalables y portables. Esta necesidad aunque hay muchas soluciones para esto hoy hablaremos de los orquestadores de contenedores y contenedores.

Primero, llegó la infraestructura física local. Las empresas que querían ofrecer un servicio online necesitaban adquirir sus propios servidores. Este modelo presentaba un sobrecostos operativo enorme para el funcionamiento de su servicio. Instalar y configurar hardware físico es lento y costoso. Además, era difícil predecir la capacidad necesaria.

La virtualización busca solucionar esto. Las VMs (máquinas virtuales) permiten crear múltiples entornos de computación en una única máquina física a través de la virtualización. Su gran ventaja es la reproducibilidad: se pueden crear plantillas de VMs preconfiguradas, haciendo el desarrollo y el despliegue de entornos consistentes mucho más rápido y confiable que con hardware físico.

Sin embargo, las VMs tienen una desventaja crucial: su peso e ineficiencia. Cada VM es prácticamente una computadora completa, requiriendo una copia completa de un sistema operativo invitado, lo que consume grandes cantidades de recursos. Cuando están activas las VM’s utilizaran sus recursos alocados. No tiene un alojamiento de recursos haciendo que tenga

mas recursos de los que necesita o menos. Cuando se encuentra inactiva la mayor parte de sus recursos se regresan a la maquina padre.

Los contenedores emergieron como la respuesta directa a las ineficiencias de las VMs. Un contenedor es similar a una VM en concepto (empaqueta una aplicación y sus dependencias), pero con una arquitectura fundamentalmente diferente. En lugar de virtualizar hardware, virtualiza el sistema operativo. Extenderemos sobre los contenedores en su sección pero una forma de verlos es como VM's pero con mejor rendimiento y mejor utilización de recursos.

Cuando las aplicaciones pasan de ser un único contenedor a un conjunto complejo de microservicios (una base de datos, un backend, un frontend, un servicio de caché, etc.), gestionarlos manualmente se vuelve imposible. Para este propósito se crearon los orquestadores de contenedores. Este funciona como un masajeador de múltiples contenders. Igualmente en su sección lo explicaremos pero es como el administrador de contenedores.

Historia

La historia de los orquestadores de contenedores inicia en 2007, durante este año Google empezó a hacer uso de contenedores internos (cgroups) a gran escala para poder ejecutar aplicaciones, esto permitió que se tuviera un mejor manejo de los recursos del sistema; un año después (2008) los cgroups se integraron a Linux Kernel, ese mismo año ocurre el lanzamiento de LXC una tecnología de virtualización que permite aislar recursos sin necesidad de una máquina virtual.

En marzo del 2013 surge Docker y en paralelo CoreOS Linux, entre 2014 y 2015 se empieza a hablar del proyecto Kubernetes así como los contenedores rtk, la versión 1.0 de Docker y el Open Container Project, que estableció los estándares comunes para las imágenes y los runtimes de los contenedores.

A mediados de 2015 ocurre el lanzamiento de Kubernetes 1.0, marcando el inicio de la modernidad para los orquestadores de contenedores, dos años después, en 2017, Kubernetes añadió soporte completo y fue adoptado por distintas plataformas, entre ellas Rancher y Cloud Foundry, mientras que Microsoft Azure Container Instances comenzó a ofrecer gestión de contenedores.

En 2018, Google presentó Kubernetes Engine (GKE) como servicio en la nube y VMware adquirió Heptio, una empresa fundada por los creadores de Kubernetes, a mitad del 2019 aparecieron los servicios administrados de Amazon EKS y Microsoft AKS, al mismo tiempo IBM adquirió Red Hat y Mirantis compró Docker Enterprise.

Finalmente, en 2021, Red Hat adquiere StackRox y Google transfirió Knative a la Cloud Native Computing Foundation (CNCF).

Virtualización y sus implementaciones

Definición y funcionamiento de una Máquina Virtual

La idea de las VM's empieza con empezar a virtualizar desde hardware, entonces se crea una computadora (con su OS, RAM, CPU, etc.) a partir de los recursos de una máquina padre. Al momento de ser creadas se les asigna una cierta cantidad de recursos. Su administración se lleva a cabo por el hipervisor ya que básicamente administra las VM's y es esencial esto cuando hay múltiples VM's.

Definición y funcionamiento de un Contenedor

El contenedor empieza a virtualizar desde el OS. Incluirá todo lo necesario para poder ejecutar una aplicación; en el momento de la creación se especifican las características del OS del contenedor, además los recursos que se ocupen de la máquina host serán dados dinámicamente.

Diferencias

En la siguiente tabla se exponen las siguientes diferencias entre los dos:

Tema	VM's	Contenedores
Virtualizacion	Virtualiza desde hardware	Virtualiza desde OS
OS	Propio del VM	Compartido con el OS
Aislacion	Total	Por proceso
Tamaño en memoria	Mucho	Poco
Rendimiento	Bueno	Muy bueno
Portabilidad	Buena	Muy buena
Escalibilidad	Mala	Buena

Ventajas y Desventajas

En general aunque los dos tipos de virtualización tiene ventajas y desventajas similares donde se diferencian es en:

- Seguridad: el entorno de las VM's es más seguro gracias a su mayor separación de la máquina host a comparación de los contenedores.
- Rendimiento: Los contenedores son mejores ya que al usar recursos dinámicamente gastaran menos. Además al estar inactivos su uso sera de recursos sera casi nulo a diferencia de las VM's que si tendrá algo.
- Escalabilidad: Los contenedores son fácilmente escalables, normalmente echo en automático. Las VM's se les necesita asignar mas recursos.
- Portabilidad: Los contenedores son más fáciles de copiar y recrear en otras máquinas host o en la mismo que las VM's.

Por las razones anteriores hoy en dia se utilizan los contenedores sobre las VM's. Aunque cabe recalcar que tienen sus casos de uso.

Orquestación

¿Qué es?

El orquestador es la forma por la cual se puede organizar múltiples contenedores de tal forma que su despliege, manejo, escalarlos, etc. se haga en automático. Esto se vuelve importante cuando se trabaja con muchos contenedores en vez de aplicaciones monolíticas. Esto se ve en particular cuando los contenedores se junta con los microservicios haciendo que existan cientos de estos.
En general lo que se tiene son clusters que son la aplicación. Adentro del cluster hay varios nodos que se encargan de alguna característica de la aplicación. Finalmente están los pods que son los contenedores, programas especializados, recursos compartidos, etc. que hacen el trabajo que se necesita hacer.

Beneficios

Al momento de usar los orquestadores los mayores beneficios serían:

- Automatización: el proceso de manejar los contenedores se encarga el orquestador. Esto hace que un humano no lo tenga que hacer y permite la realización de aplicaciones con muchos contenedores posibles.
- Optimización: Los orquestadores pueden utilizar sus recursos de forma que se aprovechen de mayor forma que host utilizara cual pod.
- Errores: Reduce los errores al no necesitar humanos para su funcionamiento
- Resiliencia: Al encontrar errores o fallos el orquestador puede levantar automáticamente un nodo o reemplazarlo.

Docker

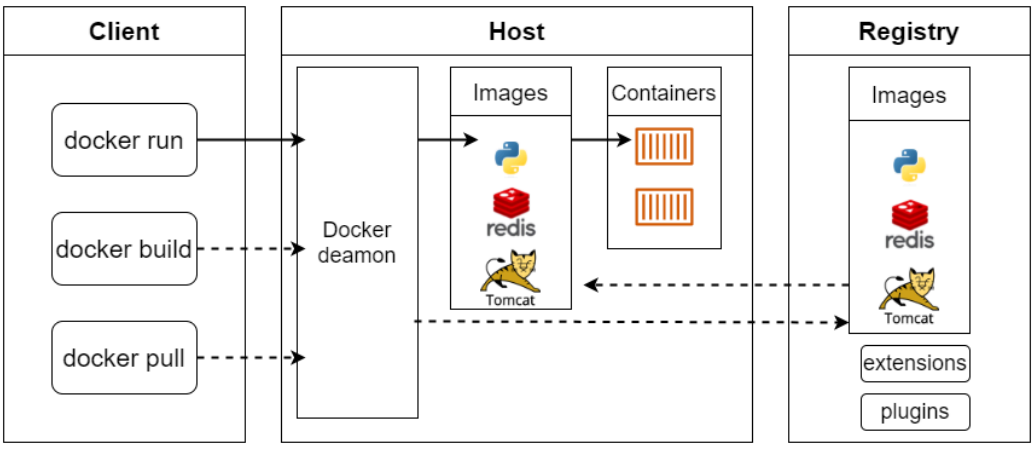
¿Qué es Docker?

Docker es una plataforma de código abierto que hace posible la simplificación del desarrollo, despliegue, pruebas, ejecución e implementación de aplicaciones en contenedores; como se mencionó anteriormente, surge en marzo del 2013 gracias a Solomon Hykes y fue construido sobre las capacidades que ofrece el kernel linux como namespaces para aislar procesos, redes y sistemas de archivos, y cgroups para limitar y monitorear el uso de recursos como CPU y memoria. Esto garantiza que las aplicaciones se ejecuten de manera segura y sin interferencias.

Arquitectura

La arquitectura de Docker se basa en un modelo cliente-servidor. El Cliente Docker (CLI) es la herramienta con la que los usuarios interactúan para ejecutar comandos, mientras que el Docker Daemon (dockerd) es el proceso en segundo plano que se encarga de construir, ejecutar y administrar contenedores, imágenes, volúmenes y redes.

Para crear un contenedor se utilizan las imágenes, que son plantillas de solo lectura con todas las instrucciones y aplicaciones necesarias para su ejecución. Estas imágenes se almacenan y comparten en los registros, que pueden ser públicos, como Docker Hub que es un repositorio, o privados dentro de una organización.



Funcionamiento

Para poder generar un contenedor de Docker el usuario tiene que introducir un comando dependiendo lo que quiera hacer, como:

- Docker Run: Se utiliza para crear e iniciar un nuevo contenedor Docker a partir de una imagen especificada.
- Docker build: Se usa para crear imágenes de Docker a partir de un Dockerfile y un contexto de compilación.
- Docker Pull: Se utiliza para descargar una imagen de un registro o repositorio de Docker.

Este comando se manda a través del Client Docker, este actúa como la interfaz de interacción del usuario, envía el comando al Docker Daemon mediante la API REST, que sirve como canal de comunicación entre ambos, después de esto el Docker Daemon, recibe la solicitud y busca la imagen especificada en un registro o repositorio de imágenes de imágenes.

Si la imagen no está disponible localmente, el demonio la descarga automáticamente desde el registro o repositorio. Una vez obtenida la imagen, el demonio crea un contenedor a partir de ella, configurando los parámetros necesarios según el comando recibido. Finalmente, el contenedor se ejecuta como un proceso aislado en el Host Docker, aprovechando los recursos compartidos del sistema operativo, pero manteniendo un entorno encapsulado que garantiza aislamiento y portabilidad. Este proceso permite que Docker pueda ejecutar aplicaciones de manera eficiente y consistente en cualquier sistema compatible.

Kubernetes

¿Qué es Kubernetes?

Kubernetes es una plataforma portable y extensible de código abierto que permite administrar cargas de trabajo y servicios, además facilita la automatización y configuración declarativa, es decir, describir el estado final deseado de un sistema, en lugar de especificar los pasos para lograrlo. Reiterando lo que se dijo en la sección de Historia, Kubernetes es lanzado en 2014 por google.

Arquitectura

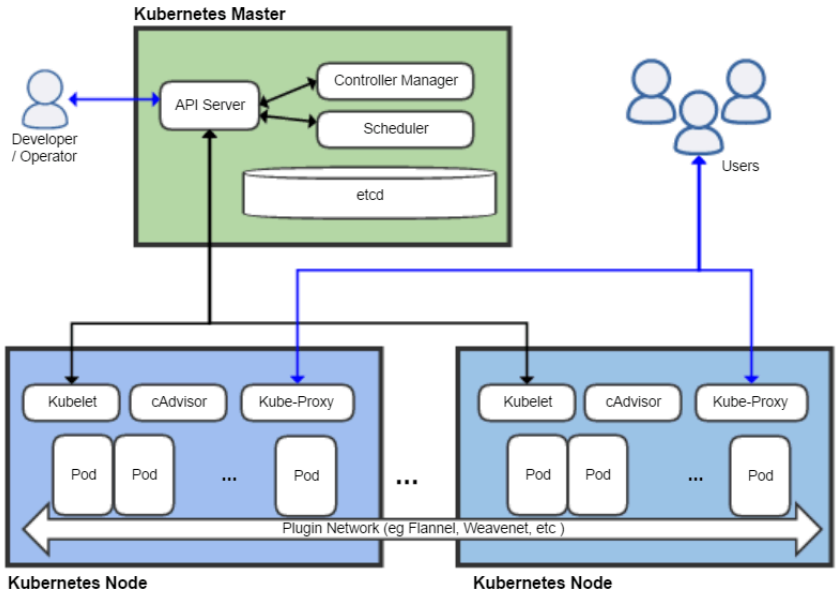
La arquitectura de Kubernetes está basada en nodos, los nodos son máquinas de trabajo, previamente conocidas como “minions”, pueden ser virtuales o físicas, esto depende del cluster.

Cada nodo es gestionado por el componente Master, el cual incluye varios elementos clave: el API Server, que actúa como el punto de entrada para todas las comunicaciones; etcd, almacenamiento de valores clave para crear el clúster; el Scheduler, que asigna los pods a los nodos más adecuados según los recursos y las restricciones; y el Controller Manager, que garantiza que el estado real del clúster coincida con el estado definido por el usuario.

Cada uno de los nodos contiene los servicios necesarios (runtime, kubelet y el kube-proxy) para poder ejecutar pods, pueden contener múltiples contenedores que comparten recursos, como la

dirección IP de red, el espacio de almacenamiento y el ciclo de vida, aunado a esto poseen un estado que se comprende de:

- Direcciones: (HostName, ExternalIP, InternalIP) dependen del proveedor de servicios en la nube y/o de la configuración en máquinas locales.
- Estados:(OutOfDisk, Ready, MemoryPressure, PIDPressure, DiskPressure, NetworkUnavailable) que se representa como un objeto JSON, estos describen el estado de los nodos que están corriendo
- Capacidad: Describe los recursos disponibles en el nodo: CPU, memoria, y el número máximo de pods que pueden ser planificados dentro del nodo.
- Información: Contiene la versión del kernel, versión de Kubernetes (versiones del kubelet y del kube-proxy), versión de Docker (si se utiliza), nombre del sistema operativo.



Funcionamiento

Kubernetes opera bajo un modelo declarativo, donde el usuario, como un desarrollador u operador, define el estado final deseado de las aplicaciones mediante archivos de configuración, generalmente en formato YAML. Estos archivos especifican objetos como Pods (unidades que contienen contenedores), Deployments (para gestionar réplicas), y Services (para la comunicación), esta declaración se envía al Master.

Dentro del Master, el API Server almacena el estado deseado en etcd, el controller manager monitorea el estado actual comparándolo con el estado deseado, mientras que el Scheduler asigna los Pods a los nodos disponibles.

En los nodos, el Kubelet es el agente que asegura que los contenedores definidos en los Pods estén ejecutándose y en buen estado, comunicándose con el API Server para poder recibir las instrucciones, por su parte el runtime recopila métricas de rendimiento de los contenedores,esto hace que el sistema a pueda tomar decisiones basándose en esa información y finalmente el Kube-Proxy gestiona las reglas de red y el balanceo de carga, permitiendo la comunicación entre Pods con el exterior a través de la red del clúster.

Durante este proceso si un nodo llegara a fallar, el Controller Manager detecta la pérdida de réplicas y junto con el Scheduler desplegará nuevos Pods en otros nodos disponibles, garantizando la tolerancia a los errores sin necesidad de una intervención manual, además, el Horizontal Pod Autoscaler puede ajustar dinámicamente el número de Pods según la carga de trabajo, optimizando recursos, mientras que el Cluster Autoscaler adapta el número de nodos si es necesario.

Conclusión

El recorrido desde la infraestructura física hasta la orquestación moderna con Docker y Kubernetes representa una evolución fundamental en la computación. Cada etapa superó las limitaciones de su predecesora: la virtualización abordó la rigidez del hardware físico, los contenedores resolvieron la ineficiencia de las máquinas virtuales, y los orquestadores añadieron inteligencia distribuida, escalabilidad y resiliencia.

Hoy, la simbiosis entre contenedores y orquestadores ha transformado radicalmente el diseño, despliegue y gestión de aplicaciones. Kubernetes, con su modelo declarativo, automatiza la recuperación ante fallos, el balanceo de carga y el escalado, convirtiendo infraestructuras complejas en sistemas cohesivos y programables. Aunque su adopción conlleva desafíos técnicos y de aprendizaje, el resultado es una infraestructura moderna, elástica y robusta. Esto permite a las organizaciones adaptarse rápidamente a la demanda, innovar con agilidad y optimizar sus recursos, consolidando así el paradigma de la computación nativa en la nube.

Bibliografia

- Amazon Web Services. (s. f.). *What's the difference between Docker and a VM?*. Recuperado el 13 de septiembre de 2025, de https://aws.amazon.com/compare/the-difference-between-docker-vm/
- Amazon Web Services. (s. f.). *Docker on AWS*. Recuperado el 13 de septiembre de 2025, de https://aws.amazon.com/es/docker/
- Docker, Inc. (s. f.). *Docker Docs*. Recuperado el 13 de septiembre de 2025, de https://docs.docker.com/
- Docker, Inc. (s. f.). *Manuals*. Recuperado el 13 de septiembre de 2025, de https://docs.docker.com/manuals/
- Hostinger Tutoriales. (2025, 2 de julio). *¿Qué es Docker? Funcionamiento y componentes básicos*. Recuperado el 13 de septiembre de 2025, de https://www.hostinger.com/es/tutoriales/que-es-docker
- Kubernetes Authors. (2024, 20 de octubre). *Cluster architecture*. *Kubernetes*. Recuperado el 13 de septiembre de 2025, de https://kubernetes.io/docs/concepts/architecture/
- Kubernetes Authors. (2025, 19 de junio). *Nodes*. *Kubernetes*. Recuperado el 13 de septiembre de 2025, de https://kubernetes.io/docs/concepts/architecture/nodes/
- Kubernetes Authors. (s. f.). *Conceptos básicos de Kubernetes*. *Kubernetes*. Recuperado el 13 de septiembre de 2025, de https://kubernetes.io/es/docs/concepts/
- Rodríguez C., J. A. (s. f.). *Manual-instalacion-docker.pdf* [Manuscrito no publicado]. Facultad de Ingeniería, Universidad Nacional Autónoma de México.