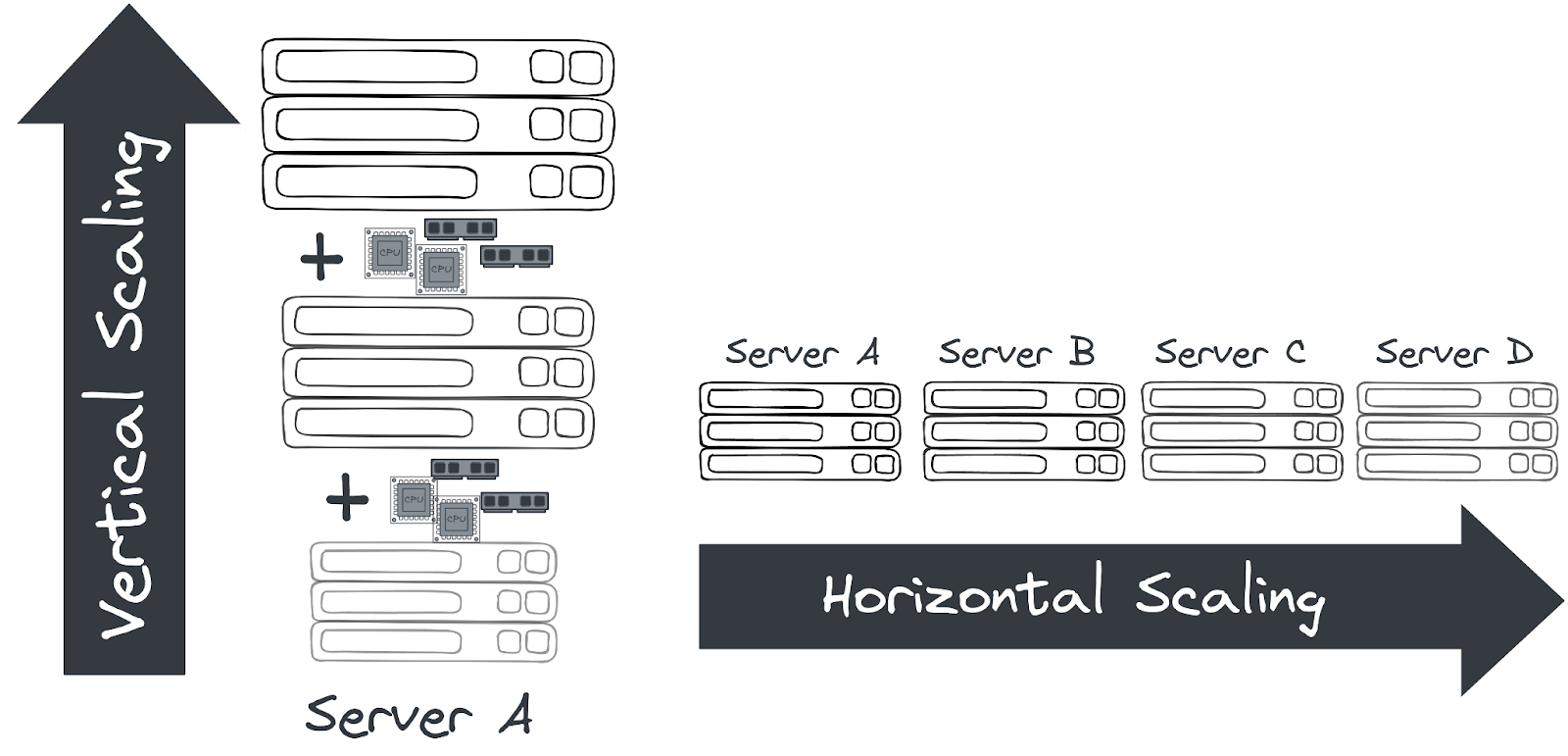
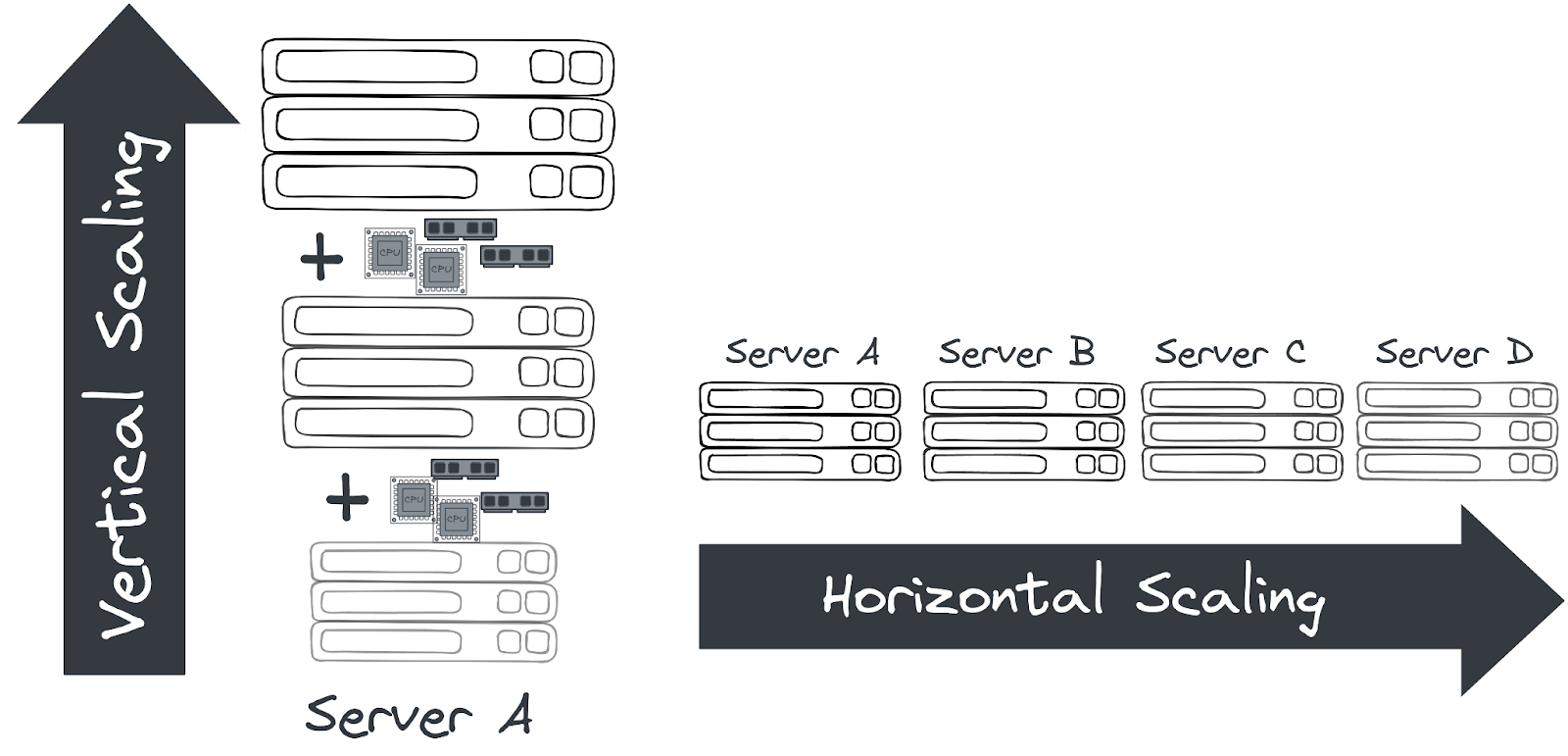
**AUTOSCALING**

El patrón de escalado automático describe el ajuste dinámico de los recursos en función de la demanda actual. La CPU y la memoria son las métricas obvias para decidir cuándo escalar las aplicaciones a medida que aumenta o disminuye la carga, pero también se pueden considerar otros métodos basados ​​en el tiempo o métricas comerciales para ampliar o reducir los servicios. Normalmente, cuando hablamos de escalado automático, hablamos de escalado horizontal, que describe el proceso de generación de nuevos recursos informáticos que pueden ser nuevas copias del proceso de su aplicación, máquinas virtuales o, de una manera menos inmediata, incluso nuevos bastidores de servidores. y otros equipos. El escalado vertical, por otro lado, describe el cambio de tamaño del hardware subyacente, que sólo funciona dentro de ciertos límites de hardware para máquinas bare metal, pero también para máquinas virtuales. Las máquinas y procesos virtuales se pueden escalar fácilmente permitiéndoles consumir más CPU y memoria; el límite superior está definido por la capacidad de computación y memoria del hardware subyacente. El hardware en si Puede ampliarse, por ejemplo, añadiendo más RAM, pero sólo hasta que todas las ranuras de RAM estén ocupadas. Para ilustrar la diferencia entre escalar vertical y horizontalmente, imagina que tienes que cargar un objeto pesado que no puedes levantar. Puedes desarrollar músculo para llevarlo tú mismo, pero tu cuerpo tiene un límite superior de fuerza. Eso es escalamiento vertical. También puedes llamar a tus amigos y pedirles que te ayuden y compartan el trabajo. Eso es escalamiento horizontal.

X

Escalado horizontal versus vertical Configurar el escalado automático en varios entornos requiere configurar un límite mínimo y máximo de instancias (máquinas virtuales o contenedores) y una métrica que desencadene el escalado. Para configurar el escalado correcto, a menudo es necesario ejecutar muchas pruebas de carga (casi de producción) y analizar el comportamiento y el equilibrio de carga cuando se escala la aplicación.

Los entornos de nube que dependen del uso basado en modelos de precios bajo demanda proporcionan plataformas muy efectivas para el escalamiento automático, con la capacidad de aprovisionar una gran cantidad de recursos en segundos o incluso escalar a cero, si los recursos no son necesarios temporalmente. Incluso si el escalado de sus aplicaciones y la infraestructura subyacente no está automatizado al principio, la capacidad de escalar su aplicación puede aumentar la disponibilidad y la resiliencia de sus servicios en entornos más tradicionales.

**SERVERLESS**

Sin servidor Al contrario de lo que sugiere el término "sin servidor", los servidores siguen siendo necesarios como base para sus aplicaciones. Los proveedores de la nube sugieren que es muy fácil implementar aplicaciones, pero requieren que prepare y configure varios recursos como una red, máquinas virtuales, sistemas operativos y balanceadores de carga para ejecutar una aplicación web simple. La idea de la informática sin servidor es aliviar a los desarrolladores de estas complicadas tareas. En pocas palabras, puede simplemente proporcionar el código de la aplicación, mientras el proveedor de la nube elige el entorno adecuado para ejecutar su aplicación. Todos los proveedores de nube populares tienen una o más ofertas comerciales de tiempos de ejecución sin servidor propietarios y un subconjunto de servicios sin servidor, también conocidos como Función como Servicio (FaaS). El proveedor de la nube abstrae la infraestructura subyacente, de modo que los desarrolladores puedan implementar software cargando su código, por ejemplo, como archivos .zip o proporcionando una imagen de contenedor. A diferencia de otros modelos de computación en la nube, la computación sin servidor se centra aún más en el aprovisionamiento y escalamiento de aplicaciones bajo demanda. El escalado automático es un concepto central importante de la tecnología sin servidor y puede incluir escalado y aprovisionamiento en función de eventos como solicitudes entrantes. Esto permite una facturación aún más precisa, que puede basarse en los eventos mencionados en lugar de la habitual facturación basada en el tiempo. En lugar de reemplazar completamente las plataformas de orquestación de contenedores o las máquinas virtuales más tradicionales, los sistemas FaaS a menudo se usan en combinación o como una extensión de las plataformas existentes, ya que permiten una implementación muy rápida y crean excelentes entornos de prueba y sandbox. Si bien las imágenes de contenedores son una excelente forma estandarizada de empaquetar software para FaaS o sistemas sin servidor, sistemas como Knative, que se construyen sobre Kubernetes, permiten ampliar las plataformas existentes con capacidades informáticas sin servidor. Esta puede ser una solución viable para el funcionamiento sin servidor en nubes privadas y entornos locales. Tenga en cuenta que esto puede facilitar el trabajo de un desarrollador, pero aumenta la complejidad del funcionamiento de una plataforma en la nube. Aunque la tecnología sin servidor tiene muchas ventajas, inicialmente tuvo problemas con la estandarización. Muchos proveedores de nube tienen ofertas patentadas que dificultan el cambio entre diferentes plataformas. Para abordar estos problemas, se fundó el proyecto CloudEvents y proporciona una especificación de cómo se deben estructurar los datos de eventos. Los eventos son la base para escalar cargas de trabajo sin servidor o activar las funciones correspondientes. Cuantos más proveedores y herramientas adopten este estándar, más fácil será utilizar arquitecturas sin servidor y basadas en eventos en múltiples plataformas. CloudEvents es una especificación para describir datos de eventos de forma común. CloudEvents busca simplificar drásticamente la declaración y entrega de eventos en todos los servicios, plataformas y más. CloudEvents es un esfuerzo nuevo y todavía está en desarrollo activo. Sin embargo, su grupo de trabajo ha recibido una sorprendente cantidad de interés de la industria, desde los principales proveedores de la nube hasta las populares empresas SaaS. La especificación ahora está bajo la Cloud Native Computing Foundation. Las aplicaciones escritas para plataformas sin servidor tienen requisitos aún más estrictos para la arquitectura nativa de la nube, pero al mismo tiempo pueden beneficiarse al máximo de ellos. Escribir aplicaciones pequeñas y sin estado las convierte en la opción perfecta para flujos de datos o eventos, tareas programadas, lógica empresarial o procesamiento por lotes.

**Open Standars**

Estándares abiertos Muchas tecnologías nativas de la nube dependen en gran medida del software de código abierto, lo que puede hacer que sea mucho más fácil comenzar y evitar la dependencia de un proveedor. Todos pueden colaborar en un proyecto de código abierto, lo que facilita la implementación de estándares en toda la industria que incluso se encuentran en productos comerciales y ofertas de proveedores de nube. Un problema común es cómo crear y distribuir paquetes de software, ya que las aplicaciones tienen muchos requisitos y dependencias para el sistema operativo subyacente y el tiempo de ejecución de la aplicación. Para superar este problema, los contenedores evolucionaron como una forma estandarizada de empaquetar y enviar software moderno. Si bien Docker se utiliza a menudo como sinónimo de tecnologías de contenedores, la comunidad se ha comprometido con el estándar industrial abierto de Open Container Initiative (OCI). Bajo el paraguas de la Fundación Linux, la Open Container Initiative proporciona dos estándares que definen la forma de construir y ejecutar contenedores. La especificación de imagen define cómo crear y empaquetar imágenes de contenedores. Mientras que la especificación de tiempo de ejecución especifica la configuración, el entorno de ejecución y el ciclo de vida de los contenedores. Una incorporación más reciente al proyecto OCI es Distribution-Spec, que proporciona un estándar para la distribución de contenido en general y de imágenes de contenedores en particular. Los estándares abiertos como este ayudan y complementan otros sistemas como Kubernetes, que es la plataforma estándar de facto para orquestar contenedores. Algunos estándares que descubrirá en los siguientes capítulos son:

Especificaciones OCI: especificación de imagen, tiempo de ejecución y distribución sobre cómo ejecutar, construir y distribuir contenedores

Interfaz de red de contenedores (CNI): una especificación sobre cómo implementar redes para contenedores.

Interfaz de ejecución de contenedores (CRI): una especificación sobre cómo implementar tiempos de ejecución de contenedores en sistemas de orquestación de contenedores.

Interfaz de almacenamiento de contenedores (CSI): una especificación sobre cómo implementar el almacenamiento en sistemas de orquestación de contenedores.

Service Mesh Interface (SMI): una especificación sobre cómo implementar Service Meshes en sistemas de orquestación de contenedores con un enfoque en Kubernetes. Siguiendo este enfoque, otros sistemas como Prometheus u OpenTelemetry evolucionaron y prosperaron en este ecosistema y proporcionan estándares adicionales para el monitoreo y la observabilidad.

**Ingeniero de Confiabilidad del Sitio (SRE)** Un rol con una definición más sólida es el de Ingeniero de Confiabilidad del Sitio (SRE). SRE se fundó alrededor de 2003 en Google y se convirtió en un trabajo importante para muchas organizaciones. El objetivo general de SRE es crear y mantener software que sea confiable y escalable. Para lograr esto, se utilizan enfoques de ingeniería de software para resolver problemas operativos y automatizar tareas operativas. Para medir el rendimiento y la confiabilidad, los SRE utilizan tres métricas principales:

* Objetivos de nivel de servicio (SLO): "Especifique un nivel objetivo para la confiabilidad de su servicio". - Un objetivo que se establezca, por ejemplo alcanzar una latencia de servicio inferior a 100ms.
* Indicadores de nivel de servicio (SLI): “Una medida cuantitativa cuidadosamente definida de algún aspecto del nivel de servicio que se brinda”; por ejemplo, cuánto tiempo realmente se debe responder a una solicitud.
* Acuerdos de nivel de servicio (SLA): “Un contrato explícito o implícito con sus usuarios que incluye las consecuencias de cumplir (o no cumplir) los SLO que contienen. Las consecuencias se reconocen más fácilmente cuando son financieras (un reembolso o una multa), pero pueden adoptar otras formas”. - Responde a la pregunta ¿Qué pasa si los SLO no soy yo?

**ARQUITECTURA FUNDAMENTALS**

Fundamentos de la arquitectura nativa de la nube

En esencia, la idea de la arquitectura nativa de la nube es optimizar su software para lograr rentabilidad, confiabilidad y un tiempo de comercialización más rápido mediante el uso de una combinación de patrones de diseño culturales, tecnológicos y arquitectónicos.

El término nativo de la nube se puede encontrar en varias definiciones, algunas de las cuales se centran en las tecnologías, mientras que otras pueden centrarse en el lado cultural de las cosas.

La Cloud Native Computing Foundation lo define de la siguiente manera:

“Las tecnologías nativas de la nube permiten a las organizaciones crear y ejecutar aplicaciones escalables en entornos modernos y dinámicos, como nubes públicas, privadas e híbridas. Los contenedores, las mallas de servicios, los microservicios, la infraestructura inmutable y las API declarativas ejemplifican este enfoque.

Estas técnicas permiten sistemas débilmente acoplados que son resilientes, manejables y observables. Combinados con una automatización sólida, permiten a los ingenieros realizar cambios de alto impacto de manera frecuente y predecible con un mínimo esfuerzo. [...]”

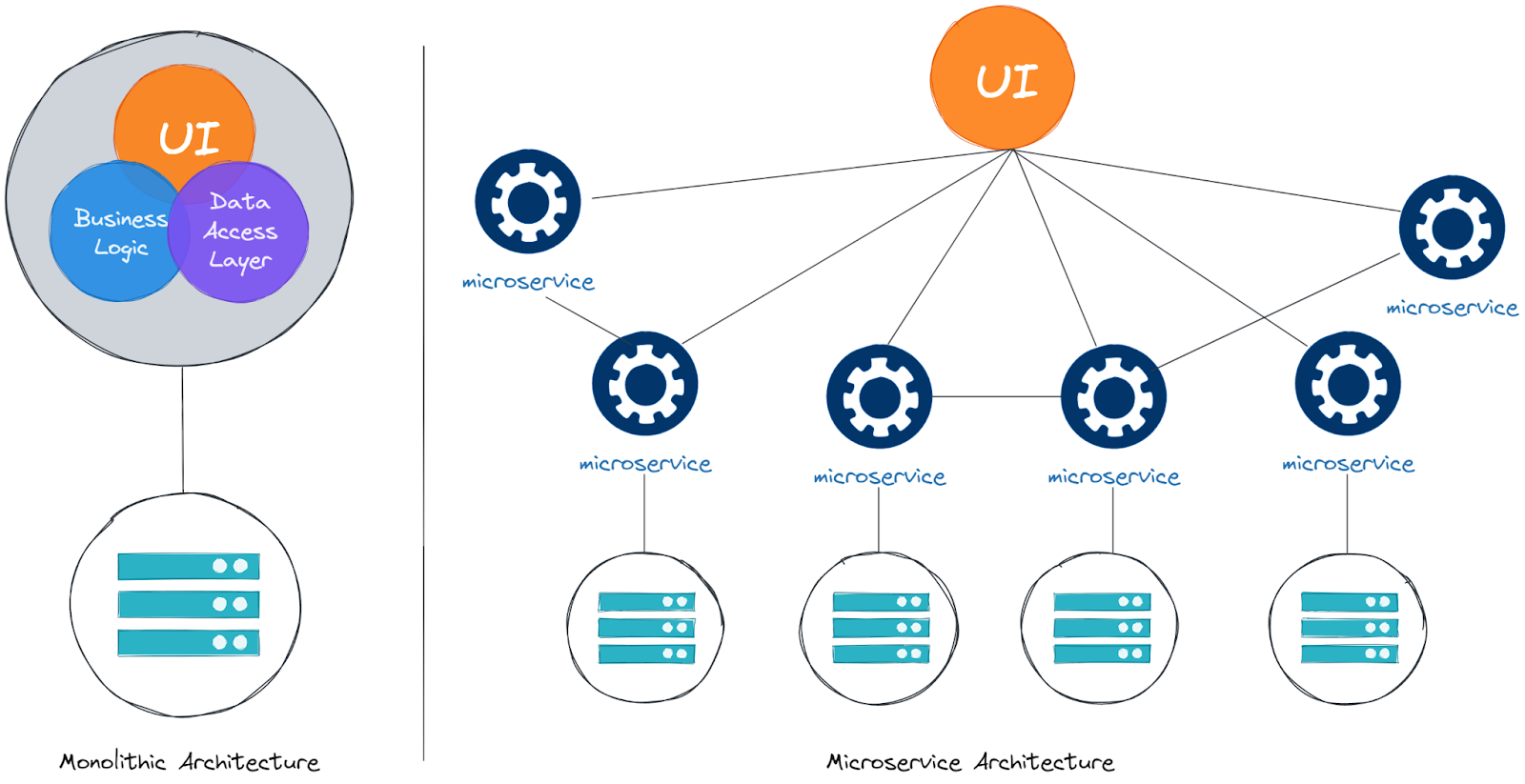
Las aplicaciones tradicionales o heredadas suelen diseñarse con un enfoque monolítico en mente, lo que significa que son autónomas e incluyen todas las funciones y componentes necesarios para realizar una tarea. Una aplicación monolítica suele tener una única base de código y se proporciona como un único archivo binario que puede ejecutarse en un servidor.

Si piensa en un software de comercio electrónico para una tienda en línea, una aplicación monolítica incluiría todas las funciones, desde la interfaz gráfica de usuario, listado de productos, carrito de compras, pago, procesamiento de pedidos y mucho más.

Si bien puede ser muy fácil desarrollar e implementar una aplicación en este formato, puede ser igualmente difícil gestionar la complejidad, escalar el desarrollo en varios equipos, implementar cambios rápidamente y poder escalar la aplicación de manera eficiente cuando se trata de muchas cosas. de carga.

La arquitectura nativa de la nube puede proporcionar soluciones para la creciente complejidad de las aplicaciones y la creciente demanda de los usuarios. La idea básica es dividir la aplicación en partes más pequeñas, lo que las hace más manejables. En lugar de proporcionar todas las funciones en una sola aplicación, tiene varias aplicaciones desacopladas que se comunican entre sí en una red. Si nos atenemos al ejemplo anterior, podría tener una aplicación para su interfaz de usuario, su pago y todo lo demás. Estas aplicaciones pequeñas e independientes con un alcance de funciones claramente definido a menudo se denominan microservicios.

Eso hace posible tener varios equipos, cada uno de los cuales es propietario de diferentes funciones de su aplicación, pero también operarlas y escalarlas individualmente. Por ejemplo, si mucha gente intenta comprar productos, puedes escalar servicios que tienen mucha carga, como el carrito de compras y el pago.



Arquitectura monolítica versus microservicios La arquitectura nativa de la nube puede tener muchas ventajas, pero también puede ser muy compleja de integrar y, por lo tanto, debe cumplir algunos requisitos para funcionar de manera eficiente.

Caracteristicas

Alto nivel de automatización Para gestionar todas las partes móviles de su aplicación nativa en la nube, se recomienda la automatización en cada paso, desde el desarrollo hasta la implementación. Esto se puede lograr mediante el uso de herramientas de automatización modernas y canalizaciones de integración continua/entrega continua (CI/CD) (hablaremos más sobre canalizaciones más adelante en el curso, pero por ahora, sepa que una canalización de CI/CD es un concepto que puede se utilizará para los múltiples pasos que se necesitan al entregar una nueva versión de su software) que están respaldados por un sistema de control de versiones como git. Crear, probar e implementar aplicaciones e infraestructura con una mínima participación humana permite realizar cambios rápidos, frecuentes e incrementales en la producción. Un sistema automatizado confiable también permite una recuperación ante desastres mucho más sencilla si tiene que reconstruir todo el sistema.

**Autosanación**

Las aplicaciones y la infraestructura fallan de vez en cuando. Esto es de esperar y, por lo tanto, los marcos de aplicaciones y los componentes de infraestructura nativos de la nube incluyen controles de estado que ayudan a monitorear su aplicación desde adentro y reiniciarla automáticamente si es necesario. Además, dado que la aplicación se ha compartimentado, existe la posibilidad de que solo partes de la aplicación dejen de funcionar o se vuelvan más lentas, mientras que otras partes no.

**Escalable** Escalar su aplicación describe el proceso de manejar más carga y al mismo tiempo brindar una experiencia de usuario agradable. Una forma de escalar puede ser iniciar varias copias de la misma aplicación y distribuir la carga entre ellas. Automatizar este comportamiento en función de métricas de la aplicación, como CPU o memoria, también puede garantizar la disponibilidad y el rendimiento de sus servicios.

**(Costo-) Eficiente**

Al igual que ampliar su aplicación para situaciones de mucho tráfico, reducir su aplicación y los modelos de precios basados ​​en el uso de los proveedores de la nube pueden ahorrar costos si el tráfico es bajo. Para optimizar el uso de su infraestructura, los sistemas de orquestación como Kubernetes pueden ayudar con una ubicación de aplicaciones más eficiente y densa.

Facil de mantener

El uso de microservicios permite dividir las aplicaciones en partes más pequeñas y hacerlas más portátiles, más fáciles de probar y distribuir entre múltiples equipos.

Seguro por defecto

Los entornos de nube suelen ser compartidos entre varios clientes o equipos, lo que requiere diferentes modelos de seguridad. En el pasado, muchos sistemas estaban divididos en diferentes zonas de seguridad que negaban el acceso a diferentes redes o personas. Una vez que esté dentro de una zona, podrá acceder a todos los sistemas internos. Patrones como la informática de confianza cero lo mitigan al requerir autenticación de cada usuario y proceso.

Una buena base y punto de partida para su viaje nativo a la nube es la aplicación de doce factores. La aplicación de **doce factores** es una guía para desarrollar **aplicaciones nativas de la nube,** que comienza con cosas simples como control de versiones de su código base, configuración consciente del entorno y patrones más sofisticados como apatridia y concurrencia de su aplicación. Si bien estos patrones y tecnologías brindan todas las ventajas si se ejecutan en la nube, también pueden ofrecer muchos beneficios cuando se aplican a sistemas locales. Por último, pero no menos importante, permiten una transición más fluida si migra sus aplicaciones e infraestructura a la nube.

**TRABAJOS REMOTOS**

Roles nativos de la nube e ingeniería de confiabilidad del sitio

Por supuesto, el cambio tecnológico y cultural que hemos visto en las últimas dos décadas ha resultado en un ajuste de tareas y descripciones de puestos. Las ofertas de trabajo anteriores incluían puestos como administrador de sistemas, redes o bases de datos, desarrollador de software o director de pruebas.

Los trabajos en computación en la nube son más difíciles de describir y las transiciones son más fluidas, ya que las responsabilidades a menudo se comparten entre varias personas provenientes de diferentes áreas y con diferentes habilidades.

**Arquitecto de la nube**

Responsable de la adopción de tecnologías en la nube, diseñando panorama e infraestructura de aplicaciones, con enfoque en seguridad, escalabilidad y mecanismos de implementación.

Ingeniero DevOps

A menudo se describe como una simple combinación de desarrollador y administrador, pero eso no le hace justicia al rol. Los ingenieros de DevOps utilizan herramientas y procesos que equilibran el desarrollo y las operaciones de software. Comenzando con enfoques para escribir, construir y probar software durante todo el ciclo de vida de la implementación.

Ingeniero de seguridad

Quizás el papel más fácil de entender. No obstante, el papel de los ingenieros de seguridad ha cambiado significativamente. Las tecnologías en la nube han creado nuevos vectores de ataque y hoy en día el rol debe vivirse de manera mucho más inclusiva y como parte integral de un equipo.

Ingeniero DevSecOps

En un esfuerzo por hacer de la seguridad una parte integral de los entornos de TI modernos, el ingeniero DevSecOps combina las funciones de los dos anteriores. Este rol se utiliza a menudo para construir puentes entre el equipo de desarrollo y de seguridad más tradicional.

Ingeniero de datos

Los ingenieros de datos enfrentan el desafío de recopilar, almacenar y analizar las grandes cantidades de datos que se recopilan o se pueden recopilar en grandes sistemas. Esto puede incluir el aprovisionamiento y la gestión de infraestructura especializada, así como el trabajo con esos datos.

Desarrollador Full-Stack

Un todoterreno que se siente cómodo en el desarrollo frontend y backend, así como en los elementos esenciales de infraestructura.

**Ingeniero de Confiabilidad del Sitio (SRE)**

Un rol con una definición más sólida es el de Ingeniero de Confiabilidad del Sitio (SRE). SRE se fundó alrededor de 2003 en Google y se convirtió en un trabajo importante para muchas organizaciones. El objetivo general de SRE es crear y mantener software que sea confiable y escalable. Para lograr esto, se utilizan enfoques de ingeniería de software para resolver problemas operativos y automatizar tareas operativas.

Para medir el rendimiento y la confiabilidad, los SRE utilizan tres métricas principales:

Objetivos de nivel de servicio (SLO): "Especifique un nivel objetivo para la confiabilidad de su servicio". - Un objetivo que se establezca, por ejemplo alcanzar una latencia de servicio inferior a 100ms.

Indicadores de nivel de servicio (SLI): “Una medida cuantitativa cuidadosamente definida de algún aspecto del nivel de servicio que se brinda”; por ejemplo, cuánto tiempo realmente se debe responder a una solicitud.

Acuerdos de nivel de servicio (SLA): “Un contrato explícito o implícito con sus usuarios que incluye las consecuencias de cumplir (o no cumplir) los SLO que contienen. Las consecuencias se reconocen más fácilmente cuando son financieras (un reembolso o una multa), pero pueden adoptar otras formas”. - Responde a la pregunta qué sucede si no se cumplen los SLO.

En torno a estas métricas, los SRE podrían definir un presupuesto de error. Un presupuesto de errores define la cantidad (o tiempo) de errores que puede tener su aplicación, antes de que se tomen medidas, como detener las implementaciones en producción.

**Comunidad y Gobernanza**

Muchos proyectos de código abierto que muchos consideran un estándar de la industria están a**lojados y respaldados por la Cloud Native Computing Foundation (CNCF).** Los proyectos se clasifican según su madurez y pasan por una etapa de pruebas y de incubación antes de graduarse. La sólida comunidad de CNCF puede respaldar proyectos durante todo su ciclo de vida, comenzando con aspectos como la visibilidad pública y la clasificación en el panorama de CNCF hasta que los proyectos maduren para su uso en producción.

La CNCF cuenta con un Comité de Supervisión Técnica (TOC) que es responsable de definir y mantener la visión técnica, aprobar nuevos proyectos, aceptar retroalimentación del comité de usuarios finales y definir prácticas comunes que deben implementarse en los proyectos de la CNCF.

Al mismo tiempo, el TOC no controla los proyectos, pero los alienta a que sean autónomos y de propiedad comunitaria y practica el principio de “gobernanza mínima viable”. Las pautas para estos proyectos incluyen cómo se mantienen, revisan y publican los proyectos, cuáles son sus usuarios y grupos de trabajo, y mucho más. Dado que los proyectos de código abierto dependen de sus respectivas comunidades, la gobernanza es muy diferente de los enfoques tradicionales. La gran libertad de las tecnologías nativas de la nube obliga a las personas a imponerse reglas y hacerlas cumplir.

Criterios de graduación CNCF v1.3

Hay un nivel de madurez asignado a cada iniciativa del CNCF. Los proyectos propuestos al CNCF deben especificar su grado de madurez preferido. Un proyecto debe recibir una mayoría calificada de dos tercios para ser aprobado como en incubación o graduado. Cualquier voto graduado se considera voto para unirse como proyecto en incubación si no hay una gran mayoría de votos para hacerlo. Cualquier voto graduado o en incubación se considera patrocinio para ingresar como proyecto sandbox si no hay una gran mayoría de votos para ingresar como proyecto en incubación. El proyecto se rechaza si no hay fondos suficientes para calificarlo para la etapa sandbox. La votación alternativa es el término para este método de elección.

**Niveles de madurez del proyecto**

Cerrar etapa Sandbox

Esta etapa es el punto de entrada para proyectos en etapa inicial. Los proyectos Sandbox deben ser proyectos en etapa inicial que el TOC del CNCF considere que justifican la experimentación. El Sandbox debería proporcionar un hogar neutral y beneficioso para este tipo de proyectos, con el fin de fomentar el desarrollo colaborativo. CNCF aspira a hacer de Sandbox el camino preferido para que proyectos en etapa inicial ingresen a CNCF. Los proyectos más maduros pueden continuar saltando directamente a la incubación, pero a medida que crece el ecosistema nativo de la nube, esperamos ver proporcionalmente más proyectos en etapa inicial.

**Etapa de incubación**

El proyecto para ser aceptado en la etapa de incubación debe haber cumplido con los requisitos de la etapa sandbox y además se ha realizado toda la debida diligencia técnica, que incluye:

Documentar que está siendo utilizado exitosamente en producción por al menos tres adoptantes directos independientes.

Tenga un número saludable de confirmadores. Un confirmador se define como alguien con el bit de confirmación; es decir, alguien que pueda aceptar contribuciones para parte o la totalidad del proyecto.

Demostrar un flujo sustancial y continuo de compromisos y contribuciones fusionadas.

Un esquema de versiones claro

Procesos de seguridad claramente documentados que explican cómo informar problemas de seguridad al proyecto y describen cómo el proyecto proporciona versiones actualizadas o parches para resolver vulnerabilidades de seguridad.

Las especificaciones deben tener al menos una implementación de referencia pública.

**Etapa de Graduación**

Para graduarse del estado sandbox o de incubación, o para que un nuevo proyecto se una como proyecto graduado, un proyecto debe cumplir con los criterios de la etapa de incubación más:

Tener comprometidos de al menos dos organizaciones.

Haber logrado y mantenido una insignia de mejores prácticas de la iniciativa de infraestructura básica.

Haber completado una auditoría de seguridad independiente y de terceros con resultados publicados de alcance y calidad similares y todas las vulnerabilidades críticas deben abordarse antes de graduarse.

Definir explícitamente un proceso de gobernanza y compromiso del proyecto.

Definir explícitamente los criterios, procesos y condiciones de baja o eméritos para los mantenedores de proyectos; o quienes puedan interactuar con la CNCF en nombre del proyecto. La lista de mantenedores debe almacenarse preferiblemente en un archivo MAINTAINERS.md y auditarse con una cadencia mínima anual.

Tenga una lista pública de adoptantes del proyecto para al menos el repositorio principal (por ejemplo, ADOPTERS.md o logotipos en el sitio web del proyecto). Para una especificación, tenga una lista de adoptantes para las implementaciones de la especificación.

Recibir una mayoría calificada del TOC para pasar a la etapa de graduación. Los proyectos pueden intentar pasar directamente de la zona de pruebas a la graduación, si pueden demostrar suficiente madurez. Los proyectos pueden permanecer en estado de incubación indefinidamente, pero normalmente se espera que se gradúen en un plazo de dos años.

loud native applications are \_\_\_\_\_\_\_\_\_\_\_?

Correct Answer

* A. Small, decoupled services
* B. Easy to maintain
* C. Self-healing
* D. **All of the above**

# Question 2.2

What are some characteristics of cloud native architecture?

Correct Answer

* A. High costs and maintenance
* **B. High automation and scalability**

**Your Answer:**

Correct

* C. High security risk and complexity
* D. All of the above

# Question 2.3

The Open Container Initiative provides container standards for \_\_\_\_\_\_\_\_\_\_.

Incorrect Answer

* A. Runtime, image, build
* **B. Runtime, image, distribution**

**Correct Answer**

* C. Image, build, distribution

Your Answer:

Incorrect

* D. Docker, build, image

# Question 2.4

What is the most common pattern for autoscaling in cloud environments?

Correct Answer

* A. Vertical Scaling
* B. Diagonal Scaling
* **C. Horizontal Scaling**

Your Answer:

Correct

* D. Parallel Scaling

# Question 2.5

Serverless computing does not require \_\_\_\_\_\_\_:

Correct Answer

* A. Servers
* **B. Provisioning and operating infrastructure**

**Your Answer:**

Correct

* C. Application Code
* D. Network

The twelve-factor app is a guideline to \_\_\_\_\_\_\_:

Correct Answer

* **A. Develop cloud native applications**

**Your Answer:**

**Correct**

* B. Build containers
* C. Deploy on Kubernetes
* D. All of the above

# Question 2.7

Which of the following is not a standard used for container orchestration?

Correct Answer

* A. Container Network Interface (CNI)
* **B. Container Deployment Interface (CDI)**

Your Answer:

Correct

* C. Container Runtime Interface (CRI)
* D. Container Storage Interface (CSI)

Which of the following is not a key metric for Site Reliability Engineers?

Correct Answer

* A. Service level agreement (SLA)
* B. Service level indicator (SLI)
* C. Service level objective (SLO)
* **D. Service level requirement (SLR)**

# **Learning Objectives**

By the end of this chapter, you should be able to:

* Explain what a container is and how it differs from virtual machines
* Describe the fundamentals of container orchestration
* Discuss the challenges of container networking and storage
* Explain the benefits of a service mesh

Uso de contenedores

La historia del desarrollo de aplicaciones es también la historia del empaquetado de estas aplicaciones para diferentes plataformas y sistemas operativos.

Tomemos como ejemplo una aplicación web sencilla escrita en el popular lenguaje de programación Python. Para ejecutar una aplicación en un servidor o en su máquina local, el sistema generalmente necesita cumplir requisitos específicos como:

Instalar y configurar el sistema operativo básico.

Instale los paquetes principales de Python para ejecutar el programa.

Instale las extensiones de Python que utiliza su programa

Configure la red para su sistema

Conéctese a sistemas de terceros como una base de datos, caché o almacenamiento.

Si bien el desarrollador es quien mejor conoce su aplicación y sus dependencias, generalmente es un administrador del sistema quien proporciona la infraestructura, instala todas las dependencias y configura el sistema en el que se ejecuta la aplicación. Este proceso puede ser muy propenso a errores y difícil de mantener, por lo que los servidores solo se configuran para un único propósito, como ejecutar una base de datos o un servidor de aplicaciones, y luego se conectan a través de la red.

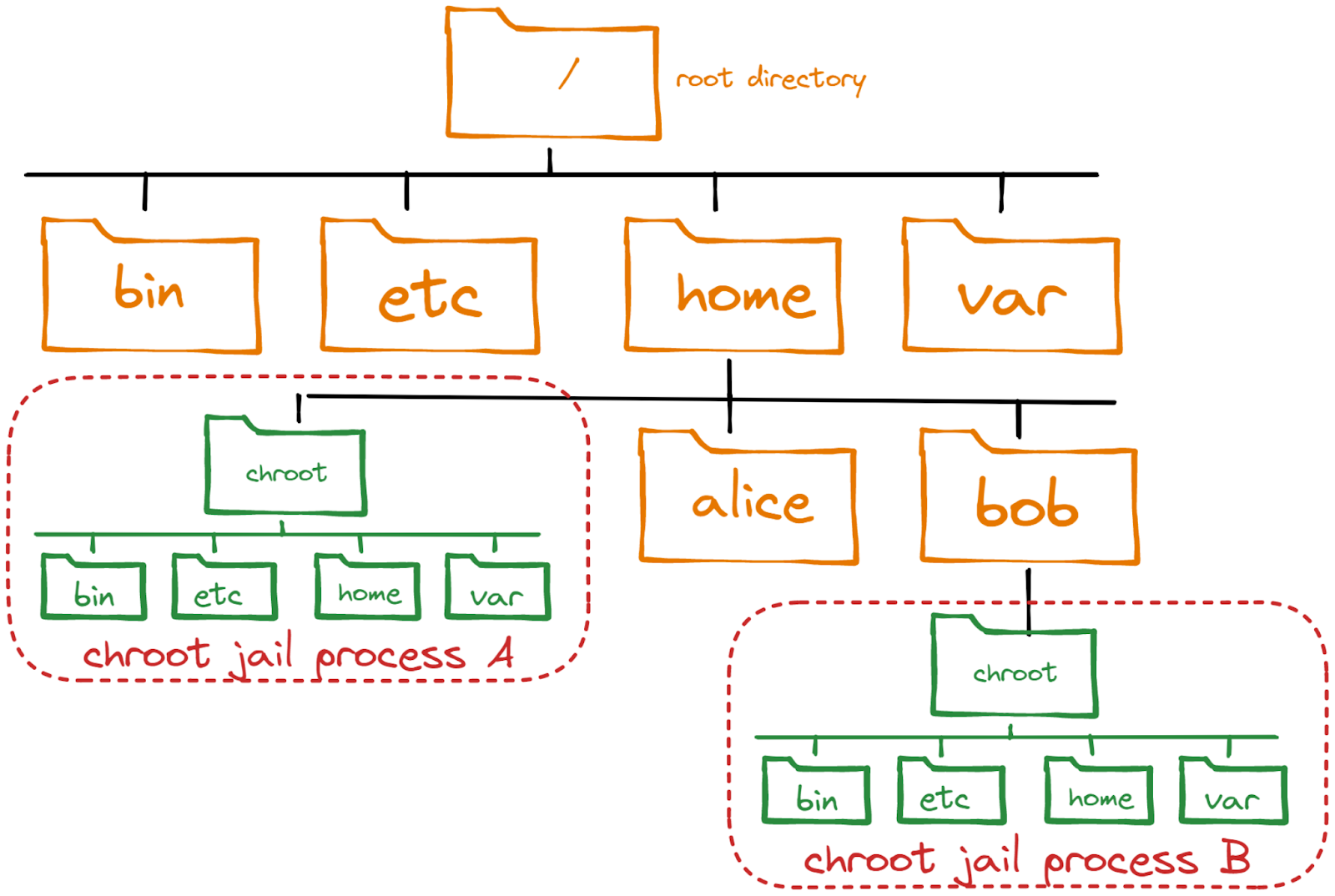
Para obtener un uso más eficiente del hardware del servidor, se pueden utilizar máquinas virtuales para emular un servidor completo con CPU, memoria, almacenamiento, redes, sistema operativo y software encima. Esto permite ejecutar múltiples servidores aislados en el mismo hardware.

Antes de la adopción generalizada de contenedores, la virtualización de servidores era la forma más eficiente de ejecutar aplicaciones aisladas y fáciles de manejar, pero como hay que ejecutar un sistema operativo completo, incluido el kernel, siempre conlleva cierta sobrecarga si es necesario ejecutar mucho. de servidores.

Los contenedores se pueden utilizar para resolver ambos problemas, gestionar las dependencias de una aplicación y ejecutarla de forma mucho más eficiente que poner en marcha muchas máquinas virtuales.

**Conceptos básicos de contenedores**

Contrariamente a la creencia popular, las tecnologías de contenedores son mucho más antiguas de lo que cabría esperar. Uno de los primeros antepasados ​​de las tecnologías de contenedores modernas es el comando **chroot que se introdujo en la versión 7 de Unix en 1979**. El comando chroot podría usarse para aislar un proceso del sistema de archivos raíz y básicamente "ocultar" los archivos del proceso y simular una nuevo directorio raíz. El entorno aislado es el llamado chroot jail, donde el proceso no puede acceder a los archivos, pero aún están presentes en el sistema.

C

Los directorios hroot se pueden crear en varios lugares del sistema de archivos.

Si bien chroot es una tecnología bastante antigua, todavía se utiliza en algunos proyectos de software populares. Las tecnologías de contenedores que tenemos hoy en día todavía incorporan este mismo concepto, pero en una versión modernizada y con muchas características adicionales.

P**ara aislar un proceso aún más de lo que puede hacer chroot, los kernels de Linux actuales proporcionan características como espacios de nombres y cgroups.**

Los espacios de nombres se utilizan para aislar varios recursos, por ejemplo la red. Se puede utilizar un espacio de nombres de red para proporcionar una abstracción completa de las interfaces de red y las tablas de enrutamiento. Esto permite que un proceso tenga su propia dirección IP. El kernel de Linux 5.6 proporciona actualmente 8 espacios de nombres:

pid: ID de proceso proporciona un proceso con su propio conjunto de ID de proceso.

net: la red permite que los procesos tengan su propia pila de red, incluida la dirección IP.

mnt: mount abstrae la vista del sistema de archivos y gestiona los puntos de montaje.

ipc: la comunicación entre procesos proporciona la separación de segmentos de memoria compartida con nombre.

usuario: proporciona al proceso su propio conjunto de ID de usuario e ID de grupo.

uts: el tiempo compartido de Unix permite que los procesos tengan su propio nombre de host y nombre de dominio.

cgroup: un espacio de nombres más nuevo que permite que un proceso tenga su propio conjunto de directorios raíz de cgroup.

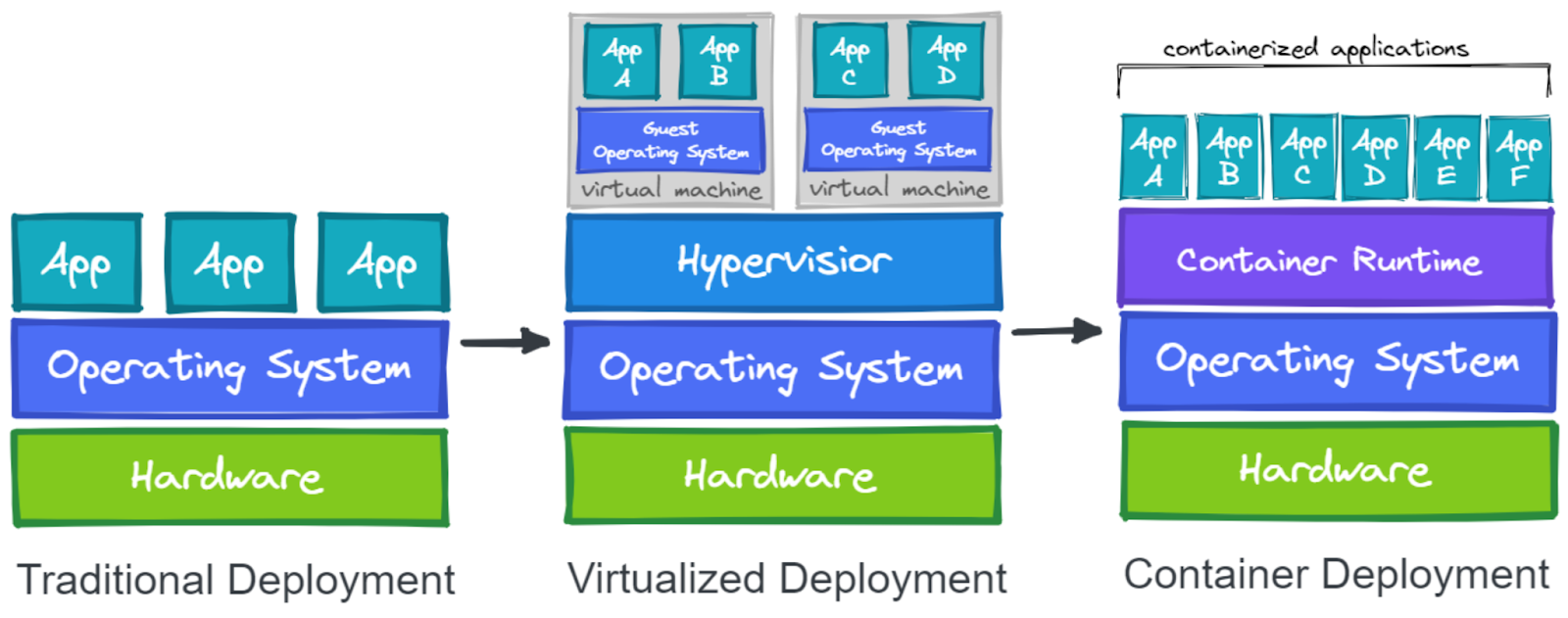
time: el espacio de nombres más nuevo se puede utilizar para virtualizar el reloj del sistema.

Los cgroups se utilizan para organizar procesos en grupos jerárquicos y asignarles recursos como memoria y CPU. Cuando desea limitar el contenedor de su aplicación a, digamos, 4 GB de memoria, se utilizan cgroups bajo el capó para garantizar estos límites.

Lanzado en 2013, Docker se convirtió en sinónimo de creación y ejecución de contenedores. Aunque Docker no inventó las tecnologías que se utilizan para ejecutar contenedores, unió las tecnologías existentes de manera inteligente para hacer que los contenedores sean más fáciles de usar y accesibles.

A primera vista, los contenedores parecen muy similares a las máquinas virtuales, pero es fundamental comprender que son muy diferentes. Mientras que las máquinas virtuales emulan una máquina completa, incluido el sistema operativo y un kernel, los contenedores comparten el kernel de la máquina host y, como se explicó, son solo procesos aislados.

Las máquinas virtuales conllevan algunos gastos generales, ya sea tiempo de arranque, tamaño o uso de recursos para ejecutar el sistema operativo. Los contenedores, por otro lado, son literalmente procesos, como el navegador que puedes iniciar en tu máquina, por lo tanto, se inician mucho más rápido y ocupan menos espacio.



Xx

I**mplementación tradicional versus implementación virtualizada versus implementación de contenedores**

En muchos casos, no se trata de si está utilizando contenedores o máquinas virtuales, sino que está utilizando ambas tecnologías para beneficiarse de la eficiencia que tienen los contenedores pero aún así utilizar las ventajas de seguridad que aporta el mayor aislamiento de las máquinas virtuales.

Ejecución de contenedores

Para ejecutar contenedores estándar de la industria, no es necesario utilizar Docker; En su lugar, puede seguir el estándar de especificaciones de tiempo de ejecución de OCI. Open Container Initiative también mantiene una implementación de referencia de tiempo de ejecución de contenedor llamada runC. Este tiempo de ejecución de bajo nivel se utiliza en una variedad de herramientas para iniciar contenedores, incluido el propio Docker.

Si es desarrollador y sabe sobre programación orientada a objetos, puede imaginar la relación entre la imagen del contenedor y el contenedor en ejecución como una clase y la creación de instancias de esa clase.

Con Docker instalado, puedes iniciar contenedores como este:

docker run nginx

La especificación de tiempo de ejecución va de la mano con la especificación de imagen, que cubriremos en el siguiente capítulo, ya que describe cómo descomprimir una imagen de contenedor y luego administrar el ciclo de vida completo del contenedor, desde la creación del entorno del contenedor hasta el inicio del proceso. , deteniéndolo y eliminándolo.

Para su máquina local, hay muchas alternativas para elegir, algunas de las cuales son solo para crear imágenes como buildah o kaniko, mientras que otras se presentan como alternativas completas a Docker, como lo hace podman.

Podman proporciona una API similar a Docker y puede usarse como reemplazo directo. Además, viene con algunas características adicionales como ejecutar contenedores sin privilegios de root y el uso del concepto Pod que descubriremos más adelante.

<https://trainingportal.linuxfoundation.org/learn/course/kubernetes-and-cloud-native-essentials-lfs250/container-orchestration/container-orchestration?page=4>

**Imágenes de Contenedor Construccion**

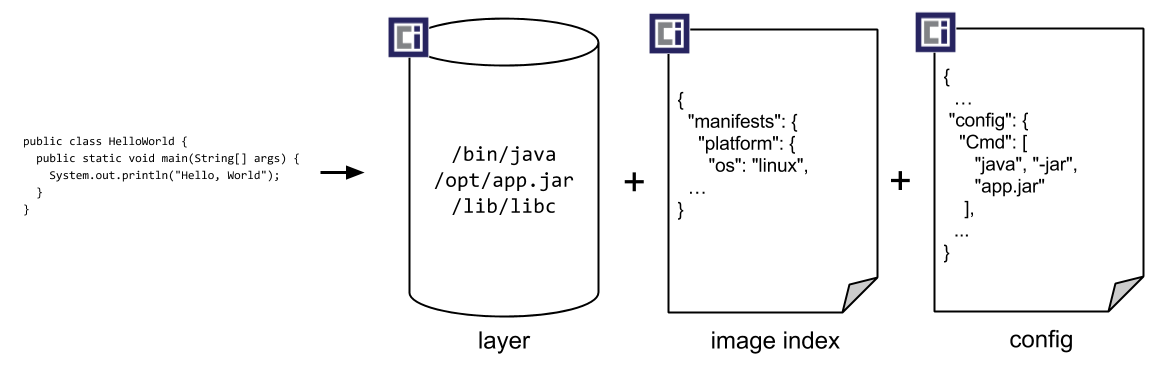
¿Por qué un contenedor se llama contenedor en primer lugar? La metáfora utilizada aquí está dirigida al uso de contenedores de transporte que están estandarizados según la norma ISO 668. El formato estándar de un contenedor de transporte hace que sea fácil de apilar en un barco portacontenedores, fácil de descargar con una grúa y en un camión, sin importar lo que suceda. adentro. Verá que muchos términos en el mundo nativo de la nube y los contenedores siguen este tema náutico.

Docker reutilizó todos los componentes para aislar procesos como espacios de nombres y cgroups, pero una pieza crucial del rompecabezas que ayudó a los contenedores a lograr su avance es la introducción de imágenes de contenedores.

Las imágenes de contenedores son lo que hace que los contenedores sean portátiles y fáciles de reutilizar en una variedad de sistemas. Docker describe una imagen de contenedor de la siguiente manera:

"Una imagen de contenedor Docker es un paquete de software ejecutable, liviano e independiente que incluye todo lo necesario para ejecutar una aplicación: código, tiempo de ejecución, herramientas del sistema, bibliotecas del sistema y configuraciones".

En 2015, el formato de imagen popularizado por Docker fue donado a la recién fundada Open Container Initiative y también se conoce como especificación de imagen OCI que se puede encontrar en GitHub. Las imágenes constan de un paquete de sistema de archivos y metadatos.

X

x

Container Images

Images can be built by reading the instructions from a buildfile called Dockerfile. The instructions are almost the same as one would use to install an application on a server. Here is an example of a Dockerfile that containerizes a Python script:

# Every container image starts with a base image.  
# This could be your favorite linux distribution  
FROM ubuntu:20.04

# Run commands to add software and libraries to your image  
# Here we install python3 and the pip package manager  
RUN apt-get update && \  
    apt-get -y install python3 python3-pip

# The copy command can be used to copy your code to the image  
# Here we copy a script called "my-app.py" to the containers filesystem  
COPY my-app.py /app/

# Defines the workdir in which the application runs  
# From this point on everything will be executed in /app  
WORKDIR /app

# The process that should be started when the container runs  
# In this case we start our python app "my-app.py"  
CMD ["python3","my-app.py"]

**Si ha instalado Docker en su máquina, puede crear la imagen con el siguiente comando:**

**docker build -t my-python-image -f Dockerfile**

Con los parámetros -t my-python-image puedes especificar una etiqueta de nombre para tu imagen, y con -f Dockerfile especificas dónde se puede encontrar tu Dockerfile. Esto brinda a los desarrolladores la capacidad de administrar todas las dependencias de su aplicación y empaquetarla lista para ejecutarse en lugar de dejar esa tarea a otra persona o equipo.

Para distribuir estas imágenes, puede utilizar un registro de contenedor. Este no es más que un servidor web donde puedes subir y descargar imágenes. Convenientemente, Docker tiene integrados los comandos push y pull:

docker push my-registry.com/my-python-image  
docker pull my-registry.com/my-python-image

Como construir container

<https://trainingportal.linuxfoundation.org/learn/course/kubernetes-and-cloud-native-essentials-lfs250/container-orchestration/container-orchestration?page=6>

**Seguridad**

Es importante comprender que los contenedores tienen requisitos de seguridad diferentes a los de las máquinas virtuales. Mucha gente confía en las propiedades de aislamiento de los contenedores, pero eso puede ser muy peligroso.

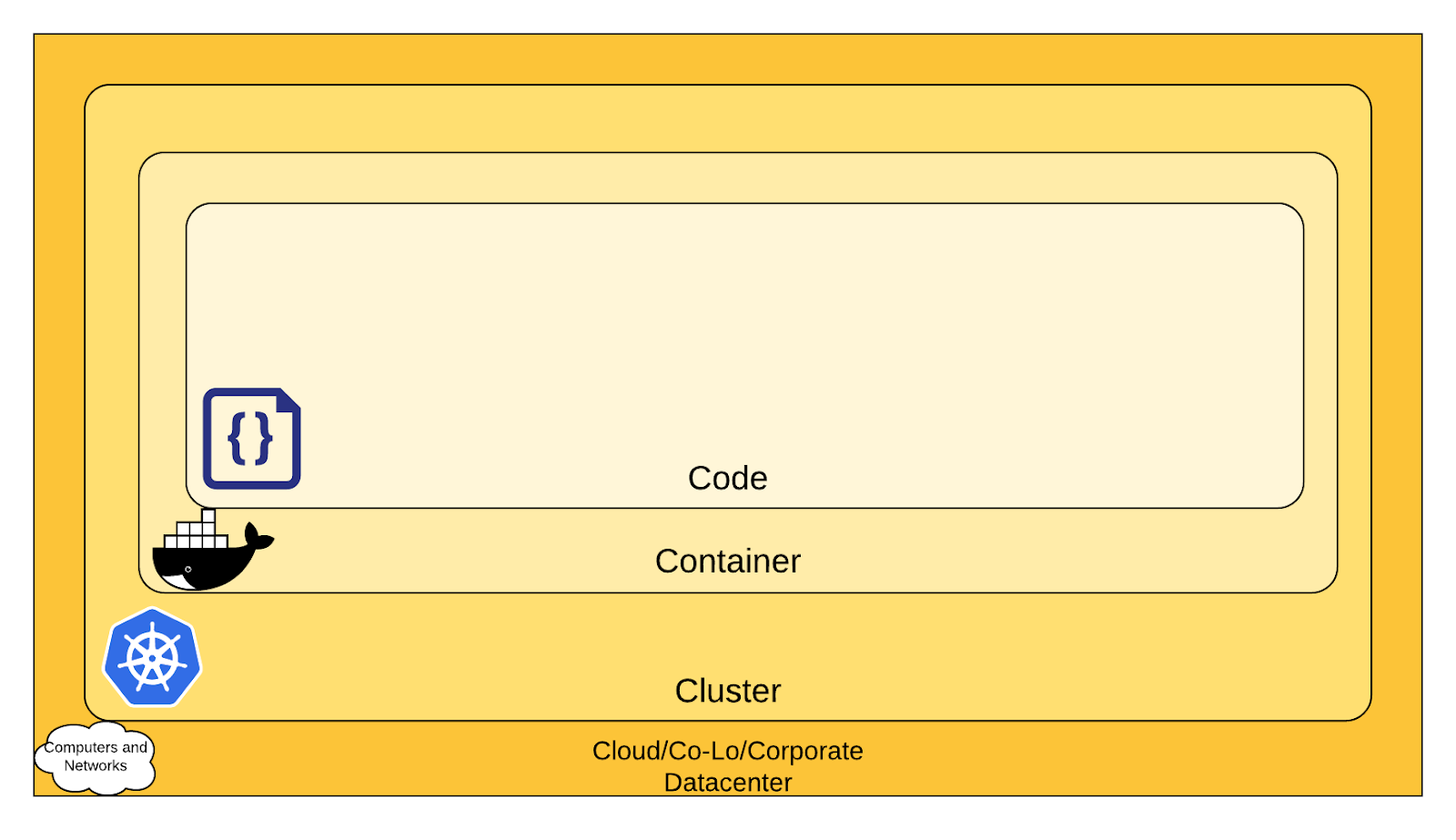
Cuando los contenedores se inician en una máquina, siempre comparten el mismo kernel, lo que luego se convierte en un riesgo para todo el sistema, si a los contenedores se les permite llamar a funciones del kernel como, por ejemplo, eliminar otros procesos o modificar la red del host mediante la creación de reglas de enrutamiento. Puede obtener más información sobre las capacidades del kernel en la documentación de Docker.

Uno de los mayores riesgos de seguridad, no sólo en el ámbito de los contenedores, es la ejecución de procesos con demasiados privilegios, especialmente al iniciar procesos como root o administrador. Desafortunadamente, este es un problema que se ignoró mucho en el pasado y existen muchos contenedores que se ejecutan como usuarios root.

Una superficie de ataque bastante nueva que se introdujo con los contenedores es el uso de imágenes públicas. Dos de los registros de imágenes públicas más populares son Docker Hub y Quay y, si bien es fantástico que proporcionen imágenes accesibles públicamente, debes asegurarte de que estas imágenes no hayan sido modificadas para incluir software malicioso.

Sysdig tiene un excelente artículo de blog sobre cómo evitar muchos problemas de seguridad y crear imágenes de contenedores seguras.

La seguridad en general no es algo que pueda lograrse únicamente en la capa contenedora. Es un proceso continuo que necesita adaptarse todo el tiempo. Las 4C de la seguridad nativa de la nube pueden dar una idea aproximada de qué capas deben protegerse si utiliza contenedores. Asegúrese de cubrir cada capa, ya que protege eficazmente la capa interna. La documentación de Kubernetes es un buen punto de partida para comprender las capas.

X

Fundamentos de la orquestación de contenedores

Es bastante fácil ejecutar algunos contenedores en su máquina local o en un único servidor, pero la forma en que se utilizan presenta nuevos desafíos con respecto a las operaciones de los contenedores. La alta eficiencia del concepto ha dado como resultado que las aplicaciones y los servicios sean cada vez más pequeños y descubrirá que las aplicaciones modernas pueden consistir en muchos contenedores.

Tener muchos contenedores pequeños que estén débilmente acoplados, aislados e independientes es la base de las llamadas arquitecturas de microservicios. Estos pequeños contenedores son pequeñas partes autónomas de la lógica empresarial que forman parte de una aplicación más grande.

Si tiene que gestionar e implementar grandes cantidades de contenedores, rápidamente llega al punto en el que necesita un sistema que ayude con la gestión de estos contenedores. Los problemas a resolver pueden incluir:

Proporcionar recursos informáticos como máquinas virtuales donde se puedan ejecutar contenedores

Programe contenedores en servidores de forma eficiente

Asigne recursos como CPU y memoria a contenedores

Gestionar la disponibilidad de contenedores y sustituirlos si fallan

Escale los contenedores si aumenta la carga.

Proporcionar redes para conectar contenedores entre sí.

Proporcione almacenamiento si los contenedores necesitan conservar datos

Los sistemas de orquestación de contenedores proporcionan una manera de construir un clúster de múltiples servidores y alojar los contenedores encima. La mayoría de los sistemas de orquestación de contenedores constan de dos partes: un plano de control que es responsable de la gestión de los contenedores y los nodos trabajadores que realmente alojan los contenedores.

A lo largo de los años, ha habido varios sistemas que se pueden utilizar para la orquestación, pero la mayoría de ellos ya no son de gran importancia hoy en día y la industria ha elegido Kubernetes como el sistema estándar para orquestar contenedores.

**Redes**

La arquitectura de microservicio depende en gran medida de la comunicación de la red. A diferencia de las aplicaciones monolíticas, un microservicio implementa una interfaz a la que se puede llamar para realizar una solicitud. Por ejemplo, podría tener un servicio que responda con una lista de productos en una aplicación de comercio electrónico.

Los espacios de nombres de red (Network namespaces )permiten que cada contenedor tenga su propia dirección IP única, por lo que varias aplicaciones pueden abrir el mismo puerto de red; por ejemplo, podría tener varios servidores web en contenedores que abran el puerto 8080.

**Para que la aplicación sea accesible desde fuera del sistema host, los contenedores tienen la capacidad de asignar un puerto desde el contenedor a un puerto desde el sistema host.**

Para permitir la comunicación entre contenedores entre hosts, podemos utilizar una red superpuesta que los coloca en una red virtual que se extiende por todos los sistemas host.

Esto hace que sea muy fácil para los contenedores comunicarse entre sí, mientras que los administradores del sistema no tienen que configurar redes y enrutamientos complejos entre hosts y contenedores.

La mayoría de las redes superpuestas también se encargan de la gestión de direcciones IP, lo que supondría mucho trabajo si se implementara manualmente. En este caso, la red superpuesta gestiona qué contenedor obtiene qué dirección IP y cómo debe fluir el tráfico para acceder a los contenedores individuales.

La mayoría de las implementaciones modernas de redes de contenedores se basan en la(Container Network Interface ) **Interfaz de red de contenedores (CNI)**. CNI es un estándar que se puede utilizar para escribir o configurar complementos de red y hace que sea muy fácil intercambiar diferentes complementos en varias plataformas de orquestación de contenedores.

**Descubrimiento de servicios y DNS**

Durante mucho tiempo, la gestión de servidores en los centros de datos tradicionales era manejable. Muchos administradores de sistemas incluso recordaban todas las direcciones IP de los sistemas importantes con los que tenían que trabajar. Grandes listas de servidores, sus nombres de host, direcciones IP y propósitos (todos mantenidos manualmente) eran una tarea diaria.

En las plataformas de orquestación de contenedores las cosas son mucho más complicadas:

Cientos o miles de contenedores con direcciones IP individuales

Los contenedores se implementan en diferentes hosts, diferentes centros de datos o incluso geolocalizaciones.

Los contenedores o servicios necesitan DNS para comunicarse. Usar direcciones IP es casi imposible

La información sobre los contenedores debe eliminarse del sistema cuando se eliminan.

La solución al problema nuevamente es la automatización. En lugar de tener una lista de servidores (o en este caso contenedores) mantenida manualmente, toda la información se coloca en un Registro de servicios **(Service Registry)**. Encontrar otros servicios en la red y solicitar información sobre ellos se llama Service Discovery.

**Area of Concern for Workload Security**

**DNS**: Los servidores DNS modernos que tienen una API de servicio se pueden utilizar para registrar nuevos servicios a medida que se crean. Este enfoque es bastante sencillo, ya que la mayoría de las organizaciones ya cuentan con servidores DNS con las capacidades adecuadas.

**Key-Value-Store (Tienda de valores clave)**

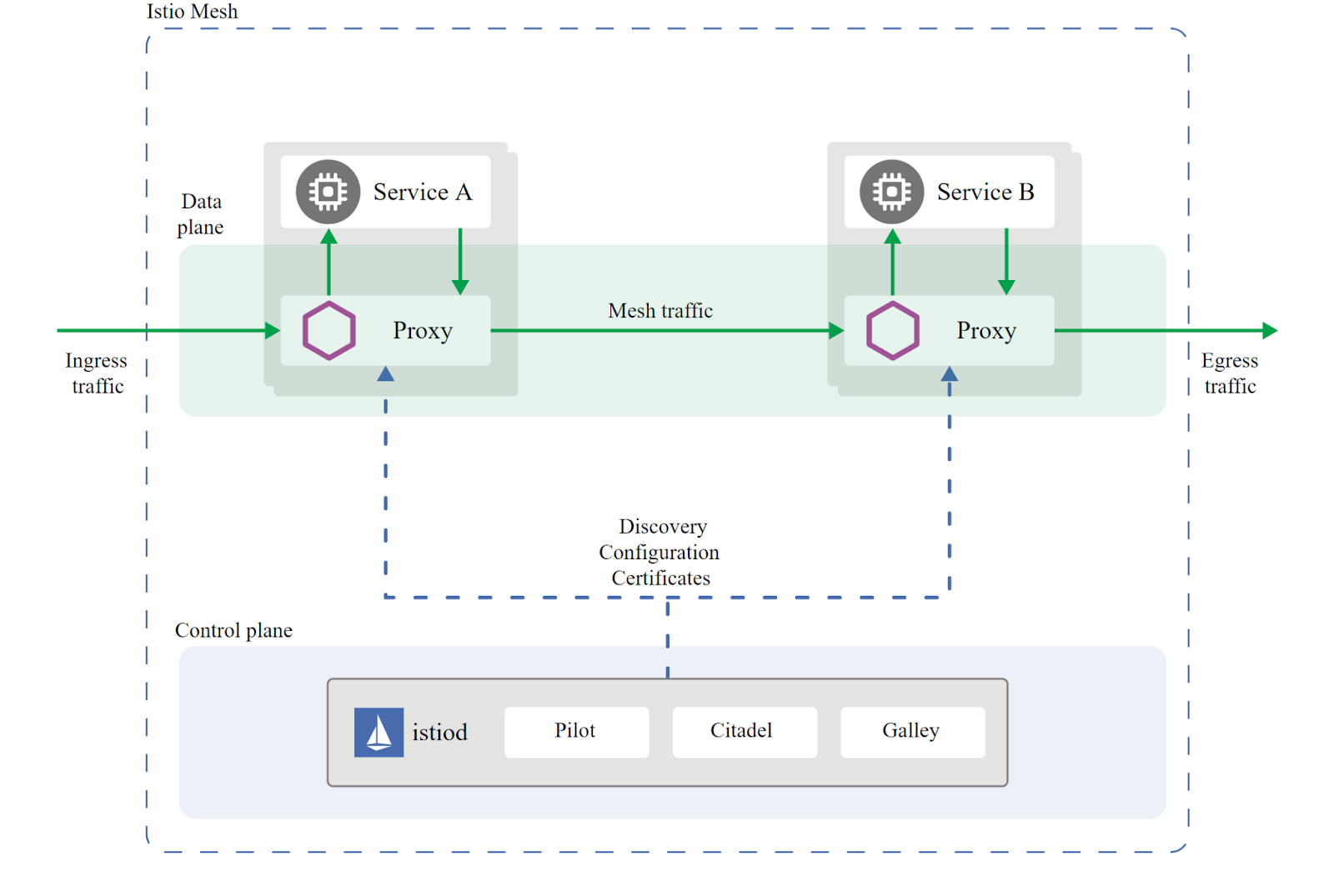
Usar un almacén de datos fuertemente consistente, especialmente para almacenar información sobre servicios. Muchos sistemas pueden funcionar con alta disponibilidad con sólidos mecanismos de conmutación por error. Las opciones populares, especialmente para la agrupación en clústeres, son etcd, Consul o Apache Zookeeper.

**Service Mesh (Malla de servicio)**

Debido a que la red es una parte tan crucial de los microservicios y contenedores, la red puede volverse muy compleja y opaca para los desarrolladores y administradores. Además de eso, se desean muchas funciones como monitoreo, control de acceso o cifrado del tráfico de red cuando los contenedores se comunican entre sí.

En lugar de implementar toda esta funcionalidad en su aplicación, puede simplemente iniciar un segundo contenedor que tenga esta funcionalidad implementada. El software que puede utilizar para administrar el tráfico de la red se llama proxy. Esta es una aplicación de servidor que se encuentra entre un cliente y un servidor y puede modificar o filtrar el tráfico de la red antes de que llegue al servidor. Los representantes populares son **nginx, haproxy or envoy**.

Llevando esta idea un paso más allá, una malla de servicios agrega un servidor proxy a cada contenedor que tenga en su arquitectura.

X

Arquitectura de Istio, recuperado de istio.io

Ahora puede utilizar los servidores proxy para manejar la comunicación de red entre sus servicios.

Tomemos el cifrado como ejemplo. Si dos o más aplicaciones cifraran su tráfico cuando se comunican entre sí, sería necesario agregar bibliotecas y configurar y administrar certificados digitales que demuestren la identidad de las aplicaciones involucradas. Esto puede suponer mucho trabajo y también puede ser propenso a errores si no se hace con especial cuidado.

Cuando se utiliza una malla de servicios, las aplicaciones no se comunican entre sí directamente, sino que el tráfico se enruta a través de servidores proxy. Las mallas de servicios más populares actualmente son istio y linkerd. Si bien tienen diferencias en la implementación, la arquitectura es la misma.

Los proxies en una malla de servicios forman el plano de datos. Aquí es donde se implementan las reglas de networking y dan forma al flujo de tráfico.

Estas reglas se gestionan de forma centralizada en el plano de control de la malla de servicios. Aquí es donde define cómo fluye el tráfico del servicio A al servicio B y qué configuración se debe aplicar a los servidores proxy.

Entonces, en lugar de escribir código e instalar bibliotecas, simplemente escribe un archivo de configuración donde le dice a la malla de servicios que el servicio A y el servicio B siempre deben comunicarse encriptados.

Luego, la configuración se carga en el plano de control y se distribuye al plano de datos para aplicar la nueva regla.

Durante mucho tiempo, el término "**service mesh**" sólo describía una idea básica de cómo se podía gestionar el tráfico en las plataformas de contenedores mediante proxies. El proyecto Service Mesh Interface (SMI) tiene como objetivo definir una especificación sobre cómo se puede implementar una red de servicios de varios proveedores.

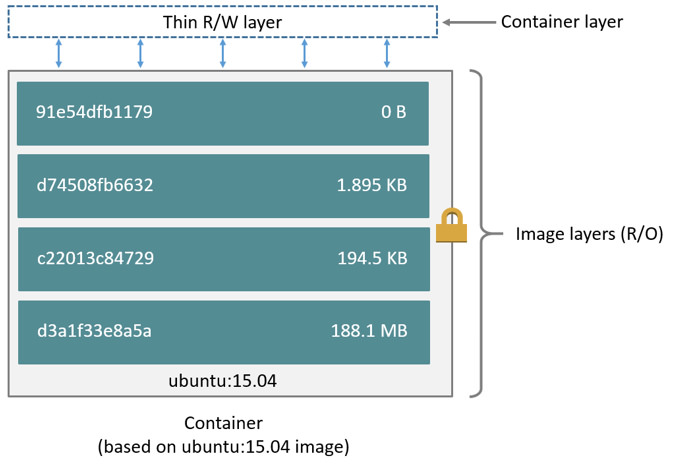
Con un fuerte enfoque en Kubernetes, su objetivo es estandarizar la experiencia del usuario final para las mallas de servicios, así como un estándar para los proveedores que quieran integrarse con Kubernetes. Puede encontrar la especificación actual en GitHub.

# **Storage**

Almacenamiento

Desde la perspectiva del almacenamiento, los contenedores tienen un defecto importante: son efímeros. Para comprender qué significa exactamente, debemos comprender qué sucede cuando un contenedor se inicia a partir de una imagen de contenedor.

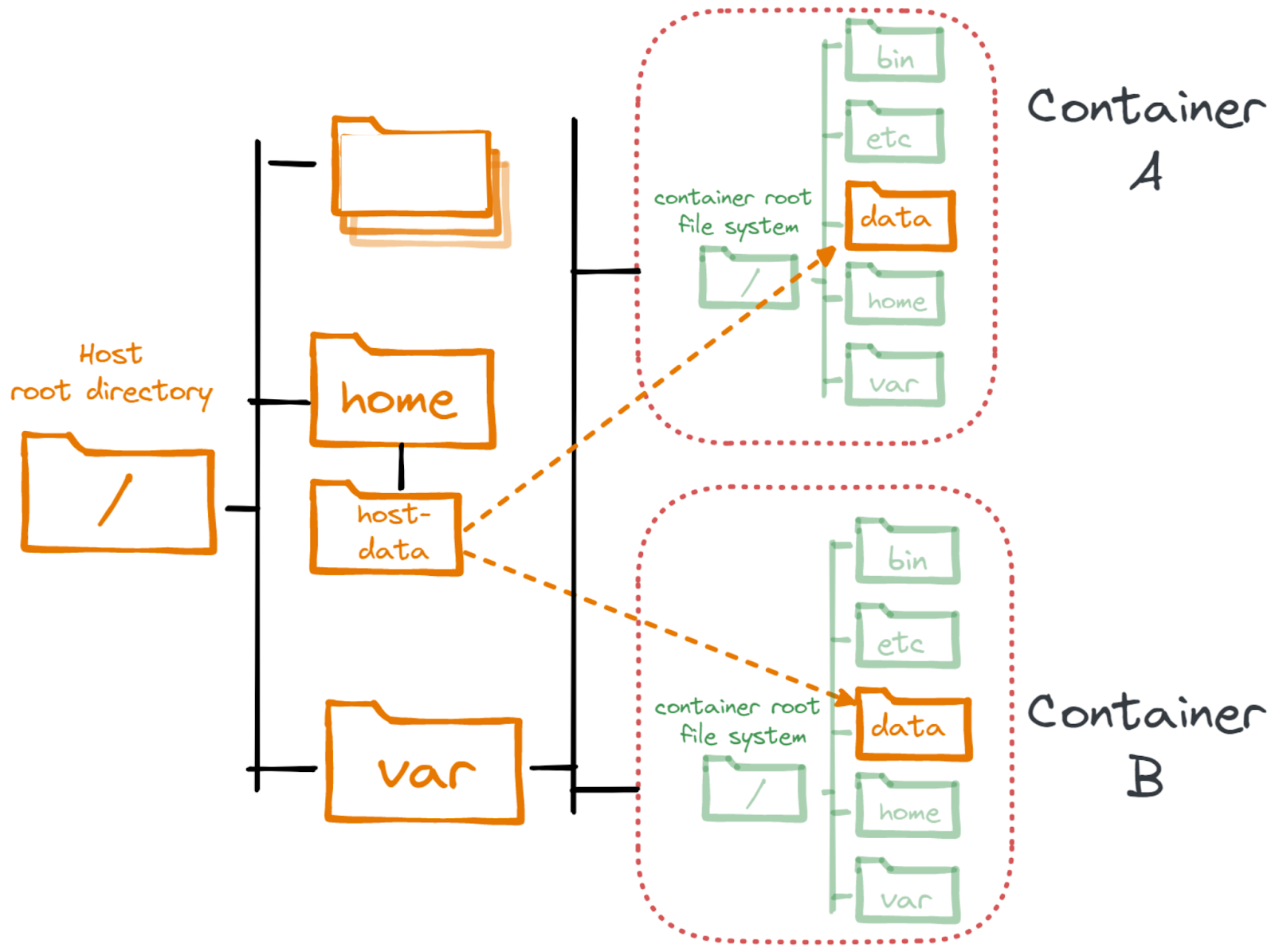
En términos generales, las imágenes de contenedor son de solo lectura y constan de diferentes capas que incluyen todo lo que agregó durante la fase de construcción. Eso garantiza que cada vez que inicia un contenedor a partir de una imagen obtenga el mismo comportamiento y funcionalidad. Como probablemente puedas imaginar, muchas aplicaciones necesitan escribir archivos. Para permitir la escritura de archivos, se coloca una capa de lectura y escritura encima de la imagen del contenedor cuando inicia un contenedor desde una imagen.

X

**Capas de contenedor, extraído de la documentación de Docker**

El problema aquí es que esta capa de lectura y escritura se pierde cuando se detiene o elimina el contenedor. Al igual que la memoria de tu computadora se borra cuando la apagas. Para conservar los datos, debe escribirlos en su disco.

Si un contenedor necesita conservar datos en un host, se puede utilizar un volumen para lograrlo. El concepto y la tecnología para esto es bastante simple: en lugar de aislar todo el sistema de archivos de un proceso, los directorios que residen en el host pasan al sistema de archivos contenedor. Si crees que esto debilita el aislamiento del contenedor, estás en lo cierto. Cuando se utilizan volúmenes de contenedor, efectivamente se otorga acceso al sistema de archivos del host.

**Los datos se comparten entre dos contenedores en el mismo host**

Cuando se organizan muchos contenedores, es posible que conservar los datos en el host donde se inició el contenedor no sea el único desafío. A menudo, es necesario acceder a los datos mediante varios contenedores que se inician en diferentes sistemas host o, cuando un contenedor se inicia en un host diferente, aún debe tener acceso a su volumen.

Los sistemas de orquestación de contenedores como Kubernetes pueden ayudar a mitigar estos problemas, pero siempre requieren un sistema de almacenamiento robusto conectado a los servidores host.

El almacenamiento se proporciona a través de un sistema de almacenamiento central. Los contenedores en el servidor A y el servidor B pueden compartir un volumen para leer y escribir datos

Para mantenerse al día con el crecimiento ininterrumpido de diversas implementaciones de almacenamiento, nuevamente, la solución fue implementar un estándar. **Container Storage Interface La interfaz de almacenamiento de contenedores (CSI) s**urgió para ofrecer una interfaz uniforme que permite conectar diferentes sistemas de almacenamiento, sin importar si se trata de almacenamiento en la nube o local.

# Additional Resources

Learn more about...

## The History of Containers

* [A Brief History of Containers: From the 1970s Till Now](https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016), by Rani Osnat (2020)
* [It's Here: Docker 1.0](https://web.archive.org/web/20160426102954/https://blog.docker.com/2014/06/its-here-docker-1-0/), by Julien Barbier (2014)

## Chroot

* [chroot](https://wiki.ubuntuusers.de/chroot/)

## Container Performance

* [Container Performance Analysis at DockerCon 2017](https://www.brendangregg.com/blog/2017-05-15/container-performance-analysis-dockercon-2017.html), by Brendan Gregg

## Best Practices on How to Build Container Images

* [Top 20 Dockerfile Best Practices](https://sysdig.com/blog/dockerfile-best-practices/), by Álvaro Iradier (2021)
* [3 simple tricks for smaller Docker images](https://learnk8s.io/blog/smaller-docker-images), by Daniele Polencic (2019)
* [Best practices for building containers](https://cloud.google.com/architecture/best-practices-for-building-containers)

## Alternatives to Classic Dockerfile Container Building

* [Buildpacks vs Jib vs Dockerfile: Comparing containerization methods](https://trainingportal.linuxfoundation.org/learn/course/kubernetes-and-cloud-native-essentials-lfs250/container-orchestration/Ál), by James Ward (2020)

## Service Discovery

* [Service Discovery in a Microservices Architecture](https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/), by Chris Richardson (2015)

## Container Networking

* [Kubernetes Networking Part 1: Networking Essentials](https://www.inovex.de/de/blog/kubernetes-networking-part-1-en/), By Simon Kurth (2021)
* [Life of a Packet (I)](https://www.youtube.com/watch?v=0Omvgd7Hg1I), by Michael Rubin (2017)
* [Computer Networking Introduction - Ethernet and IP (Heavily Illustrated)](https://iximiuz.com/en/posts/computer-networking-101/), by Ivan Velichko (2021)

## Container Storage

* [Managing Persistence for Docker Containers](https://thenewstack.io/methods-dealing-container-storage/), by Janakiram MSV (2016)

## Container and Kubernetes Security

* [Secure containerized environments with updated thread matrix for Kubernetes](https://www.microsoft.com/security/blog/2021/03/23/secure-containerized-environments-with-updated-threat-matrix-for-kubernetes/), by Yossi Weizman (2021)

## Docker Container Playground

* [Play with Docker](https://labs.play-with-docker.com/)

**KUBERNETES FUNDAMENTALS**

Descripción general del capítulo

Kubernetes es una plataforma de orquestación de contenedores de código abierto muy popular que se puede utilizar para automatizar la implementación, el escalado y la gestión de cargas de trabajo en contenedores. En este capítulo, descubriremos la arquitectura básica de un clúster de Kubernetes y sus componentes. Para aprender aún más conceptos básicos de Kubernetes, puede realizar el curso gratuito Introducción a Kubernetes (LFS158x) de la Fundación Linux en edX. https://training.linuxfoundation.org/training/introduction-to-kubernetes/?\_gl=1\*17svjuq\*\_ga\*OTM0ODg5MjE3LjE2ODU1OTU4OTQ.\*\_ga\_EMX7DDZMX4\*MTY5NDQzODE5MS4xNi4xLjE2OTQ0NDQwOTIuNTUuMC4w

Originalmente diseñado y desarrollado por Google, Kubernetes se volvió de código abierto en 2014 y, junto con la versión v1.0, Kubernetes fue donado a la recién formada Cloud Native Computing Foundation como el primer proyecto. Muchas tecnologías nativas de la nube evolucionan en torno a Kubernetes, ya sean herramientas de bajo nivel como tiempos de ejecución de contenedores, monitoreo o herramientas de entrega de aplicaciones.

Los profesionales con habilidades en Kubernetes, ya sea administración o desarrollo, son muy buscados ya que Kubernetes se ha convertido en la pieza central de muchas plataformas en la nube modernas.

Objetivos de aprendizaje

Al final de este capítulo, debería poder:

Discutir la arquitectura básica de Kubernetes.

Explicar los diferentes componentes del plano de control y los nodos trabajadores.

Comprenda cómo comenzar con la configuración de Kubernetes

Analice cómo Kubernetes ejecuta contenedores

Discutir los conceptos de programación de Kubernetes.

Arquitectura de Kubernetes

Kubernetes se utiliza a menudo como un clúster, lo que significa que se distribuye en varios servidores que trabajan en diferentes tareas y distribuyen la carga de un sistema. Esta es una decisión de diseño inicial basada en los requisitos de Google, donde cada semana se ponen en marcha miles de millones de contenedores. Dada la alta escalabilidad horizontal de Kubernetes, es posible tener clústeres con miles de nodos de servidores en múltiples centros de datos y regiones.

Desde una perspectiva de alto nivel, los clústeres de Kubernetes constan de dos tipos de nodos de servidor diferentes que conforman un clúster:

Control plane node(s) **Nodo(s) del plano de control**

Estos son los cerebros de la operación. Los nodos del plano de control contienen varios componentes que administran el clúster y controlan diversas tareas como la implementación, la programación y la autorreparación de cargas de trabajo en contenedores.

Worker nodes **Nodos trabajadores**

Los nodos trabajadores son donde se ejecutan las aplicaciones en su clúster. Este es el único trabajo de los nodos trabajadores y no tienen implementada ninguna lógica adicional. Su comportamiento, como si tuvieran que iniciar un contenedor, está completamente controlado por el nodo del plano de control.

**Configuración de Kubernetes**

La configuración de un clúster de Kubernetes se puede lograr con muchos métodos diferentes. Crear un "clúster" de pruebas puede ser muy fácil con las herramientas adecuadas:

* [Minikube](https://minikube.sigs.k8s.io/docs/)
* [kind](https://kind.sigs.k8s.io/)
* [MicroK8s](https://microk8s.io/)

Si desea configurar un clúster de nivel de producción en su propio hardware o máquinas virtuales, puede elegir uno de los diversos instaladores:

* [kubeadm](https://kubernetes.io/docs/reference/setup-tools/kubeadm/)
* [kops](https://github.com/kubernetes/kops)
* [kubespray](https://github.com/kubernetes-sigs/kubespray)

Algunos proveedores comenzaron a empaquetar Kubernetes en una distribución e incluso ofrecen soporte comercial:

* [Rancher](https://rancher.com/)
* [k3s](https://k3s.io/)
* [OpenShift](https://www.redhat.com/en/technologies/cloud-computing/openshift)
* [VMWare Tanzu](https://tanzu.vmware.com/tanzu)

Las distribuciones a menudo eligen un enfoque obstinado y ofrecen herramientas adicionales mientras utilizan Kubernetes como pieza central de su marco.

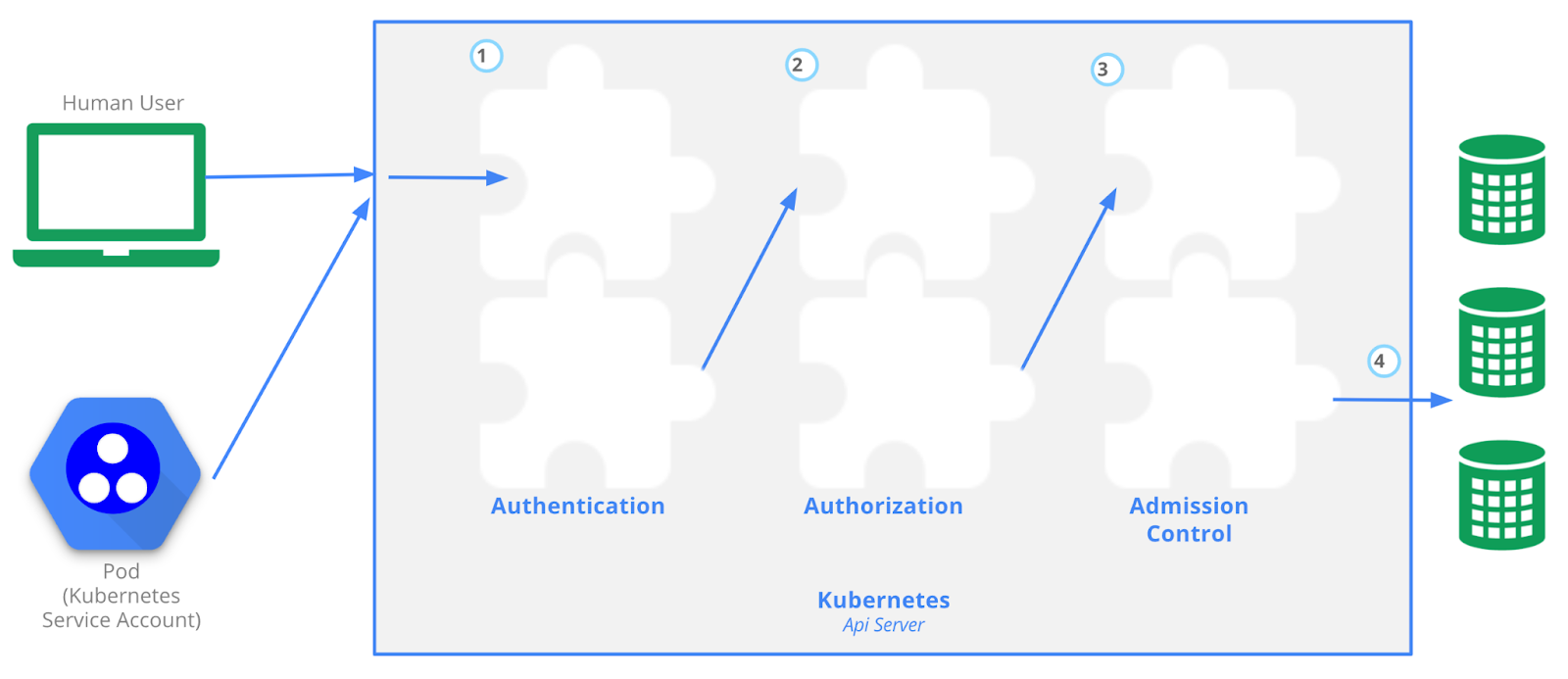
Si no desea instalarlo y administrarlo usted mismo, puede consumirlo desde un proveedor de nube:

* [Amazon (EKS)](https://aws.amazon.com/eks/)
* [Google (GKE)](https://cloud.google.com/kubernetes-engine)
* [Microsoft (AKS)](https://azure.microsoft.com/en-us/services/kubernetes-service)
* [DigitalOcean (DOKS)](https://www.digitalocean.com/products/kubernetes/)

Puede aprender cómo configurar su propio clúster de Kubernetes con Minikube en este

**API de Kubernetes**

La API de Kubernetes es el componente más importante de un clúster de Kubernetes. Sin él, la comunicación con el clúster no es posible, cada usuario y cada componente del clúster necesita el servidor API.

X

x

Descripción general del control de acceso, obtenido de la documentación de Kubernetes

Antes de que Kubernetes procese una solicitud, debe pasar por tres etapas:

Autenticación

El solicitante debe presentar un medio de identidad para autenticarse en la API. Comúnmente se realiza con un certificado firmado digitalmente (X.509) o con un sistema de gestión de identidad externo. Los usuarios de Kubernetes siempre se administran externamente. Las cuentas de servicio se pueden utilizar para autenticar a los usuarios técnicos.

**Autorización**

Se decide qué se le permite hacer al solicitante. En Kubernetes esto se puede hacer con el control de acceso basado en roles (RBAC).

**Control de admisión**

En el último paso, se pueden utilizar controladores de admisión para modificar o validar la solicitud. Por ejemplo, si un usuario intenta utilizar una imagen de contenedor de un registro no confiable, un controlador de admisión podría bloquear esta solicitud. Se pueden utilizar herramientas como Open Policy Agent para gestionar el control de admisión de forma externa.

Como muchas otras API, la API de Kubernetes se implementa como una interfaz RESTful que se expone a través de HTTPS. A través de la API, un usuario o servicio puede crear, modificar, eliminar o recuperar recursos que residen en Kubernetes.

**Ejecutar contenedores en Kubernetes**

¿En qué se diferencia la ejecución de un contenedor en su máquina local de la ejecución de contenedores en Kubernetes? En Kubernetes, en lugar de iniciar contenedores directamente, define Pods como la unidad informática más pequeña y Kubernetes la traduce en un contenedor en ejecución. Aprenderemos más sobre Pods más adelante, por ahora imagínelo como una envoltura alrededor de un contenedor.

Cuando creas un objeto Pod en Kubernetes, varios componentes participan en ese proceso, hasta que obtienes contenedores ejecutando un nodo.

Aquí hay un ejemplo usando contenedor:

c

Ejecución de contenedores en Kubernetes

En un esfuerzo por permitir el uso de otros tiempos de ejecución de contenedores además de Docker, Kubernetes introdujo la Container Runtime Interface (CRI) en 2016.

Haga clic para expandir cada cuadro y obtener más información sobre los tiempos de ejecución de contenedores disponibles con CRI.

**Containerd e**s una implementación ligera y eficaz para ejecutar contenedores. Posiblemente el tiempo de ejecución de contenedores más popular en este momento. Lo utilizan los principales proveedores de nube para los productos Kubernetes como servicio.

**CRI-O** fue creado por Red Hat y con una base de código similar estrechamente relacionada con podman y buildah.

**Docker**

El estándar durante mucho tiempo, pero nunca se creó realmente para la orquestación de contenedores. El uso de Docker como tiempo de ejecución para Kubernetes ha quedado obsoleto y eliminado en Kubernetes 1.24. Kubernetes tiene un excelente artículo de blog que responde a todas las preguntas sobre el tema.

La idea de Containerd y CRI-O era muy simple: proporcionar un tiempo de ejecución que solo contenga lo absolutamente esencial para ejecutar contenedores. Sin embargo, tienen características adicionales, como la capacidad de integrarse con herramientas de espacio aislado en tiempo de ejecución de contenedores. Estas herramientas intentan resolver el problema de seguridad que surge al compartir el kernel entre múltiples contenedores. Las herramientas más comunes en este momento son:

[gvisor](https://github.com/google/gvisor)

Creado por Google, proporciona un núcleo de aplicación que se encuentra entre el proceso en contenedor y el núcleo del host.

[Kata Containers](https://katacontainers.io/)

Un tiempo de ejecución seguro que proporciona una máquina virtual liviana, pero se comporta como un contenedor.

Las redes de Kubernetes pueden ser muy complicadas y difíciles de entender. Muchos de estos conceptos no están relacionados con Kubernetes y se trataron en el capítulo Orquestación de contenedores. Nuevamente, tenemos que lidiar con el problema de que muchos contenedores necesitan comunicarse a través de muchos nodos. Kubernetes distingue entre cuatro problemas de red diferentes que deben resolverse:

Container-to-Container communications  
This can be solved by the Pod concept as we'll learn later.

1. Pod-to-Pod communications  
   This can be solved with an overlay network.
2. Pod-to-Service communications  
   Se implementa mediante kube-proxy y filtro de paquetes en el nodo.
3. External-to-Service communications  
   Se implementa mediante kube-proxy y filtro de paquetes en el nodo.

Hay diferentes formas de implementar redes en Kubernetes, pero también tres requisitos importantes:

* Todos los pods pueden comunicarse entre sí a través de nodos.
* Todos los nodos pueden comunicarse con todos los pods.
* Sin traducción de direcciones de red (NAT).

Para implementar redes, puede elegir entre una variedad de proveedores de redes como:

* [Project Calico](https://www.tigera.io/project-calico/)
* [Weave](https://www.weave.works/oss/net/)
* [Cilium](https://cilium.io/)

En Kubernetes, cada Pod tiene su propia dirección IP, por lo que no es necesario realizar ninguna configuración manual. Además, la mayoría de las configuraciones de Kubernetes incluyen un complemento de servidor DNS llamado core-dns, que puede proporcionar descubrimiento de servicios y resolución de nombres dentro del clúster.

Por diseño, cada pod puede comunicarse con otros pods en el clúster de Kubernetes; sin embargo, si desea controlar el flujo de tráfico a nivel de dirección IP o puerto, debe utilizar políticas de red. Las políticas de red actúan como firewalls internos del clúster. Se pueden definir políticas de red para un conjunto de pods o espacios de nombres con la ayuda de un selector para especificar qué tráfico está permitido hacia y desde los pods que coinciden con el selector. Las políticas de red basadas en IP se definen con bloques de IP (rangos CIRD). Las políticas de red se implementan mediante el complemento de red. Para utilizar políticas de red, debe utilizar una solución de red que admita NetworkPolicy. Crear un recurso NetworkPolicy sin un controlador que lo implemente no tendrá ningún efecto.

**SchedulingPlanificación**

En su forma más básica, la programación es una subcategoría de la orquestación de contenedores y describe el proceso de elegir automáticamente el nodo (trabajador) correcto para ejecutar una carga de trabajo en contenedores. En el pasado, la programación era más una tarea manual en la que un administrador del sistema elegía el servidor adecuado para una aplicación realizando un seguimiento de los servidores disponibles, su capacidad y otras propiedades, como su ubicación.

En un clúster de Kubernetes, kube-scheduler es el componente que toma la decisión de programación, pero no es responsable de iniciar realmente la carga de trabajo. El proceso de programación en Kubernetes siempre comienza cuando se crea un nuevo objeto Pod. Recuerde que Kubernetes utiliza un enfoque declarativo, donde el Pod solo se describe primero, luego el programador selecciona un nodo donde el Kubelet y el tiempo de ejecución del contenedor realmente iniciarán el Pod.

Un error común acerca de Kubernetes es que tiene alguna forma de "inteligencia artificial" que analiza la carga de trabajo y mueve los Pods en función del consumo de recursos, el tipo de carga de trabajo y otros factores. La verdad es que un usuario debe proporcionar información sobre los requisitos de la aplicación, incluidas las solicitudes de CPU y memoria y las propiedades de un nodo. Por ejemplo, un usuario podría solicitar que su aplicación requiera dos núcleos de CPU, cuatro gigabytes de memoria y preferiblemente debería programarse en un nodo con discos rápidos.

El programador utilizará esa información para filtrar todos los nodos que cumplan con estos requisitos. Si varios nodos cumplen los requisitos por igual, Kubernetes programará el Pod en el nodo con la menor cantidad de Pods. Este también es el comportamiento predeterminado si un usuario no ha especificado ningún requisito adicional.

Es posible que no se pueda establecer el estado deseado, por ejemplo, porque los nodos trabajadores no tienen recursos suficientes para ejecutar su aplicación. En este caso, el planificador volverá a intentar encontrar un nodo apropiado hasta que se pueda establecer el estado.

# Additional Resources

Learn more about...

## Kubernetes history and the Borg Heritage

* [From Google to the world: The Kubernetes origin story](https://cloud.google.com/blog/products/containers-kubernetes/from-google-to-the-world-the-kubernetes-origin-story), by Craig McLuckie (2016)
* [Large-scale cluster management at Google with Borg](https://research.google/pubs/pub43438/), by Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, John Wilkes (2015)

## Kubernetes Architecture

* [Kubernetes Architecture explained | Kubernetes Tutorial 15](https://www.youtube.com/watch?v=umXEmn3cMWY)

## RBAC

* [Demystifying RBAC in Kubernetes](https://www.cncf.io/blog/2018/08/01/demystifying-rbac-in-kubernetes/), by Kaitlyn Barnard

## Container Runtime Interface

* [Introducing Container Runtime Interface (CRI) in Kubernetes](https://kubernetes.io/blog/2016/12/container-runtime-interface-cri-in-kubernetes/) (2016)

## Kubernetes networking and CNI

* [What is Kubernetes networking?](https://www.vmware.com/topics/glossary/content/kubernetes-networking)

## Internals of Kubernetes Scheduling

* [A Deep Dive into Kubernetes Scheduling](https://thenewstack.io/a-deep-dive-into-kubernetes-scheduling/), by Ron Sobol (2020)

## Kubernetes Security Tools

* [Popeye](https://github.com/derailed/popeye)
* [kubeaudit](https://github.com/Shopify/kubeaudit)
* [kube-bench](https://github.com/aquasecurity/kube-bench)

## Kubernetes Playground

* [Play with Kubernetes](https://labs.play-with-k8s.com/)

Kubernetes was originally designed and developed by \_\_\_\_\_\_\_\_\_.

Correct Answer

* A. Google

Your Answer:

Correct

What are the two types of Kubernetes nodes? Select all answers that apply.

**A. Control Plane Node**

B. Data Node

C. Security Node

**D. Worker Node**

The Kubernetes agent running on each worker node is called \_\_\_\_\_\_\_.

Correct Answer

* A. systemd
* B.**kubelet**

Your Answer:

Correct

# Question 4.4

What tool can be used to set up a Kubernetes cluster?

Correct Answer

* A. Minikube
* B. Kubeadm
* C. Rancher
* **D. All of the above**

**Your Answer:**

C

# Question 4.5

Sort the three stages a request needs to go through in the api-server.

Correct Answer

* A. Admission Control, Authorization, Authentication
* B. Authentication, Admission Control, Authorization
* **C. Authentication, Authorization, Admission Control**

**Your Answer:**

**Correct**

* D. Authorization, Authentication, Admission Control

Which container runtime is marked as deprecated by Kubernetes?

Correct Answer

* A. CRI-O
* B. containerd
* **C. Docker**

**Your Answer:**

**Correct**

* D. gvisor

# Question 4.7

Which of the following is NOT a requirement in Kubernetes networking?

Correct Answer

* A. All pods can communicate with each other across nodes
* B. All nodes can communicate with all pods
* **C. All pods communicate encrypted**

**Your Answer:**

**Correct**

* D. No Network Address Translation (NAT)

Which Kubernetes component is responsible for scheduling?

Correct Answer

* A. kube-apiserver
* **B. kube-scheduler**

**Your Answer:**

**Correct**

* C. kube-controller-manager
* D. kube-proxy

**\*\*\***

**capitulo 5**

Descripción general del capítulo En este capítulo, aprenderemos sobre los diferentes objetos de Kubernetes, su propósito y cómo interactuar con ellos. Después de configurar un clúster o utilizar uno ya existente, podemos comenzar a implementar alguna carga de trabajo. La unidad informática más pequeña de Kubernetes no es un contenedor, sino un objeto Pod. Dicho esto, un Pod no es la única abstracción que utilizamos para la carga de trabajo. Kubernetes tiene una variedad de objetos de carga de trabajo que controlan cómo se implementan, escalan y administran los Pods. Implementar la carga de trabajo no es la única tarea que debe realizar un desarrollador o administrador. Kubernetes tiene soluciones para algunos problemas inherentes a los contenedores y la orquestación, ya sea gestión de configuración, redes entre nodos, enrutamiento de tráfico externo, equilibrio de carga o escalado de pods.

[TRADUCIR ESTA PÁGINA](chrome-extension://aapbdbdomjkkjkaonfhkkikfgjllcleb/popup.html) [ABRIR EN GOOGLE TRADUCTOR](https://translate.google.com/?source=gtx_m" \l "en/es/Chapter Overview%0AIn this chapter%2C we will learn about the different Kubernetes objects%2C their purpose and how to interact with them.%0A%0AAfter setting up a cluster or using an already existing one%2C we can start deploying some workload. The smallest compute unit in Kubernetes is not a container%2C but a Pod object. That being%2C a Pod is not the only abstraction we use for workload. Kubernetes has a variety of workload objects that control how Pods are deployed%2C scaled and managed.%0A%0ADeploying the workload is not the only task a developer or administrator has to perform. Kubernetes has solutions for some inherent problems with containers and orchestration%2C be it configuration management%2C cross-node networking%2C routing of external traffic%2C load balancing or scaling of the pods.%0A%0A)

# Chapter Overview

In this chapter, we will learn about the different Kubernetes objects, their purpose and how to interact with them.

After setting up a cluster or using an already existing one, we can start deploying some workload. The smallest compute unit in Kubernetes is not a container, but a Pod object. That being, a Pod is not the only abstraction we use for workload. Kubernetes has a variety of workload objects that control how Pods are deployed, scaled and managed.

Deploying the workload is not the only task a developer or administrator has to perform. Kubernetes has solutions for some inherent problems with containers and orchestration, be it configuration management, cross-node networking, routing of external traffic, load balancing or scaling of the pods.

# Learning Objectives

By the end of this chapter, you should be able to:

* Explain what a Kubernetes object is and how to describe it
* Discuss the Pod concept and the problems it solves
* Understand how to scale and schedule Pods with workload resources
* Understand how to abstract Pods with services and how to expose them

Objetivos de aprendizaje Al final de este capítulo, debería poder: Explica qué es un objeto de Kubernetes y cómo describirlo. Discutir el concepto Pod y los problemas que resuelve. Comprender cómo escalar y programar pods con recursos de carga de trabajo Comprenda cómo abstraer Pods con servicios y cómo exponerlos.

Kubernetes Objects

One of the core concepts of Kubernetes is providing a lot of mostly abstract resources, also called objects, that you can use to describe how your workload should be handled. Some of them are used to handle problems of container orchestration, like scheduling and self-healing, others are there to solve some inherent problems of containers.

Kubernetes objects can be distinguished between workload-oriented objects that are used for handling container workloads and infrastructure-oriented objects, that for example handle configuration, networking and security. Some of these objects can be put into a namespace, while others are available across the whole cluster.

As a user, we can describe these objects in the popular data-serialization language YAML and send them to the api-server, where they get validated before they are created.

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

selector:

matchLabels:

app: nginx

replicas: 2 # tells deployment to run 2 pods matching the template

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.19

ports:

- containerPort: 80

The fields highlighted in red are required fields. They include:

apiVersion

Each object can be versioned. That means the data structure of the object can change between different versions.

kind

The kind of object that should be created.

metadata

Data that can be used to identify it. A name is required for each object and must be unique. You can use namespaces if you need multiple objects with the same name.

spec

The specification of the object. Here you can describe your desired state. Be cautious, since the structure for the object can change with its version!

Creating, modifying or deleting an object is only a record of intent where you describe the state your objects should be in, you’re not actively starting pods or even containers for that matter like you would do on your local machine and get direct feedback if it worked or not.

Objetos Kubernetes Uno de los conceptos centrales de Kubernetes es proporcionar una gran cantidad de recursos, en su mayoría abstractos, también llamados objetos, que puede utilizar para describir cómo se debe manejar su carga de trabajo. Algunos de ellos se utilizan para manejar problemas de orquestación de contenedores, como la programación y la autorreparación, otros están ahí para resolver algunos problemas inherentes de los contenedores. Los objetos de Kubernetes se pueden distinguir entre objetos orientados a cargas de trabajo que se utilizan para manejar cargas de trabajo de contenedores y objetos orientados a infraestructura, que por ejemplo manejan configuración, redes y seguridad. Algunos de estos objetos se pueden colocar en un espacio de nombres, mientras que otros están disponibles en todo el clúster. Como usuario, podemos describir estos objetos en el popular lenguaje de serialización de datos YAML y enviarlos al servidor API, donde se validan antes de crearlos.

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: nginx-deployment  
spec:   
 selector:  
 matchLabels:  
 app: nginx  
 replicas: 2 # tells deployment to run 2 pods matching the template  
 template:  
    metadata:  
      labels:  
        app: nginx  
    spec:  
      containers:  
      - name: nginx  
        image: nginx:1.19  
        ports:  
        - containerPort: 80

Los campos resaltados en rojo son campos obligatorios. Incluyen:

api version: Cada objeto puede ser versionado. Eso significa que la estructura de datos del objeto puede cambiar entre diferentes versiones. **Kind:** El tipo de objeto que se debe crear.

**metadatos** Datos que pueden utilizarse para identificarlo. Se requiere un nombre para cada objeto y debe ser único. Puede utilizar espacios de nombres si necesita varios objetos con el mismo nombre.

**Spec:** La especificación del objeto. Aquí puede describir el estado deseado. ¡Tenga cuidado, ya que la estructura del objeto puede cambiar con su versión!

Crear, modificar o eliminar un objeto es solo un registro de intención en el que usted describe el estado en el que deben estar sus objetos; no está iniciando activamente pods o incluso contenedores como lo haría en su máquina local y obteniendo comentarios directos si funcionó o no.

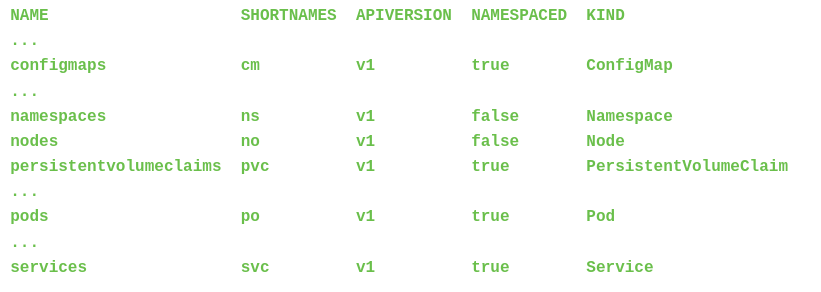
# Interacting with Kubernetes

To access the API, users can use the official command line interface client called kubectl. Lets look at some basic commands for everyday Kubernetes usage.

NOTE: You can learn how to install kubectl in the [official documentation](https://kubernetes.io/docs/tasks/tools/" \l "kubectl).

You can list the available objects in your cluster with the following command:

Interactuando con Kubernetes Para acceder a la API, los usuarios pueden utilizar el cliente de interfaz de línea de comandos oficial llamado kubectl. Veamos algunos comandos básicos para el uso diario de Kubernetes. NOTA: Puede aprender cómo instalar kubectl en la documentación oficial. Puede enumerar los objetos disponibles en su clúster con el siguiente comando:



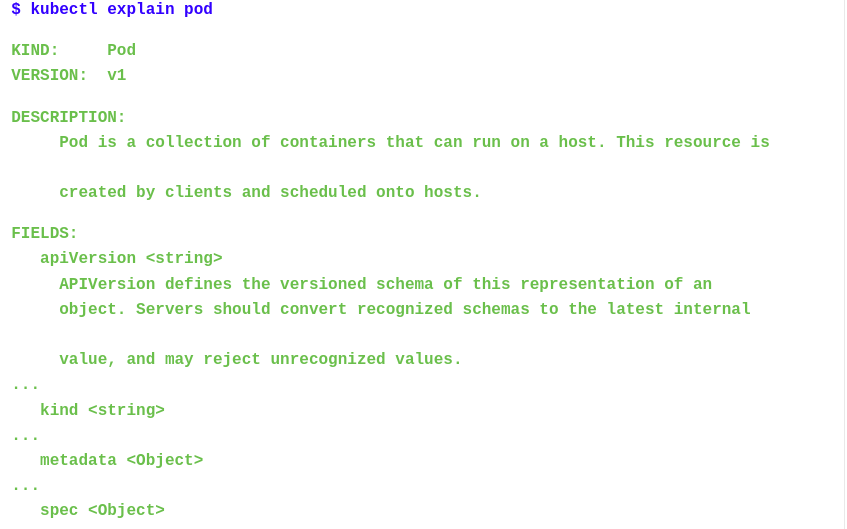
x

Note how objects have short names. That is very helpful for objects with longer names like configmaps or persistentvolumeclaims. The table also shows which objects are namespaced and in what version they’re available.

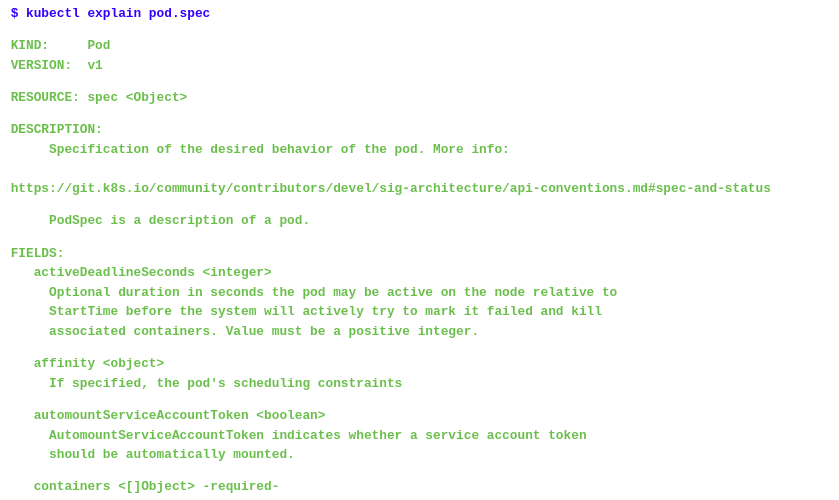
If you want to know more about an object, kubectl has a built-in explanation function!

Let’s learn more about pods:

Observe cómo los objetos tienen nombres cortos. Esto es muy útil para objetos con nombres más largos como mapas de configuración o reclamos de volumen persistentes. La tabla también muestra qué objetos tienen espacios de nombres y en qué versión están disponibles. Si desea saber más sobre un objeto, ¡kubectl tiene una función de explicación incorporada! Aprendamos más sobre las pods:

c

To learn more about the pod spec, you can drill down in the object definition. Use the format: <type>. Para obtener más información sobre la especificación del pod, puede profundizar en la definición del objeto. Utilice el formato: <type>

Let’s take a look at the basic kubectl commands. You can use the --help flag to view them: Echemos un vistazo a los comandos básicos de kubectl. Puedes usar la bandera --help para verlos:

x

To create an object in Kubernetes from a YAML file you can use the following command:

Para crear un objeto en Kubernetes a partir de un archivo YAML, puede utilizar el siguiente comando:

kubectl create -f <your-file>.yaml

There are plenty of graphic user interfaces and dashboards for Kubernetes that allow a visual interaction with the cluster.  
Hay muchas interfaces gráficas de usuario y paneles de control para Kubernetes que permiten una interacción visual con el clúster.

Cc

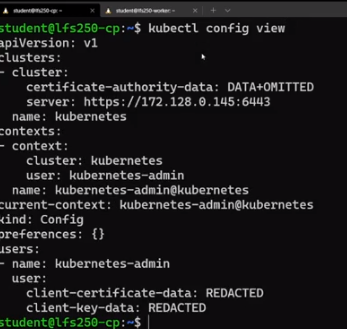
Other tools for interaction with Kubernetes:

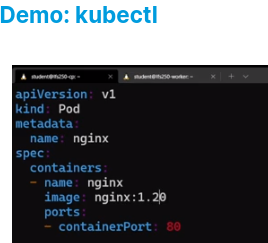
* [kubernetes/dashboard](https://github.com/kubernetes/dashboard)
* [derailed/k9s](https://github.com/derailed/k9s)
* [Lens](https://k8slens.dev/)
* [VMware Tanzu Octant](https://github.com/vmware-tanzu/octant)

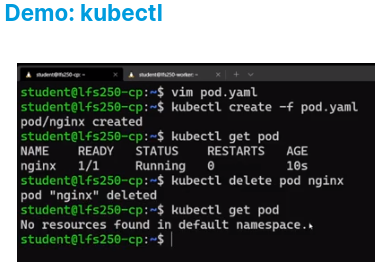
Despite the numerous CLI tools and GUIs, there are also advanced tools that allow the creation of templates and the packaging of Kubernetes objects. Probably the most frequently used tool in connection with Kubernetes today is Helm.

Helm is a package manager for Kubernetes, which allows easier updates and interaction with objects. Helm packages Kubernetes objects in so-called Charts, which can be shared with others via a registry. To get started with Kubernetes, you can search the ArtifactHub to find your favorite software packages, ready to deploy.

A pesar de las numerosas herramientas CLI y GUI, también existen herramientas avanzadas que permiten la creación de plantillas y el empaquetado de objetos de Kubernetes. Probablemente la herramienta más utilizada en relación con Kubernetes en la actualidad sea Helm. Helm es un administrador de paquetes para Kubernetes, que permite actualizaciones e interacción con objetos más sencillas. Helm empaqueta objetos de Kubernetes en los llamados Gráficos, que se pueden compartir con otros a través de un registro. Para comenzar con Kubernetes, puede buscar en ArtifactHub sus paquetes de software favoritos, listos para implementar.







Vv

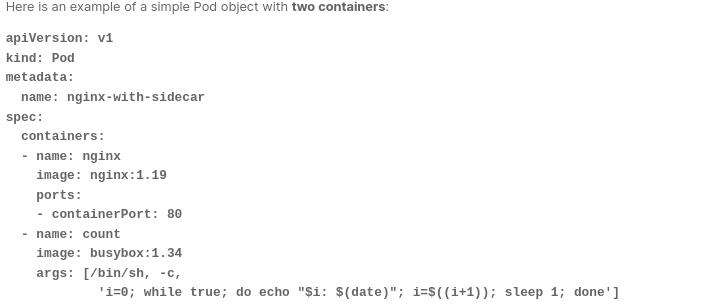
# Pod Concept

The most important object in Kubernetes is a Pod. A pod describes a unit of one or more containers that share an isolation layer of namespaces and cgroups. It is the smallest deployable unit in Kubernetes, which also means that Kubernetes is not interacting with containers directly. The pod concept was introduced to allow running a combination of multiple processes that are interdependent. All containers inside a pod share an IP address and can share via the filesystem.

**Concepto de POD**

El objeto más importante de Kubernetes es un Pod. Un pod describe una unidad de uno o más contenedores que comparten una capa de aislamiento de espacios de nombres y cgroups. Es la unidad desplegable más pequeña de Kubernetes, lo que también significa que Kubernetes no interactúa directamente con los contenedores. El concepto de pod se introdujo para permitir ejecutar una combinación de múltiples procesos que son interdependientes. Todos los contenedores dentro de un pod comparten una dirección IP y pueden compartir a través del sistema de archivos.



C

You could add as many containers to your main application as you want. But be careful since you lose the ability to scale them individually! Using a second container that supports your main application is called a sidecar container.

All containers defined are started at the same time with no ordering, but you also have the ability to use initContainers to start containers before your main application starts. In this example, the init container init-myservice tries to reach another service. Once it completes, the main container is started.

Puede agregar tantos contenedores a su aplicación principal como desee. ¡Pero ten cuidado porque perderás la capacidad de escalarlos individualmente! El uso de un segundo contenedor que admita su aplicación principal se denomina contenedor complementario. Todos los contenedores definidos se inician al mismo tiempo sin realizar pedidos, pero también tiene la posibilidad de usar initContainers para iniciar contenedores antes de que se inicie su aplicación principal. En este ejemplo, el contenedor de inicio init-myservice intenta llegar a otro servicio. Una vez que se completa, se inicia el contenedor principal.

X

Make sure to explore the documentation about Pods, since there are many more settings to discover. Some examples of important settings that can be set for every container in a Pod are:

* resources: Set a resource request and a maximum limit for CPU and Memory.
* livenessProbe: Configure a health check that periodically checks if your application is still alive. Containers can be restarted if the check fails.
* securityContext: Set user & group settings, as well as kernel capabiliti

Asegúrate de explorar la documentación sobre Pods, ya que hay muchas más configuraciones por descubrir. Algunos ejemplos de configuraciones importantes que se pueden establecer para cada contenedor en un Pod son: recursos: establece una solicitud de recursos y un límite máximo para CPU y memoria. livenessProbe: configure una verificación de estado que verifique periódicamente si su aplicación aún está activa. Los contenedores se pueden reiniciar si la verificación falla. securityContext: establece la configuración de usuario y grupo, así como las capacidades del kernel

# Pod Lifecycle

Pods follow a defined lifecycle, starting in the Pending phase, moving through Running if at least one of its primary containers starts OK, and then through either the Succeeded or Failed phases depending on whether any container in the Pod terminated in failure.

Ciclo de vida del pod Los pods siguen un ciclo de vida definido, comenzando en la fase Pendiente, pasando por En ejecución si al menos uno de sus contenedores principales comienza correctamente, y luego a través de las fases Correcta o Fallida dependiendo de si algún contenedor en el Pod terminó en falla.

**Pending**

The Pod has been accepted by the Kubernetes cluster, but one or more of the containers has not been set up and made ready to run. This includes time a Pod spends waiting to be scheduled, as well as the time spent downloading container images over the network.

El clúster de Kubernetes aceptó el pod, pero uno o más contenedores no se configuraron ni prepararon para ejecutarse. Esto incluye el tiempo que pasa un Pod esperando ser programado, así como el tiempo dedicado a descargar imágenes de contenedores a través de la red.

**Running**

The Pod has been bound to a node, and all of the containers have been created. At least one container is still running, or is in the process of starting or restarting.

El Pod se vinculó a un nodo y se crearon todos los contenedores. Al menos un contenedor todavía se está ejecutando o está en proceso de iniciarse o reiniciarse.

Succeeded

All containers in the Pod have terminated in success, and will not be restarted.

T

Tuvo éxito Todos los contenedores del Pod finalizaron con éxito y no se reiniciarán.

Failed

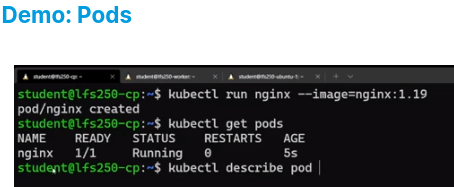
All containers in the Pod have terminated, and at least one container has terminated in failure. That is, the container either exited with non-zero status or was terminated by the system.

Fallido Todos los contenedores del Pod han finalizado y al menos un contenedor ha finalizado con error. Es decir, el contenedor salió con un estado distinto de cero o el sistema lo canceló.

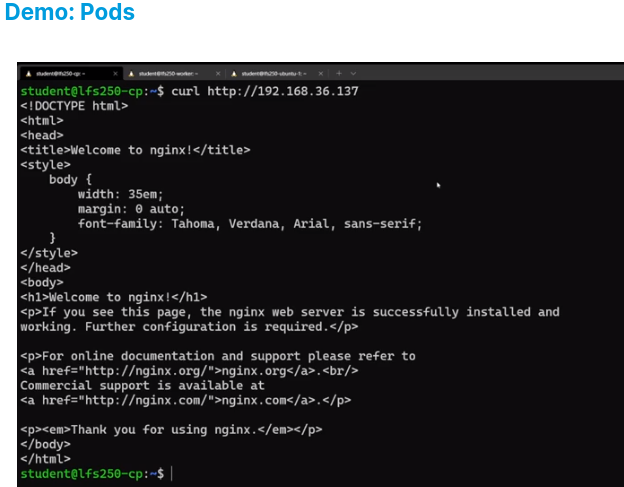
Unknown

For some reason, the state of the Pod could not be obtained. This phase typically occurs due to an error in communicating with the node where the Pod should be running.

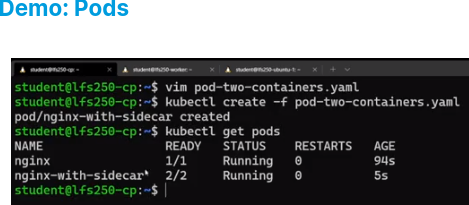
Desconocido Por alguna razón, no se pudo obtener el estado del Pod. Esta fase generalmente ocurre debido a un error en la comunicación con el nodo donde debería ejecutarse el Pod.

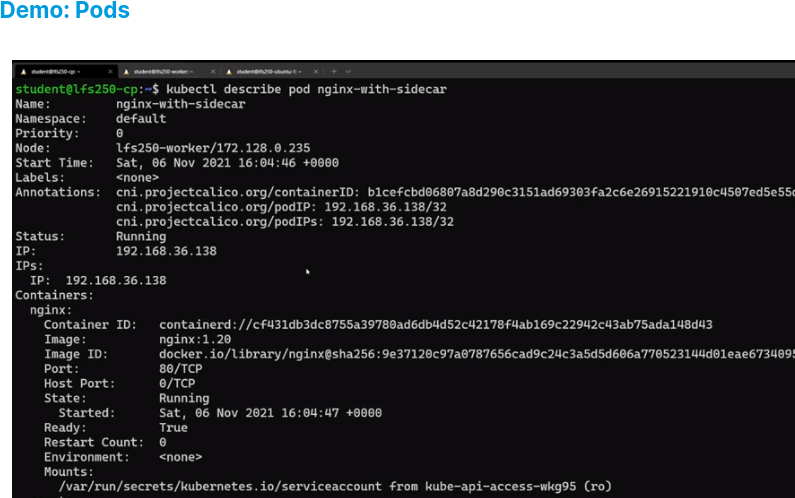


S







ddd

# Workload Objects

Working just with Pods would not be flexible enough in a container orchestration platform. For example, if a Pod is lost because a node failed, it is gone forever. To make sure that a defined number of Pod copies runs all the time, we can use controller objects that manage the pod for us.

Objetos de carga de trabajo Trabajar solo con Pods no sería lo suficientemente flexible en una plataforma de orquestación de contenedores. Por ejemplo, si un Pod se pierde porque falló un nodo, desaparecerá para siempre. Para asegurarnos de que se ejecute una cantidad definida de copias de Pod todo el tiempo, podemos usar objetos de controlador que administren el pod por nosotros.

**Kubernetes Objects**

**ReplicaSet**

A controller object that ensures a desired number of pods is running at any given time. ReplicaSets can be used to scale out applications and improve their availability. They do this by starting multiple copies of a pod definition.

Conjunto de réplicas Un objeto controlador que garantiza que se esté ejecutando una cantidad deseada de pods en un momento dado. ReplicaSets se puede utilizar para escalar aplicaciones y mejorar su disponibilidad. Para ello, inician varias copias de una definición de pod.

**Deployment**

The most feature-rich object in Kubernetes. A Deployment can be used to describe the complete application lifecycle, they do this by managing multiple ReplicaSets that get updated when the application is changed by providing a new container image, for example. Deployments are perfect to run stateless applications in Kubernetes.

Despliegue El objeto con más funciones de Kubernetes. Se puede utilizar una implementación para describir el ciclo de vida completo de la aplicación; lo hacen administrando múltiples ReplicaSets que se actualizan cuando se cambia la aplicación al proporcionar una nueva imagen de contenedor, por ejemplo. Las implementaciones son perfectas para ejecutar aplicaciones sin estado en Kubernetes.

**StatefulSet**

Considered a bad practice for a long time, StatefulSets can be used to run stateful applications like databases on Kubernetes. Stateful applications have special requirements that don't fit the ephemeral nature of pods and containers. In contrast to Deployments, StatefulSets try to retain IP addresses of pods and give them a stable name, persistent storage and more graceful handling of scaling and updates.

Conjunto con estado StatefulSets, considerado una mala práctica durante mucho tiempo, se puede utilizar para ejecutar aplicaciones con estado, como bases de datos, en Kubernetes. Las aplicaciones con estado tienen requisitos especiales que no se ajustan a la naturaleza efímera de los pods y contenedores. A diferencia de las implementaciones, los StatefulSets intentan conservar las direcciones IP de los pods y darles un nombre estable, almacenamiento persistente y un manejo más elegante del escalado y las actualizaciones.

DaemonSet

Ensures that a copy of a Pod runs on all (or some) nodes of your cluster. DaemonSets are perfect to run infrastructure-related workload, for example monitoring or logging tools.

Conjunto de demonios Garantiza que se ejecute una copia de un Pod en todos (o algunos) los nodos de su clúster. Los DaemonSets son perfectos para ejecutar cargas de trabajo relacionadas con la infraestructura, por ejemplo, herramientas de monitoreo o registro.

Job

Creates one or more Pods that execute a task and terminate afterwards. Job objects are perfect to run one-shot scripts like database migrations or administrative tasks.

Crea uno o más Pods que ejecutan una tarea y finalizan luego. Los objetos de trabajo son perfectos para ejecutar scripts de una sola vez, como migraciones de bases de datos o tareas administrativas.

CronJob

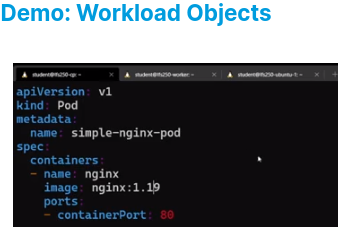
CronJobs add a time-based configuration to jobs. This allows running Jobs periodically, for example doing a backup job every night at 4am.

CronJob CronJobs agrega una configuración basada en tiempo a los trabajos. Esto permite ejecutar trabajos periódicamente, por ejemplo, realizar un trabajo de copia de seguridad todas las noches a las 4 a.m.

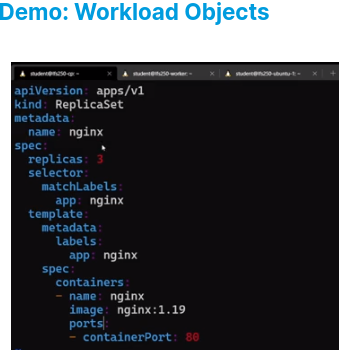
## Interactive Tutorial - Deploy an App and explore it

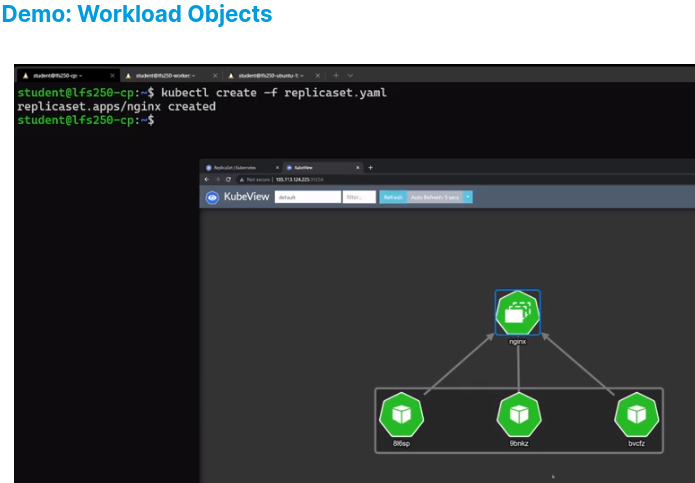
You can learn how to deploy an application in your Minikube cluster in the [second part of the interactive tutorial available in the Kubernetes documentation](https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/).

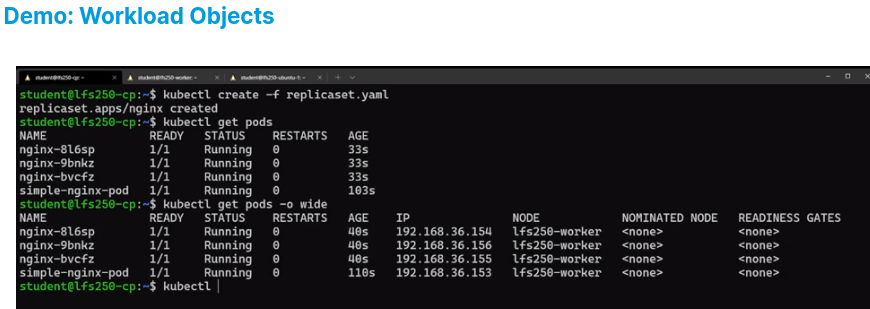
Apply what you have learned from "Interacting with Kubernetes" to explore your app in the [third part of the interactive tutorial](https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/).

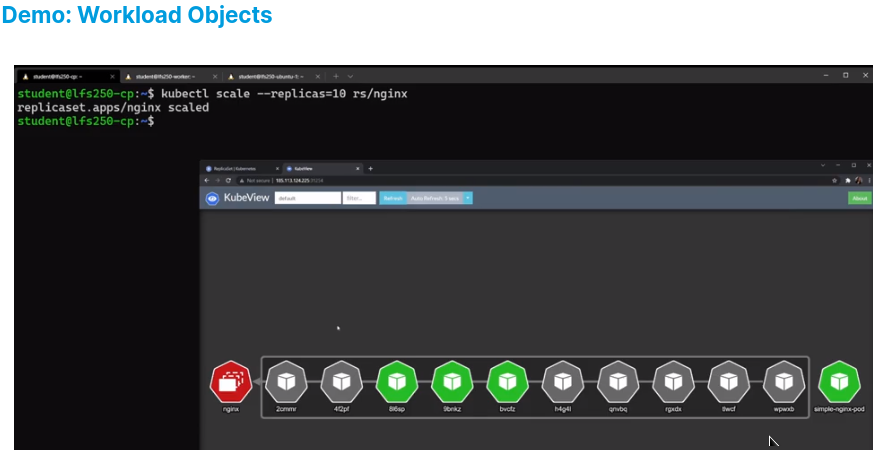


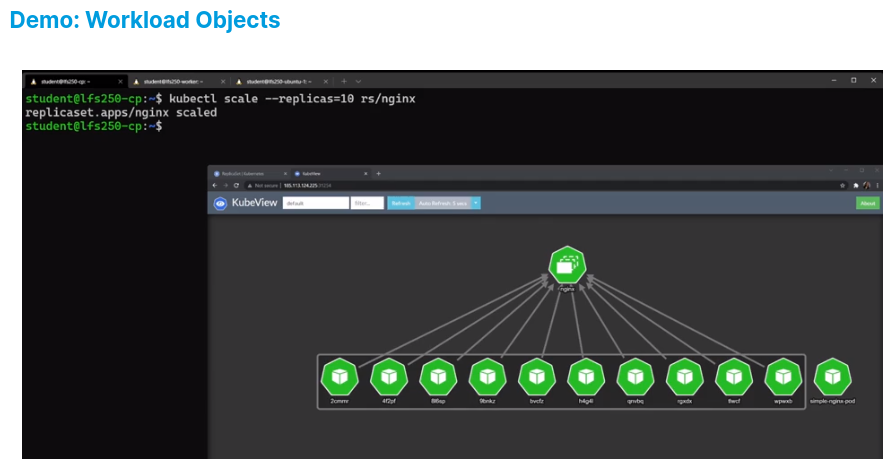
Xx

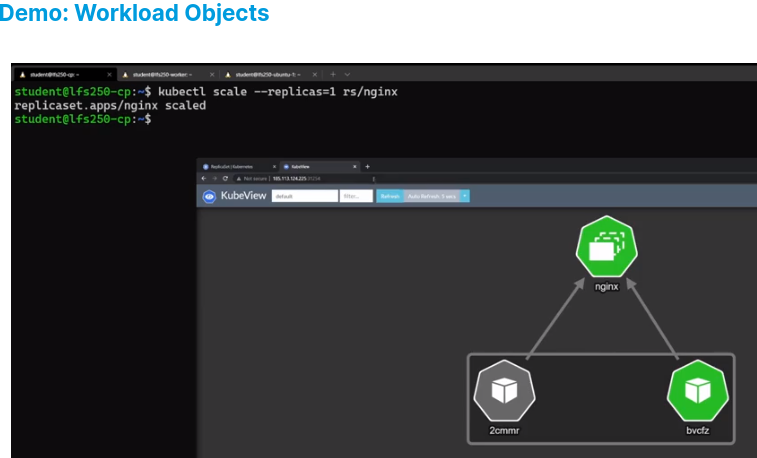


c

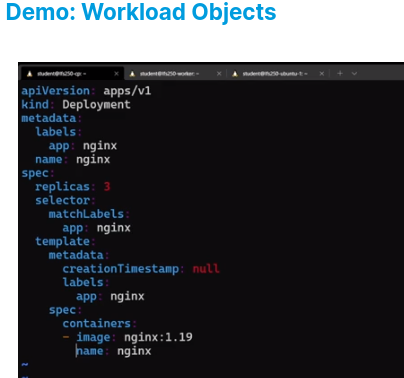
c

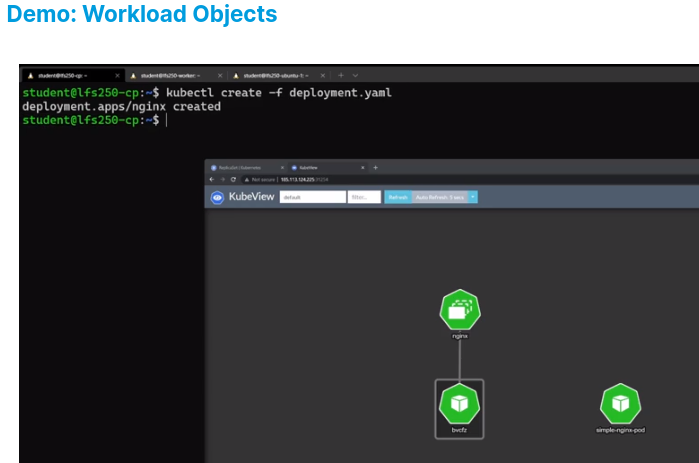
x

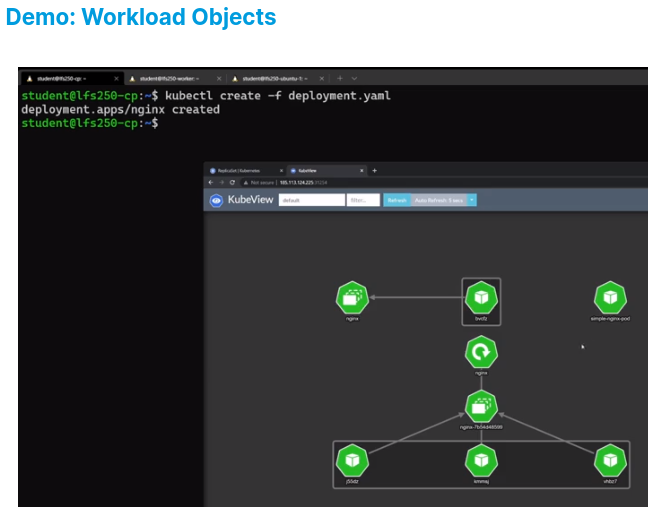
c

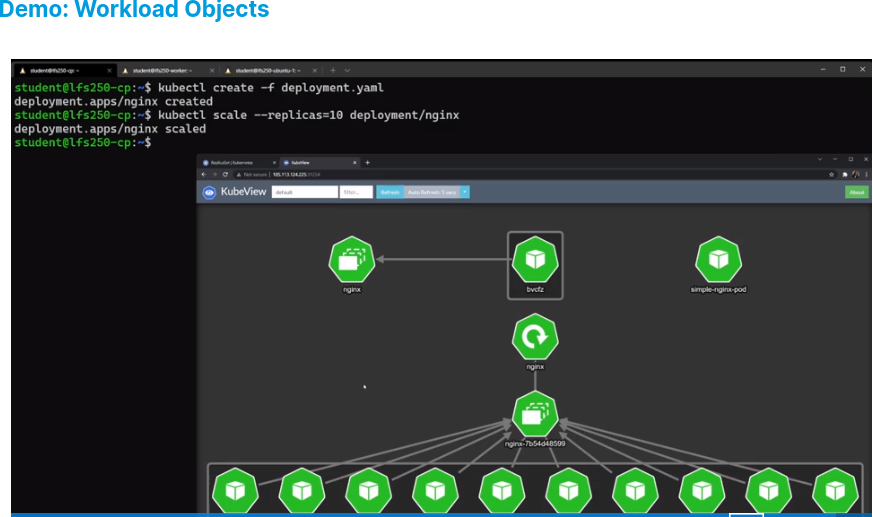


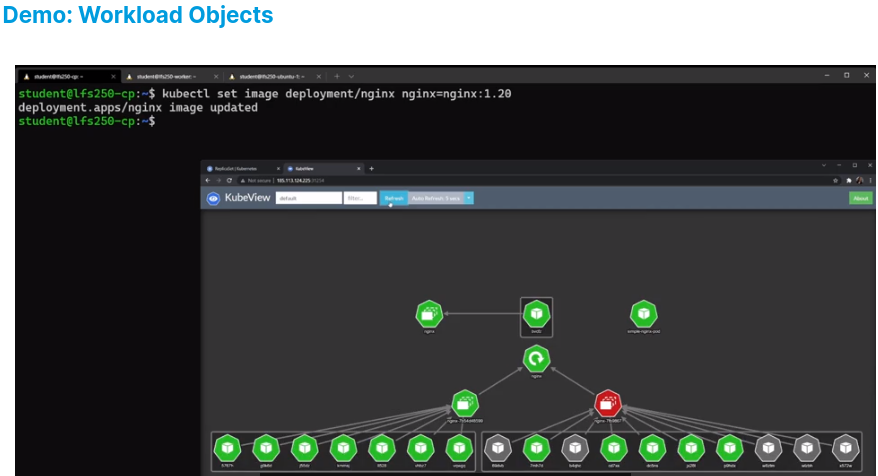
deployment







c

c

# Networking Objects

Since a lot of Pods would require a lot of manual network configuration, we can use Service and Ingress objects to define and abstract networking.

Objetos de red Dado que muchos Pods requerirían mucha configuración de red manual, podemos usar objetos Service e Ingress para definir y abstraer la red.

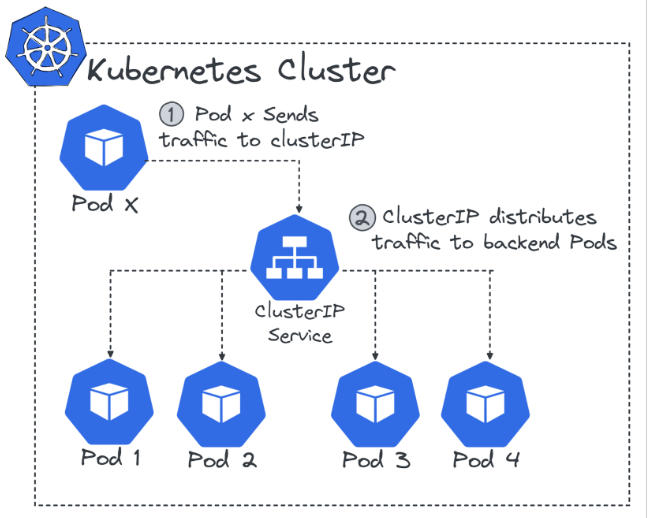
Services can be used to expose a set of pods as a network service. Click to expand each box and learn more about each of the four distinct types of services we can use.

Los servicios se pueden utilizar para exponer un conjunto de pods como un servicio de red. Haga clic para expandir cada cuadro y obtener más información sobre cada uno de los cuatro tipos distintos de servicios que podemos utilizar.

**ClusterIP**

The most common service type. A ClusterIP is a virtual IP inside Kubernetes that can be used as a single endpoint for a set of pods. This service type can be used as a round-robin load balancer.

P de clúster El tipo de servicio más común. Una ClusterIP es una IP virtual dentro de Kubernetes que se puede utilizar como punto final único para un conjunto de pods. Este tipo de servicio se puede utilizar como equilibrador de carga por turnos.

­

**NodePort**

The NodePort service type extends the ClusterIP by adding simple routing rules. It opens a port (default between 30000-32767) on every node in the cluster and maps it to the ClusterIP. This service type allows routing external traffic to the cluster.

Puerto de nodo El tipo de servicio NodePort amplía ClusterIP agregando reglas de enrutamiento simples. Abre un puerto (predeterminado entre 30000 y 32767) en cada nodo del clúster y lo asigna a ClusterIP. Este tipo de servicio permite enrutar el tráfico externo al clúster.

**LoadBalancer**

The LoadBalancer service type extends the NodePort by deploying an external LoadBalancer instance. This will only work if you’re in an environment that has an API to configure a LoadBalancer instance, like GCP, AWS, Azure or even OpenStack.

Equilibrador de carga El tipo de servicio LoadBalancer amplía NodePort mediante la implementación de una instancia externa de LoadBalancer. Esto solo funcionará si estás en un entorno que tiene una API para configurar una instancia de LoadBalancer, como GCP, AWS, Azure o incluso OpenStack.

ExternalName

A special service type that has no routing whatsoever. ExternalName is using the Kubernetes internal DNS server to create a DNS alias. You can use this to create a simple alias to resolve a rather complicated hostname like: my-cool-database-az1-uid123.cloud-provider-i-like.com. This is especially useful if you want to reach external resources from your Kubernetes cluster.

Nombre externo Un tipo de servicio especial que no tiene ruta alguna. ExternalName utiliza el servidor DNS interno de Kubernetes para crear un alias de DNS. Puede usar esto para crear un alias simple para resolver un nombre de host bastante complicado como: my-cool-database-az1-uid123.cloud-provider-i-like.com. Esto es especialmente útil si desea acceder a recursos externos desde su clúster de Kubernetes.

**Headless Services**

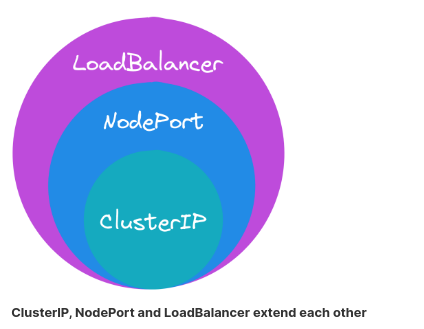
Sometimes you don't need load-balancing and a single Service IP. In this case, you can create what are termed "headless" Services, by explicitly specifying "None" for the cluster IP (.spec.clusterIP).

You can use a headless Service to interface with other service discovery mechanisms, without being tied to Kubernetes' implementation.

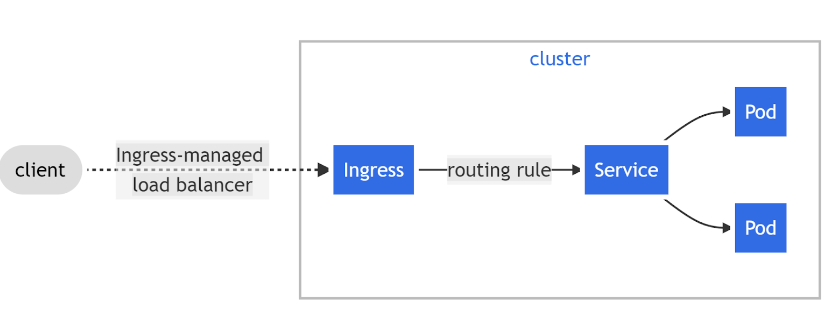
For headless Services, a cluster IP is not allocated, kube-proxy does not handle these Services, and there is no load balancing or proxying done by the platform for them. How DNS is automatically configured depends on whether the Service has selectors defined with or without selectors.

Example: A StatefulSet controller can use the Headless Service to control the domain of its pods, where stable network id is the need and not load-balancing.

Servicios sin cabeza A veces no necesita equilibrio de carga ni una única IP de servicio. En este caso, puede crear lo que se denomina Servicios "sin cabeza", especificando explícitamente "Ninguno" para la IP del clúster (.spec.clusterIP). Puede utilizar un Servicio sin cabeza para interactuar con otros mecanismos de descubrimiento de servicios, sin estar vinculado a la implementación de Kubernetes. Para los servicios sin cabeza, no se asigna una IP de clúster, kube-proxy no maneja estos servicios y la plataforma no realiza equilibrio de carga ni proxy para ellos. La forma en que se configura DNS automáticamente depende de si el Servicio tiene selectores definidos con o sin selectores. Ejemplo: un controlador StatefulSet puede usar el servicio sin cabeza para controlar el dominio de sus pods, donde la identificación de red estable es la necesidad y no el equilibrio de carga.



f you need even more flexibility to expose applications, you can use an Ingress object. Ingress provides a means to expose HTTP and HTTPS routes from outside of the cluster for a service within the cluster. It does this by configuring routing rules that a user can set and implement with an ingress controller.



Standard features of ingress controllers may include:

* LoadBalancing
* TLS offloading/termination
* Name-based virtual hosting
* Path-based routing

A lot of ingress controllers even provide more features, like:

* Redirects
* Custom errors
* Authentication
* Session affinity
* Monitoring
* Logging
* Weighted routing
* Rate limiting.

Kubernetes also provides a cluster internal firewall with the NetworkPolicy concept. NetworkPolicies are a simple IP firewall (OSI Layer 3 or 4) that can control traffic based on rules. You can define rules for incoming (ingress) and outgoing traffic (egress). A typical use case for NetworkPolicies would be restricting the traffic between two different namespaces.

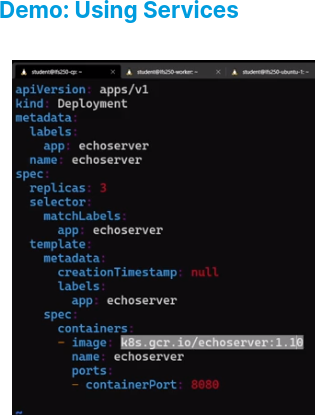
## Interactive Tutorial - Expose Your App

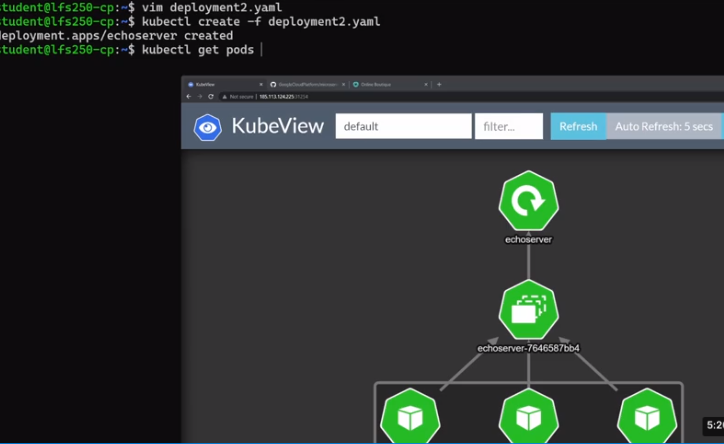
You can now learn how to [expose your application with a Service](https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/) in the fourth part of the interactive tutorial available in the Kubernetes documentation.

as características estándar de los controladores de ingreso pueden incluir:

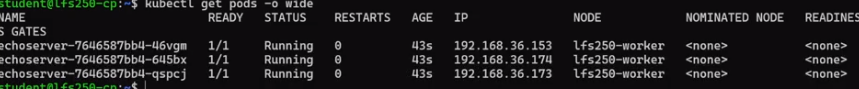
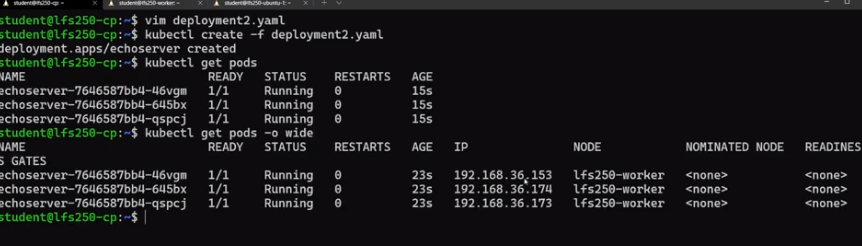
Balanceo de carga Descarga/terminación de TLS Alojamiento virtual basado en nombres Enrutamiento basado en rutas

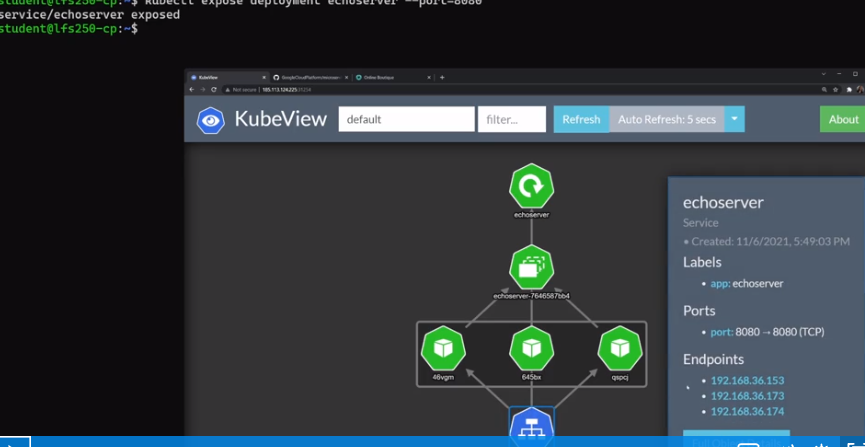
Muchos controladores de ingreso incluso ofrecen más funciones, como: Redirecciones Errores personalizados Autenticación Afinidad de sesión Supervisión Inicio sesión Enrutamiento ponderado Limitación de velocidad. Kubernetes también proporciona un firewall interno del clúster con el concepto NetworkPolicy. NetworkPolicies son un firewall IP simple (OSI Layer 3 o 4) que puede controlar el tráfico según reglas. Puede definir reglas para el tráfico entrante (entrada) y saliente (salida). Un caso de uso típico de NetworkPolicies sería restringir el tráfico entre dos espacios de nombres diferentes.

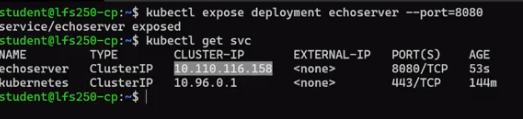


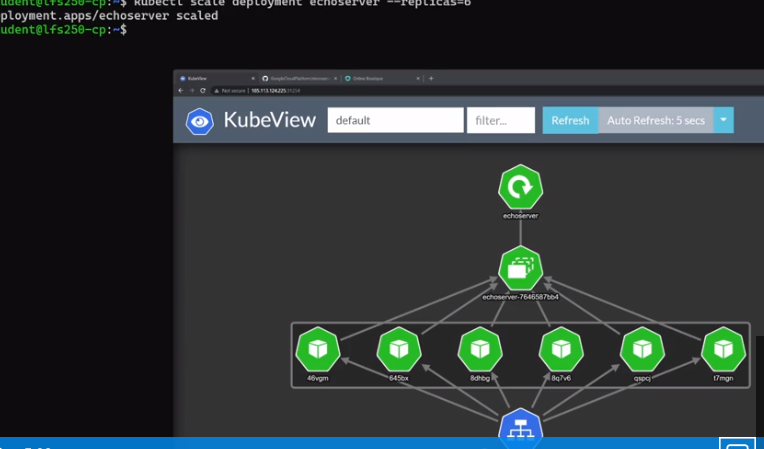


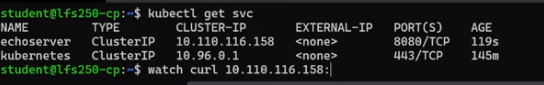
X









c

# Volume & Storage Objects

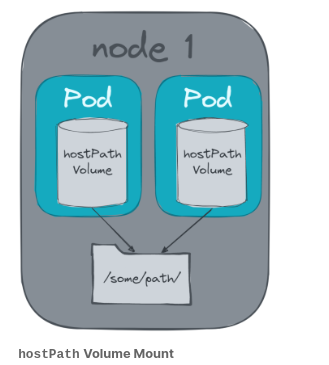
As mentioned earlier, containers were not designed with persistent storage in mind, especially when that storage spans across multiple nodes. Kubernetes introduces a few solutions, but note that these solutions do not automatically remove all of the complexities of managing storage with containers.

Containers already had the concept of mounting volumes, but since we’re not working with containers directly, Kubernetes made volumes part of a Pod, just like containers are.

Here’s an example of a hostPath volume mount that is similar to a host mount introduced by Docker:

Objetos de volumen y almacenamiento Como se mencionó anteriormente, los contenedores no se diseñaron teniendo en cuenta el almacenamiento persistente, especialmente cuando ese almacenamiento abarca varios nodos. Kubernetes presenta algunas soluciones, pero tenga en cuenta que estas soluciones no eliminan automáticamente todas las complejidades de administrar el almacenamiento con contenedores. Los contenedores ya tenían el concepto de montar volúmenes, pero como no trabajamos con contenedores directamente, Kubernetes hizo que los volúmenes formaran parte de un Pod, al igual que los contenedores. A continuación se muestra un ejemplo de un montaje de volumen hostPath que es similar a un montaje de host introducido por Docker:





Volumes allow sharing data between multiple containers within the same Pod. This concept allows for great flexibility when you want to use a sidecar pattern. The second purpose they serve is preventing data loss when a Pod crashes and is restarted on the same node. Pods are started in a clean state, but all data is lost unless written to a volume.

Unfortunately, a cluster environment with multiple servers requires even more flexibility when it comes to persistent storage. Depending on the environment, we can use cloud block storage like [Amazon EBS](https://aws.amazon.com/ebs/), [Google Persistent Disks](https://cloud.google.com/persistent-disk), [Azure Disk Storage](https://azure.microsoft.com/en-us/services/storage/disks/) or consume from storage systems like [Ceph](https://ceph.io/en/), [GlusterFS](https://www.gluster.org/) or more traditional systems like [NFS](https://en.wikipedia.org/wiki/Network_File_System).

These are only a few examples of storage that can be used in Kubernetes. To make the user experience more uniform, Kubernetes is using the [Container Storage Interface (CSI)](https://github.com/container-storage-interface/spec) which allows the storage vendor to write a plugin (storage driver) that can be used in Kubernetes.

To use this abstraction, we have two more objects that can be used:

* PersistentVolumes (PV)  
  An abstract description for a slice of storage. The object configuration holds information like type of volume, volume size, access mode and unique identifiers and information how to mount it.
* PersistentVolumeClaims (PVC)  
  A request for storage by a user. If the cluster has multiple persistent volumes, the user can create a PVC which will reserve a persistent volume according to the user's needs.

Los volúmenes permiten compartir datos entre múltiples contenedores dentro de un mismo Pod. Este concepto permite una gran flexibilidad cuando se desea utilizar un patrón de sidecar. El segundo propósito al que sirven es evitar la pérdida de datos cuando un Pod falla y se reinicia en el mismo nodo. Los pods se inician en un estado limpio, pero todos los datos se pierden a menos que se escriban en un volumen. Desafortunadamente, un entorno de clúster con múltiples servidores requiere aún más flexibilidad cuando se trata de almacenamiento persistente. Dependiendo del entorno, podemos utilizar almacenamiento en bloques en la nube como Amazon EBS, Google Persistent Disks, Azure Disk Storage o consumir desde sistemas de almacenamiento como Ceph, GlusterFS o sistemas más tradicionales como NFS. Estos son sólo algunos ejemplos de almacenamiento que se pueden utilizar en Kubernetes. Para que la experiencia del usuario sea más uniforme, Kubernetes utiliza la interfaz de almacenamiento de contenedores (CSI), que permite al proveedor de almacenamiento escribir un complemento (controlador de almacenamiento) que se puede usar en Kubernetes.

ara usar esta abstracción, tenemos dos objetos más que se pueden usar: **Volúmenes persistentes (PV)** Una descripción abstracta de una porción de almacenamiento. La configuración del objeto contiene información como el tipo de volumen, el tamaño del volumen, el modo de acceso y los identificadores únicos e información sobre cómo montarlo.

**Reclamaciones de volumen persistente (PVC)** Una solicitud de almacenamiento por parte de un usuario. Si el clúster tiene varios volúmenes persistentes, el usuario puede crear un PVC que reservará un volumen persistente según las necesidades del usuario.



The example shows a PersistentVolume that uses an AWS EBS volume implemented with a CSI driver. After the PersistentVolume is provisioned, a developer can reserve it with a PersistentVolumeClaim. The last step is using the PVC as a volume in a Pod, just like the hostPath example we saw before.

It is possible to operate storage clusters directly in Kubernetes. Projects like [Rook](https://rook.io/) provide cloud-native storage orchestration and integrate with battle tested storage solutions like Ceph.

El ejemplo muestra un PersistentVolume que utiliza un volumen de AWS EBS implementado con un controlador CSI. Una vez aprovisionado el PersistentVolume, un desarrollador puede reservarlo con PersistentVolumeClaim. El último paso es usar el PVC como volumen en un Pod, tal como en el ejemplo de hostPath que vimos antes. Es posible operar clústeres de almacenamiento directamente en Kubernetes. Proyectos como Rook brindan orquestación de almacenamiento nativa de la nube y se integran con soluciones de almacenamiento probadas como Ceph.



# Configuration Objects

The twelve factor app recommends [storing configuration in the environment](https://12factor.net/config). But what does that mean exactly? Running an application requires more than the application code and some libraries. Applications have config files, connect to other services, databases, storage systems or caches and that requires configuration like [connection strings](https://en.wikipedia.org/wiki/Connection_string).

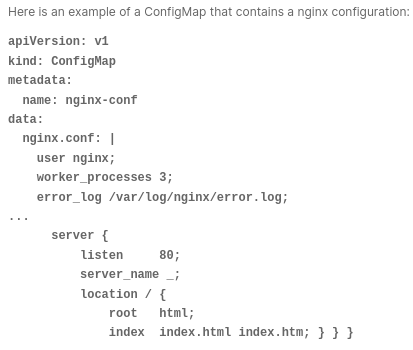
It is considered bad practice to incorporate the configuration directly into the container build. Any configuration change would require the entire image to be rebuilt and the entire container or pod to be redeployed. This problem gets only worse when multiple environments (development, staging, production) are used and images are being built for each and every environment. The twelve factor app explains this problem in more detail: [Dev/prod parity](https://12factor.net/dev-prod-parity).

In Kubernetes, this problem is solved by decoupling the configuration from the Pods with a ConfigMap. ConfigMaps can be used to store whole configuration files or variables as key-value pairs. There are two possible ways to use a ConfigMap:

* Mount a ConfigMap as a volume in Pod
* Map variables from a ConfigMap to environment variables of a Pod.

Here is an example of a ConfigMap that contains a nginx configuration:

Objetos de configuración La aplicación de doce factores recomienda almacenar la configuración en el entorno. ¿Pero qué significa eso exactamente? Ejecutar una aplicación requiere más que el código de la aplicación y algunas bibliotecas. Las aplicaciones tienen archivos de configuración, se conectan a otros servicios, bases de datos, sistemas de almacenamiento o cachés y eso requiere configuración como cadenas de conexión. Se considera una mala práctica incorporar la configuración directamente en la compilación del contenedor. Cualquier cambio de configuración requeriría reconstruir toda la imagen y volver a implementar todo el contenedor o pod. Este problema solo empeora cuando se utilizan múltiples entornos (desarrollo, puesta en escena, producción) y se crean imágenes para todos y cada uno de los entornos. La aplicación de doce factores explica este problema con más detalle: paridad desarrollo/producción. En Kubernetes, este problema se soluciona desacoplando la configuración de los Pods con un ConfigMap. ConfigMaps se puede utilizar para almacenar variables o archivos de configuración completos como pares clave-valor. Hay dos formas posibles de utilizar un ConfigMap: Montar un ConfigMap como volumen en Pod Asigne variables de un ConfigMap a variables de entorno de un Pod. A continuación se muestra un ejemplo de un ConfigMap que contiene una configuración de nginx:



Una vez creado el ConfigMap, puedes usarlo en un Pod:



Right from the beginning Kubernetes also provided an object to store sensitive information like passwords, keys or other credentials. These objects are called Secrets. Secrets are very much related to ConfigMaps and basically their only difference is that secrets are base64 encoded.

There is an on-going debate about the risk of using Secrets, since their - in contrast to their name - not considered secure.

In cloud-native environments purpose-built secret management tools have emerged that integrate very well with Kubernetes. One example would be [HashiCorp Vault](https://www.vaultproject.io/).

Desde el principio, Kubernetes también proporcionó un objeto para almacenar información confidencial como contraseñas, claves u otras credenciales. Estos objetos se llaman Secretos. Los secretos están muy relacionados con ConfigMaps y básicamente su única diferencia es que los secretos están codificados en base64. Existe un debate en curso sobre el riesgo de utilizar Secrets, ya que, a diferencia de su nombre, no se consideran seguros. En entornos nativos de la nube, han surgido herramientas de gestión secreta especialmente diseñadas que se integran muy bien con Kubernetes. Un ejemplo sería HashiCorp Vault.

# **Autoscaling Objects**

to scale the workload in a Kubernetes cluster, we can use three different Autoscaling mechanisms. To learn more about them, click on each box.

Para escalar la carga de trabajo en un clúster de Kubernetes, podemos utilizar tres mecanismos de Autoescalado diferentes. Para conocer más sobre ellos, haga clic en cada casilla.

Horizontal Pod Autoscaler (HPA)

[Horizontal Pod Autoscaler (HPA)](https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/) is the most used autoscaler in Kubernetes. The HPA can watch Deployments or ReplicaSets and increase the number of Replicas if a certain threshold is reached. Imaging your Pod can use 500MiB of memory and you configured a threshold of 80%. If the usage is over 400MiB (80%), a second Pod will get scheduled. Now you have a capacity of 1000MiB. If 800MiB is used, a third Pod will get scheduled and so on.

Escalador automático de pods horizontales (HPA) Horizontal Pod Autoscaler (HPA) es el escalador automático más utilizado en Kubernetes. El HPA puede observar implementaciones o conjuntos de réplicas y aumentar la cantidad de réplicas si se alcanza un cierto umbral. La creación de imágenes de su Pod puede utilizar 500 MiB de memoria y usted configuró un umbral del 80 %. Si el uso supera los 400MiB (80%), se programará un segundo Pod. Ahora tienes una capacidad de 1000MiB. Si se utilizan 800MiB, se programará un tercer Pod y así sucesivamente.

Cluster Autoscaler

Of course, there is no point in starting more and more Replicas of Pods, if the Cluster capacity is fixed. The [Cluster Autoscaler](https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler) can add new worker nodes to the cluster if the demand increases. The Cluster Autoscaler works great in tandem with the Horizontal Autoscaler.

Escalador automático de clústeres Por supuesto, no tiene sentido iniciar más y más réplicas de Pods si la capacidad del clúster es fija. Cluster Autoscaler puede agregar nuevos nodos trabajadores al clúster si la demanda aumenta. El escalador automático de clúster funciona muy bien en conjunto con el escalador automático horizontal.

Vertical Pod Autoscaler

The [Vertical Pod Autoscaler](https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler) is relatively new and allows Pods to increase the resource requests and limits dynamically. As we discussed earlier, vertical scaling is limited by the node capacity.

Escalador automático de pods verticales El escalador automático de pods vertical es relativamente nuevo y permite a los pods aumentar las solicitudes y los límites de recursos de forma dinámica. Como comentamos anteriormente, el escalamiento vertical está limitado por la capacidad del nodo.

Unfortunately, (horizontal) autoscaling in Kubernetes is NOT available out of the box and requires installing an add-on called [metrics-server](https://github.com/kubernetes-sigs/metrics-server).

It is possible though to replace the metrics-server with [Prometheus Adapter for Kubernetes Metrics APIs](https://github.com/kubernetes-sigs/prometheus-adapter). The prometheus-adapter allows you to use custom metrics in Kubernetes and scale up or down based on things like requests or number of users on your system.

Rather than relying solely on metrics, projects like [KEDA](https://keda.sh/) can be used to scale the Kubernetes workload based on events triggered by external systems. KEDA stands for Kubernetes-based Event Driven Autoscaler and was started in 2019 as a partnership between Microsoft and Red Hat. Similar to the HPA, KEDA can scale deployments, ReplicaSets, pods, etc., but also other objects such as Kubernetes jobs. With a large selection of out-of-the-box scalers, KEDA can scale to special triggers such as a database query or even the number of pods in a Kubernetes cluster.

Desafortunadamente, el escalado automático (horizontal) en Kubernetes NO está disponible de fábrica y requiere la instalación de un complemento llamado metrics-server. Sin embargo, es posible reemplazar el servidor de métricas con el adaptador Prometheus para las API de métricas de Kubernetes. El adaptador prometheus le permite utilizar métricas personalizadas en Kubernetes y ampliar o reducir según aspectos como solicitudes o la cantidad de usuarios en su sistema. En lugar de depender únicamente de métricas, proyectos como KEDA se pueden utilizar para escalar la carga de trabajo de Kubernetes en función de eventos desencadenados por sistemas externos. KEDA significa Event Driven Autoscaler basado en Kubernetes y se inició en 2019 como una asociación entre Microsoft y Red Hat. Al igual que HPA, KEDA puede escalar implementaciones, ReplicaSets, pods, etc., pero también otros objetos como trabajos de Kubernetes. Con una gran selección de escaladores listos para usar, KEDA puede escalar a activadores especiales, como una consulta de base de datos o incluso la cantidad de pods en un clúster de Kubernetes.

# **Scheduling Objects**

**The scheduler is the control process which assigns Pods to Nodes. The scheduler determines which Nodes are valid placements for each Pod in the scheduling queue according to constraints and available resources. The scheduler then ranks each valid Node and binds the Pod to a suitable Node. Multiple different schedulers may be used within a cluster; kube-scheduler is the default implementation.**

The default scheduler does a good job of scheduling the pods across the nodes in the cluster, however there are scenarios where you want to restrict the pod on particular nodes or prefer to run on particular nodes. There are several ways of doing this, the recommended way is to make use of Label Selectors to facilitate the selection.

El planificador es el proceso de control que asigna Pods a Nodos. El planificador determina qué Nodos son ubicaciones válidas para cada Pod en la cola de programación según las restricciones y los recursos disponibles. Luego, el programador clasifica cada Nodo válido y vincula el Pod a un Nodo adecuado. Se pueden utilizar varios programadores diferentes dentro de un clúster; kube-scheduler es la implementación predeterminada. El programador predeterminado hace un buen trabajo al programar los pods en los nodos del clúster; sin embargo, hay escenarios en los que desea restringir el pod en nodos concretos o prefiere ejecutarlo en nodos concretos. Hay varias formas de hacer esto, la forma recomendada es utilizar selectores de etiquetas para facilitar la selección.

**Methods**

**nodeSelector field matching against node labels**

nodeSelector is the simplest recommended form of node selection constraint. You can add the nodeSelector field to your Pod specification and specify the node labels you want the target node to have. Kubernetes only schedules the Pod onto nodes that have each of the labels you specify.

Campo nodeSelector que coincide con las etiquetas de nodo nodeSelector es la forma recomendada más simple de restricción de selección de nodos. Puede agregar el campo nodeSelector a su especificación de Pod y especificar las etiquetas de nodo que desea que tenga el nodo de destino. Kubernetes solo programa el Pod en nodos que tienen cada una de las etiquetas que usted especifica.

**Affinity and anti-affinity**

Affinity and anti-affinity expands the types of constraints you can define and give you more control over the selection logic. You can indicate that a rule is soft or preferred, so that the scheduler still schedules the Pod even if it can't find a matching node.

Afinidad y antiafinidad La afinidad y la antiafinidad amplían los tipos de restricciones que puede definir y le brindan más control sobre la lógica de selección. Puede indicar que una regla es flexible o preferida, de modo que el programador aún programe el Pod incluso si no puede encontrar un nodo coincidente.

**nodeName field**

nodeName is a more direct form of node selection than affinity or nodeSelector. nodeName is a field in the Pod spec. If the nodeName field is not empty, the scheduler ignores the Pod and the kubelet on the named node tries to place the Pod on that node. Using nodeName overrules using nodeSelector or affinity and anti-affinity rules.

campo nombre de nodo nodeName es una forma más directa de selección de nodos que afinidad o nodeSelector. nodeName es un campo en la especificación Pod. Si el campo nodeName no está vacío, el programador ignora el Pod y el kubelet en el nodo nombrado intenta colocar el Pod en ese nodo. El uso de nodeName anula el uso de nodeSelector o reglas de afinidad y antiafinidad.

Pod topology spread constraints

You can use topology spread constraints to control how Pods are spread across your cluster among failure-domains such as regions, zones, nodes, or among any other topology domains that you define. You might do this to improve performance, expected availability, or overall utilization.

Restricciones de distribución de la topología del pod Puede utilizar restricciones de distribución de topología para controlar cómo se distribuyen los pods en su clúster entre dominios de falla, como regiones, zonas, nodos o entre cualquier otro dominio de topología que defina. Puede hacer esto para mejorar el rendimiento, la disponibilidad esperada o la utilización general.

## **Taints and Tolerations**

Node affinity is a property of Pods that attracts them to a set of nodes (either as a preference or a hard requirement). Taints are the opposite -- they allow a node to repel a set of pods.

Tolerations are applied to pods. Tolerations allow the scheduler to schedule pods with matching taints. Tolerations allow scheduling, but don't guarantee scheduling: the scheduler also evaluates other parameters as part of its function.

Taints and tolerations work together to ensure that pods are not scheduled onto inappropriate nodes. One or more taints are applied to a node; this marks that the node should not accept any pods that do not tolerate the taints.

A taint consists of a key, value, and effect. As an argument here, it is expressed as key=value:effect.

Contaminaciones y tolerancias La afinidad de nodos es una propiedad de los Pods que los atrae a un conjunto de nodos (ya sea como una preferencia o como un requisito estricto). Las manchas son todo lo contrario: permiten que un nodo rechace un conjunto de cápsulas. Se aplican tolerancias a las vainas. Las tolerancias permiten al programador programar pods con taints coincidentes. Las tolerancias permiten la programación, pero no la garantizan: el programador también evalúa otros parámetros como parte de su función. Las contaminaciones y las tolerancias trabajan juntas para garantizar que los pods no se programen en nodos inapropiados. Se aplican una o más manchas a un nodo; esto marca que el nodo no debe aceptar ningún pod que no tolere las contaminaciones. Una contaminación consta de una clave, un valor y un efecto. Como argumento aquí, se expresa como clave=valor:efecto.

For example:

kubectl taint node worker region=useast2:NoSchedule

The key must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores, up to 253 characters.

The value is optional. If given, it must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores, up to 63 characters.

The effect must be NoSchedule, PreferNoSchedule or NoExecute and Currently taint can only apply to nodes.

Toleration for a pod is specified in the PodSpec. A toleration "matches" a taint if the keys are the same and the effects are the same, and thus a pod with toleration would be able to schedule onto nodes.

La clave debe comenzar con una letra o un número y puede contener letras, números, guiones, puntos y guiones bajos, hasta 253 caracteres. El valor es opcional. Si se proporciona, debe comenzar con una letra o un número y puede contener letras, números, guiones, puntos y guiones bajos, hasta 63 caracteres. El efecto debe ser NoSchedule, PreferNoSchedule o NoExecute y actualmente la contaminación solo puede aplicarse a los nodos. La tolerancia para un pod se especifica en PodSpec. Una tolerancia "coincide" con una contaminación si las claves son las mismas y los efectos son los mismos y, por lo tanto, un módulo con tolerancia podría programarse en nodos

For example:

tolerations:  
- key: "region"  
  operator: "Equal"  
  value: "useast2"  
  effect: "NoSchedule"

# Kubernetes Security

Particularly in a distributed system like Kubernetes, security is a broad and complicated subject. It takes a constant effort to keep a cloud system secure. We must consider more than simply Kubernetes and consider the hardware, software, and configuration options for the complete environment as more and more apps migrate to the cloud. Hardware, firmware, and operating system binaries must be secured starting in the design phase itself.

Once the platform is hardened, the kube-apiserver has a list of considerations, tools, and settings to limit access and formalize access in an easy to understand manner.

Because Kubernetes is a network-intensive environment, it is crucial to secure the network using typical firewall techniques from outside the Kubernetes cluster and using pod-to-pod encryption, a NetworkPolicy and other measures from within the Kubernetes cluster.

Minimizing base images, insisting on container immutability, and static and runtime analysis of tools is also an important part of security which often begins with developers and is implemented in the CI/CD pipeline prior to an image being used in a production cluster.

Tools like AppArmor and SELinux should also be used to further protect the environment from malicious containers.

Security is more than just ‘settings and configuration’. It is an ongoing process of issue detection using intrusion detection tools and behavioral analytics. There needs to be an ongoing process of assessment, prevention, detection, and reaction following written and often updated policies.

Seguridad de Kubernetes Particularmente en un sistema distribuido como Kubernetes, la seguridad es un tema amplio y complicado. Se necesita un esfuerzo constante para mantener seguro un sistema en la nube. Debemos considerar algo más que Kubernetes y considerar el hardware, el software y las opciones de configuración para el entorno completo a medida que más y más aplicaciones migran a la nube. Los binarios de hardware, firmware y sistema operativo deben protegerse desde la propia fase de diseño. Una vez que la plataforma está reforzada, kube-apiserver tiene una lista de consideraciones, herramientas y configuraciones para limitar el acceso y formalizar el acceso de una manera fácil de entender. Debido a que Kubernetes es un entorno de red intensiva, es fundamental proteger la red utilizando técnicas de firewall típicas desde fuera del clúster de Kubernetes y utilizando cifrado de pod a pod, una política de red y otras medidas desde dentro del clúster de Kubernetes. Minimizar las imágenes base, insistir en la inmutabilidad de los contenedores y el análisis estático y en tiempo de ejecución de las herramientas también es una parte importante de la seguridad que a menudo comienza con los desarrolladores y se implementa en el proceso de CI/CD antes de que se utilice una imagen en un clúster de producción. También se deben utilizar herramientas como AppArmor y SELinux para proteger aún más el entorno de contenedores maliciosos. La seguridad es más que solo "ajustes y configuración". Es un proceso continuo de detección de problemas mediante herramientas de detección de intrusiones y análisis de comportamiento. Es necesario que haya un proceso continuo de evaluación, prevención, detección y reacción siguiendo políticas escritas y, a menudo, actualizadas.

## Accessing the API

To perform any action in a Kubernetes cluster, you need to access the API and go through three main steps.

### 1. Authentication (token)

The type of authentication used is defined in the kube-apiserver startup options. Below are four examples of a subset of configuration options that would need to be set depending on what choice of authentication mechanism you choose:

Accediendo a la API Para realizar cualquier acción en un clúster de Kubernetes, debe acceder a la API y seguir tres pasos principales. 1. Autenticación (token) El tipo de autenticación utilizado se define en las opciones de inicio de kube-apiserver. A continuación se muestran cuatro ejemplos de un subconjunto de opciones de configuración que deberían establecerse según el mecanismo de autenticación que elija:

--basic-auth-file

* --oidc-issuer-url
* --token-auth-file
* --authorization-webhook-config-file

To learn more about authentication, see the [official documentation](https://kubernetes.io/docs/admin/authentication/).

### 2. Authorization (RBAC)

RBAC stands for Role Based Access Control. All resources are modeled API objects in Kubernetes, from Pods to Namespaces. They also belong to API Groups such as core and apps. These resources allow operations such as Create, Read, Update, and Delete (CRUD), which we have been working with so far.

In YAML files, operations are referred to as verbs. We will add more API elements to these fundamental ones, which subsequently can be controlled by RBAC.

Rules are operations which can act upon an API group.

Roles are a group of rules which affect, or scope, a single namespace, whereas ClusterRoles have a scope of the entire cluster.

Each operation can act upon one of three subjects: User Accounts, which don’t exist as API objects; Service Accounts, and Groups, which are known as clusterrolebinding when using kubectl.

[RBAC](https://kubernetes.io/docs/admin/authorization/rbac) is then writing rules to allow or deny operations by users, roles or groups upon resources.

. Autorización (RBAC) RBAC significa Control de acceso basado en roles. Todos los recursos son objetos API modelados en Kubernetes, desde Pods hasta Namespaces. También pertenecen a grupos de API como core y apps. Estos recursos permiten operaciones como Crear, Leer, Actualizar y Eliminar (CRUD), con las que hemos estado trabajando hasta ahora. En los archivos YAML, las operaciones se denominan verbos. Agregaremos más elementos API a estos fundamentales, que posteriormente podrán ser controlados por RBAC. Las reglas son operaciones que pueden actuar sobre un grupo de API. Los roles son un grupo de reglas que afectan o abarcan un único espacio de nombres, mientras que ClusterRoles tienen un alcance de todo el clúster. Cada operación puede actuar sobre uno de tres temas: Cuentas de usuario, que no existen como objetos API; Cuentas de servicio y grupos, que se conocen como clusterrolebinding cuando se usa kubectl. Luego, RBAC escribe reglas para permitir o denegar operaciones por parte de usuarios, roles o grupos sobre recursos.

### Admission Controllers

The last step in letting an API request into Kubernetes is Admission Control.

Admission controllers are pieces of software that can access the content of the objects being created by the requests. They can modify the content or validate it, and potentially deny the request.

Admission controllers are needed for certain features to work properly. Controllers have been added as Kubernetes has matured. As of the v1.12 release, the kube-apiserver uses a compiled-in set of controllers. Instead of passing a list, we can enable or disable particular controllers. If you want to use a controller not available by default, you would need to download source and compile.

The first controller is Initializers, which will allow dynamic modification of the API request, providing great flexibility. Each admission controller functionality is explained in the documentation. For example, the ResourceQuota controller will ensure that the object created does not violate any of the existing quotas.

Also, tools like the [Open Policy Agent](https://www.openpolicyagent.org/) can be used to manage admission control externally. The Open Policy Agent (OPA, pronounced “oh-pa”) is an open source, general-purpose policy engine that unifies policy enforcement across the stack. OPA provides a high-level declarative language that lets you specify policy as code and simple APIs to offload policy decision-making from your software. You can use OPA to enforce policies in microservices, Kubernetes, CI/CD pipelines, API gateways, and more.

Controladores de admisión El último paso para permitir que una solicitud de API ingrese a Kubernetes es el Control de admisión. Los controladores de admisión son piezas de software que pueden acceder al contenido de los objetos creados por las solicitudes. Pueden modificar el contenido o validarlo y potencialmente rechazar la solicitud. Se necesitan controladores de admisión para que ciertas funciones funcionen correctamente. Se han agregado controladores a medida que Kubernetes maduró. A partir de la versión v1.12, kube-apiserver utiliza un conjunto de controladores compilados. En lugar de pasar una lista, podemos habilitar o deshabilitar controladores particulares. Si desea utilizar un controlador que no está disponible de forma predeterminada, deberá descargar el código fuente y compilarlo. El primer controlador es Inicializadores, que permitirá la modificación dinámica de la solicitud API, brindando una gran flexibilidad. Cada funcionalidad del controlador de admisión se explica en la documentación. Por ejemplo, el controlador ResourceQuota se asegurará de que el objeto creado no viole ninguna de las cuotas existentes. Además, se pueden utilizar herramientas como Open Policy Agent para gestionar el control de admisión de forma externa. El Open Policy Agent (OPA, pronunciado “oh-pa”) es un motor de políticas de código abierto y de propósito general que unifica la aplicación de políticas en toda la pila. OPA proporciona un lenguaje declarativo de alto nivel que le permite especificar políticas como código y API simples para descargar la toma de decisiones de políticas desde su software. Puede utilizar OPA para aplicar políticas en microservicios, Kubernetes, canalizaciones de CI/CD, puertas de enlace API y más.

# Additional Resources

Learn more about...

## Differences between Containers and Pods

* [What are Kubernetes Pods Anyway?](https://www.ianlewis.org/en/what-are-kubernetes-pods-anyway), by Ian Lewis (2017)
* [Containers vs. Pods - Taking a Deeper Look](https://iximiuz.com/en/posts/containers-vs-pods/), by Ivan Velichko (2021)

## kubectl tips & tricks

* [kubectl Cheat Sheet](https://kubernetes.io/docs/reference/kubectl/cheatsheet/)

## Storage and CSI in Kubernetes

* [Container Storage Interface (CSI) for Kubernetes GA](https://kubernetes.io/blog/2019/01/15/container-storage-interface-ga/), by Saad Ali (2019)
* [Kubernetes Storage: Ephemeral Inline Volumes, Volume Cloning, Snapshots and more!](https://www.inovex.de/de/blog/kubernetes-storage-volume-cloning-ephemeral-inline-volumes-snapshots/), by Henning Eggers (2020)

## Autoscaling in Kubernetes

* [Architecting Kubernetes clusters - choosing the best autoscaling strategy](https://learnk8s.io/kubernetes-autoscaling-strategies), by Daniele Polencic (2021)

preguntas y respuestas

# Question 5.1

Kubernetes objects can be described in a data-serialization language called \_\_\_\_\_\_\_\_\_\_\_\_\_.

Correct Answer

* A. HTML
* B. PHP
* C. YAML

Your Answer:

Correct

* D. Python

# Question 5.2

Which of the following is not a required field in a Kubernetes object?

Incorrect Answer

* A. kind

Your Answer:

Incorrect

* B. spec
* **C. containers**

**Correct Answer**

* D. metadata

# Question 5.3

What is the name of the official Kubernetes command line interface?

Correct Answer

* A. containerctl
* **B. kubectl**

**Your Answer:**

Correct

* C. kube-tool
* D. podctl

# Question 5.4

What is the smallest deployable compute unit of Kubernetes?

Correct Answer

* A. Container
* B. Deployment
* **C. Pod**

**Your Answer:**

**Correct**

* D. ReplicaSet

# Question 5.5

How do you configure a container to run in a pod before your main container(s)?

Correct Answer

* A. bootstrapContainer
* **B. initContainer**

**Your Answer:**

**Correct**

* C. priorityContainer
* D. startContainer

Which workload object does not exist in Kubernetes?

Correct Answer

* A. ReplicaSet
* B. Deployment
* **C. ApplicationSet**

**Your Answer:**

Correct

* D. StatefulSet

Which service types exist in Kubernetes?

Correct Answer

* A. NodePort
* B. ClusterIP
* C. LoadBalancer
* D. ExternalName
* **E. All of the above**

# Question 5.8

Ingress objects can be used to configure HTTP(S) routing rules. True or False?

* A.

True

Correct Answer

* B.

False

# Chapter Overview

Deploying an application on any kind of platform has gone a long way over the years. In the beginning, applications maybe would be executed on the same machine they were written on, later transported by physical medium (floppy disk, usb-stick, CD), and now we're checking in code on a server, that builds and application, put it in a container and directly deploy it to a platform like Kubernetes.

The way we deliver our applications is heavily influenced by the DevOps movement that had its breakthrough in the late 2000s. The DevOps movement is a culture change that brought a lot of new methods and technologies to the industry.

Arguably the most important change was the automation of the deployment process, which allowed very fast, more frequent and higher quality software deployments. In this chapter, we will learn about these methods also known as Continuous Integration/Continuous Delivery (CI/CD) and how they further advanced in new tools and practices like GitOps.

Descripción general del capítulo La implementación de una aplicación en cualquier tipo de plataforma ha recorrido un largo camino a lo largo de los años. Al principio, las aplicaciones tal vez se ejecutaban en la misma máquina en la que fueron escritas, luego se transportaban en un medio físico (disquete, memoria USB, CD), y ahora estamos verificando el código en un servidor, que construye y aplica. , colóquelo en un contenedor e impleméntelo directamente en una plataforma como Kubernetes. La forma en que entregamos nuestras aplicaciones está fuertemente influenciada por el movimiento DevOps que tuvo su gran avance a finales de la década de 2000. El movimiento DevOps es un cambio cultural que trajo muchos métodos y tecnologías nuevos a la industria. Podría decirse que el cambio más importante fue la automatización del proceso de implementación, que permitió implementaciones de software muy rápidas, más frecuentes y de mayor calidad. En este capítulo, aprenderemos sobre estos métodos, también conocidos como Integración Continua/Entrega Continua (CI/CD) y cómo avanzaron en nuevas herramientas y prácticas como GitOps.

# Learning Objectives

By the end of this chapter, you should be able to:

* Discuss the importance of automation in integration and delivery of applications
* Understand the need for Git and version control systems
* Explain what a CI/CD pipeline is
* Discuss the idea of Infrastructure as Code (IaS)
* Discuss the principles of GitOps and how it integrates with Kubernetes

bjetivos de aprendizaje Al final de este capítulo, debería poder: Discutir la importancia de la automatización en la integración y entrega de aplicaciones. Comprender la necesidad de Git y los sistemas de control de versiones. Explicar qué es una canalización de CI/CD Discutir la idea de Infraestructura como Código (IaS) Analice los principios de GitOps y cómo se integra con Kubernetes.

# Application Delivery Fundamentals

Every application starts its lifecycle with code that is written. Source code is not only the basis of an application, but also intellectual property and therefore the capital of most companies or individuals. We figured out a long time ago that the best way to manage source code is a version control system.

In 2005 Linus Torvalds created Git, which is the standard version control system that almost everybody is using today. Git is a decentralized system that can be used to track changes in your source code. In essence, Git can work with copies of the code, so called in branches or forks where you can work in, before your changes get merged back in a main branch.

Make sure to check out [this web page](https://git-scm.com/) to learn more about git since it's a powerful tool that is an industry standard and used daily by almost all developers and administrators.

With your source code in check, the next step before delivering your application is to build it, which could also include the building of a container image, as described in the Container Orchestration chapter.

To ensure high quality of your application, the next step should be extensive and automatic testing of the app to make sure all functionality is still in place after someone makes changes.

The last step is delivering the application to the platform it should run on. If your target platform is Kubernetes, you can write a YAML file to deploy your application while your newly built container image can be pushed to a container registry, where Kubernetes will download it for you.

Today source code is not the only thing managed in a version control system.To make full use of cloud resources, the principle of [Infrastructure as Code (IaC)](https://en.wikipedia.org/wiki/Infrastructure_as_code) became popular. Instead of installing infrastructure manually you describe it in files and use the cloud vendors API to set up your infrastructure. This allows developers to be more involved in the setup of the infrastructure.

undamentos de entrega de aplicaciones Cada aplicación comienza su ciclo de vida con código escrito. El código fuente no es sólo la base de una aplicación, sino también propiedad intelectual y por tanto el capital de la mayoría de empresas o particulares. Hace mucho tiempo descubrimos que la mejor manera de administrar el código fuente es un sistema de control de versiones. En 2005, Linus Torvalds creó Git, que es el sistema de control de versiones estándar que casi todo el mundo utiliza hoy en día. Git es un sistema descentralizado que se puede utilizar para rastrear cambios en su código fuente. En esencia, Git puede trabajar con copias del código, llamadas así en ramas o bifurcaciones donde puedes trabajar, antes de que tus cambios se vuelvan a fusionar en una rama principal. Asegúrese de visitar esta página web para obtener más información sobre git, ya que es una herramienta poderosa que es un estándar de la industria y que casi todos los desarrolladores y administradores utilizan a diario. Con su código fuente bajo control, el siguiente paso antes de entregar su aplicación es compilarla, lo que también podría incluir la creación de una imagen de contenedor, como se describe en el capítulo Orquestación de contenedores. Para garantizar una alta calidad de su aplicación, el siguiente paso debe ser una prueba exhaustiva y automática de la aplicación para asegurarse de que todas las funciones sigan funcionando después de que alguien realice cambios. El último paso es entregar la aplicación a la plataforma en la que debería ejecutarse. Si su plataforma de destino es Kubernetes, puede escribir un archivo YAML para implementar su aplicación mientras su imagen de contenedor recién creada se puede enviar a un registro de contenedor, donde Kubernetes la descargará por usted. Hoy en día, el código fuente no es lo único que se gestiona en un sistema de control de versiones. Para aprovechar al máximo los recursos de la nube, se hizo popular el principio de Infraestructura como Código (IaC). En lugar de instalar la infraestructura manualmente, la describe en archivos y utiliza la API de los proveedores de la nube para configurar su infraestructura. Esto permite a los desarrolladores participar más en la configuración de la infraestructura.

# CI/CD

With services getting smaller and deployments getting more frequent, a logical and important step was the automation of the deployment process. The DevOps movement has highlighted the importance of frequent and rapid deployments. In traditional setups, a deployment would include developers and administrators, a lot of error-prone manual steps and the constant fear that something would break.

Automation is the key to overcoming these barriers, and today we know and use the principles of Continuous Integration/Continuous Delivery (CI/CD), which describe the different steps in the deployment of an application, configuration or even infrastructure.

Continuous Integration is the first part of the process and describes the permanent building and testing of the written code. High automation and usage of version control allows multiple developers and teams to work on the same code base.

Continuous Delivery is the second part of the process and automates the deployment of the pre-built software. In cloud environments, you will often see that software is deployed to Development or Staging environments, before it gets released and delivered to a production system.

To automate the whole workflow, you can use a CI/CD pipeline, which is actually nothing more than the scripted form of all the steps involved, running on a server or even in a container. Pipelines should be integrated with a version control system that manages changes to the code base.

Every time a new revision of your code is ready to be deployed, the pipeline starts to execute scripts that build your code, run tests, deploy them to servers and even perform security and compliance checks.

Besides the generic scripting of the pipeline steps, modern CI/CD tools have a lot more functionality like direct interaction and feedback from a system like Kubernetes.

Popular CI/CD tools include:

CI/CD Con los servicios cada vez más pequeños y las implementaciones cada vez más frecuentes, un paso lógico e importante fue la automatización del proceso de implementación. El movimiento DevOps ha resaltado la importancia de implementaciones frecuentes y rápidas. En las configuraciones tradicionales, una implementación incluiría desarrolladores y administradores, muchos pasos manuales propensos a errores y el temor constante de que algo se rompiera. La automatización es la clave para superar estas barreras, y hoy conocemos y utilizamos los principios de Integración Continua/Entrega Continua (CI/CD), que describen los diferentes pasos en el despliegue de una aplicación, configuración o incluso infraestructura. La integración continua es la primera parte del proceso y describe la construcción y prueba permanente del código escrito. La alta automatización y el uso del control de versiones permiten que varios desarrolladores y equipos trabajen en la misma base de código. La entrega continua es la segunda parte del proceso y automatiza la implementación del software prediseñado. En entornos de nube, a menudo verá que el software se implementa en entornos de desarrollo o de prueba, antes de ser lanzado y entregado a un sistema de producción. Para automatizar todo el flujo de trabajo, puede utilizar una canalización de CI/CD, que en realidad no es más que la forma escrita de todos los pasos involucrados, ejecutándose en un servidor o incluso en un contenedor. Las canalizaciones deben integrarse con un sistema de control de versiones que gestione los cambios en el código base. Cada vez que una nueva revisión de su código está lista para implementarse, la canalización comienza a ejecutar scripts que crean su código, ejecutan pruebas, las implementan en servidores e incluso realizan comprobaciones de seguridad y cumplimiento. Además de las secuencias de comandos genéricas de los pasos del proceso, las herramientas CI/CD modernas tienen muchas más funciones, como interacción directa y retroalimentación de un sistema como Kubernetes. Las herramientas CI/CD populares incluyen:

# GitOps

Infrastructure as Code was a real revolution in increasing the quality and speed of providing infrastructure, and it works so well that today, configuration, network, policies, or security can be described as code, and often even live in the same repository as the software.

GitOps takes the idea of Git as the single source of truth a step further and integrates the provisioning and change process of infrastructure with version control operations.

If code was branched and should be merged back into the main branch, you can create a merge or pull request that can be reviewed by other developers before actually merging it. This has been a best practice for a long time in software development, and also includes running a CI pipeline for every change that should be made. In GitOps, these merge requests are used to manage infrastructure changes.

Spinnaker

• GitLab

• Jenkins

• Jenkins X

• Tekton CD

• ArgoCD

To get more insight into DevOps, Site Reliability Engineering and Infrastructure as Code, we highly recommend you take Introduction to DevOps and Site Reliability Engineering (LFS162x), a free course on edX.

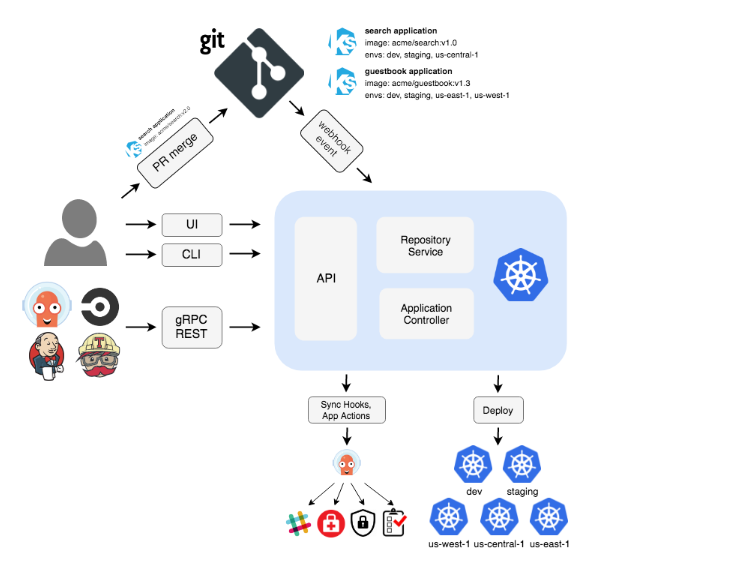
There are two different approaches how a CI/CD pipeline can implement the changes you want to make:

* Push-based  
  The pipeline is started and runs tools that make the changes in the platform. Changes can be triggered by a commit or merge request.
* Pull-based  
  An agent watches the git repository for changes and compares the definition in the repository with the actual running state. If changes are detected, the agent applies the changes to the infrastructure.

Two examples of popular GitOps frameworks that use the pull-based approach are [Flux](https://fluxcd.io/) and [ArgoCD](https://argo-cd.readthedocs.io/). ArgoCD is implemented as a Kubernetes controller, while Flux is built with the GitOps Toolkit, a set of APIs and controllers that can be used to extend Flux, or even build a custom delivery platform.

The ArgoCD architecture gives a good overview of how GitOps can be implemented.

GitOps La infraestructura como código fue una verdadera revolución en el aumento de la calidad y la velocidad de la provisión de infraestructura, y funciona tan bien que hoy en día, la configuración, la red, las políticas o la seguridad pueden describirse como código y, a menudo, incluso vivir en el mismo repositorio que el software. . GitOps lleva la idea de Git como la única fuente de verdad un paso más allá e integra el proceso de aprovisionamiento y cambio de infraestructura con operaciones de control de versiones. Si el código se bifurcó y debe fusionarse nuevamente en la rama principal, puede crear una solicitud de fusión o extracción que otros desarrolladores puedan revisar antes de fusionarlo. Esta ha sido una práctica recomendada durante mucho tiempo en el desarrollo de software y también incluye ejecutar un canal de CI para cada cambio que se debe realizar. En GitOps, estas solicitudes de fusión se utilizan para gestionar cambios de infraestructura. Hay dos enfoques diferentes sobre cómo una canalización de CI/CD puede implementar los cambios que desea realizar: Basado en push La canalización se inicia y ejecuta herramientas que realizan los cambios en la plataforma. Los cambios pueden desencadenarse mediante una solicitud de confirmación o fusión. Basado en extracción Un agente observa el repositorio de git en busca de cambios y compara la definición en el repositorio con el estado de ejecución real. Si se detectan cambios, el agente los aplica a la infraestructura. Dos ejemplos de marcos GitOps populares que utilizan el enfoque basado en extracción son Flux y ArgoCD. ArgoCD se implementa como un controlador de Kubernetes, mientras que Flux se construye con GitOps Toolkit, un conjunto de API y controladores que se pueden usar para ampliar Flux o incluso crear una plataforma de entrega personalizada. La arquitectura ArgoCD ofrece una buena descripción general de cómo se puede implementar GitOps.

X

# Additional Resources

Learn more about...

## 10 Deploys Per Day - Start of the DevOps movement at Flickr

* [Velocity 09: John Allspaw and Paul Hammond, "10+ Deploys Per Day"](https://www.youtube.com/watch?v=LdOe18KhtT4)
* [10+ Deploys Per Day: Dev and Ops Cooperation at Flickr](https://www.slideshare.net/jallspaw/10-deploys-per-day-dev-and-ops-cooperation-at-flickr), by John Allspaw and Paul Hammond

## Learn git in a playful way

* [Oh My Git! An open source game about learning Git!](https://ohmygit.org/)
* [Learn Git Branching](https://learngitbranching.js.org/)

## Infrastructure as Code

* [Delivering Cloud Native Infrastructure as Code](https://www.pulumi.com/why-pulumi/delivering-cloud-native-infrastructure-as-code/)
* [Unlocking the Cloud Operating Model: Provisioning](https://www.hashicorp.com/resources/unlocking-the-cloud-operating-model-provisioning)

## Beginners guide to CI/CD

# Question 6.1

What is the name of a popular version control tool?

Incorrect Answer

* A. Docker
* B. git

Correct Answer

* C. Kubernetes

Your Answer:

Incorrect

* D. containerd

# Question 6.2

Version control systems can be used as a basis to manage \_\_\_\_\_\_\_\_\_\_.

Correct Answer

* A. Source code
* B. Configuration
* C. Infrastructure
* D. All of the above

# Question 6.3

What does CI/CD stand for?

Correct Answer

* A. Continuous Installation/Continuous Downtime
* B. Continuous Integration/Continuous Delivery

Your Answer:

Correct

* C. Container Interface/Container Deployment
* D. Continuous Infrastructure/Continuous Deployment

# Question 6.4

How can you automate the build, test and deployment of an application?

Correct Answer

* A. CI/CD pipeline

Your Answer:

Correct

* B. CI/CD branch
* C. CI/CD repository
* D. CI/CD network

# Question 6.5

Flux is built with the \_\_\_\_\_\_\_\_\_\_\_\_.

Correct Answer

* A. Ci/CD Toolkit
* B. Cluster Toolkit
* C. GitOps Toolkit

Your Answer:

Correct

* D. Infrastructure Toolkit

# Question 6.6

Flux and ArgoCD are popular GitOps tools. They use a push-based approach. True or False?

* A.

True

Your Answer:

Incorrect

* B.

False

Correct Answer

# Learning Objectives

By the end of this chapter, you should be able to:

* Explain why observability is a key discipline of cloud computing
* Discuss metrics, logs and traces
* Understand how to show logs of containerized applications
* Explain how Prometheus can be used to collect and store metrics
* Understand how to optimize cloud costs

bjetivos de aprendizaje Al final de este capítulo, debería poder: Explique por qué la observabilidad es una disciplina clave de la computación en la nube. Analizar métricas, registros y seguimientos Comprender cómo mostrar registros de aplicaciones en contenedores Explicar cómo se puede utilizar Prometheus para recopilar y almacenar métricas. Comprenda cómo optimizar los costos de la nube

# Observability

Observability is often used synonymously with monitoring, but monitoring is only one of the subcategories of cloud native observability and does not do justice to its scope. The term observability is closely related to the [control theory](https://en.wikipedia.org/wiki/Control_theory) which deals with behavior of dynamic systems. In essence, the control theory describes how external outputs of systems can be measured to manipulate the behavior of the system.

A popular example is the cruise control system of a car. You set the desired speed of the car, which is constantly measured and which can be observed by a person with the speedometer. In order to maintain the speed in changed conditions, e.g. when driving up a mountain, the power of the motor must be adapted to maintain the speed.

In IT systems the same principle can be applied for autoscaling. You set the desired utilization of the system and trigger scaling events based on the load of the system.

Automating your systems in such a way can be very challenging and is not the most important usage of observability. When we deal with container orchestration and microservices, the biggest challenge is keeping track of the systems, how they interact with each other and how they behave when under load or in an error state.

Observability should give answers to questions like:

* Is the system stable or does it change its state when manipulated?
* Is the system sensitive to change, e.g. if some services have high latency?
* Do certain metrics in the system exceed their limits?
* Why does a request to the system fail?
* Are there any bottlenecks in the system?

The higher goal of observability is to allow analysis of the collected data. This helps to get a better understanding of the system and react to error states. This more technological side of things is closely related to modern agile software development that also uses feedback loops in which you analyze the behavior of software and adapt it constantly based on the outcome.

Observabilidad La observabilidad se utiliza a menudo como sinónimo de monitoreo, pero el monitoreo es solo una de las subcategorías de la observabilidad nativa de la nube y no hace justicia a su alcance. El término observabilidad está estrechamente relacionado con la teoría del control que se ocupa del comportamiento de los sistemas dinámicos. En esencia, la teoría del control describe cómo se pueden medir los resultados externos de los sistemas para manipular el comportamiento del sistema. Un ejemplo popular es el sistema de control de crucero de un automóvil. Usted establece la velocidad deseada del vehículo, que se mide constantemente y que una persona puede observar con el velocímetro. Para mantener la velocidad en condiciones cambiantes, p. Al subir una montaña, la potencia del motor debe adaptarse para mantener la velocidad. En los sistemas de TI se puede aplicar el mismo principio para el escalado automático. Usted establece la utilización deseada del sistema y activa eventos de escalado en función de la carga del sistema. Automatizar sus sistemas de esta manera puede ser un gran desafío y no es el uso más importante de la observabilidad. Cuando nos ocupamos de la orquestación de contenedores y los microservicios, el mayor desafío es realizar un seguimiento de los sistemas, cómo interactúan entre sí y cómo se comportan cuando están bajo carga o en estado de error. La observabilidad debería dar respuestas a preguntas como: ¿Es el sistema estable o cambia de estado cuando se manipula? ¿Es el sistema sensible al cambio, p.e. ¿Si algunos servicios tienen alta latencia? ¿Ciertas métricas del sistema exceden sus límites? ¿Por qué falla una solicitud al sistema? ¿Existen cuellos de botella en el sistema? El objetivo superior de la observabilidad es permitir el análisis de los datos recopilados. Esto ayuda a comprender mejor el sistema y reaccionar ante estados de error. Este lado más tecnológico de las cosas está estrechamente relacionado con el desarrollo de software ágil y moderno que también utiliza ciclos de retroalimentación en los que se analiza el comportamiento del software y se adapta constantemente en función del resultado.

# Telemetry

The term telemetry has Greek roots and means remote or distance (tele) and measuring (metry). Measuring and collecting data points and then transferring it to another system is of course not exclusive to cloud native or even IT systems. A good example would be a weather station with a data-logger that measures the temperature, humidity, wind speed and more at a certain point and then transmits it to another system that can process and display the data.

In container systems, each and every application should have tools built in that generate information data, which is then collected and transferred in a centralized system. The data can be divided into three categories.

Click on each card to flip it and learn more about each of the three data categories

elemetria El término telemetría tiene raíces griegas y significa remoto o distancia (tele) y medición (metría). Por supuesto, medir y recopilar puntos de datos y luego transferirlos a otro sistema no es exclusivo de los sistemas nativos de la nube o incluso de TI. Un buen ejemplo sería una estación meteorológica con un registrador de datos que mide la temperatura, la humedad, la velocidad del viento y más en un punto determinado y luego los transmite a otro sistema que puede procesar y mostrar los datos. En los sistemas de contenedores, todas y cada una de las aplicaciones deben tener herramientas integradas que generen datos de información, que luego se recopilan y transfieren en un sistema centralizado. Los datos se pueden dividir en tres categorías. Haga clic en cada tarjeta para darle la vuelta y obtener más información sobre cada una de las tres categorías de datos.

Log

These are messages that are emitted from an application when errors, warnings or debug information should be presented. A simple log entry could be the start and end of a specific task that the application performed.

Estos son mensajes que se emiten desde una aplicación cuando se deben presentar errores, advertencias o información de depuración. Una simple entrada de registro podría ser el inicio y el final de una tarea específica que realizó la aplicación.

Metric

Metrics are quantitative measurements taken over time. This could be the number of requests or an error rate.

Las métricas son medidas cuantitativas tomadas a lo largo del tiempo. Esta podría ser la cantidad de solicitudes o una tasa de error.

Trace

They track the progression of a request while it’s passing through the system. Traces are used in a distributed system that can provide information about when a request was processed by a service and how long it took.

Realizan un seguimiento de la progresión de una solicitud mientras pasa por el sistema. Los seguimientos se utilizan en un sistema distribuido que puede proporcionar información sobre cuándo un servicio procesó una solicitud y cuánto tiempo tardó.

A lot of traditional systems didn’t even bother to transmit the data like logs to a centralized system and to view the logs you had to connect to the system and read it directly from files.

In a distributed system with hundreds or thousands of services, this would mean a lot of effort and troubleshooting would be very time consuming.

Muchos sistemas tradicionales ni siquiera se molestaban en transmitir datos como registros a un sistema centralizado y para ver los registros había que conectarse al sistema y leerlos directamente desde los archivos. En un sistema distribuido con cientos o miles de servicios, esto significaría mucho esfuerzo y la resolución de problemas llevaría mucho tiempo.

# Logging

Today, application frameworks and programming languages come with extensive logging tools built-in, which makes it very easy to log to a file with different log levels based on the severity of the log message.

The documentation for the Python programming language provides the following [example](https://docs.python.org/3/howto/logging.html" \l "logging-to-a-file):

import logging  
logging.basicConfig(filename='example.log', encoding='utf-8', level=logging.DEBUG)  
logging.debug('This message should go to the log file')  
logging.info('So should this')  
logging.warning('And this, too')  
logging.error('And non-ASCII stuff, too, like Øresund and Malmö')

Unix and Linux programs provide three I/O streams from which two can be used to output logs from a container:

* standard input (stdin): Input to a program e.g. via keyboard
* standard output (stdout): The output a program writes on the screen
* standard error (stderr): Errors that a program writes on the screen

If you want to learn more about I/O streams and how they originated, make sure to visit the [stdin(3) - Linux manual page](https://man7.org/linux/man-pages/man3/stdout.3.html).

Command line tools like docker, kubectl or podman provide a command to show the logs of containerized processes if you let them log directly to the console or to /dev/stdout and /dev/stderr.

An example to view the logs of a container named nginx:

$ docker logs nginx

/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration  
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/  
/docker-entrypoint.sh: Launching  
/docker-entrypoint.d/10-listen-on-ipv6-by-default.sh  
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf  
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf  
/docker-entrypoint.sh: Launching  
/docker-entrypoint.d/20-envsubst-on-templates.sh  
/docker-entrypoint.sh: Launching  
/docker-entrypoint.d/30-tune-worker-processes.sh  
/docker-entrypoint.sh: Configuration complete; ready for start up  
2021/10/20 13:22:44 [notice] 1#1: using the "epoll" event method  
2021/10/20 13:22:44 [notice] 1#1: nginx/1.21.3

To stream the logs in real time, you could add the -f parameter to the command. Kubernetes provides the same functionality with the kubectl command line tool. The documentation of the **[kubectl logs](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands" \l "logs)** command provides some examples:

# Return snapshot logs from pod nginx with only one container  
kubectl logs nginx

# Return snapshot of previous terminated ruby container logs from pod web-1  
kubectl logs -p -c ruby web-1

# Begin streaming the logs of the ruby container in pod web-1  
kubectl logs -f -c ruby web-1

# Display only the most recent 20 lines of output in pod nginx  
kubectl logs --tail=20 nginx

# Show all logs from pod nginx written in the last hour  
kubectl logs --since=1h nginx

These methods allow for a direct interaction with a single container. But to manage the huge amount of data, these logs need to be shipped to a system that stores the logs. To ship the logs, different methods can be used:

* Node-level logging  
  The most efficient way to collect logs. An administrator configures a log shipping tool that collects logs and ships them to a central store.
* Logging via sidecar container  
  The application has a sidecar container that collects the logs and ships them to a central store.
* Application-level logging  
  The application pushes the logs directly to the central store. While this seems very convenient at first, it requires configuring the logging adapter in every application that runs in a cluster.

There are several tools to choose from to ship and store the logs. The first two methods can be done by tools like [**fluentd**](https://www.fluentd.org/) or [**filebeat**](https://www.elastic.co/beats/filebeat).

Popular choices to store logs are [**OpenSearch**](https://opensearch.org/) or [**Grafana Loki**](https://grafana.com/oss/loki/). To find more datastores, you can visit the [**fluentd documentation on possible log targets**](https://www.fluentd.org/dataoutputs).

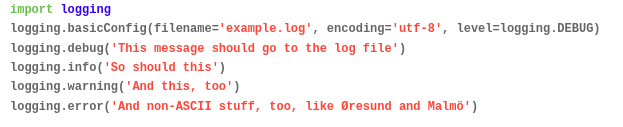
To make logs easy to process and searchable make sure you log in a structured format like JSON instead of plaintext. The major cloud vendors provide good documentation on the importance of structured logging and how to implement it:

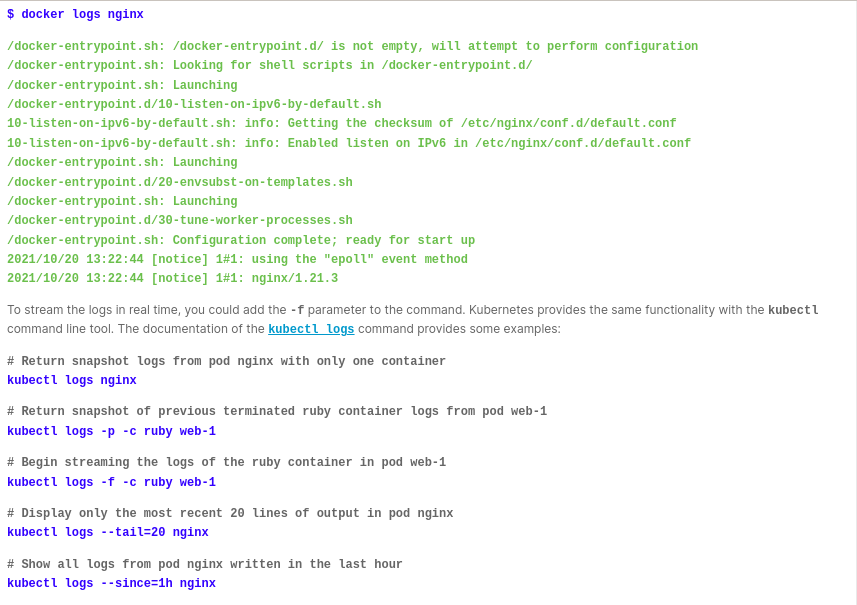
* [**Structured logging (Google Cloud documentation)**](https://cloud.google.com/logging/docs/structured-logging)
* **[Structured logging (Microsoft Azure documentation)](https://docs.microsoft.com/en-us/azure/architecture/example-scenario/logging/unified-logging" \l "structured-logging)**

Inicio sesión Hoy en día, los marcos de aplicaciones y los lenguajes de programación vienen con amplias herramientas de registro integradas, lo que hace que sea muy fácil iniciar sesión en un archivo con diferentes niveles de registro según la gravedad del mensaje de registro. La documentación del lenguaje de programación Python proporciona el siguiente ejemplo:

os programas Unix y Linux proporcionan tres flujos de E/S de los cuales dos pueden usarse para generar registros desde un contenedor: entrada estándar (stdin): entrada a un programa, p. a través del teclado Salida estándar (stdout): La salida que un programa escribe en la pantalla. error estándar (stderr): Errores que un programa escribe en la pantalla Si desea obtener más información sobre los flujos de E/S y cómo se originaron, asegúrese de visitar la página del manual stdin(3) - Linux. Las herramientas de línea de comandos como Docker, Kubectl o Podman proporcionan un comando para mostrar los registros de los procesos en contenedores si les permite iniciar sesión directamente en la consola o en /dev/stdout y /dev/stderr. Un ejemplo para ver los registros de un contenedor llamado nginx:

[TRADUCIR ESTA PÁGINA](chrome-extension://aapbdbdomjkkjkaonfhkkikfgjllcleb/popup.html) [ABRIR EN GOOGLE TRADUCTOR](https://translate.google.com/?source=gtx_m" \l "en/es/Unix and Linux programs provide three I%2FO streams from which two can be used to output logs from a container%3A%0A%0Astandard input (stdin)%3A Input to a program e.g. via keyboard%0Astandard output (stdout)%3A The output a program writes on the screen%0Astandard error (stderr)%3A Errors that a program writes on the screen%0AIf you want to learn more about I%2FO streams and how they originated%2C make sure to visit the stdin(3) - Linux manual page.%0A%0ACommand line tools like docker%2C kubectl or podman provide a command to show the logs of containerized processes if you let them log directly to the console or to %2Fdev%2Fstdout and %2Fdev%2Fstderr.%0A%0AAn example to view the logs of a container named nginx%3A)

x

x

# Logging

Today, application frameworks and programming languages come with extensive logging tools built-in, which makes it very easy to log to a file with different log levels based on the severity of the log message.

The documentation for the Python programming language provides the following [example](https://docs.python.org/3/howto/logging.html" \l "logging-to-a-file):

import logging  
logging.basicConfig(filename='example.log', encoding='utf-8', level=logging.DEBUG)  
logging.debug('This message should go to the log file')  
logging.info('So should this')  
logging.warning('And this, too')  
logging.error('And non-ASCII stuff, too, like Øresund and Malmö')

Unix and Linux programs provide three I/O streams from which two can be used to output logs from a container:

* standard input (stdin): Input to a program e.g. via keyboard
* standard output (stdout): The output a program writes on the screen
* standard error (stderr): Errors that a program writes on the screen

If you want to learn more about I/O streams and how they originated, make sure to visit the [stdin(3) - Linux manual page](https://man7.org/linux/man-pages/man3/stdout.3.html).

Command line tools like docker, kubectl or podman provide a command to show the logs of containerized processes if you let them log directly to the console or to /dev/stdout and /dev/stderr.

An example to view the logs of a container named nginx:

$ docker logs nginx

/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration  
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/  
/docker-entrypoint.sh: Launching  
/docker-entrypoint.d/10-listen-on-ipv6-by-default.sh  
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf  
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf  
/docker-entrypoint.sh: Launching  
/docker-entrypoint.d/20-envsubst-on-templates.sh  
/docker-entrypoint.sh: Launching  
/docker-entrypoint.d/30-tune-worker-processes.sh  
/docker-entrypoint.sh: Configuration complete; ready for start up  
2021/10/20 13:22:44 [notice] 1#1: using the "epoll" event method  
2021/10/20 13:22:44 [notice] 1#1: nginx/1.21.3

To stream the logs in real time, you could add the -f parameter to the command. Kubernetes provides the same functionality with the kubectl command line tool. The documentation of the **[kubectl logs](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands" \l "logs)** command provides some examples:

# Return snapshot logs from pod nginx with only one container  
kubectl logs nginx

# Return snapshot of previous terminated ruby container logs from pod web-1  
kubectl logs -p -c ruby web-1

# Begin streaming the logs of the ruby container in pod web-1  
kubectl logs -f -c ruby web-1

# Display only the most recent 20 lines of output in pod nginx  
kubectl logs --tail=20 nginx

# Show all logs from pod nginx written in the last hour  
kubectl logs --since=1h nginx

These methods allow for a direct interaction with a single container. But to manage the huge amount of data, these logs need to be shipped to a system that stores the logs. To ship the logs, different methods can be used:

* Node-level logging  
  The most efficient way to collect logs. An administrator configures a log shipping tool that collects logs and ships them to a central store.
* Logging via sidecar container  
  The application has a sidecar container that collects the logs and ships them to a central store.
* Application-level logging  
  The application pushes the logs directly to the central store. While this seems very convenient at first, it requires configuring the logging adapter in every application that runs in a cluster.

There are several tools to choose from to ship and store the logs. The first two methods can be done by tools like [**fluentd**](https://www.fluentd.org/) or [**filebeat**](https://www.elastic.co/beats/filebeat).

Popular choices to store logs are [**OpenSearch**](https://opensearch.org/) or [**Grafana Loki**](https://grafana.com/oss/loki/). To find more datastores, you can visit the [**fluentd documentation on possible log targets**](https://www.fluentd.org/dataoutputs).

To make logs easy to process and searchable make sure you log in a structured format like JSON instead of plaintext. The major cloud vendors provide good documentation on the importance of structured logging and how to implement it:

Estos métodos permiten una interacción directa con un solo contenedor. Pero para gestionar la enorme cantidad de datos, estos registros deben enviarse a un sistema que los almacene. Para enviar los troncos se pueden utilizar diferentes métodos: Registro a nivel de nodo La forma más eficiente de recopilar registros. Un administrador configura una herramienta de envío de registros que recopila registros y los envía a un almacén central. Registro a través de contenedor sidecar La aplicación tiene un contenedor lateral que recoge los registros y los envía a un almacén central. Registro a nivel de aplicación La aplicación envía los registros directamente al almacén central. Si bien esto parece muy conveniente al principio, requiere configurar el adaptador de registro en cada aplicación que se ejecuta en un clúster. Hay varias herramientas para elegir para enviar y almacenar los registros. Los dos primeros métodos se pueden realizar con herramientas como fluentd o filebeat. Las opciones populares para almacenar registros son OpenSearch o Grafana Loki. Para encontrar más almacenes de datos, puede visitar la documentación de fluentd sobre posibles destinos de registro. Para que los registros sean fáciles de procesar y poder buscar, asegúrese de iniciar sesión en un formato estructurado como JSON en lugar de texto sin formato. Los principales proveedores de nube proporcionan buena documentación sobre la importancia del registro estructurado y cómo implementarlo:

Prometheus

[Prometheus](https://prometheus.io/) is an open source monitoring system, originally developed at SoundCloud, which became the second CNCF hosted project in 2016. Over time, it became a very popular monitoring solution and is now a standard tool that integrates especially well in the Kubernetes and container ecosystem.

Prometheus can collect metrics that were emitted by applications and servers as time series data - these are very simple sets of data that include a timestamp, label and the measurement itself. The Prometheus data model provides [four core metrics](https://prometheus.io/docs/concepts/metric_types/):

* Counter: A value that increases, like a request or error count
* Gauge: Values that increase or decrease, like memory size
* Histogram: A sample of observations, like request duration or response size
* Summary: Similar to a histogram, but also provides the total count of observations.

To expose these metrics, applications can expose an HTTP endpoint under /metrics instead of implementing it yourself. You can use the existing client libraries:

* Go
* Java or Scala
* Python
* Ruby.

You can also use one of the many unofficial client libraries listed in the [Prometheus documentation](https://prometheus.io/docs/instrumenting/clientlibs/).

The data exposed could look like this:

Prometeo Prometheus es un sistema de monitoreo de código abierto, desarrollado originalmente en SoundCloud, que se convirtió en el segundo proyecto alojado en CNCF en 2016. Con el tiempo, se convirtió en una solución de monitoreo muy popular y ahora es una herramienta estándar que se integra especialmente bien en Kubernetes y el ecosistema de contenedores. Prometheus puede recopilar métricas emitidas por aplicaciones y servidores como datos de series de tiempo; estos son conjuntos de datos muy simples que incluyen una marca de tiempo, una etiqueta y la medición en sí. El modelo de datos de Prometheus proporciona cuatro métricas principales: Contador: un valor que aumenta, como una solicitud o un recuento de errores. Calibre: valores que aumentan o disminuyen, como el tamaño de la memoria. Histograma: una muestra de observaciones, como la duración de la solicitud o el tamaño de la respuesta. Resumen: similar a un histograma, pero también proporciona el recuento total de observaciones. Para exponer estas métricas, las aplicaciones pueden exponer un punto final HTTP en /metrics en lugar de implementarlo usted mismo. Puede utilizar las bibliotecas cliente existentes: Ir Java o Scala Pitón Rubí. También puede utilizar una de las muchas bibliotecas cliente no oficiales que figuran en la documentación de Prometheus. Los datos expuestos podrían verse así:

# HELP queue\_length The number of items in the queue.  
# TYPE queue\_length  
gauge queue\_length 42  
# HELP http\_requests\_total The total number of handled HTTP requests.  
# TYPE http\_requests\_total counter  
http\_requests\_total 7734  
# HELP http\_request\_duration\_seconds A histogram of the HTTP request durations in seconds.  
# TYPE http\_request\_duration\_seconds histogram http\_request\_duration\_seconds\_bucket{le="0.05"} 4599  
http\_request\_duration\_seconds\_sum 88364.234  
http\_request\_duration\_seconds\_count 227420  
# HELP http\_request\_duration\_seconds A summary of the HTTP request durations in seconds.  
# TYPE http\_request\_duration\_seconds summary  
http\_request\_duration\_seconds{quantile="0.5"} 0.052  
http\_request\_duration\_seconds\_sum 88364.234  
http\_request\_duration\_seconds\_count 227420

Prometheus has built-in support for Kubernetes and can be configured to automatically discover all services in your cluster and collect the metric data in a defined interval to save them in a time series database.

To query data that is stored in the time series database, Prometheus provides a querying language called [PromQL (Prometheus Query Language)](https://prometheus.io/docs/prometheus/latest/querying/basics/). A user can use PromQL to select and aggregate data in real time and view it in the built-in Prometheus user interface, which offers a simple graphical or tabular view.

Here are some examples taken from the [Prometheus documentation](https://prometheus.io/docs/prometheus/latest/querying/examples/):

# Return all time series with the metric http\_requests\_total and the given job and handler labels:  
http\_requests\_total{job="apiserver", handler="/api/comments"}

Or a sample function that gives a rate over time:

# Return the per-second rate for all time series with the http\_requests\_total metric name, as measured over the last 5 minutes:  
rate(http\_requests\_total[5m])

You can use these functions to get an indication on how a certain value increases or decreases over time. That will help in analyzing errors or predicting failures for an application.

Of course, monitoring only makes sense if you use the data collected. The most used companion for Prometheus is Grafana, which can be used to build dashboards from the collected metrics. You can use Grafana for many more data sources and not only Prometheus, although that is the most used one.

Prometheus tiene soporte integrado para Kubernetes y se puede configurar para descubrir automáticamente todos los servicios en su clúster y recopilar los datos métricos en un intervalo definido para guardarlos en una base de datos de series temporales. Para consultar datos almacenados en la base de datos de series temporales, Prometheus proporciona un lenguaje de consulta llamado PromQL (Prometheus Query Language). Un usuario puede utilizar PromQL para seleccionar y agregar datos en tiempo real y verlos en la interfaz de usuario integrada de Prometheus, que ofrece una vista gráfica o tabular simple. Aquí hay algunos ejemplos tomados de la documentación de Prometheus: # Devuelve todas las series temporales con la métrica http\_requests\_total y las etiquetas de trabajo y controlador proporcionadas: http\_requests\_total{job="apiserver", handler="/api/comments"} O una función de muestra que proporciona una tasa a lo largo del tiempo: # Devuelve la tasa por segundo para todas las series temporales con el nombre de la métrica http\_requests\_total, medida durante los últimos 5 minutos: tasa(http\_solicitudes\_total[5m]) Puede utilizar estas funciones para obtener una indicación de cómo un determinado valor aumenta o disminuye con el tiempo. Eso ayudará a analizar errores o predecir fallas en una aplicación. Por supuesto, el seguimiento sólo tiene sentido si se utilizan los datos recopilados. El complemento más utilizado para Prometheus es Grafana, que se puede utilizar para crear paneles a partir de las métricas recopiladas. Puedes usar Grafana para muchas más fuentes de datos y no solo Prometheus, aunque esa es la más utilizada.

X

Another tool from the Prometheus ecosystem is the [**Alertmanager**](https://prometheus.io/docs/alerting/latest/alertmanager/). The Prometheus server itself allows you to configure alerts when certain metrics reach or pass a threshold. When the alert is firing, Alertmanager can send a notification out to your favorite persistent chat tool, e-mail or specialized tools that are made for alerting and on-call management.

Here is an example for a alerting rule in Prometheus:

groups:  
- name: example  
  rules:  
  - alert: HighRequestLatency  
    expr: job:request\_latency\_seconds:mean5m{job="myjob"} > 0.5 for: 10m  
    labels:  
      severity: page  
    annotations:  
      summary: High request latency

x

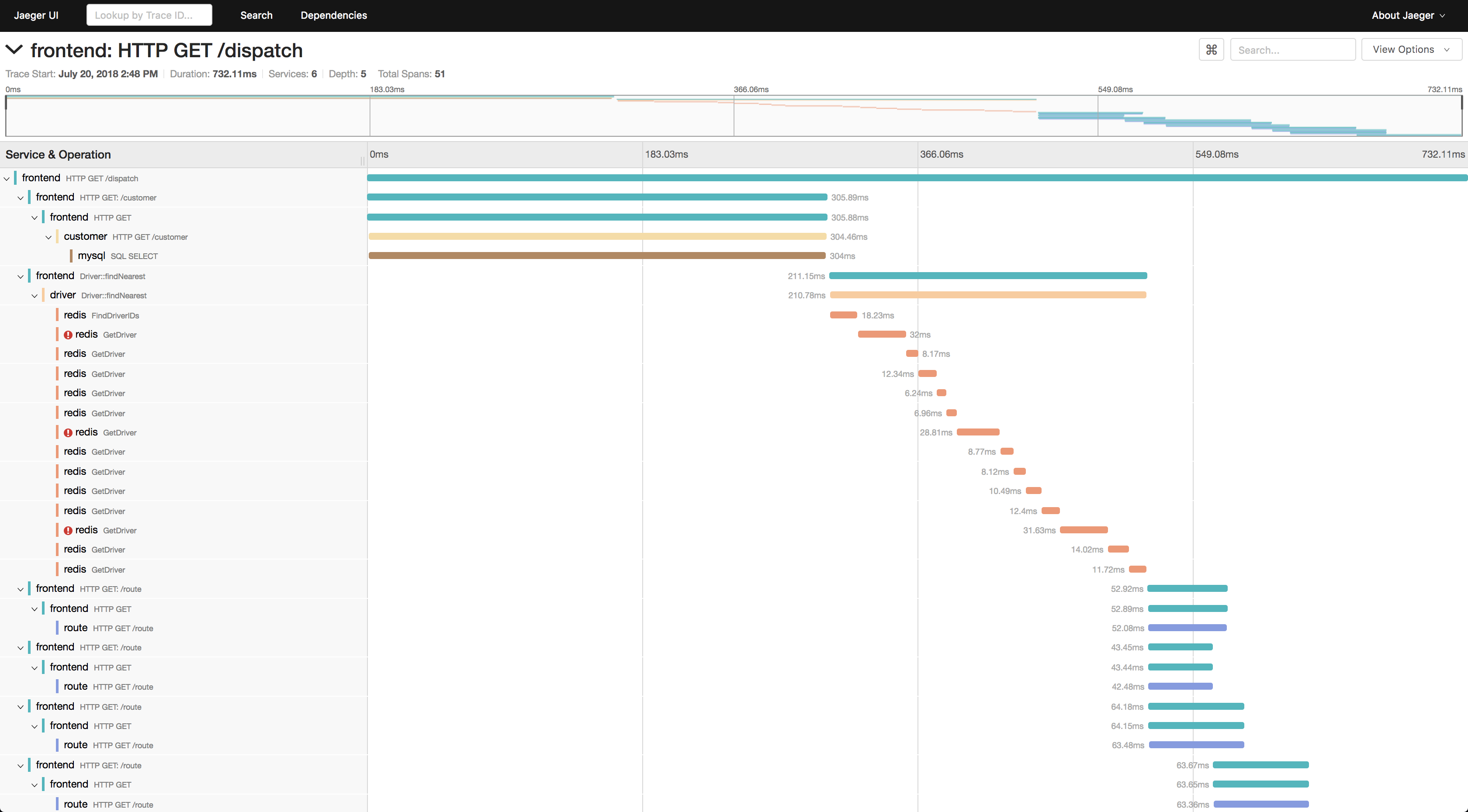
# **Tracing**

Logging and Monitoring with the collection of metrics are not particularly new methods. The same thing cannot be said for (distributed) tracing. Metrics and logs are essential and can give a good overview of individual services, but to understand how a request is processed in a microservice architecture, traces can be of good use.

A trace describes the tracking of a request while it passes through the services. A trace consists of multiple units of work which represent the different events that occur while the request is passing the system. Each application can contribute a span to the trace, which can include information like start and finish time, name, tags or a log message.

These traces can be stored and analyzed in a tracing system like [Jaeger](https://www.jaegertracing.io/).

Rastreo El registro y la supervisión con la recopilación de métricas no son métodos particularmente nuevos. No se puede decir lo mismo del rastreo (distribuido). Las métricas y los registros son esenciales y pueden brindar una buena descripción general de los servicios individuales, pero para comprender cómo se procesa una solicitud en una arquitectura de microservicio, los seguimientos pueden resultar útiles. Un seguimiento describe el seguimiento de una solicitud mientras pasa por los servicios. Un seguimiento consta de múltiples unidades de trabajo que representan los diferentes eventos que ocurren mientras la solicitud pasa por el sistema. Cada aplicación puede contribuir con un lapso al seguimiento, que puede incluir información como hora de inicio y finalización, nombre, etiquetas o un mensaje de registro. Estos rastros se pueden almacenar y analizar en un sistema de rastreo como Jaeger.

Xcc

While tracing was a new technology and method that was geared towards cloud native environments, there were again problems in the area of standardization. In 2019, the [**OpenTracing**](https://opentracing.io/) and [**OpenCensus**](https://opencensus.io/) projects merged to form the [**OpenTelemetry**](https://opentelemetry.io/) project, which is now also a CNCF project.

OpenTelemetry is a set of application programming interfaces (APIs), software development kits (SDKs) and tools that can be used to integrate telemetry such as metrics, protocols, but especially traces into applications and infrastructures. The OpenTelemetry clients can be used to export telemetry data in a standardized format to central platforms like Jaeger. Existing tools can be found in the [OpenTelemetry documentation](https://opentelemetry.io/docs/).

Si bien el rastreo era una tecnología y un método nuevos orientados a entornos nativos de la nube, nuevamente había problemas en el ámbito de la estandarización. En 2019, los proyectos OpenTracing y OpenCensus se fusionaron para formar el proyecto OpenTelemetry, que ahora también es un proyecto CNCF. OpenTelemetry es un conjunto de interfaces de programación de aplicaciones (API), kits de desarrollo de software (SDK) y herramientas que se pueden utilizar para integrar telemetría, como métricas, protocolos, pero especialmente seguimientos en aplicaciones e infraestructuras. Los clientes OpenTelemetry se pueden utilizar para exportar datos de telemetría en un formato estandarizado a plataformas centrales como Jaeger. Las herramientas existentes se pueden encontrar en la documentación de OpenTelemetry.

# **Cost Management**

The possibilities of cloud computing allow us to draw from a theoretically infinite pool of resources and only pay for them when they are really needed. Since cloud providers don’t offer their services "pro-bono", the key to cost optimization in the cloud is to analyze what is really needed and, if possible, automate the scheduling of the resources needed.

anejo de costos Las posibilidades de la computación en la nube nos permiten aprovechar un conjunto de recursos teóricamente infinito y pagar por ellos sólo cuando realmente se necesitan. Dado que los proveedores de la nube no ofrecen sus servicios "pro-bono", la clave para la optimización de costes en la nube es analizar qué se necesita realmente y, si es posible, automatizar la programación de los recursos necesarios.

There are several ways to do automatic and manual optimization. Click to expand each box and learn more about these ways.

Hay varias formas de realizar una optimización automática y manual. Haga clic para expandir cada cuadro y obtener más información sobre estas formas.

Identify wasted and unused resources

With a good monitoring of your resource usage, it is very easy to find unused resources or servers that don’t have a lot of idle time. A lot of cloud vendors have cost explorers that can break down costs for individual services. Autoscaling helps to shut down instances that are not needed.

dentificar recursos desperdiciados y no utilizados Con un buen seguimiento del uso de sus recursos, es muy fácil encontrar recursos o servidores no utilizados que no tengan mucho tiempo de inactividad. Muchos proveedores de nube tienen exploradores de costos que pueden desglosar los costos de servicios individuales. El ajuste de escala automático ayuda a cerrar instancias que no son necesarias.

Right-Sizing

When you start out, it can be a good idea to choose servers and systems with a lot more power than actually needed. Again, good monitoring can give you indications over time of how much resources are actually needed for your application. This is an ongoing process where you should always adapt to the load you really need. Don’t buy powerful machines if you only need half of their capacity.

amaño correcto Al empezar, puede ser una buena idea elegir servidores y sistemas con mucha más potencia de la que realmente se necesita. Una vez más, un buen seguimiento puede brindarle indicaciones a lo largo del tiempo sobre cuántos recursos se necesitan realmente para su aplicación. Este es un proceso continuo en el que siempre debes adaptarte a la carga que realmente necesitas. No compres máquinas potentes si sólo necesitas la mitad de su capacidad

Reserved Instances

On-demand pricing models are great if you really need resources on-demand. Otherwise, you’re probably paying a lot for the "on-demand" service. A method to save a lot of money is to reserve resources and even pay for them upfront. This is a great pricing model if you have a good estimate about the resources you need, maybe even for years in advance.

nstancias reservadas Los modelos de precios bajo demanda son excelentes si realmente necesita recursos bajo demanda. De lo contrario, probablemente estés pagando mucho por el servicio "bajo demanda". Un método para ahorrar mucho dinero es reservar recursos e incluso pagarlos por adelantado. Este es un excelente modelo de precios si tiene una buena estimación de los recursos que necesita, tal vez incluso con años de anticipación.

Spot Instances

If you have a batch job or heavy load for a short amount of time, you can use spot instances to save money. The idea of spot instances is that you get unused resources that have been over-provisioned by the cloud vendor for very low prices. The "problem" is that these resources are not reserved for you, and can be terminated on short notice to be used by someone else paying "full price".

Instancias puntuales Si tiene un trabajo por lotes o una carga pesada durante un período corto de tiempo, puede utilizar instancias puntuales para ahorrar dinero. La idea de las instancias puntuales es que se obtienen recursos no utilizados que el proveedor de la nube ha aprovisionado en exceso a precios muy bajos. El "problema" es que estos recursos no están reservados para usted y pueden cancelarse con poca antelación para que los utilice otra persona que pague el "precio completo".

All these methods can be combined to be more cost-efficient. It is usually no problem to mix on-demand, reserved and spot instances.

# Additional Resources

Learn more about...

## Cloud Native Observability

* [The Cloud Native Landscape: Observability and Analysis](https://thenewstack.io/the-cloud-native-landscape-observability-and-analysis/)

## Prometheus

* [Prometheus Cheat Sheet - Basics (Metrics, Labels, Time Series, Scraping)](https://iximiuz.com/en/posts/prometheus-metrics-labels-time-series/), by Ivan Velichko (2021)

## Prometheus at scale

* [Thanos](https://thanos.io/)
* [Cortex](https://cortexmetrics.io/)

## Logging for Containers

* [Use the native logging mechanisms of containers](https://cloud.google.com/architecture/best-practices-for-operating-containers" \l "use_the_native_logging_mechanisms_of_containers) (Google Cloud)

## Right-Sizing and cost optimization

* [Right Sizing](https://aws.amazon.com/aws-cost-management/aws-cost-optimization/right-sizing/) (Amazon AWS)
* [Cloud cost optimization: principles for lasting success](https://cloud.google.com/blog/topics/cost-management/principles-of-cloud-cost-optimization) (Google Cloud)

Observability is the same as monitoring. True or False?

* A.

True

Your Answer:

Incorrect

* B.

False

Correct Answer

Which of the following is typical telemetry data?

Correct Answer

* A. Metrics
* B. Logs
* C. Traces
* D. All of the above

# Question 7.3

How can you show the logs of a previously terminated container named ruby in the web-1 pod?

Incorrect Answer

* A. kubectl logs web-1 ruby
* B. kubectl logs -p web-1 ruby

Your Answer:

Incorrect

* C. kubectl web-1 ruby -p
* **D. kubectl logs -p -c ruby web-1**

Correct Answer

# Question 7.4

What do you call the pattern where you add a second container to your pod to collect and ship logs?

Correct Answer

* A. Cluster-level logging
* B. Node-level logging
* **C. Sidecar container logging**

**Your Answer:**

**Correct**

* D. Application-level logging

What is a good format for structured logging?

Correct Answer

* A. YAML
* B. JSON

Your Answer:

Correct

* C. XML
* D. HTML

# Question 7.6

What kind of software is Prometheus?

Correct Answer

* A. Software to manage containers
* B. Software to collect and store metrics

Your Answer:

Correct

* C. Software to manage virtual machines
* D. Software to collect and store logs

# Question 7.7

Which Prometheus metric type should be used for a value that only increases (e.g., an error\_count)?

Incorrect Answer

* A. Histogram

Your Answer:

Incorrect

* B. Gauge
* **C. Counter**

Correct Answer

* D. Summary

Which methods can be used to optimize cloud costs? Select all answers that apply.

Incorrect Answer

* A. Buying dedicated bare-metal servers
* **B. Right-Sizing**

Correct Answer

* C. Geo-replication
* D. Reserving instances