

## **01 . CURSO INTRODUCTORIO**

# **Objetivos de aprendizaje del curso**

Al final de este curso, aprenderá lo siguiente:

- La historia y evolución de Kubernetes.
- Su arquitectura y componentes de alto nivel.
- La API, los recursos más importantes que componen la API y cómo utilizarlos.
- Cómo implementar y administrar una aplicación.
- Algunas funciones próximas que impulsarán su productividad.

## **Audiencia y requisitos del curso**

### **Audiencia**

Este curso está dirigido a administradores de Linux o desarrolladores de software que comienzan a trabajar con contenedores y se preguntan cómo administrarlos en producción. En este curso, aprenderá los principios clave que lo pondrán en el camino hacia la gestión de aplicaciones en contenedores en producción.

### **Habilidades de conocimiento**

Para aprovechar al máximo este curso, necesitará lo siguiente:

- Un buen conocimiento de Linux
- Familiaridad con la línea de comando
- Familiaridad con los administradores de paquetes
- Familiaridad con Git y GitHub
- Acceso a un servidor Linux o una computadora de escritorio / portátil Linux
- VirtualBox en su máquina o acceso a una nube pública.

## Entorno de software

El material producido por The Linux Foundation es de distribución flexible. Esto significa que las explicaciones técnicas, los laboratorios y los procedimientos deberían funcionar en la mayoría de las distribuciones de Linux modernas, y no promocionamos productos vendidos por ningún proveedor específico (aunque podemos mencionarlos para escenarios específicos).

En la práctica, la mayor parte de nuestro material está escrito teniendo en cuenta las tres principales familias de distribución de Linux:

- Debian / Ubuntu
- Red Hat / Fedora
- openSUSE / SUSE.

Las distribuciones utilizadas por nuestros estudiantes tienden a ser una de estas tres alternativas, o un producto que se deriva de ellas.

## Entorno de laboratorio

Los ejercicios de laboratorio se escribieron y probaron utilizando instancias de Ubuntu que se ejecutan en Google Cloud Platform. Se han escrito para que sean independientes del proveedor, por lo que pueden ejecutarse en AWS, hardware local o dentro de máquinas virtuales, para brindarle la mayor flexibilidad y opciones.

Cada nodo tiene 3 CPU virtuales y 7,5 G de memoria, con Ubuntu 18.04. Los nodos más pequeños deberían funcionar, pero debería esperar una respuesta lenta. También son posibles otras imágenes del sistema operativo, pero puede haber una ligera diferencia en algunas salidas de comando.

El uso de GCP requiere la configuración de una cuenta e incurrirá en gastos si se utilizan nodos del tamaño sugerido. Para obtener más información, consulte la [Guía de inicio rápido con una máquina virtual Linux](#).

<https://cloud.google.com/compute/docs/quickstart-linux>

Amazon Web Service (AWS) es otro proveedor de nodos basados en la nube y requiere una cuenta; incurrirá en gastos por nodos del tamaño sugerido. Puede encontrar videos e información sobre [cómo iniciar una máquina virtual Linux](#) en el sitio web de AWS.

<https://aws.amazon.com/es/getting-started/hands-on/launch-a-virtual-machine/>

También se pueden utilizar máquinas virtuales como KVM, VirtualBox o VMWare para los sistemas de laboratorio. Poner las VM en una red privada puede facilitar la resolución de problemas. A partir de Kubernetes v1.16.1, el tamaño mínimo (como en apenas funciona) para VirtualBox es 3vCPU / memoria 4G / SO mínimo 5G para el master y 1vCPU / memoria 2G / SO mínimo 5G para el nodo worker.

Finalmente, el uso de nodos bare-metal, con acceso a Internet, también funcionará para los ejercicios de laboratorio.

## Recursos del curso

---

Los recursos para este curso se pueden encontrar en línea. Actualizar este curso lleva tiempo. Por lo tanto, si hay algún cambio entre actualizaciones, siempre puede acceder a las actualizaciones del curso, así como a los recursos del curso en línea:

- Vaya al sitio web de formación de la Fundación Linux para obtener [recursos del curso](#)

<https://training.linuxfoundation.org/cm/LFS258/>

- El ID de usuario es **LFtraining** y la contraseña es **Penguin2014** .

## ¿Qué distribución elegir?

# Which Distribution to Choose?

 THE **LINUX** FOUNDATION

## Which Distribution to Choose?

You should ask yourself several questions when choosing a new distribution:

- Has your employer already standardized?
- Do you want to learn more?
- Do you want to certify?

While there are many reasons that may force you to focus on one Linux distribution versus another, we encourage you to gain experience on all of them. You will quickly notice that technical differences are mainly about package management systems, software versions and file locations. Once you get a grasp of those differences, it becomes relatively painless to switch from one Linux distribution to another.

Some tools and utilities have vendor-supplied front-ends, especially for more particular or complex reporting. The steps included in the text may need to be modified to run on a different platform.

 THE **LINUX** FOUNDATION

## Debian Family

The Debian distribution is the upstream for several other distributions, including Ubuntu, Linux Mint and others. Debian is a pure open source project, and focuses on a key aspect: **stability**. It also provides the largest and most complete software repository to its users.

Ubuntu aims at providing a good compromise between long term stability and ease of use. Since Ubuntu gets most of its packages from Debian's unstable branch, Ubuntu also has access to a very large software repository. For those reasons, we decided to use Ubuntu as the reference Debian-based distribution for our lab exercises:

- Commonly used on both servers and desktops
- DPKG-based, uses apt-get and front-ends for installing and updating
- Upstream for Ubuntu, Linux Mint and others
- Current material based upon the latest release of Ubuntu and should work well with later versions
- x86 and x86-64
  - Long Term Release (LTS)

## Red Hat/Fedora Family

Fedora is the community distribution that forms the basis of Red Hat Enterprise Linux, CentOS, Scientific Linux and Oracle Linux. Fedora contains significantly more software than Red Hat's enterprise version. One reason for this is that a diverse community is involved in building Fedora; it is not just one company.

The Fedora community produces new versions every six months or so. For this reason, we decided to standardize the Red Hat/Fedora part of the course material on the latest version of CentOS 7, which provides much longer release cycles. Once installed, CentOS is also virtually identical to Red Hat Enterprise Linux (RHEL), which is the most popular Linux distribution in enterprise environments:

- Current material is based upon the latest release of Red Hat Enterprise Linux (RHEL) - 7.x at the time of publication, and should work well with later versions
- Supports x86, x86-64, Itanium, PowerPC and IBM System Z
- RPM-based, uses yum (or dnf) to install and update
- Long release cycle; targets enterprise server environments
- Upstream for CentOS, Scientific Linux and Oracle Linux

## openSUSE Family

The relationship between openSUSE and SUSE Linux Enterprise Server is similar to the one we just described between Fedora and Red Hat Enterprise Linux. In this case, however, we decided to use openSUSE as the reference distribution for the openSUSE family, due to the difficulty of obtaining a free version of SUSE Linux Enterprise Server. The two products are extremely similar and material that covers openSUSE can typically be applied to SUSE Linux Enterprise Server with no problem:

- Current material is based upon the latest release of openSUSE, and should work well with later versions.
- RPM-based, uses zypper to install and update
- YaST available for administration purposes
- x86 and x86-64
- Upstream for SUSE Linux Enterprise Server (SLES)

## New Distribution Similarities

Current trends and changes to the distributions have reduced some of the differences between the distributions.

- **systemd** (system startup and service management)  
systemd is used by the most common distributions, replacing the SysVinit and Upstart packages. Replaces service and chkconfig commands.
- **journald** (manages system logs)  
journald is a systemd service that collects and stores logging data. It creates and maintains structured, indexed journals based on logging information that is received from a variety of sources. Depending on the distribution, text-based system logs may be replaced.
- **firewalld** (firewall management daemon)  
firewalld provides a dynamically managed firewall with support for network/firewall zones to define the trust level of network connections or interfaces. It has support for IPv4, IPv6 firewall settings and for Ethernet bridges. This replaces the iptables configurations.
- **ip** (network display and configuration tool)  
The ip program is part of the net-tools package, and is designed to be a replacement for the ifconfig command. The ip command will show or manipulate routing, network devices, routing information and tunnels.

Dado que las utilidades son comunes en todas las distribuciones, el contenido del curso y la información del laboratorio utilizarán estas utilidades.

Si su elección de distribución o versión no admite estos comandos, traduzca en consecuencia.

Los siguientes documentos pueden ser de ayuda para traducir comandos más antiguos a sus contrapartes de systemd:

[https://fedoraproject.org/wiki/SysVinit\\_to\\_Systemd\\_Cheatsheet](https://fedoraproject.org/wiki/SysVinit_to_Systemd_Cheatsheet)

<https://wiki.debian.org/systemd/CheatSheet>

[https://en.opensuse.org/openSUSE:Cheat\\_sheet\\_13.1#Services](https://en.opensuse.org/openSUSE:Cheat_sheet_13.1#Services)

## Práctica de laboratorio 1.1 - Configuración del sistema para sudo

Ver: 95j9tla4zde3-LFS258-labs\_V2020-10-19.pdf

## 02 . CONCEPTOS BÁSICOS DE KUBERNETES

# Objetivos de aprendizaje

Al final de este capítulo, debería poder:

- Habla sobre Kubernetes.
- Aprenda la terminología básica de Kubernetes.
- Analice las herramientas de configuración.
- Conozca qué recursos comunitarios están disponibles.

## ¿Qué es Kubernetes?

---

Ejecutar un contenedor en una computadora portátil es relativamente simple. Pero conectar contenedores en múltiples hosts, escalarlos, implementar aplicaciones sin tiempo de inactividad y descubrir servicios entre varios aspectos, puede ser difícil.

Kubernetes aborda esos desafíos desde el principio con un conjunto de primitivas y una poderosa API abierta y extensible. La capacidad de agregar nuevos objetos y controladores permite una fácil personalización para diversas necesidades de producción.

Según el [sitio](https://kubernetes.io) web kubernetes.io, Kubernetes es:

*"un sistema de código abierto para automatizar la implementación, el escalado y la gestión de aplicaciones en contenedores".*

Un aspecto clave de Kubernetes es que se basa en 15 años de experiencia en Google en un proyecto llamado borg.

La infraestructura de Google comenzó a alcanzar una gran escala antes de que las máquinas virtuales se generalizaran en el centro de datos, y los contenedores proporcionaron una solución detallada para empaquetar clústeres de manera eficiente.



La eficiencia en el uso de clústeres y la administración de aplicaciones distribuidas ha sido el núcleo de los desafíos de Google.



En griego, **κυβερνήτης** significa el timonel o piloto del barco. Manteniendo el tema marítimo de los contenedores Docker, Kubernetes es el piloto de un barco de contenedores. Debido a la dificultad para pronunciar el nombre, muchos usarán un apodo, K8s, ya que Kubernetes tiene ocho letras entre K y S. El apodo se pronuncia como el de Kate.

## Componentes de Kubernetes

---

La implementación de contenedores y el uso de Kubernetes pueden requerir un cambio en el enfoque de desarrollo y administración del sistema para implementar aplicaciones. En un entorno tradicional, una aplicación (como un servidor web) sería una aplicación monolítica ubicada en un servidor dedicado. A medida que aumenta el tráfico web, la aplicación se ajustará y quizás se trasladará a un hardware cada vez más grande. Después de un par de años, es posible que se hayan realizado muchas personalizaciones para satisfacer las necesidades actuales de tráfico web.

En lugar de utilizar un servidor grande, Kubernetes aborda el mismo problema mediante la implementación de una gran cantidad de pequeños servidores web o microservicios. Los lados del servidor y del cliente de la aplicación están escritos para esperar que haya muchos agentes posibles disponibles para responder a una solicitud. También es importante que los clientes esperen que los procesos del servidor mueran y sean reemplazados, dando lugar a una implementación transitoria del servidor. En

lugar de un gran servidor web Apache con muchos demonios httpd respondiendo a las solicitudes de página, habría muchos servidores nginx, cada uno respondiendo.

La naturaleza transitoria de los servicios más pequeños también permite el desacoplamiento. Cada aspecto de la aplicación tradicional se reemplaza por un microservicio o agente dedicado, pero transitorio. Para unir a estos agentes, o sus reemplazos, utilizamos servicios y llamadas API. Un servicio vincula el tráfico de un agente a otro (por ejemplo, un servidor web frontend a una base de datos backend) y maneja una nueva IP u otra información, en caso de que una muera y sea reemplazada.

La comunicación está totalmente basada en llamadas a la API, lo que permite flexibilidad. La información de configuración del clúster se almacena en formato JSON dentro de etcd, pero la comunidad suele escribirla en YAML. Los agentes de Kubernetes convierten YAML a JSON antes de la persistencia en la base de datos.



Kubernetes está escrito en Go Language, un lenguaje portátil que es como una hibridación entre C ++, Python y Java. Algunos afirman que incorpora las mejores (mientras que otros afirman que las peores) partes de cada uno.

## Desafíos

---

Los contenedores proporcionan una excelente manera de empaquetar, enviar y ejecutar aplicaciones, ese es el lema de Docker.

La experiencia del desarrollador se ha mejorado enormemente gracias a los contenedores. Los contenedores, y Docker específicamente, han permitido a los desarrolladores la facilidad para crear imágenes de contenedores, la simplicidad de compartir imágenes a través de registros de Docker y brindar una experiencia de usuario poderosa para administrar contenedores.

Sin embargo, administrar contenedores a escala y diseñar una aplicación distribuida basada en los principios de los microservicios puede ser un desafío.

Un primer paso inteligente es decidir sobre una canalización de integración continua / entrega continua (CI / CD) para crear, probar y verificar imágenes de contenedores. Herramientas como [Spinnaker](#) , [Jenkins](#) y [Helm](#) pueden resultar útiles, entre otras posibles herramientas. Esto ayudará con los desafíos de un entorno dinámico.

Luego, necesita un grupo de máquinas que actúen como su infraestructura base en la que ejecutar sus contenedores. También necesita un sistema para lanzar sus contenedores y vigilarlos cuando las cosas fallan y reemplazarlos según sea necesario. Los rolling updates y los rollbacks fáciles de contenedores son una característica importante y, finalmente, destruyen el recurso cuando ya no se necesitan.

Todas estas acciones requieren una red y almacenamiento flexibles, escalables y fáciles de usar. A medida que los contenedores se lanzan en cualquier nodo worker, la red debe unir el recurso a otros contenedores, sin dejar de mantener el tráfico seguro de otros. También necesitamos una estructura de almacenamiento que proporcione y mantenga o recicle el almacenamiento de manera transparente.

Cuando Kubernetes responde a estas preocupaciones, uno de los mayores desafíos para la adopción son las propias aplicaciones, que se ejecutan dentro del contenedor. Deben escribirse o reescribirse para que sean realmente transitorios. Una buena pregunta para reflexionar: si implementaran Chaos Monkey, que podría terminar *cualquier* contenedor en cualquier momento, ¿se darían cuenta tus clientes?

## Otras soluciones

---

Construido en código abierto y fácilmente extensible, Kubernetes es definitivamente una solución para administrar aplicaciones en contenedores. También existen otras soluciones.



### Docker Swarm

[Docker Swarm](#) es la solución proporcionada por Docker Inc. Recientemente se ha rediseñado y se basa en [SwarmKit](#) . Está integrado con Docker Engine.



### Apache Mesos

Apache Mesos es un programador de centro de datos, que puede ejecutar contenedores mediante el uso de marcos . Marathon es el marco que le permite orquestar contenedores.



### Nomad

[Nomad](#) de HashiCorp, los creadores de Vagrant y Consul, es otra solución para administrar aplicaciones en contenedores. Nomad programa las tareas definidas en *Jobs* . Tiene un controlador Docker que le permite definir un contenedor en ejecución como una tarea.



## Rancher

Rancher es un sistema agnóstico del orquestador de contenedores, que proporciona una interfaz de panel único para administrar aplicaciones. Es compatible con Mesos, Swarm, Kubernetes.

# Herencia Borg

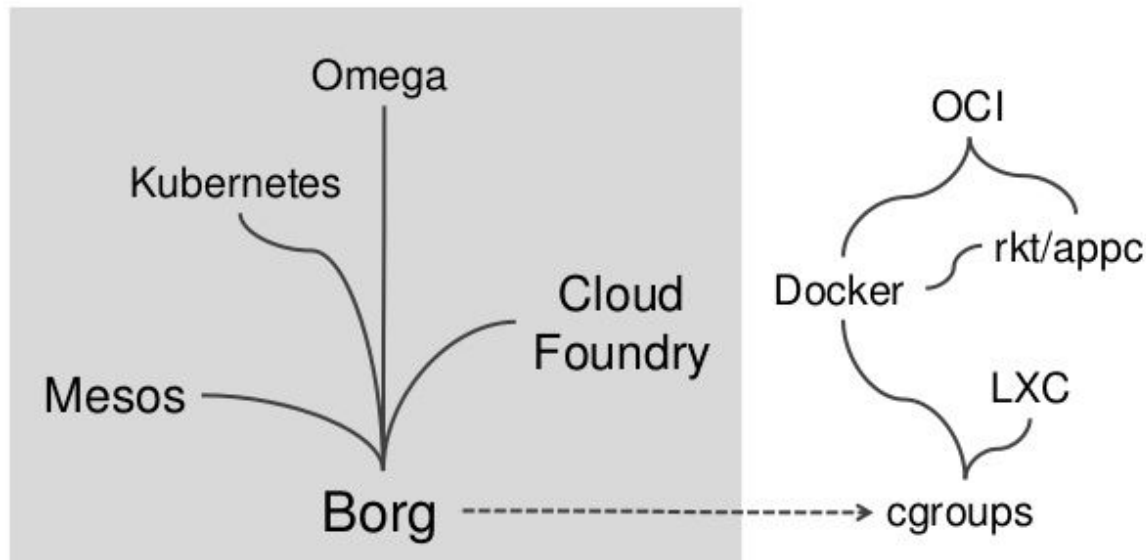
---

Lo que distingue principalmente a Kubernetes de otros sistemas es su herencia. Kubernetes está inspirado en Borg, el sistema interno que utiliza Google para administrar sus aplicaciones (por ejemplo, Gmail, Apps, GCE).

Con Google vertiendo las valiosas lecciones que aprendieron al escribir y operar Borg durante más de 15 años en Kubernetes, esto hace que Kubernetes sea una opción segura al tener que decidir qué sistema usar para administrar contenedores. Si bien es una herramienta poderosa, parte del crecimiento actual de Kubernetes es facilitar el trabajo y el manejo de cargas de trabajo que no se encuentran en un centro de datos de Google.

Para obtener más información sobre las ideas detrás de Kubernetes, puede leer el documento *[Administración de clústeres a gran escala en Google con Borg](https://research.google/pubs/pub43438/)*.

<https://research.google/pubs/pub43438/>



CLOUD FOUNDRY FOUNDATION

### The Kubernetes Lineage

Chip Childers, Cloud Foundry Foundation

Retrieved from [The Platform for Forging Cloud Native Applications](https://www.slideshare.net/chipchilders/cloud-foundry-the-platform-for-forging-cloud-native-applications) presentation

<https://www.slideshare.net/chipchilders/cloud-foundry-the-platform-for-forging-cloud-native-applications>

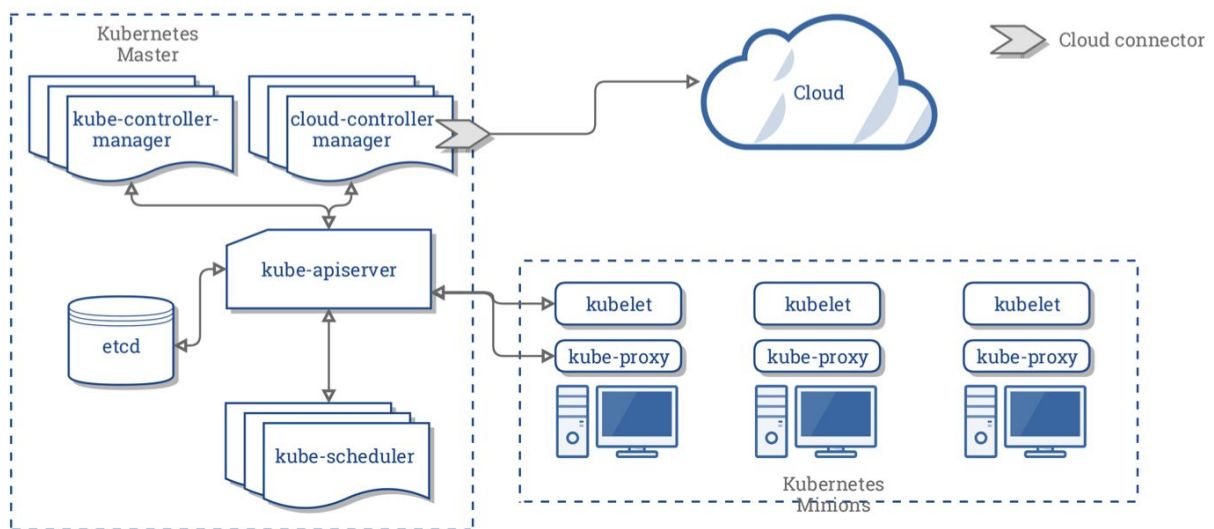
Borg ha inspirado los sistemas de centros de datos actuales, así como las tecnologías subyacentes que se utilizan en el tiempo de ejecución de contenedores hoy. Google contribuyó con cgroups al kernel de Linux en 2007; limita los recursos utilizados por la colección de procesos. Tanto los cgroups como los namespaces de Linux están en el corazón de los contenedores de hoy, incluido Docker.

Mesos se inspiró en las discusiones con Google cuando Borg todavía era un secreto. De hecho, Mesos crea un programador de varios niveles, cuyo objetivo es utilizar mejor un clúster de centro de datos.

Cloud Foundry Foundation adopta los principios de aplicación de 12 factores. Estos principios brindan una excelente guía para crear aplicaciones web que se puedan escalar fácilmente, se puedan implementar en la nube y cuya construcción esté automatizada. Borg y Kubernetes también abordan estos principios.

# Arquitectura de Kubernetes

Para desmitificar Kubernetes rápidamente, echemos un vistazo al gráfico de la *arquitectura de Kubernetes*, que muestra un diagrama de arquitectura de alto nivel de los componentes del sistema. No se muestran todos los componentes. Cada nodo que ejecute un contenedor tendría **kubelet** y **kube-proxy**, por ejemplo.



## Arquitectura de Kubernetes

Obtenido de la documentación de Kubernetes : [conceptos subyacentes al administrador del controlador de la nube](https://kubernetes.io/docs/concepts/architecture/cloud-controller/)

<https://kubernetes.io/docs/concepts/architecture/cloud-controller/>

En su forma más simple, Kubernetes está compuesto por un administrador central (también conocido como **master**) y algunos nodos **workers**, una vez llamados minions (veremos en un capítulo siguiente cómo se puede ejecutar todo en un solo nodo con fines de prueba). El administrador ejecuta un API server, un scheduler, varios controlers y un sistema de almacenamiento para mantener el estado del clúster, la configuración del contenedor y la configuración de la red.

Kubernetes expone una API a través del API server. Puede comunicarse con la API mediante un cliente local llamado **kubectl** o puede escribir su propio cliente y usar comandos **curl** . El **kube-scheduler** envía las solicitudes para ejecutar contenedores que llegan a la API y encuentra un nodo adecuado para ejecutar ese contenedor. Cada nodo del clúster ejecuta dos procesos: un kubelet y un kube-proxy. El kubelet recibe solicitudes para ejecutar los contenedores, administra los recursos necesarios y los supervisa en el nodo local. El kubelet interactúa con el motor de contenedor local, que es Docker de forma predeterminada, pero podría ser rkt o cri-o, que está ganando popularidad.

El **kube-proxy** crea y administra reglas de red para exponer el contenedor en la red.

El uso de un esquema de comunicación basado en API permite contenedores y nodos trabajadores que no son de Linux. El soporte para Windows Server 2019 se graduó a *estable* con la versión 1.14. Solo los nodos de Linux pueden ser masters en un clúster.

## Terminología

---

Hemos aprendido que Kubernetes es un sistema de orquestación para implementar y administrar contenedores. Los contenedores no se gestionan individualmente; en cambio, son parte de un objeto más grande llamado **Pod** . Un Pod consta de uno o más contenedores que comparten una dirección IP, acceso al almacenamiento y espacio de nombres. Por lo general, un contenedor en un pod ejecuta una aplicación, mientras que otros contenedores admiten la aplicación principal.

Kubernetes usa namespaces para mantener los objetos distintos entre sí, para el control de recursos y consideraciones multi-tenant. Algunos objetos tienen cluster-scoped, otros tienen un namespace a la vez. Como el namespace es una



segregación de recursos, los pods necesitarían aprovechar los servicios para comunicarse.

La orquestación se gestiona a través de una serie de watch-loops, también denominados controllers u **operators**. Cada controllers interroga al **kube-apiserver** por un estado de objeto particular, luego modifica el objeto hasta que el estado declarado coincide con el estado actual. Estos controllers se compilan en **kube-controller-manager**, pero se pueden agregar otros utilizando definiciones de recursos personalizadas. El **operator** predeterminado y feature-filled para contenedores es una **Deployment**. Un deployment no funciona directamente con pods. En su lugar, administra **ReplicaSets**. ReplicaSet es un operator que creará o finalizará pods enviando un podSpec. El podSpec se envía al kubelet, que luego interactúa con el motor del contenedor, Docker por defecto, para generar o terminar un contenedor hasta que se esté ejecutando el número solicitado.

El operador del servicio solicita las direcciones IP y los endpoints existentes y administrará la conectividad de la red según las labels. Se utilizan para comunicarse entre pods, espacios de nombres y fuera del clúster. También existen Jobs y CronJobs para manejar tareas únicas o recurrentes, entre otras.

Administrar fácilmente miles de pods en cientos de nodos puede ser una tarea difícil. Para facilitar la gestión, podemos utilizar **labels**, cadenas arbitrarias que pasan a formar parte de los metadatos del objeto. Estos se pueden usar para verificar o cambiar el estado de los objetos sin tener que conocer nombres individuales o UID. Los nodos pueden tener **taints** para desalentar las asignaciones de pods, a menos que el pod tenga una **toleration** en sus metadatos.

También hay espacio en los metadatos para los **annotations** que permanecen con el objeto pero que los comandos de Kubernetes no pueden usar. Esta información podría ser utilizada por agentes externos u otras herramientas.

# Innovación

---

Desde sus inicios, Kubernetes ha experimentado un tremendo ritmo de innovación y adopción. La comunidad de desarrolladores, usuarios, evaluadores y defensores crece continuamente todos los días. El software también se está moviendo a un ritmo extremadamente rápido, lo que incluso está poniendo a prueba a GitHub:

- Dado a código abierto en junio de 2014
- Miles de colaboradores
- Más de 83k commits
- Más de 28k en Slack
- Actualmente, en un ciclo de lanzamiento principal de tres meses
- Cambios constantes.

# Comunidad de usuarios

---

Kubernetes se está adoptando a un ritmo muy rápido. Para obtener más información, debe consultar los [estudios de caso](#) presentados en el sitio web de Kubernetes. Ebay, Box, Pearson y Wikimedia han compartido sus historias.

<https://kubernetes.io/case-studies/>

[Pokemon Go](#) , el juego para móviles de más rápido crecimiento, también se ejecuta en Google Container Engine (GKE), el servicio Kubernetes de Google Cloud Platform (GCP).

### Kubernetes Users



## Herramientas

---

Hay varias herramientas que puede utilizar para trabajar con Kubernetes. A medida que el proyecto ha crecido, se ponen a disposición nuevas herramientas, mientras que las antiguas están en desuso. **Minikube** es una herramienta muy simple destinada a ejecutarse dentro de VirtualBox. Si tiene recursos limitados y no quiere muchas molestias, es la forma más fácil de empezar a trabajar. Lo mencionamos para aquellos que no están interesados en un entorno de producción típico, pero quieren utilizar la herramienta.

Nuestros laboratorios se centrarán en el uso de **kubeadm** y **kubectl** , que son herramientas muy potentes y complejas.

También existen herramientas de terceros, como Helm, una herramienta sencilla para usar gráficos de Kubernetes, y Kompose para traducir archivos de Docker Compose en objetos de Kubernetes. ¡Espere que estas herramientas cambien a menudo!

## Fundación de computación nativa en la nube (CNCF)

---

Kubernetes es un software de código abierto con una licencia de Apache. Google donó Kubernetes a un proyecto colaborativo recién formado dentro de The Linux Foundation en [julio de 2015](#), cuando Kubernetes alcanzó la versión v1.0. Este proyecto se conoce como [Cloud Native Computing Foundation \(CNCF\)](#).

CNCF no se trata solo de Kubernetes, sino que actúa como el organismo rector del software de código abierto que resuelve problemas específicos que enfrentan las aplicaciones nativas de la nube (es decir, aplicaciones que están escritas específicamente para un entorno de nube).

CNCF tiene muchos miembros corporativos que colaboran, como Cisco, Cloud Foundry Foundation, AT&T, Box, Goldman Sachs y muchos otros.



**Nota:** Dado que CNCF ahora posee los derechos de autor de Kubernetes, los colaboradores de la fuente deben firmar un acuerdo de licencia de colaborador (CLA) con CNCF, al igual que cualquier colaborador de un proyecto con licencia de Apache firma un CLA con Apache Software Foundation.

# Recomendaciones de recursos

---

Si desea ir más allá de esta introducción general a Kubernetes, le recomendamos algunas cosas:

:



Leer el [periódico Borg](https://research.google/pubs/pub43438/) <https://research.google/pubs/pub43438/>

Escuche a [John Wilkes](https://www.gcppodcast.com/post/episode-46-borg-and-k8s-with-john-wilkes/) hablando sobre Borg y Kubernetes  
<https://www.gcppodcast.com/post/episode-46-borg-and-k8s-with-john-wilkes/>

Agregue el Hangout de la [comunidad de](https://github.com/kubernetes/community) Kubernetes a su calendario y asista al menos una vez. <https://github.com/kubernetes/community>

Únase a la comunidad de [Slack](https://slack.kubernetes.io/) y acceda al **canal # kubernetes-users** .  
<http://slack.kubernetes.io/>

Echa un vistazo a la [comunidad](https://stackoverflow.com/nocaptcha?s=c09580b5-0588-42bf-937a-5ffc5af224ac) muy activa de [Stack Overflow](https://stackoverflow.com/nocaptcha?s=c09580b5-0588-42bf-937a-5ffc5af224ac)  
[.https://stackoverflow.com/nocaptcha?s=c09580b5-0588-42bf-937a-5ffc5af224ac](https://stackoverflow.com/nocaptcha?s=c09580b5-0588-42bf-937a-5ffc5af224ac)

# Laboratorio 2.1 - Ver recursos en línea

Ver: 95j9tla4zde3-LFS258-labs\_V2020-10-19.pdf

Felicitaciones por completar el *Capítulo 2: Conceptos básicos de Kubernetes*. Responda este cuestionario para comprobar su comprensión de los conceptos que ha aprendido hasta ahora.

## Pregunta 2.1

---

¿Cuáles de los siguientes forman parte de un Pod?

- **A**. Uno o más contenedores
- **B**. Dirección IP compartida
- **C**. Un namespace
- **D**. Todas las anteriores

## Pregunta 2.2

---

¿Qué empresa desarrolló Borg como proyecto interno?

- **A**. Amazon
- **B**. Google
- **C**. IBM
- **D**. Toyota

## Pregunta 2.3

---

¿En qué base de datos se almacenan los objetos y el estado del clúster?

- **A .** ZooKeeper
- **B .** MySQL
- **C .** etcd
- **D .** Couchbase

## Pregunta 2.4

---

La orquestación se gestiona mediante una serie de watch-loops o controladores. Cada uno interroga al \_\_\_\_\_ por un estado de objeto particular.

- **A .** kube-apiserver
- **B .** etcd
- **C .** kubelet
- **D .** ntpd

### 03 . INSTALACION Y CONFIGURACION

=

## Objetivos de aprendizaje

Al final de este capítulo, debería poder:

- Descarga las herramientas de instalación y configuración.
- Instale un master de Kubernetes y haga crecer un clúster.
- Configure una solución de red para comunicaciones seguras.
- Analice las consideraciones de implementación de alta disponibilidad.

## Herramientas de instalación

---

Este capítulo trata sobre la instalación y configuración de Kubernetes. Vamos a revisar algunos mecanismos de instalación que puede utilizar para crear su propio clúster de Kubernetes.

Para comenzar sin tener que sumergirse de inmediato en la instalación y configuración de un clúster, existen dos opciones principales.

Una forma es utilizar Google Container Engine (GKE), un servicio en la nube de Google Cloud Platform, que le permite solicitar un clúster de Kubernetes con la última versión estable.

Otra forma sencilla de empezar es utilizar Minikube. Es un único binario que se implementa en el software Oracle VirtualBox, que puede ejecutarse en varios sistemas operativos. Si bien Minikube es un nodo local y único, le brindará una plataforma de aprendizaje, pruebas y desarrollo. [MicroK8s](#) es una herramienta más nueva desarrollada por Canonical y destinada a una fácil instalación. Dirigido a instalaciones similares a dispositivos, actualmente se ejecuta en Ubuntu 16.04 y 18.04.



Para poder usar el clúster de Kubernetes, deberá tener instalada la línea de comandos de Kubernetes, llamada **kubectl** . Esto se ejecuta localmente en su máquina y apunta al API server endpoint. Le permite crear, administrar y eliminar todos los recursos de Kubernetes (por ejemplo, Pods, Implementaciones, Servicios). Es una CLI poderosa que usaremos durante el resto de este curso. Entonces, debes familiarizarte con él.

En este curso, usaremos **kubeadm** , la herramienta sugerida por la comunidad del proyecto Kubernetes, que facilita la instalación de Kubernetes y evita los instaladores específicos del proveedor. Hacer que un clúster se ejecute implica dos comandos: **kubeadm init** , que se ejecuta en un nodo master, y luego, **kubeadm join** , que se ejecuta en su worker o nodos masters redundantes, y su propio clúster arranca. La flexibilidad de estas herramientas permite implementar Kubernetes en varios lugares. Los ejercicios de laboratorio utilizan este método.

También hablaremos de otros mecanismos de instalación, como kubespray o kops, otra forma de crear un clúster de Kubernetes en AWS. Notaremos que puede crear su archivo de unidad systemd de una manera muy tradicional. Además, puede usar una imagen de contenedor llamada **hiperkube** , que contiene todos los binarios clave de Kubernetes, de modo que pueda ejecutar un clúster de Kubernetes con solo iniciar algunos contenedores en sus nodos.

## Instalación de kubectl

---

Para configurar y administrar su clúster, probablemente usará el comando **kubectl** . También puede usar llamadas RESTful o el idioma Go.

Las distribuciones de Enterprise Linux tienen las diversas utilidades de Kubernetes y otros archivos disponibles en sus repositorios. Por ejemplo, en RHEL 7 / CentOS 7, encontrará **kubectl** en el paquete **kubernetes-client** .

Puede (si es necesario) descargar el código de [GitHub](#) y seguir los pasos habituales para compilar e instalar `kubect1` .

<https://github.com/kubernetes/kubernetes/tree/master/pkg/kubect1>

Esta línea de comando usará `$HOME/.kube/config` como archivo de configuración. Contiene todos los endpoints de Kubernetes que podría usar. Si lo examina, verá definiciones de clúster (es decir, IP endpoints), credenciales y contextos.

Un *contexto* es una combinación de un clúster y credenciales de usuario. Puede pasar estos parámetros en la línea de comando o cambiar el shell entre contextos con un comando, como en:

```
$ kubectl config use-context foobar
```

Esto es útil cuando se pasa de un entorno local a un clúster en la nube, o de un clúster a otro, por ejemplo, del desarrollo a la producción.

## Usando Google Kubernetes Engine (GKE)

---

Google toma cada versión de Kubernetes a través de pruebas rigurosas y las pone a disposición a través de su servicio GKE. Para poder usar GKE, necesitará lo siguiente:

- Una cuenta en [Google Cloud](#) .
- Un método de pago por los servicios que utilizará.
- El cliente de línea de comandos de `gcloud` .

Existe una extensa documentación para instalarlo. Elija su método de instalación favorito y configúrelo. Para obtener más detalles, puede visitar la [página web Installing Cloud SDK](#) . <https://cloud.google.com/sdk/install#linux>

Luego podrá seguir la [guía de inicio rápido de GKE](https://cloud.google.com/kubernetes-engine/docs/quickstart) y estará listo para crear su primer clúster de Kubernetes: <https://cloud.google.com/kubernetes-engine/docs/quickstart>

```
$ gcloud container clusters create linuxfoundation
```

```
$ gcloud container clusters list
```

```
$ kubectl get nodes
```

Al instalar `gcloud`, habrá instalado `kubectl` automáticamente. En los comandos anteriores, creamos el clúster, lo enumeramos y luego, enumeramos los nodos del clúster con `kubectl`.

Una vez que haya terminado, ***no olvide eliminar su clúster***; de lo contrario, se le seguirá cobrando por ello:

```
$ gcloud container clusters delete linuxfoundation
```

## Usando Minikube

---

También puede usar Minikube, un proyecto de código abierto dentro de la [organización GitHub Kubernetes](https://github.com/kubernetes/minikube) <https://github.com/kubernetes/minikube>. Si bien puede descargar una versión de GitHub, siguiendo las instrucciones enumeradas, puede ser más fácil descargar un binario precompilado. Asegúrese de verificar y obtener la última versión.

Por ejemplo, para obtener la versión v.0.22.2, haga lo siguiente:

```
$ curl -Lo minikube
```

```
https://storage.googleapis.com/minikube/releases/latest/minikube  
-darwin-amd64
```

```
$ chmod +x minikube
```

```
$ sudo mv minikube /usr/local/bin
```

Con Minikube ahora instalado, iniciar Kubernetes en su máquina local es muy fácil:

```
$ minikube start
```

```
$ kubectl get nodes
```

Esto iniciará una máquina virtual VirtualBox que contendrá una implementación de Kubernetes de un solo nodo y el motor Docker. Internamente, minikube ejecuta un único binario de Go llamado `localkube`. Este binario ejecuta todos los componentes de Kubernetes juntos. Esto hace que Minikube sea más simple que una implementación completa de Kubernetes. Además, Minikube VM también ejecuta Docker, para poder ejecutar contenedores.

## Instalación con kubeadm

---

Una vez que se familiarice con Kubernetes usando Minikube, es posible que desee comenzar a construir un clúster real. Actualmente, el método más sencillo es usar **kubeadm**, que apareció en Kubernetes v1.4.0, y se puede usar para iniciar un clúster rápidamente. A medida que la comunidad se ha centrado en **kubeadm**, ha pasado de beta a estable y ha añadido alta disponibilidad con v1.15.0.

El sitio web de Kubernetes proporciona documentación sobre [cómo usar kubeadm para crear un clúster](https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/).

<https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/>

Los repositorios de paquetes están disponibles para Ubuntu 18.04 y CentOS 7.1. Los paquetes aún no están disponibles para Ubuntu 18.04, pero funcionarán como verá en los ejercicios de laboratorio.

Para unir otros nodos al clúster, necesitará al menos un token y un hash SHA256. Esta información es devuelta por el comando `kubeadm init`. Una vez que el master se haya inicializado, aplicaría un complemento de red. Pasos principales:

- Ejecute `kubeadm init` en el nodo principal.
- Cree una red para los criterios de IP-per-Pod.
- Ejecute `kubeadm join --token token head-node-IP` en los nodos workers.

También puede crear la red con `kubectrl` utilizando un manifiesto de recursos de la red.

Por ejemplo, para usar la red Weave, haría lo siguiente:

```
$ kubectrl create -f https://git.io/weave-kube
```

Una vez que se completen todos los pasos, los workers y otros nodos masters se unan, tendrá un clúster de Kubernetes de múltiples nodos funcional y podrá usar `kubectrl` para interactuar con él.

## kubeadm-upgrade

---

Si crea su clúster con `kubeadm`, también tiene la opción de actualizar el clúster mediante el comando `kubeadm upgrade`. Si bien la mayoría opta por permanecer con una versión el mayor tiempo posible y, a menudo, se saltan varias versiones, esto ofrece una ruta útil para las actualizaciones regulares por razones de seguridad.

- `plan`

Esto comparará la versión instalada con la más reciente que se encuentre en el repositorio y verificará si el clúster se puede actualizar.

- **apply**

Actualiza el primer nodo del plano de control del clúster a la versión especificada.

- **diff**

Similar a `apply --dry-run`, este comando mostrará las diferencias aplicadas durante una actualización.

- **node**

Esto permite actualizar la configuración local de kubelet en los nodos workers, o los control planes de otros nodos masters si hay más de uno. Además, accederá a un comando de `phase` para recorrer el proceso de actualización.

Proceso de actualización general:

- Actualiza el software
- Verifique la versión del software
- Drena el control plane
- Ver la actualización planificada
- Aplicar la actualización
- Desenganche el control plane para permitir el scheduled de los pods .

Los pasos detallados se pueden encontrar en la documentación de Kubernetes:

[Actualización de clústeres de kubeadm](#) .

<https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/>

## Instalación de un Pod Network

---

Antes de inicializar el clúster de Kubernetes, se debe considerar la red y evitar los conflictos de IP. Hay varias opciones de Pod Network, en diferentes niveles de desarrollo y conjunto de características.

Muchos de los proyectos mencionarán la Interfaz de red de contenedores (CNI), que es un proyecto de CNCF. Actualmente, varios tiempos de ejecución de contenedores utilizan CNI. Como estándar para manejar la gestión de la implementación y la limpieza de los recursos de red, CNI se volverá más popular.

## Opciones de Pod Networking



Calico

Una red plana de Capa 3 que se comunica sin encapsulación de IP, utilizada en producción con software como Kubernetes, OpenShift, Docker, Mesos y OpenStack. Visto como un modelo de red simple y flexible, se adapta bien a entornos grandes. Otra opción de red, Canal, también parte de este proyecto, permite la integración con Flannel. Permite la implementación de políticas de red.

Para obtener más detalles, consulte la [página web](https://www.projectcalico.org/) del Proyecto Calico  
[.https://www.projectcalico.org/](https://www.projectcalico.org/)



## Flannel

Una red IPv4 de capa 3 entre los nodos de un clúster. Desarrollado por CoreOS, tiene una larga historia con Kubernetes. Centrado en el tráfico entre hosts, no en cómo los contenedores configuran las redes locales, puede usar uno de varios mecanismos de backend, como VXLAN. Un agente de flanneld en cada nodo asigna concesiones de subred para el host. Si bien se puede configurar después de la implementación, es mucho más fácil antes de agregar cualquier Pod.

Puede obtener más información sobre [Flannel](https://github.com/coreos/flannel) en sus páginas de GitHub.  
<https://github.com/coreos/flannel>



## Kube-Router

Binario único lleno de funciones que pretende " *hacerlo todo* ". El proyecto se encuentra en la etapa alfa, pero promete ofrecer un equilibrador de carga distribuido, un firewall y un enrutador diseñados específicamente para Kubernetes.

Para obtener más detalles, consulte la [página web de Kube-Router](https://www.kube-router.io/) .  
<https://www.kube-router.io/>





## Romana

Este es otro proyecto destinado a la automatización de redes y seguridad para aplicaciones nativas de la nube. Dirigido a grandes clústeres, topología compatible con IPAM e integración con clústeres de kops.

Para obtener más información, consulte la página web de [Romana](#) GitHub.

<https://github.com/romana/romana>



## Weave Net

Por lo general, se usa como un complemento para un clúster de Kubernetes habilitado para CNI.

Para obtener más información, consulte la página web de [Weave Net](#) .

<https://www.weave.works/oss/net/>

# Más herramientas de instalación

---

Dado que Kubernetes es, después de todo, como cualquier otra aplicación que instale en un servidor (ya sea físico o virtual), se pueden utilizar todos los sistemas de administración de configuración (por ejemplo, Chef, Puppet, Ansible, Terraform). Varias recetas están disponibles en Internet.

La mejor manera de aprender a instalar Kubernetes mediante comandos manuales paso a paso es examinar el [tutorial de Kelsey Hightower](https://github.com/kelseyhightower/kubernetes-the-hard-way) .

<https://github.com/kelseyhightower/kubernetes-the-hard-way>

## Ejemplos de herramientas de instalación



### Kubespray

kubespray está ahora en la incubadora de Kubernetes. Es un libro de jugadas avanzado de Ansible que le permite configurar un clúster de Kubernetes en varios sistemas operativos y utilizar diferentes proveedores de red. Alguna vez fue conocido como kargo.

Para obtener más información sobre [kubespray](https://github.com/kubernetes-sigs/kubespray) , consulte su página de GitHub.  
<https://github.com/kubernetes-sigs/kubespray>



## Kops

kops (Kubernetes Operations) le permite crear un clúster de Kubernetes en AWS a través de una única línea de comandos. También en versión beta para GKE y alfa para VMware.

Obtenga más información sobre [kops](https://github.com/kubernetes/kops) en su página de GitHub.  
<https://github.com/kubernetes/kops>



## kube-aws

kube-aws es una herramienta de línea de comandos que utiliza AWS Cloud Formation para aprovisionar un clúster de Kubernetes en AWS.

Para obtener más detalles sobre [kube-aws](https://kubernetes-incubator.github.io/kube-aws/) , consulte su página web.  
<https://kubernetes-incubator.github.io/kube-aws/>



## Kubicorn

kubicorn es una herramienta que aprovecha el uso de kubernetes para construir un clúster. Afirma no tener dependencia del DNS, se ejecuta en varios sistemas operativos y utiliza instantáneas para capturar un clúster y moverlo.

Para obtener más información, consulte la página web de [kubicorn](http://kubicorn.io/) . <http://kubicorn.io/>

## Consideraciones de instalación

---

Para comenzar el proceso de instalación, debe comenzar a experimentar con una implementación de un solo nodo. Este nodo único ejecutará todos los componentes de Kubernetes (por ejemplo, servidor de API, controlador, programador, kubelet y kube-proxy). Puede hacer esto con Minikube, por ejemplo.

Una vez que desee implementar en un clúster de servidores (físicos o virtuales), tendrá muchas opciones que tomar, al igual que con cualquier otro sistema distribuido:

- ¿Qué proveedor debo utilizar? ¿Una nube pública o privada? ¿Físico o virtual?
- ¿Qué sistema operativo debo utilizar? Kubernetes se ejecuta en la mayoría de los sistemas operativos (por ejemplo, Debian, Ubuntu, CentOS, etc.), además de en sistemas operativos optimizados para contenedores (por ejemplo, CoreOS, Atomic).
- ¿Qué solución de red debo utilizar? ¿Necesito una superposición?
- ¿Dónde debo ejecutar mi clúster etcd?
- ¿Puedo configurar nodos principales de alta disponibilidad (HA)?

Para obtener más información sobre cómo elegir las mejores opciones, puede leer la página de documentación de *Introducción* . <https://kubernetes.io/docs/setup/>

Con systemd convirtiéndose en el sistema de inicio dominante en Linux, sus componentes de Kubernetes terminarán ejecutándose como *archivos de unidad de systemd* en la mayoría de los casos. O se ejecutarán mediante un kubelet que se ejecuta en el nodo principal (es decir, **kubadm** ).

Los ejercicios de laboratorio de este curso se escribieron utilizando nodos de Google Compute Engine (GCE). Cada nodo tiene 2 vCPU y 7,5 GB de memoria, con Ubuntu 16.04. Los nodos más pequeños deberían funcionar, pero debería esperar una respuesta lenta. También son posibles otras imágenes del sistema operativo, pero puede haber ligeras diferencias en algunas salidas de comando. El uso de GCE requiere la creación de una cuenta e incurrirá en gastos si utiliza nodos del tamaño sugerido. Puede ver las páginas de *Introducción* para obtener más detalles. <https://cloud.google.com/compute/docs/quickstart-linux>

Amazon Web Services (AWS) es otro proveedor de nodos basados en la nube y requiere una cuenta; incurrirá en gastos por nodos del tamaño sugerido. Puede

encontrar videos e [información sobre cómo comenzar en línea](#).

<https://aws.amazon.com/es/getting-started/hands-on/launch-a-virtual-machine/>

Las máquinas virtuales como KVM, VirtualBox o VMware también se pueden utilizar para sistemas de laboratorio. Poner las VM en una red privada puede facilitar la resolución de problemas.

Finalmente, el uso de nodos bare metal con acceso a Internet también funcionará para los ejercicios de laboratorio.

## Configuraciones de implementación principales

---

En un nivel alto, tiene cuatro configuraciones de implementación principales:

- **Un solo nodo**

Con una implementación de un solo nodo, todos los componentes se ejecutan en el mismo servidor. Esto es excelente para probar, aprender y desarrollar en Kubernetes.

- **Un solo nodo principal, varios workers**

La adición de más workers, un único nodo principal y varios workers normalmente consistirá en una instancia de un solo nodo etcd que se ejecuta en el nodo principal con la API, el scheduler y el controller-manager.

- **Múltiples nodos principales con HA, varios workers**

Múltiples nodos principales en una configuración HA y varios workers agregan

más durabilidad al clúster. El servidor API estará dirigido por un load balancer, el load balancer y el controller-manager elegirán un líder (que se configura a través de flags). La configuración de etcd aún puede ser de un solo nodo.

- **HA etcd, nodos principales de HA, varios workers**

La configuración más avanzada y resistente sería un clúster de HA etcd, con nodos principales de HA y varios workers. Además, etcd se ejecutaría como un verdadero clúster, que proporcionaría HA y se ejecutaría en nodos separados de los nodos principales de Kubernetes.

Cuál de los cuatro usará dependerá de qué tan avanzado esté en su viaje de Kubernetes, pero también de cuáles son sus objetivos.

El uso de Kubernetes Federation también ofrece alta disponibilidad. Varios clústeres se unen con un plano de control común que permite el movimiento de recursos de un clúster a otro administrativamente o después de una falla. Si bien la Federación tiene algunos problemas, hay esperanzas de que la [v2](#) sea un producto más sólido.

## Archivo de unidad systemd para Kubernetes

---

En cualquiera de estas configuraciones, ejecutará algunos de los componentes como un demonio del sistema estándar. Como ejemplo, puede ver aquí un archivo de unidad systemd de muestra para ejecutar el controller-manager. El uso de **kubeadm** creará un

demonio del sistema para kubelet, mientras que el resto se implementará como contenedores:

```
- name: kube-controller-manager.service
```

```
    command: start
```

```
    content: |
```

```
        [Unit]
```

```
            Description=Kubernetes Controller Manager
```

```
            Documentation=https://github.com/kubernetes/kubernetes
```

```
            Requires=kube-apiserver.service
```

```
            After=kube-apiserver.service
```

```
        [Service]
```

```
            ExecStartPre=/usr/bin/curl -L -o
/opt/bin/kube-controller-manager -z
/opt/bin/kube-controller-manager
https://storage.googleapis.com/kubernetes-release/release/v1.7.6
/bin/linux/amd64/kube-controller-manager
```

```
            ExecStartPre=/usr/bin/chmod +x
/opt/bin/kube-controller-manager
```

```
            ExecStart=/opt/bin/kube-controller-manager \
```

```
--service-account-private-key-file=/opt/bin/kube-serviceaccount.
key \
```

```
            --root-ca-file=/var/run/kubernetes/apiserver.crt \
```



```
--master=127.0.0.1:8080 \
```

...

Este no es de ninguna manera un archivo unitario perfecto. Descarga el binario del controlador de la versión publicada de Kubernetes y establece algunos indicadores para que se ejecuten.

A medida que profundice en la configuración de cada componente, se familiarizará no solo con su configuración, sino también con las diversas opciones existentes, incluidas las de autenticación, autorización, HA, container runtime, etc. Espere que cambien.

Por ejemplo, el servidor API es altamente configurable. La documentación de Kubernetes proporciona más detalles sobre kube-apiserver .

<https://kubernetes.io/docs/reference/command-line-tools-reference/kube-apiserver/>

## Usando Hyperkube

---

Si bien puede ejecutar todos los componentes como demonios del sistema normales en archivos unitarios, también puede ejecutar el API server, el scheduler y el controller-manager como contenedores. Esto es lo que hace **kubeadm** .

Similar a minikube, hay un práctico binario todo en uno llamado **hyperkube** , que está disponible como una imagen de contenedor (por ejemplo,

`gcr.io/google_containers/hyperkube:v1.16.7` ). Esto está alojado por Google, por lo que es posible que deba agregar un nuevo repositorio para que Docker sepa dónde extraer el archivo. Se están actualizando y se están realizando cambios en los subcomandos. Consulte la **ayuda** de la versión actual para obtener más información. Puede encontrar la versión actual del software aquí:

<https://console.cloud.google.com/gcr/images/google-containers/GLOBAL/hyperkube> .

Este método de instalación consiste en ejecutar un kubelet como un demonio del sistema y configurarlo para leer en manifiestos que especifican cómo ejecutar los otros componentes (es decir, el API server, el scheduler, etcd, el scheduler). En estos manifiestos, se utiliza la imagen de hiperkubo. El kubelet los vigilará y se asegurará de que se reinicien si mueren.

Para tener una idea de esto, simplemente puede descargar la imagen del hiperkubo y ejecutar un contenedor para obtener ayuda sobre el uso:

```
$ docker run --rm gcr.io/google_containers/hyperkube:v1.16.7  
/hyperkube kube-apiserver --help
```

```
$ docker run --rm gcr.io/google_containers/hyperkube:v1.16.7  
/hyperkube kube-scheduler --help
```

```
$ docker run --rm gcr.io/google_containers/hyperkube:v1.16.7  
/hyperkube kube-controller-manager --help
```

Esta también es una muy buena manera de comenzar a aprender los distintos indicadores de configuración.

# Compilando desde la fuente

---

La [lista de versiones binarias](https://github.com/kubernetes/kubernetes/releases) <https://github.com/kubernetes/kubernetes/releases> está disponible en GitHub. Junto con **gcloud** , **minikube** y **kubeadmin** , cubren varios escenarios para comenzar con Kubernetes.

Kubernetes también se puede compilar desde la fuente con relativa rapidez. Puede clonar el repositorio de GitHub y luego usar **Makefile** para compilar los binarios. Puede compilarlos de forma nativa en su plataforma si tiene un entorno Golang configurado correctamente, o mediante contenedores Docker si está en un [host Docker](#) .

Para compilar de forma nativa con Golang, primero [instale Golang](#) . Los archivos de descarga y las instrucciones se pueden encontrar en línea. <https://golang.org/doc/install>

Una vez que Golang esté funcionando, puede clonar el repositorio de **kubernetes** , con un tamaño de alrededor de 500 MB. Cambie al directorio y use **make** :

```
$ cd $GOPATH
```

```
$ git clone https://github.com/kubernetes/kubernetes
```

```
$ cd kubernetes
```

```
$ make
```

En un host de Docker, clone el repositorio en cualquier lugar que desee y use el comando **make quick-release** . La compilación se realizará en contenedores Docker.

El directorio **\_output/bin** contendrá los binarios recién construidos.

Puede encontrar algunos materiales a través de [GitLab](#) , pero la mayoría de los recursos están en [GitHub](#) en este momento.

## **Lab 3.1 - Instalar Kubernetes**

## **Laboratorio 3.2 - Haga crecer el clúster**

## **Práctica de laboratorio 3.3: Finalización de la configuración del clúster**

## **Lab 3.4: Implementar una aplicación simple**

## **Práctica de laboratorio 3.5: Acceso desde fuera del clúster**

Ver: 95j9tla4zde3-LFS258-labs\_V2020-10-19.pdf

Felicitaciones por completar el *Capítulo 3 - Instalación y configuración* . Responda este cuestionario para comprobar su comprensión de los conceptos que ha aprendido hasta ahora.

## Pregunta 3.1

---

¿Para qué se **utiliza el** comando **kubeadm** ?

- **A** . Asignar un administrador al clúster
  - **B** . Iniciar un nuevo Pod
  - **C** . Crea un clúster y agrega nodos
  - **D** . Todas las anteriores
- 

## Pregunta 3.2

---

¿Cuál de los siguientes es el binario principal para trabajar con objetos de un clúster de Kubernetes?

- **A** . OpenStack
- **B** . Hacer
- **C** . adminCreate
- **D** . kubectl

## Pregunta 3.3

¿Cuántas redes de pod puede tener por clúster?

- **A** . 1
- **B** . 2
- **C** . 3
- **D** . 4

## Pregunta 3.4

---

El archivo `~/.kube/config` contiene \_\_\_\_\_.

- |                                  |
|----------------------------------|
| • <b>A.</b> Endpoints            |
| • <b>B.</b> SSL keys             |
| • <b>C.</b> Contexts             |
| • <b>D.</b> Todas las anteriores |

- 04. ARQUITECTURA DE KUBERNETES
- .

## Objetivos de aprendizaje

---

Al final de este capítulo, debería poder:

- Analice los componentes principales de un clúster de Kubernetes.
- Conozca los detalles del agente maestro kube-apiserver.
- Explique cómo la base de datos etcd mantiene el estado y la configuración del clúster.
- Estudie al agente local de kubelet.
- Examine cómo se utilizan los controladores para administrar el estado del clúster.
- Descubra qué es un pod para el clúster.
- Examine las configuraciones de red de un clúster.
- Analice los servicios de Kubernetes.

# Componentes principales

---

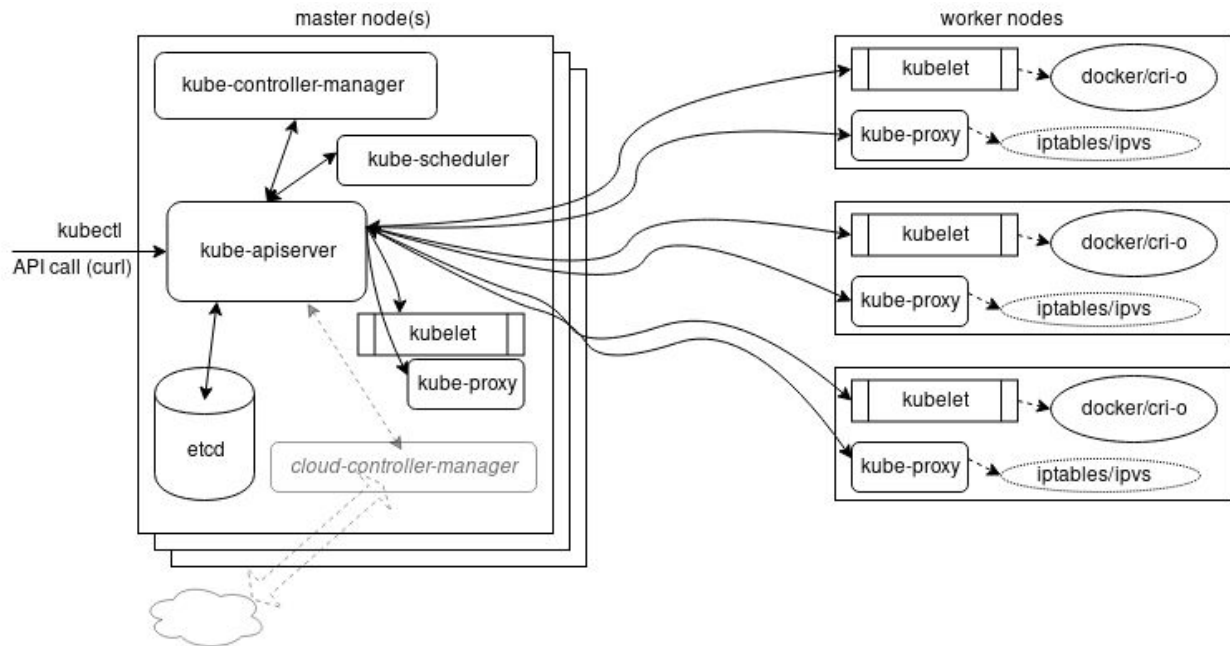
Kubernetes tiene los siguientes componentes principales:

- Nodos masters y workers
- Controllers
- Services
- Pods de containers
- Namespaces and quotas
- Network and policies
- Storage.

Un clúster de Kubernetes está formado por un nodo principal y un conjunto de nodos worker . El clúster se maneja a través de llamadas API a los Controllers, tanto del tráfico interior como exterior. A continuación, analizaremos más de cerca estos componentes.

La mayoría de los procesos se ejecutan dentro de un contenedor. Existen algunas diferencias, según el proveedor y la herramienta utilizada para construir el clúster.

Al actualizar un clúster, tenga en cuenta que cada uno de estos componentes está desarrollado para trabajar en conjunto por varios equipos. Se debe tener cuidado para asegurar una combinación adecuada de versiones.



## Arquitectura de Kubernetes

# Master Node

El master de Kubernetes ejecuta varios procesos de administrador y servidor para el clúster. Entre los componentes del nodo master se encuentran **kube-apiserver**, **kube-Scheduler** y la base de datos **etcd**. A medida que el software ha madurado, se han creado nuevos componentes para manejar necesidades específicas, como el **cloud-controller-manager**; maneja tareas que alguna vez fueron manejadas por **kube-controller-manager** para interactuar con otras herramientas, como Rancher o DigitalOcean para la administración y generación de informes de clústeres de terceros.

Hay varios complementos que se han vuelto esenciales para un clúster de producción típico, como los servicios DNS. Otras son soluciones de terceros en las que Kubernetes aún no ha desarrollado un componente local, como el registro a nivel de clúster y la supervisión de recursos.

Como concepto, los distintos módulos responsables de garantizar que el estado actual del clúster coincida con el estado deseado se denominan *control plane*.



Al crear un clúster con kubeadm, systemd administra el proceso de kubelet. Una vez que se ejecute, iniciará todos los pod que se encuentran en `/etc/kubernetes/manifests/` .

## Kube-apiserver

El **kube-apiserver** es fundamental para el funcionamiento del clúster de Kubernetes. Todas las llamadas, tanto internas como externas, se gestionan a través de este agente. Todas las acciones son aceptadas y validadas por este agente, y es la única conexión a la base de datos **etcd** . Valida y configura datos para objetos API y servicios de operaciones REST. Como resultado, actúa como un proceso maestro para todo el clúster y actúa como una interfaz del estado compartido del clúster.

Comenzando como una característica alfa en v1.16 es la capacidad de separar el tráfico iniciado por el usuario del tráfico iniciado por el servidor. Hasta que se desarrollen estas funciones, la mayoría de los complementos de red mezclan el tráfico, que tiene ramificaciones de rendimiento, capacidad y seguridad.

## Kube-scheduler

El **Kube-scheduler** usa un algoritmo para determinar qué nodo albergará un Pod de contenedores. El programador intentará ver los recursos disponibles (como los volúmenes) para vincular y luego intentará y volverá a intentar implementar el Pod según la disponibilidad y el éxito. Hay varias formas en que puede afectar el algoritmo, o en su lugar se puede utilizar un programador personalizado. También puede vincular un Pod a un nodo en particular, aunque el Pod puede permanecer en estado pendiente debido a otras configuraciones. Una de las primeras configuraciones a las que se hace referencia es si el Pod se puede implementar dentro de las restricciones de cuota actuales. Si es así, las manchas y tolerancias y las etiquetas de los Pods se utilizan junto con las de los nodos para determinar la ubicación adecuada.

Los [detalles del programador se pueden encontrar en GitHub](#) .

<https://raw.githubusercontent.com/kubernetes/kubernetes/master/pkg/scheduler/scheduler.go>

## etcd-database

El estado del clúster, la red y otra información persistente se mantiene en una **base de datos etcd** o, más exactamente, en un almacén de valores-clave de árbol b +. En lugar de buscar y cambiar una entrada, los valores siempre se añaden al final. Las copias anteriores de los datos se marcan para su posterior eliminación mediante un proceso de compactación. Funciona con **curl** y otras bibliotecas HTTP, y proporciona consultas de observación fiables.

Las solicitudes simultáneas para actualizar un valor viajan a través de **kube-apiserver** , que luego pasa la solicitud a **etcd** en una serie. La primera solicitud actualizará la base de datos. La segunda solicitud ya no tendría el mismo

número de versión, en cuyo caso **kube-apiserver** respondería con un error 409 al solicitante. No hay lógica más allá de esa respuesta en el lado del servidor, lo que significa que el cliente debe esperar esto y actuar sobre la negación para actualizar.

Existe una base de datos *maestra* junto con posibles *seguidores* . Se comunican entre sí de forma continua para determinar cuál será el maestro y determinar otro en caso de falla. Si bien es muy rápido y potencialmente duradero, ha habido algunos contratiempos con las nuevas herramientas, como **kubeadm** , y funciones como las actualizaciones de todo el clúster.

Si bien la mayoría de los objetos de Kubernetes están diseñados para desacoplarse, la excepción es un microservicio transitorio que se puede terminar sin mucha preocupación, **etc.** Tal como está, el estado persistente de todo el clúster debe estar protegido y asegurado. Antes de las actualizaciones o el mantenimiento, debe planificar realizar una copia de seguridad de **etcd** . El comando `etcdctl` permite **guardar** y **restaurar instantáneas** .

## Other Agents

El **kube-controller-manager** es un demonio de bucle de control de núcleo, que interactúa con el **kube-apiserver** para determinar el estado del cluster. Si el estado no coincide, el administrador se comunicará con el controlador necesario para que coincida con el estado deseado. Hay varios controladores en uso, como endpoints, namespace, y replication. La lista completa se ha ampliado a medida que Kubernetes ha madurado.

Permaneciendo en beta en v1.19, **Cloud-Controller-Manager ( ccm )** interactúa con agentes fuera de la nube. Maneja tareas una vez manejadas por **kube-controller-manager**. Esto permite cambios más rápidos sin alterar el proceso de control central de Kubernetes. Cada kubelet debe usar la **configuración** `--cloud-provider-external` pasada al binario. También puede desarrollar su propio ccm, que se puede implementar como un daemonset como una implementación en el árbol o como una instalación independiente fuera del árbol. El **cloud-controller-manager** es un agente opcional que requiere algunos pasos para habilitarlo. Puede obtener más información sobre [Cloud-Controller-Manager en línea](https://kubernetes.io/docs/tasks/administer-cluster/running-cloud-controller/).  
<https://kubernetes.io/docs/tasks/administer-cluster/running-cloud-controller/>

Para manejar consultas de DNS, descubrimiento de servicios de Kubernetes y otras funciones, el servidor CoreDNS ha reemplazado a **kube-dns** . Usando cadenas de complementos, uno de los muchos proporcionados o personalizados, el servidor se puede ampliar fácilmente.

# Worker Nodes

Todos los nodos ejecutan kubelet y kube-proxy, así como el motor de contenedores, como Docker o cri-o. Se implementan otros demonios de administración para vigilar estos agentes o brindar servicios que aún no se incluyen con Kubernetes.

El kubelet interactúa con el Docker Engine subyacente también instalado en todos los nodos y se asegura de que los contenedores que deben ejecutarse se estén ejecutando. El kube-proxy se encarga de gestionar la conectividad de la red a los contenedores. Lo hace mediante el uso de entradas de iptables. También tiene el modo de espacio de usuario, en el que supervisa los Services y Endpoints mediante un puerto aleatorio para el tráfico de proxy y una función alfa de ipvs.

También puede ejecutar una alternativa al motor Docker: cri-o o algunos motores menos usados. Para saber cómo puede hacerlo, debe consultar la documentación en línea. En versiones futuras, es muy probable que Kubernetes admita motores de tiempo de ejecución de contenedores adicionales.

Supervisord es un monitor de procesos liviano que se utiliza en entornos Linux tradicionales para monitorear y notificar sobre otros procesos. En un clúster que no es systemd, este demonio se puede usar para monitorear los procesos de kubelet y docker. Intentará reiniciarlos si fallan y registrar eventos. Si bien no es parte de una instalación típica, algunos pueden agregar este monitor para generar informes adicionales.

Kubernetes aún no tiene registros de todo el clúster. En su lugar, se utiliza otro proyecto de CNCF, llamado [Fluentd](https://www.fluentd.org/) . <https://www.fluentd.org/> . Cuando se implementa, proporciona una capa de registro unificada para el clúster, que filtra, almacena en búfer y enruta los mensajes.

Las métricas de todo el clúster son otra área con funcionalidad limitada. El metrics-server SIG proporciona un uso básico de la CPU y la memoria de los nodos y pods. Para obtener más métricas, muchos usan el proyecto Prometheus.

# Kubelet

---

El agente de kubelet es el que más pesa en los cambios y la configuración en los nodos worker. Acepta las llamadas a la API para especificaciones de pod (un `PodSpec` es un archivo JSON o YAML que describe un pod). Funcionará para configurar el nodo local hasta que se cumpla la especificación.

Si un Pod requiere acceso al almacenamiento, Secretos o ConfigMaps, el kubelet garantizará el acceso o la creación. También devuelve el estado al kube-apiserver para una eventual persistencia.

- Utiliza `PodSpec`
- Monta volúmenes en Pod
- Descargas secretos
- Pasa la solicitud al motor de contenedores local
- Informa el estado de los pods y del nodo al clúster.

Kubelet llama a otros componentes, como el Administrador de topología, que usa *sugerencias* de otros componentes para configurar asignaciones de recursos con reconocimiento de topología, como CPU y aceleradores de hardware. Como función alfa, no está habilitada de forma predeterminada.

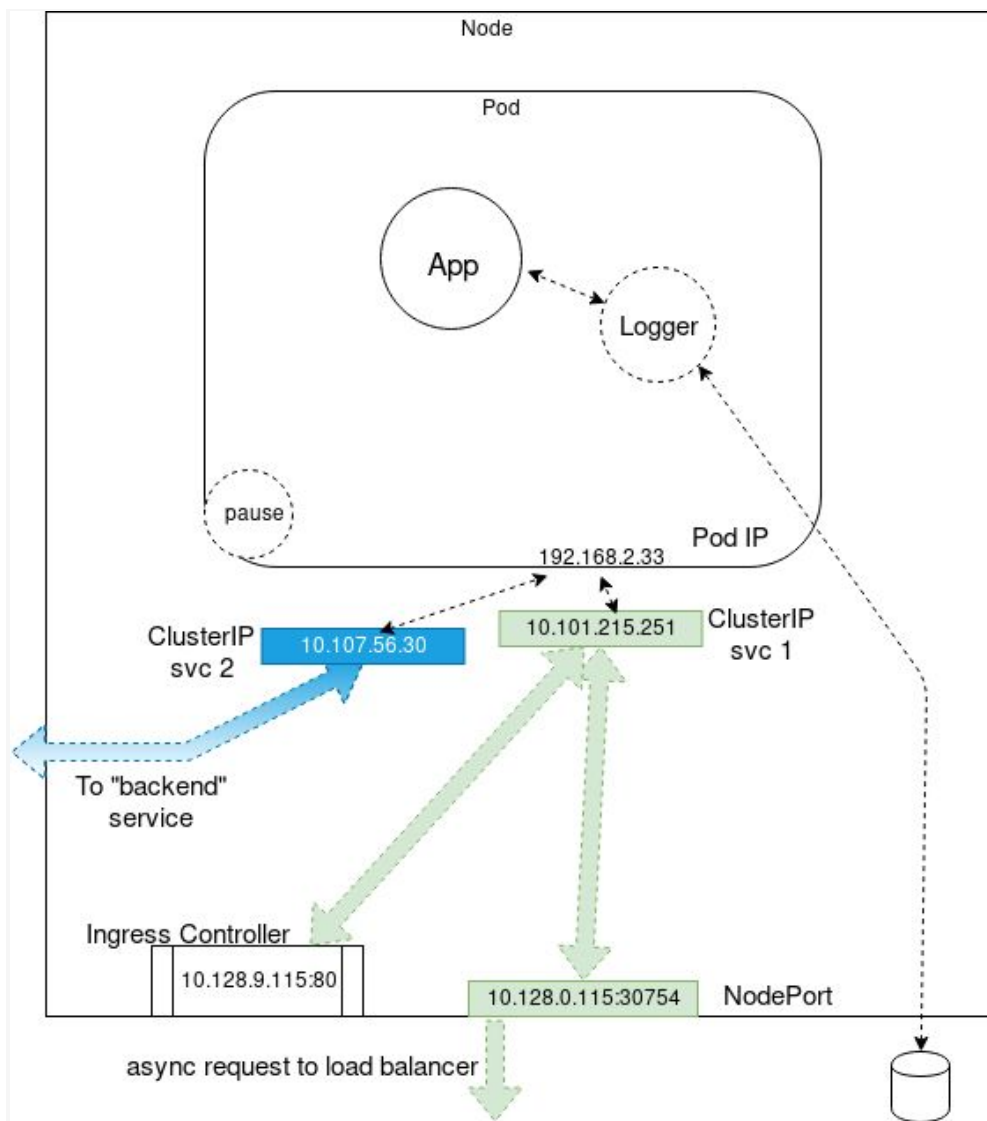
## Services

Con cada objeto y agente desacoplado, necesitamos un agente flexible y escalable que conecte los recursos y se reconecte, en caso de que algo muera y se genere un reemplazo. Cada servicio es un microservicio que maneja una parte particular de tráfico, como un solo NodePort o un LoadBalancer para distribuir las solicitudes entrantes entre muchos pods.

Un servicio también maneja políticas de acceso para solicitudes entrantes, útiles para el control de recursos, así como para la seguridad.

- Conectar Pods juntos
- Exponer pods a Internet
- Configuración de desacoplamiento
- Defina la política de acceso al pod.

Podemos utilizar un servicio para conectar un pod a otro o al exterior del clúster.



**Red de servicio**

Este gráfico muestra un pod con un contenedor principal, app, con un sidecar Logger opcional. También se ve el pause container, que utiliza el clúster para reservar la dirección IP en el namespace antes de iniciar los otros pods. Este contenedor no se ve desde Kubernetes, pero se puede ver usando **docker** y **crictl** .

Este gráfico también muestra una ClusterIP que se utiliza para conectarse dentro del clúster, no la IP del clúster. Como muestra el gráfico, esto se puede usar para conectarse a un NodePort para fuera del clúster, un IngressController o proxy, u otro pod o pods "backend".

## Controllers

Un concepto importante para la orquestación es el uso de controllers. Varios controllers se envían con Kubernetes, y usted también puede crear los suyos propios. Una vista simplificada de un controlador es un agente, o *Informer*, y una tienda secundaria. Usando una cola DeltaFIFO, se comparan el origen y el flujo descendente. Un proceso de loop recibe un `obj` u objeto, que es una matriz de deltas de la cola FIFO. Siempre que el delta no sea del tipo `Deleted`, la lógica del controlador se usa para crear o modificar algún objeto hasta que coincida con la especificación.

El *informador*, que utiliza el servidor API como fuente, solicita el estado de un objeto a través de una llamada API. Los datos se almacenan en caché para minimizar las transacciones del servidor API. Un agente similar es *SharedInformer* ; Los objetos a menudo son utilizados por muchos otros objetos. Crea una caché compartida del estado para múltiples solicitudes.

Un *Workqueue* utiliza una clave para repartir tareas a varios workers. Por lo general, se utilizan las colas de trabajo Go estándar de limitación de velocidad, retraso y cola de tiempo.

Los `endpoints` , `namespace`, and `serviceaccounts` controllers administran los recursos del mismo nombre para los pods.

# Pods

El objetivo de Kubernetes es orquestar el ciclo de vida de un contenedor. No interactuamos con contenedores particulares. En cambio, la unidad más pequeña con la que podemos trabajar es un *Pod* . Algunos dirían una vaina de ballenas o guisantes en una vaina. Debido a los recursos compartidos, el diseño de un Pod generalmente sigue una arquitectura de un proceso por contenedor.

Los contenedores de un Pod se inician en paralelo. Como resultado, no hay forma de determinar qué contenedor estará disponible primero dentro de un pod. El uso de **InitContainers** puede ordenar el inicio, hasta cierto punto. Para admitir un solo proceso que se ejecuta en un contenedor, es posible que necesite un registro, un proxy o un adaptador especial. Estas tareas a menudo las manejan otros contenedores en el mismo pod.

Solo hay una dirección IP por Pod, para casi todos los complementos de red. Si hay más de un contenedor en un pod, deben compartir la IP. Para comunicarse entre sí, pueden usar IPC, la interfaz de bucle invertido o un sistema de archivos compartido.

Si bien los pods a menudo se implementan con un contenedor de aplicaciones en cada uno, una razón común para tener varios contenedores en un pod es el registro. Puede encontrar el término **sidecar** para un contenedor dedicado a realizar una tarea auxiliar, como manejar registros y responder a solicitudes, ya que el contenedor de la aplicación principal puede no tener esta capacidad. El término **sidecar** , como **ambassador** y **adapter**, no tiene una configuración especial, pero se refiere al concepto de qué se incluyen las cápsulas secundarias.

# Rewrite Legacy Applications

## Reescribir aplicaciones heredadas

El traslado de aplicaciones heredadas a Kubernetes a menudo plantea la pregunta de si la aplicación debe estar en contenedores tal como está o reescribirse como un microservicio transitorio desacoplado. El costo y el tiempo de reescritura de aplicaciones heredadas pueden ser altos, pero también es valioso aprovechar la flexibilidad de Kubernetes.

## Containers

Si bien la orquestación de Kubernetes no permite la manipulación directa a nivel de contenedor, podemos administrar los recursos que los contenedores pueden consumir.

En la sección de recursos de PodSpec , puede pasar parámetros que se pasarán al tiempo de ejecución del contenedor en el nodo programado:

```
resources:
```

```
  limits:
```

```
    cpu: "1"
```

```
    memory: "4Gi"
```

```
  requests:
```

```
    cpu: "0.5"
```

```
    memory: "500Mi"
```



Otra forma de administrar el uso de recursos de los contenedores es mediante la creación de un objeto `ResourceQuota`, que permite establecer límites rígidos y flexibles en un namespace. Las cuotas permiten la gestión de más recursos además de la CPU y la memoria y permite limitar varios objetos.

Una función beta en la versión 1.12 usa el campo `scopeSelector` en la especificación de cuota para ejecutar un pod con una prioridad específica si tiene el `priorityClassName` apropiado en su especificación de pod.

## Init Containers

No todos los contenedores son iguales. Los contenedores estándar se envían al motor de contenedores al mismo tiempo y pueden comenzar en cualquier orden.

LivenessProbes, ReadinessProbes y StatefulSets se pueden usar para determinar el orden, pero pueden agregar complejidad. Otra opción puede ser un **Init container**, que debe completarse antes de que se inicien los contenedores de aplicaciones. Si el contenedor de inicio falla, se reiniciará hasta que se complete, sin que se ejecute el contenedor de la aplicación.

El init container puede tener una vista diferente de la configuración de seguridad y almacenamiento, lo que permite utilizar utilidades y comandos que la aplicación no podría utilizar. Los init container pueden contener código o utilidades que no están en una aplicación. También tiene una seguridad independiente de los contenedores de aplicaciones.

El siguiente código ejecutará el init container hasta que el comando `ls` tenga éxito; entonces se iniciará el contenedor de la base de datos.

`spec:`

`containers:`

`- name: main-app`

`image: databaseD`

`initContainers:`

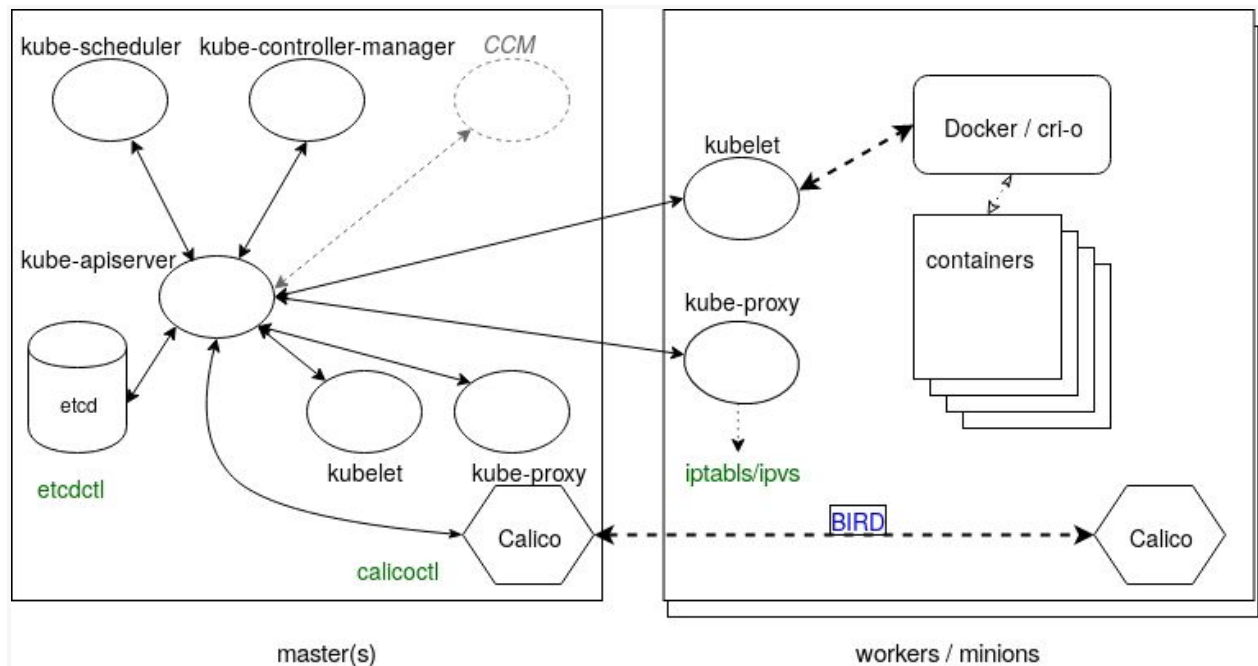
`- name: wait-database`

`image: busybox`

`command: ['sh', '-c', 'until ls /db/dir ; do sleep 5; done;']`

## Revisión de componente

Ahora que hemos visto algunos de los componentes, echemos otro vistazo con algunas de las conexiones que se muestran. No todas las conexiones se muestran en el diagrama siguiente. Tenga en cuenta que todos los componentes se comunican con kube-apiserver. Solo kube-apiserver se comunica con la base de datos etcd.



## Revisión arquitectónica de Kubernetes

También vemos algunos comandos, que es posible que debamos instalar por separado para trabajar con varios componentes. Hay un comando **etcdctl** para interrogar a la base de datos y **calicoctl** para ver más sobre cómo está configurada la red. Podemos ver a Felix, que es el agente principal de Calico en cada máquina. Este agente, o demonio, es responsable de la supervisión y gestión de la interfaz, la programación de rutas, la configuración de ACL y los informes de estado.

BIRD es un demonio de enrutamiento IP dinámico utilizado por Felix para leer el estado del enrutamiento y distribuir esa información a otros nodos del clúster. Esto permite que un cliente se conecte a cualquier nodo y, finalmente, se conecte a la carga de trabajo en un contenedor, incluso si no se contactó originalmente con el nodo.

# Nodo

Un nodo es un objeto API creado fuera del clúster que representa una instancia. Si bien un master debe ser Linux, los nodos workers también pueden ser Microsoft Windows Server 2019. Una vez que el nodo tiene instalado el software necesario, se ingiere en el Api Server.

Por el momento, puede crear un nodo master con `kubeadm init` y nodos de workers pasando `join`. En un futuro próximo, se podrán unir nodos masters secundarios y / o nodos etcd.

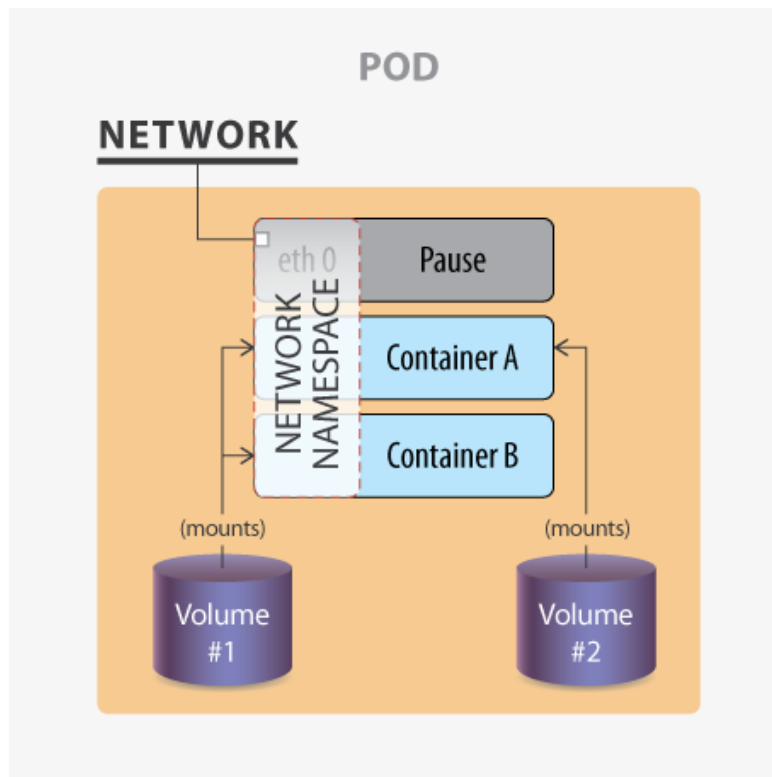
Si kube-apiserver no puede comunicarse con el kubelet en un nodo durante 5 minutos, el `NodeLease` predeterminado programará el nodo para su eliminación y `NodeStatus` cambiará de *listo*. Los pods serán desalojados una vez que se restablezca la conexión. El clúster ya no los elimina a la fuerza ni los reprograma.

Cada objeto de nodo existe en el espacio de nombres `kube-node-lease`. Para eliminar un nodo del clúster, primero use `kubectl delete node <nombre-nodo>` para eliminarlo del API server. Esto hará que los pods sean evacuados. Luego, use `kubeadm reset` para eliminar la información específica del clúster. Es posible que también deba eliminar la información de iptables, dependiendo de si planea reutilizar el nodo.

Para ver el uso de CPU, memoria y otros recursos, solicitudes y límites, use el comando `kubectl describe node`. La salida mostrará la capacidad y los pods permitidos, así como detalles sobre los pods actuales y la utilización de recursos.

# IP única por pod

Un **pod** representa un grupo de contenedores coubicados con algunos volúmenes de datos asociados. Todos los contenedores de un pod comparten el mismo espacio de nombres de red.



## Pod Network

El gráfico muestra un pod con dos contenedores, A y B, y dos volúmenes de datos, 1 y 2. Los contenedores A y B comparten el espacio de nombres de red de un tercer contenedor, conocido como *pause container*. El *pause container* se usa para obtener una dirección IP, luego todos los contenedores en el pod usarán su namespace de red. Los volúmenes 1 y 2 se muestran para completar.

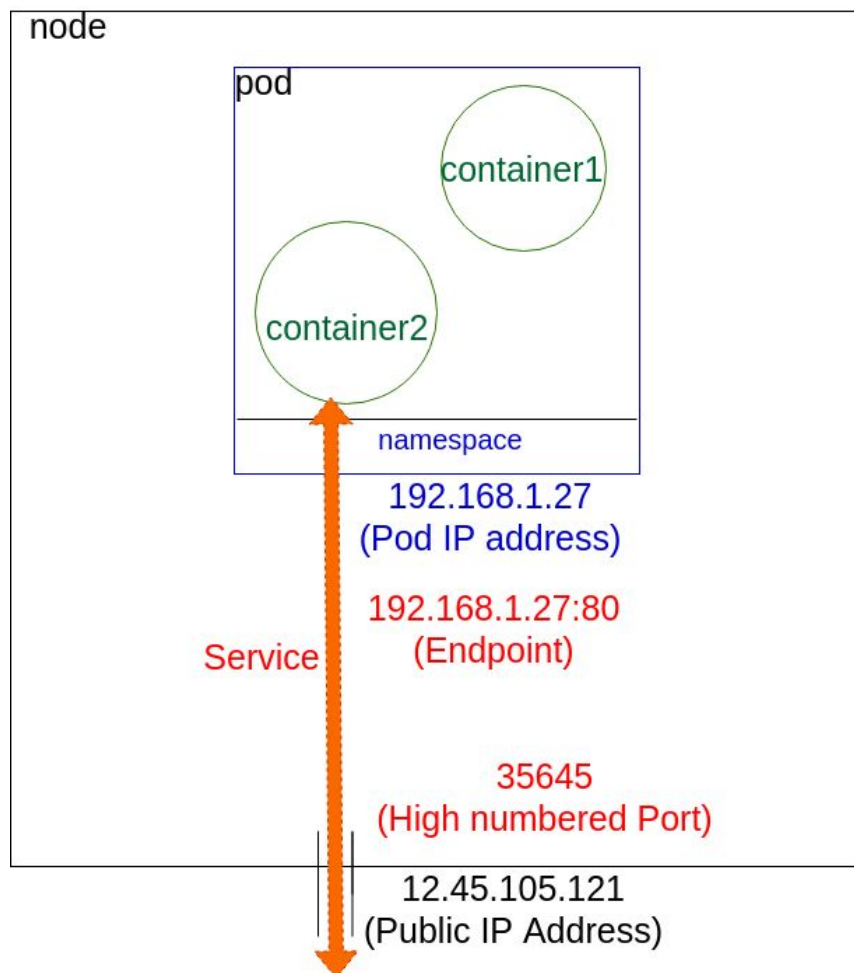
Para comunicarse entre sí, los contenedores dentro de los pods pueden usar la interfaz de loopback, escribir en archivos en un sistema de archivos común o mediante comunicación entre procesos (IPC). Ahora hay un complemento de red de HPE Labs

que permite múltiples direcciones IP por pod, pero esta característica no ha superado este nuevo complemento.

Comenzando como una característica alfa en 1.16 es la capacidad de usar IPv4 e IPv6 para pods y servicios. En la versión actual, al crear un servicio, debe crear la red para cada familia de direcciones por separado.

## Contenedor a camino exterior

Este gráfico muestra un nodo con un solo pod de contenedor doble. Un servicio NodePort conecta el Pod a la red externa.



Container Network

Aunque hay dos contenedores, comparten el mismo namespace y la misma dirección IP, que sería configurada por `kubect` trabajando con `kube-proxy`. La dirección IP se asigna antes de que se inicien los contenedores y se insertará en los contenedores. El contenedor tendrá una interfaz como `eth0@tun10`. Esta IP está configurada durante la vida útil del pod.

El endpoint se crea al mismo tiempo que el service. Tenga en cuenta que utiliza la dirección IP del módulo, pero también incluye un puerto. El servicio conecta el tráfico de red desde un puerto de alto número de nodos hasta el punto final utilizando iptables con ipvs en el camino. Kube-controller-manager maneja los bucles de vigilancia para monitorear la necesidad de endpoints y services, así como cualquier actualización o eliminación.

## Configuración de red

Hacer que todos los componentes anteriores se ejecuten es una tarea común para los administradores de sistemas que están acostumbrados a la administración de la configuración. Pero, para obtener un clúster de Kubernetes completamente funcional, la red también deberá configurarse correctamente.

Se puede ver una explicación detallada sobre el modelo de red de Kubernetes en la [página Cluster Networking](#) en la documentación de Kubernetes.

<https://kubernetes.io/docs/concepts/cluster-administration/networking/>

Si tiene experiencia en la implementación de máquinas virtuales (VM) basadas en soluciones IaaS, esto le resultará familiar. La única advertencia es que, en Kubernetes, la unidad de cálculo más baja no es un contenedor, sino lo que llamamos un pod.

Un pod es un grupo de contenedores que comparten la misma dirección IP. Desde una perspectiva de red, un pod puede verse como una máquina virtual de hosts físicos. La

red necesita asignar direcciones IP a los pods y debe proporcionar rutas de tráfico entre todos los pods en cualquier nodo.

Los tres principales desafíos de redes que se deben resolver en un sistema de orquestación de contenedores son:

- Comunicación acoplada de contenedor a contenedor (resuelto por el concepto de pod).
- Comunicación pod-to-pod.
- Comunicación externa al pod (resuelto por el concepto de servicios, que discutiremos más adelante).

Kubernetes espera que la configuración de la red permita que la comunicación de pod a pod esté disponible; no lo hará por ti.

Tim Hockin, uno de los desarrolladores principales de Kubernetes, ha creado una plataforma de diapositivas muy útil para comprender las redes de Kubernetes: *una guía ilustrada para las redes de Kubernetes*.

<https://speakerdeck.com/thockin/illustrated-guide-to-kubernetes-networking>

## Archivo de configuración de red CNI

Para proporcionar redes de contenedores, Kubernetes está estandarizando la especificación Container Network Interface (CNI). Desde la v1.6.0, el objetivo de kubeadm (la herramienta de arranque del clúster de Kubernetes) ha sido utilizar CNI, pero es posible que deba volver a compilar para hacerlo.

CNI es una especificación emergente con bibliotecas asociadas para escribir complementos que configuran la red de contenedores y eliminan los recursos asignados cuando se elimina el contenedor. Su objetivo es proporcionar una interfaz común entre las diversas soluciones de red y tiempos de ejecución de contenedores. Como la especificación CNI es independiente del idioma, hay muchos complementos de Amazon ECS, SR-IOV, Cloud Foundry y más.



Con CNI, puede escribir un archivo de configuración de red:

```
{  
  "cniVersion": "0.2.0",  
  "name": "mynet",  
  "type": "bridge",  
  "bridge": "cni0",  
  "isGateway": true,  
  "ipMasq": true,  
  "ipam": {  
    "type": "host-local",  
    "subnet": "10.22.0.0/16",  
    "routes": [  
      { "dst": "0.0.0.0/0" }  
    ]  
  }  
}
```

Esta configuración define un puente de Linux estándar llamado **cni0** , que proporcionará direcciones IP en la subred 10.22.0.0/16. El complemento de puente configurará las interfaces de red en los espacios de nombres correctos para definir correctamente la red de contenedores.

El **archivo README** principal del repositorio CNI GitHub tiene más información.

<https://github.com/containernetworking/cni>

# Comunicación Pod-to-Pod

Si bien se puede utilizar un complemento CNI para configurar la red de un pod y proporcionar una única IP por pod, CNI no lo ayuda con la comunicación pod-a-pod entre nodos.

El requisito de Kubernetes es el siguiente:

- 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).

Básicamente, todas las IP involucradas (nodos y pods) son enrutables sin NAT. Esto se puede lograr en la infraestructura de red física si tiene acceso a ella (por ejemplo, GKE). O, esto se puede lograr con una superposición definida por software con soluciones como:

- Weave
- Flannel
- Calico
- Romana.

Consulte esta [página de documentación](#) o la [lista de complementos de red](#) para obtener una lista más completa.

<https://kubernetes.io/docs/concepts/cluster-administration/networking/>

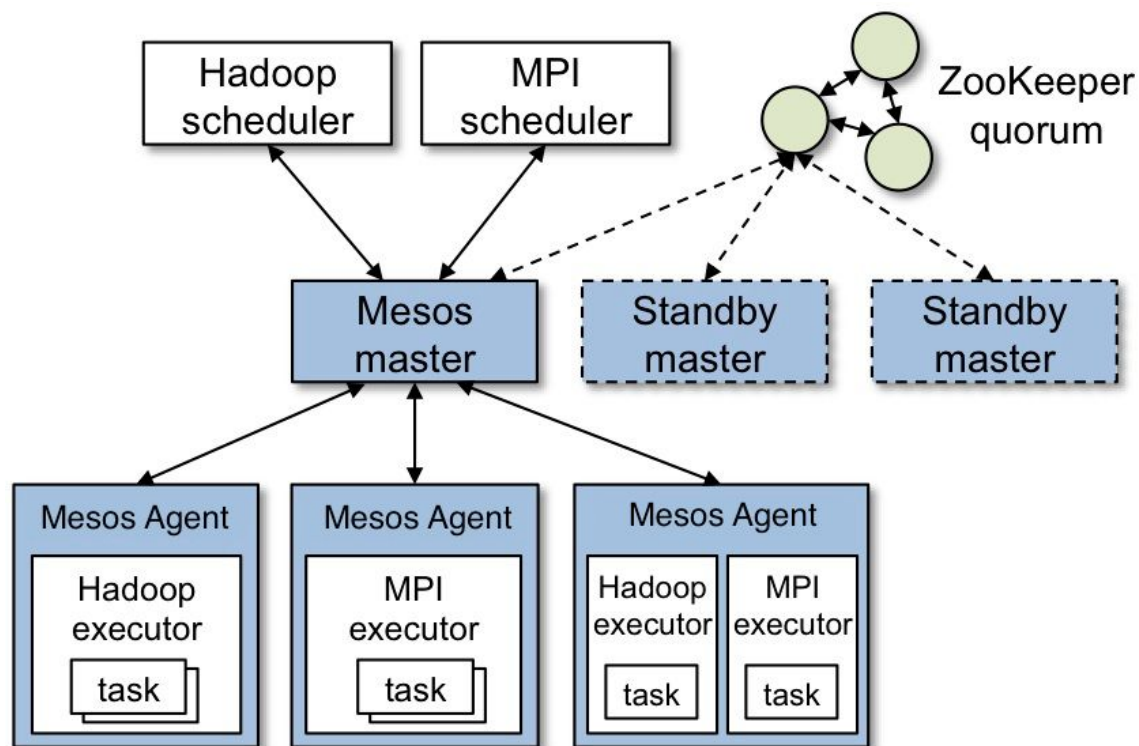
<https://kubernetes.io/docs/concepts/cluster-administration/addons/>

# Mesos

En un nivel alto, no hay nada diferente entre Kubernetes y otros sistemas de clústeres.

Un administrador central expone una API, un programador coloca las cargas de trabajo en un conjunto de nodos y el estado del clúster se almacena en una capa persistente.

Por ejemplo, puede comparar Kubernetes con Mesos y verá las similitudes. En Kubernetes, sin embargo, la capa de persistencia se implementa con etcd, en lugar de Zookeeper para Mesos.



## Arquitectura de Mesos

The Apache Software Foundation

Obtenido del [sitio web de Mesos](http://mesos.apache.org/documentation/latest/architecture/)

<http://mesos.apache.org/documentation/latest/architecture/>

También debe considerar sistemas como OpenStack y CloudStack. Piense en lo que se ejecuta en su nodo principal y en lo que se ejecuta en sus nodos de trabajo. ¿Cómo mantienen el estado? ¿Cómo manejan las redes? Si está familiarizado con esos sistemas, Kubernetes no parecerá tan diferente.

Lo que realmente distingue a Kubernetes son sus características orientadas hacia la tolerancia a fallas, el autodescubrimiento y el escalado, junto con una mentalidad que se basa exclusivamente en API.

## **Práctica de laboratorio 4.1 -**

## **Mantenimiento básico de nodos**

## **Práctica de laboratorio 4.2: Trabajo con restricciones de memoria y CPU**

## **Práctica de laboratorio 4.3: Límites de recursos para un espacio de nombres**

Ver: 95j9tla4zde3-LFS258-labs\_V2020-10-19.pdf

## Question 4.1

---

¿Cuál es el objeto o unidad más pequeño con el que podemos trabajar en Kubernetes?

- **A.** Container
- **B.** Pod
- **C.** ReplicaSet
- **D.** Deployment

## Question 4.2

¿Cuántas direcciones IP se pueden configurar para un Pod?

- **A.** 1
- **B.** 2
- **C.** 8
- **D.** None

## Question 4.3

¿Cuál es el agente de configuración principal en un servidor maestro?

- **A.** watcher
- **B.** kubelet
- **C.** loop
- **D.** kube-apiserver

## Question 4.4

¿Cuál es el agente principal en un nodo trabajador?

- **A.** watcher
- **B.** kubelet
- **C.** loop
- **D.** kube-apiserver

## Question 4.5

¿Qué objeto conecta otros recursos y maneja el tráfico de entrada y salida?

- **A.** Pod
- **B.** Controller
- **C.** etcd
- **D.** Service

## - 05 . API Y ACCESO

- =

# Objetivos de aprendizaje

---

Al final de este capítulo, debería poder:

- Comprender la arquitectura basada en API REST.
- Trabaja con anotaciones.
- Comprende una plantilla de Pod simple.
- Utilice kubectl con mayor detalle para solucionar problemas.
- Separe los recursos del clúster mediante espacios de nombres.

## Acceso API

Kubernetes tiene una potente API basada en REST. Toda la arquitectura está impulsada por API. Saber dónde encontrar los endpoints de los recursos y comprender cómo cambia la API entre versiones puede ser importante para las tareas administrativas en curso, ya que hay muchos cambios y un crecimiento en curso. A partir de la v1.16, el servidor de API ya no acepta los objetos obsoletos.

Como aprendimos en el capítulo *Arquitectura de Kubernetes* , el agente principal para la comunicación entre los agentes del clúster y desde fuera del clúster es kube-apiserver. Una consulta `curl` al agente expondrá los grupos de API actuales. Los grupos pueden tener múltiples versiones, que evolucionan independientemente de otros grupos y siguen un formato de nombre de dominio con varios nombres reservados, como dominios de una sola palabra, el grupo vacío y cualquier nombre que termine en `.k8s.io` .

# RESTful

kubectl realiza llamadas a la API en su nombre, respondiendo a los verbos HTTP típicos ( `GET` , `POST` , `DELETE` ). También puede realizar llamadas de forma externa, utilizando `curl` u otro programa. Con los certificados y claves adecuados, puede realizar solicitudes o pasar archivos JSON para realizar cambios de configuración.

```
$ curl --cert userbob.pem --key userBob-key.pem \
--cacert /path/to/ca.pem \
https://k8sServer:6443/api/v1/pods
```

La capacidad de hacerse pasar por otros usuarios o grupos, sujeto a la configuración de RBAC, permite una autenticación de anulación manual. Esto puede resultar útil para depurar las políticas de autorización de otros usuarios.

## Comprobación de acceso

Si bien hay más detalles sobre la seguridad en un capítulo posterior, es útil verificar las autorizaciones actuales, tanto como administrador como como otro usuario. A continuación, se muestra lo que podría hacer el usuario `bob` en el `default` namespace y el `developer` namespace , utilizando el subcomando `auth can-i` para realizar consultas:

```
$ kubectl auth can-i create deployments
```

```
yes
```

```
$ kubectl auth can-i create deployments --as bob
```

```
no
```



```
$ kubectl auth can-i create deployments --as bob --namespace developer
```

```
yes
```

Actualmente hay tres API que se pueden aplicar para establecer quién y qué se puede consultar:

- **SelfSubjectAccessReview**  
opinión acceso para cualquier usuario, útiles para delegar a otros.
- **LocalSubjectAccessReview**  
Review está restringida a un namespace específico.
- **SelfSubjectRulesReview**  
Una revisión que muestra permite acciones de un usuario dentro de un namespace particular.

El uso de **reconcile** permite verificar la autorización necesaria para crear un objeto a partir de un archivo. Ningún resultado indica que se permitiría la creación.

## Simultaneidad optimista

La serialización predeterminada para las llamadas a la API debe ser JSON. Hay un esfuerzo por utilizar la serialización *protobuf* de Google , pero sigue siendo experimental. Si bien podemos trabajar con archivos en formato YAML, se convierten desde y hacia JSON.

Kubernetes usa el valor **resourceVersion** para determinar las actualizaciones de la API e implementar la simultaneidad optimista. En otras palabras, un objeto no se bloquea desde el momento en que se lee hasta que se escribe.

En cambio, tras una llamada actualizada a un objeto, se verifica `resourceVersion` y se devuelve un `409 CONFLICT`, en caso de que el número haya cambiado. La `resourceVersion` se respalda actualmente mediante el parámetro `modifiedIndex` en la base de datos etcd y es única para el namespace, kind y server. Las operaciones que no cambian un objeto, como `WATCH` o `GET`, no actualizan este valor.

## Usar anotaciones

Las etiquetas se utilizan para trabajar con objetos o colecciones de objetos; las anotaciones no lo son.

En cambio, las anotaciones permiten incluir metadatos con un objeto que puede ser útil fuera de la interacción del objeto de Kubernetes. Al igual que las etiquetas, son clave para los mapas de valor. También pueden contener más información y más información legible por humanos que las etiquetas.

Tener este tipo de metadatos se puede usar para rastrear información como una marca de tiempo, punteros a objetos relacionados de otros ecosistemas o incluso un correo electrónico del desarrollador responsable de la creación de ese objeto.

De lo contrario, los datos de las anotaciones podrían guardarse en una base de datos exterior, pero eso limitaría la flexibilidad de los datos. Cuanto más se incluyan estos metadatos, más fácil será integrar herramientas de administración e implementación o bibliotecas cliente compartidas.

Por ejemplo, para anotar solo pods dentro de un espacio de nombres, puede sobrescribir la anotación y, finalmente, eliminarla:

```
$ kubectl annotate pods --all description='Production Pods' -n prod
```

```
$ kubectl annotate --overwrite pod webpod description="Old Production Pods" -n prod
```

```
$ kubectl -n prod annotate pod webpod description-
```

## Simple Pod

Como se discutió anteriormente, un Pod es la unidad de procesamiento más baja y el objeto individual con el que podemos trabajar en Kubernetes. Puede ser un solo contenedor, pero a menudo constará de un contenedor de aplicación principal y uno o más contenedores de soporte.

A continuación, se muestra un ejemplo de un manifiesto de pod simple en formato YAML. Puede ver la **apiVersion** (debe coincidir con el grupo de API existente), el **tipo** (el tipo de objeto a crear), los **metadatos** (al menos un nombre) y su **especificación** (qué crear y parámetros), que definen el contenedor que realmente se ejecuta en este pod:

```
apiVersion: v1

kind: Pod

metadata:
  name: firstpod

spec:
  containers:
    - image: nginx
      name: stan
```

Puede usar el comando `kubectl create` para crear este pod en Kubernetes. Una vez creado, puede comprobar su estado con `kubectl get pods`. La salida se omite para ahorrar espacio:

```
$ kubectl create -f simple.yaml

$ kubectl get pods

$ kubectl get pod firstpod -o yaml

$ kubectl get pod firstpod -o json
```

## Administra los recursos de la API con kubectl

Kubernetes expone recursos a través de llamadas a la API RESTful, lo que permite que todos los recursos se administren a través de HTTP, JSON o incluso XML, siendo el protocolo típico HTTP. El estado de los recursos se puede cambiar utilizando verbos HTTP estándar (por ejemplo , `GET` , `POST` , `PATCH` , `DELETE` , etc.).

**kubectl** tiene un argumento en modo detallado que muestra detalles de dónde obtiene el comando y actualiza la información. Otra salida incluye comandos `curl` que puede usar para obtener el mismo resultado. Si bien la verbosidad acepta niveles desde cero hasta cualquier número, actualmente no hay un valor de verbosidad mayor que diez. Puede consultar esto para `kubectl get` . La salida a continuación se ha formateado para mayor claridad:

```
$ kubectl --v=10 get pods firstpod

....

I1215 17:46:47.860958 29909 round_tripper.go:417]

curl -k -v -XGET -H "Accept: application/json"

-H "User-Agent: kubectl/v1.8.5 (linux/amd64) kubernetes/cce11c6"

https://10.128.0.3:6443/api/v1/namespaces/default/pods/firstpod

....
```

Si elimina este pod, verá que el método HTTP cambia de `XGET` a `XDELETE` .

```
$ kubectl --v=10 delete pods firstpod

....

I1215 17:49:32.166115 30452 round_tripper.go:417]
curl -k -v -XDELETE -H "Accept: application/json, */*"
-H "User-Agent: kubectl/v1.8.5 (linux/amd64) kubernetes/cce11c6"
https://10.128.0.3:6443/api/v1/namespaces/default/pods/firstpod
```

## Acceso desde fuera del clúster

La herramienta principal utilizada desde la línea de comando será **kubectl** , que llama a **curl** en su nombre. También puede usar el comando **curl** desde fuera del clúster para ver o realizar cambios.

La información básica del servidor, con la información del certificado TLS redactada, se puede encontrar en la salida de

```
$ kubectl config view
```

Si ve el resultado detallado de una página anterior, notará que la primera línea hace referencia a un archivo de configuración del que se extrae esta información, `~ /`  
`~/.kube/config`:

```
I1215 17:35:46.725407 27695 loader.go:357]
```

```
Config loaded from file /home/student/.kube/config
```

Sin el certificate authority, key y el certificado de este archivo, solo se pueden usar comandos **curl** inseguros , que no expondrán mucho debido a la configuración de

seguridad. Usaremos `curl` para acceder a nuestro clúster usando TLS en un próximo laboratorio.

## ~/.kube/config

Eche un vistazo a la salida a continuación:

```
apiVersion: v1

clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdF.....
    server: https://10.128.0.3:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes

kind: Config

preferences: {}

users:
- name: kubernetes-admin
  user:
```

```
client-certificate-data: LS0tLS1CRUdJTib.....
```

```
client-key-data: LS0tLS1CRUdJTi.....
```

La salida anterior muestra 19 líneas de salida, con cada una de las claves muy truncadas. Si bien las claves pueden parecer similares, un examen detenido muestra que son distintas.

### apiVersion:

Al igual que con otros objetos, esto indica al kube-apiserver dónde asignar los datos.

### Clusters:

Esta clave contiene el nombre del clúster, así como a dónde enviar las llamadas a la API. El `certificate-authority-data` se pasan para autenticar la solicitud curl.

### Context:

Esta es una configuración que permite un fácil acceso a múltiples clústeres, posiblemente como varios usuarios, desde un archivo de configuración. Se puede utilizar para configurar el `namespace`, `user`, and `cluster`.

### Current-context:

Esto muestra qué clúster y usuario usaría el comando `kubect1`. Estas configuraciones también se pueden pasar por comando.

### Kind:

Todos los objetos de Kubernetes deben tener esta configuración; en este caso, una declaración de tipo de objeto `Config`.

### Preferences:

Actualmente no se utiliza, esta es una configuración opcional para el comando `kubectl`, como colorear la salida.

#### Users:

Un apodo asociado con las credenciales del cliente, que pueden ser la clave y el certificado del cliente, el nombre de usuario y la contraseña, y un token. El token y el nombre de usuario / contraseña son mutuamente excluyentes. Estos se pueden configurar mediante el comando `kubectl config set-credentials`.

## Namespaces

El término *namespaces* se utiliza para hacer referencia tanto a la función del kernel como a la segregación de objetos API por parte de Kubernetes. Ambos son medios para diferenciar los recursos.

Cada llamada a la API incluye un namespace, utilizando el **valor predeterminado** si no se declara lo contrario:

`https://10.128.0.3:6443/api/v1/namespaces/default/pods`.

Los namespaces, una característica del kernel de Linux que segrega los recursos del sistema, están destinados a aislar varios grupos y los recursos con los que tienen acceso para trabajar a través de cuotas. Con el tiempo, las políticas de control de acceso también funcionarán en los límites del namespace. Se podrían usar etiquetas para agrupar recursos por razones administrativas.

***Hay cuatro namespaces cuando se crea un clúster por primera vez.***

#### Default:

Aquí es donde se asumen todos los recursos, a menos que se establezca lo contrario.

#### Kube-node-lease:

Este es el namespace donde se guarda la información de concesión del nodo trabajador.



### Kube-public:

Un namespace legible por todos, incluso aquellos no autenticados. A menudo se incluye información general en este namespace.

### Kube-system:

Este namespace contiene pods de infraestructura.

Si desea ver todos los recursos en un sistema, debe pasar la opción `--all-namespaces` al comando `kubectl`.

## Trabajar con namespaces

Eche un vistazo a los siguientes comandos:

```
$ kubectl get ns
$ kubectl create ns linuxcon
$ kubectl describe ns linuxcon
$ kubectl get ns/linuxcon -o yaml
$ kubectl delete ns/linuxcon
```

Los comandos anteriores muestran cómo ver, crear y eliminar namespaces. Tenga en cuenta que el subcomando `describe` muestra varias configuraciones, como etiquetas, anotaciones, cuotas de recursos y límites de recursos, que analizaremos más adelante en el curso.

Una vez que se ha creado un namespace, puede hacer referencia a él a través de YAML al crear un recurso:

```
$ cat redis.yaml
```

```
apiVersion: V1

kind: Pod

metadata:

  name: redis

  namespace: linuxcon
```

## Recursos de API con kubectl

Todos los recursos de API expuestos están disponibles a través de `kubectl` . Para obtener más información, haga `kubectl help`.

`kubectl [command] [type] [Name] [flag]`

Espere que la lista a continuación cambie:

|  |   |  |
|--|---|--|
| <code>all</code>   | <code>events (ev)</code>                    | <code>podsecuritypolicies (psp)</code>   |
| <code>certificatesigningrequests (csr)</code>                | <code>horizontalpodautoscalers (hpa)</code> | <code>podtemplates</code>                |
| <code>clusterrolebindings</code>                             | <code>ingresses (ing)</code>                | <code>replicasets (rs)</code>            |
| <code>clusterroles</code>                                    | <code>jobs</code>                           | <code>replicationcontrollers (rc)</code> |
| <code>clusters</code> (valid only for federation apiservers) | <code>limitranges (limits)</code>           | <code>resourcequotas (quota)</code>      |
| <code>componentstatuses (cs)</code>                          | <code>namespaces (ns)</code>                | <code>rolebindings</code>                |
| <code>configmaps (cm)</code>                                 | <code>networkpolicies (netpol)</code>       | <code>roles</code>                       |
| <code>controllerrevisions</code>                             | <code>nodes (no)</code>                     | <code>secrets</code>                     |

|   |   |                                   |
|---|---|-----------------------------------|
| <code>cronjobs</code>                       | <code>persistentvolumeclaims (pvc)</code> | <code>serviceaccounts (sa)</code> |
| <code>customresourcedefinition (crd)</code> | <code>persistentvolumes (pv)</code>       | <code>services (svc)</code>       |
| <code>daemonsets (ds)</code>                | <code>poddisruptionbudgets (pdb)</code>   | <code>statefulsets</code>         |
| <code>deployments (deploy)</code>           | <code>podpreset</code>                    | <code>storageclasses</code>       |
| <code>endpoints (ep)</code>                 | <code>Pods (po)</code>                    |                                   |

## Métodos de recursos adicionales

Además de la gestión básica de recursos a través de REST, la API también proporciona algunos puntos finales extremadamente útiles para ciertos recursos.

Por ejemplo, puede acceder a los registros de un contenedor, ejecutar en él y ver los cambios en él con los siguientes puntos finales:

```
$ curl --cert /tmp/client.pem --key /tmp/client-key.pem \
--cacert /tmp/ca.pem -v -XGET \
https://10.128.0.3:6443/api/v1/namespaces/default/pods/firstpod/log
```

Este sería el mismo que el siguiente. Si el contenedor no tiene salida estándar, no habrá registros.

```
$ kubectl logs firstpod
```

Hay otras llamadas que podría realizar siguiendo los distintos grupos de API en su clúster:

```
GET /api/v1/namespaces/{namespace}/pods/{name}/exec
```

```
GET /api/v1/namespaces/{namespace}/pods/{name}/log
```

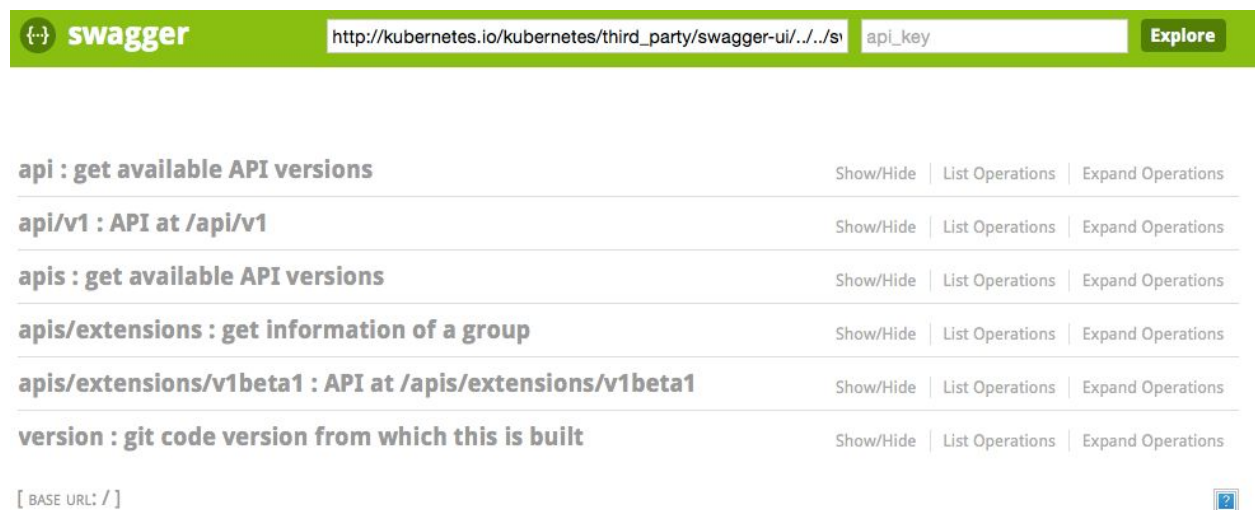
```
GET /api/v1/watch/namespaces/{namespace}/pods/{name}
```

## Swagger y OpenAPI

Toda la API de Kubernetes utiliza una especificación Swagger. Esto está evolucionando hacia la iniciativa OpenAPI. Es extremadamente útil, ya que permite, por ejemplo, generar automáticamente el código del cliente. Todas las definiciones de recursos estables están disponibles en el sitio de documentación.

Puede explorar algunos de los grupos de API a través de una interfaz de usuario Swagger en la página web de especificación de OpenAPI .

<https://swagger.io/specification/>



| api : get available API versions                          | Show/Hide | List Operations | Expand Operations |
|---|-----------|-----------------|-------------------|
| api/v1 : API at /api/v1                                   | Show/Hide | List Operations | Expand Operations |
| apis : get available API versions                         | Show/Hide | List Operations | Expand Operations |
| apis/extensions : get information of a group              | Show/Hide | List Operations | Expand Operations |
| apis/extensions/v1beta1 : API at /apis/extensions/v1beta1 | Show/Hide | List Operations | Expand Operations |
| version : git code version from which this is built       | Show/Hide | List Operations | Expand Operations |

[ BASE URL: / ]

## API Maturity

El uso de grupos de API y diferentes versiones permite que el desarrollo avance sin cambios en un grupo existente de API. Esto permite un crecimiento y una separación

del trabajo más fáciles entre equipos separados. Si bien se intenta mantener cierta coherencia entre la API y las versiones de software, solo están vinculadas indirectamente.

El uso de JSON y el esquema de serialización Protobuf de Google seguirá las mismas pautas de publicación.

#### Alpha:

Una versión de nivel Alpha, indicada con *alpha* en el nombre, puede tener errores y está desactivada de forma predeterminada. Las funciones pueden cambiar o desaparecer en cualquier momento. Utilice estas funciones únicamente en un clúster de prueba que a menudo se reconstruye.

#### Beta:

El nivel Beta, que se encuentra con *beta* en el nombre, tiene un código más probado y está habilitado de forma predeterminada. También garantiza que, a medida que avancen los cambios, se probará la compatibilidad con versiones anteriores entre versiones. No se ha adoptado ni probado lo suficiente como para llamarlo estable. Puede esperar algunos errores y problemas.

#### Stable:

El uso de la versión estable , que se indica solo con un número entero que puede estar precedido por la letra *v* , es para API estables.

## Práctica de laboratorio 5.1: Configuración del acceso TLS

### Lab 5.2: Explore las llamadas a la API

Ver: 95j9tla4zde3-LFS258-labs\_V2020-10-19.pdf

## Pregunta 5.1

Kubernetes utiliza una arquitectura basada en API RESTful, que acepta verbos HTTP estándar. ¿Verdadero o falso?

- **A** . Cierto
- **B** . Falso

## Question 5.2

\_\_\_\_\_ permite que se incluyan metadatos con un objeto que puede ser útil fuera de la interacción del objeto de Kubernetes.

- **A.** Annotations
- **B.** Labels
- **C.** kubectl
- **D.** Controllers

## Question 5.3

¿Cuál de los siguientes debe incluirse en una plantilla de pod?

- **A.** kind
- **B.** metadata
- **C.** spec
- **D.** apiVersion
- **E.** All of the above

## Question 5.4

¿Qué se debe agregar al comando para afectar cada namespace con kubectl ?

- **A.** This cannot be done
- **B.** --every-object
- **C.** --all-namespaces
- **D.** --all

## Question 5.5

Todos los objetos están restringidos a un solo namespace. ¿Verdadero o falso?

- **A.** True
- **B.** False

- **06. API OBJECTS**

-

## Objetivos de aprendizaje

---

Al final de este capítulo, debería poder:

- Explore las versiones de API.
- Discuta el cambio y el desarrollo rápidos.
- Implementar y configurar una aplicación mediante una implementación.
- Examine las primitivas para una aplicación autocurativa.
- Escale una aplicación.

## Visión general

Este capítulo trata sobre recursos u objetos API adicionales. Conoceremos los recursos del grupo API `v1` , entre otros. La estabilidad aumenta y el código se vuelve más

estable a medida que los objetos pasan de las versiones alfa a beta y luego a la v1, lo que indica estabilidad.

DaemonSets, que asegura un Pod en cada nodo, y StatefulSets, que pegan un contenedor a un nodo y actúan como una implementación, han progresado hacia la estabilidad de **apps/v1**. Los Jobs y CronJobs ahora están en **batch/v1**.

El control de acceso basado en roles (RBAC), esencial para la seguridad, ha dado el salto de **v1alpha1** al estado estable **v1** en una versión.

Como proyecto de rápido movimiento que realiza un seguimiento de los cambios, cualquier cambio posible puede ser una parte importante de la administración continua del sistema. Las notas de la versión, así como las discusiones sobre las notas de la versión, se pueden encontrar en los subdirectorios dependientes de la versión en el [repositorio de seguimiento de características para las versiones de Kubernetes en GitHub](https://github.com/kubernetes/enhancements) <https://github.com/kubernetes/enhancements>. Por ejemplo, el estado de la función de la versión v1.17 se puede encontrar en línea, en la [página Notas de la versión de Kubernetes v1.17.0](#).



A partir de la v1.16, las versiones de objetos de API obsoletas responderán

con un error en lugar de ser aceptadas. Este es un cambio importante con respecto al comportamiento histórico.

## Grupo de API v1

El grupo de API v1 ya no es un solo grupo, sino una colección de grupos para cada categoría de objeto principal. Por ejemplo, hay una v1 grupo, un **storage.k8s.io/v1** grupo, y un **rbac.authorization.k8s.io/v1**, etc. Actualmente, hay ocho v1 grupos.

## Objects



**Node:** Representa una máquina, física o virtual, que forma parte de su clúster de Kubernetes. Puede obtener más información sobre los nodos con el comando **kubectl get nodes** . Puede activar y desactivar la programación de un nodo con los comandos **kubectl cordon / uncordon** .

**Service Account:** Proporciona un identificador para los procesos que se ejecutan en un pod para acceder al servidor de API y realiza las acciones para las que está autorizado.

**Resource Quota:** Es una herramienta extremadamente útil que le permite definir cuotas por namespaces. Por ejemplo, si desea limitar un namespace específico para ejecutar solo una cantidad determinada de pods, puede escribir un manifiesto de cuota de recursos , crearlo con **kubectl** y la cuota se aplicará.

**Endpoint:** Generalmente, no administra los endpoints. Representan el conjunto de IP de los pods que coinciden con un servicio en particular. Son útiles cuando se desea comprobar que un servicio realmente coincide con algunos módulos en ejecución. Si un endpoint está vacío, significa que no hay pods coincidentes y es muy probable que haya algún problema con la definición de servicio.

## Descubriendo grupos de API

Podemos echar un vistazo más de cerca al resultado de la solicitud de las API actuales. Cada uno de los valores de nombre se puede agregar a la URL para ver detalles de ese grupo. Por ejemplo, puede profundizar para encontrar objetos incluidos en esta URL: `https://localhost:6443/apis/apiregistration.k8s.io/v1beta1`.

Si sigue esta URL, encontrará solo un recurso, con el nombre de apiservices. Si parece aparecer dos veces, la salida más baja es para el estado. Notarás que hay diferentes

verbos o acciones para cada uno. Otra entrada es si este objeto tiene un espacio de nombres o está restringido a un solo espacio de nombres. En este caso, no lo es.

```
$ curl https://localhost:6443/apis --header "Authorization: Bearer $token" -k
```

```
{
  "kind": "APIGroupList",
  "apiVersion": "v1",
  "groups": [
    {
      "name": "apiregistration.k8s.io",
      "versions": [
        {
          "groupVersion": "apiregistration.k8s.io/v1",
          "version": "v1"
        }
      ],
      "preferredVersion": {
        "groupVersion": "apiregistration.k8s.io/v1",
        "version": "v1"
      }
    }
  ]
}
```

Luego puede curvar cada uno de estos URI y descubrir objetos API adicionales, sus características y verbos asociados.

## Implementar una aplicación

Usando el comando `kubectl create`, podemos implementar rápidamente una aplicación. Hemos analizado los pods creados al ejecutar la aplicación, como nginx.

Mirando más de cerca, encontrará que se creó una implementación, que administra un ReplicaSet, que luego implementa el Pod.

## Objetos:

**Deployment:** Es un controlador que gestiona el estado de ReplicaSets y los pods que contiene. El control de nivel superior permite una mayor flexibilidad con las actualizaciones y la administración. A menos que tenga una buena razón, utilice una implementación.

**ReplicaSet:** Orquesta el ciclo de vida y las actualizaciones de cada pod. Estas son versiones más recientes de Replication Controllers, que solo se diferencian en la compatibilidad con el selector.

**Pod:** Como ya hemos mencionado, es la unidad más baja que podemos gestionar; ejecuta el contenedor de la aplicación y posiblemente los contenedores de soporte.

## DaemonSets

Si desea tener una aplicación de registro en cada nodo, un DaemonSet puede ser una buena opción. El controlador asegura que un solo pod, del mismo tipo, se ejecute en cada nodo del clúster. Cuando se agrega un nuevo nodo al clúster, se inicia un Pod, igual que el implementado en los otros nodos. Cuando se elimina el nodo, DaemonSet se asegura de que se elimine el Pod local. Los DaemonSets se utilizan a menudo para registros, métricas y pods de seguridad, y se pueden configurar para evitar nodos.

Como de costumbre, obtienes todas las operaciones CRUD a través de `kubectl` :

```
$ kubectl get daemonsets
```

```
$ kubectl get ds
```

## StatefulSets

Según la documentación de Kubernetes, un StatefulSet es el objeto de la API de carga de trabajo que se utiliza para administrar aplicaciones con estado. Los pods implementados con un StatefulSet usan la misma especificación de pod. En qué se diferencia de una implementación es que un StatefulSet considera que cada pod es único y proporciona pedidos para la implementación de pod.

Para rastrear cada Pod como un objeto único, los controladores usan una identidad compuesta de almacenamiento estable, identidad de red estable y un ordinal. Esta identidad permanece con el nodo, independientemente del nodo en el que se ejecute el Pod en un momento dado.

El esquema de implementación predeterminado es secuencial, comenzando con 0, como `app-0` , `app-1` , `app-2` , etc. Un pod siguiente no se iniciará hasta que el pod actual alcance un estado en ejecución y listo. No se despliegan en paralelo.

StatefulSets es estable a partir de Kubernetes v1.9.

## Autoscaling

En el grupo de ajuste de autoscaling encontramos los **Horizontal Pod Autoscalers (HPA)**. Este es un recurso estable. Los HPA escalan automáticamente los controladores de replicación, los conjuntos de réplicas o las implementaciones en función de un objetivo del 50% de uso de la CPU de forma predeterminada. El kubelet verifica el uso cada 30 segundos y la llamada a la API de Metrics Server lo recupera cada minuto. HPA verifica con Metrics Server cada 30 segundos. Si se agrega o quita un Pod, HPA espera 180 segundos antes de tomar medidas adicionales.

Se pueden utilizar y consultar otras métricas a través de REST. El autoscaling no recopila las métricas, solo realiza una solicitud de información agregada y aumenta o disminuye la cantidad de réplicas para que coincida con la configuración.

El **Cluster Autoscaler (CA)** agrega o quita nodos al clúster, en función de la imposibilidad de implementar un Pod o de tener nodos con baja utilización durante al

menos 10 minutos. Esto permite solicitudes dinámicas de recursos del proveedor de la nube y minimiza los gastos de los nodos no utilizados. Si está utilizando CA, los nodos deben agregarse y eliminarse mediante los comandos `cluster-autoscaler-`. El escalado hacia arriba y hacia abajo de los nodos se verifica cada 10 segundos, pero las decisiones se toman en un nodo cada 10 minutos. Si falla una reducción, el grupo se volverá a verificar en 3 minutos y el nodo que falla será elegible en cinco minutos. El tiempo total para asignar un nuevo nodo depende en gran medida del proveedor de la nube.

Otro proyecto aún en desarrollo es el Vertical Pod Autoscaler. Este componente ajustará la cantidad de CPU y memoria solicitada por los pods.

## Jobs

Los jobs forman parte del **grupo de batch** API. Se utilizan para ejecutar un número determinado de pods hasta su finalización. Si un módulo falla, se reiniciará hasta que se alcance el número de finalización.

Si bien pueden verse como una forma de realizar procesamiento por lotes en Kubernetes, también se pueden usar para ejecutar pods únicos. Una especificación de trabajo tendrá un paralelismo y una clave de finalización. Si se omiten, se establecerán en uno. Si están presentes, el número de paralelismo establecerá el número de pods que se pueden ejecutar simultáneamente y el número de finalización establecerá cuántos pods deben ejecutarse correctamente para que el trabajo en sí se considere completado. Se pueden implementar varios patrones de trabajo, como una cola de trabajo tradicional.

Los cronjobs funcionan de manera similar a los jobs de Linux, con la misma sintaxis de tiempo. Hay algunos casos en los que un job no se ejecutará durante un período de tiempo o podría ejecutarse dos veces; como resultado, el pod solicitado debe ser idempotente.

Un campo de especificación de opción es `.spec.concurrencyPolicy`, que determina cómo manejar los jobs existentes, en caso de que expire el segmento de tiempo. Si se establece en `Allow`, se ejecutará otro job simultáneo de forma predeterminada. Si se establece en `Forbid`, el job actual continúa y se omite el nuevo job. Un valor de `Replace` cancela el job actual e inicia un nuevo job en su lugar.

## RBAC

Los últimos recursos de API que veremos están en el grupo `rbac.authorization.k8s.io`. En realidad, tenemos cuatro recursos: `ClusterRole`, `Role`, `ClusterRoleBinding` y `RoleBinding`. Se utilizan para el control de acceso basado en roles (RBAC) a Kubernetes.

```
$ curl localhost:8080/apis/rbac.authorization.k8s.io/v1
```

```
...
```

```
  "groupVersion": "rbac.authorization.k8s.io/v1",
```

```
  "resources": [
```

```
...
```

```
    "kind": "ClusterRoleBinding"
```

```
...
```

```
    "kind": "ClusterRole"
```

```
...
```

```
    "kind": "RoleBinding"
```

```
...
```

```
    "kind": "Role"
```

```
...
```

Estos recursos nos permiten definir roles dentro de un clúster y asociar usuarios a estos roles. Por ejemplo, podemos definir un rol para alguien que solo puede leer pods en un namespace específico, o un rol que puede crear deployments, pero no servicios. Hablaremos más sobre RBAC más adelante en el curso.

## Lab 6.1 - RESTful API Access

## Lab 6.2 - Using the Proxy

## Lab 6.3 - Working with Jobs

Ver: 95j9tla4zde3-LFS258-labs\_V2020-10-19.pdf

### Pregunta 6.1

Todas las versiones de la API deben considerarse estables. ¿Verdadero o falso?

- **A** . Cierto
- **B** . Falso

### Pregunta 6.2

¿Cuál de los siguientes es el objeto sugerido para implementar y escalar una aplicación?

- **A**. Pod
- **B**. Container
- **C**. Deployment
- **D**. Service

### Pregunta 6.3

Desde el objeto más pequeño hasta el más grande, ¿cuál es el orden correcto de los siguientes objetos de Kubernetes?

- **A.** Container, ReplicaSet, Pod, Deployment
- **B.** Pod, Container, ReplicaSet, Deployment
- **C.** Container, Pod, ReplicaSet, Deployment
- **D.** Container, Pod, Deployment, ReplicaSet

## Pregunta 6.4

¿Cuántos pods ejecuta un DaemonSet en cada nodo?

- **A.** Ninguna
- **B.** One
- **C.** Depende de la configuración de la réplica
- **D.** Ninguna de las anteriores

## Pregunta 6.5

Los Deployments manejan el escalado de una aplicación según la configuración administrativa. ¿Cuál de las siguientes opciones escala los recursos según el uso de la CPU (50% de forma predeterminada)?

- **A.** ReplicaSet
- **B.** Deployment
- **C.** Horizontal Pod Autoscaling
- **D.** Vertical Node Autoscaling

## Pregunta 6.6

¿A qué grupo de API pertenecen Jobs y CronJobs?

- **A.** v1



- **B.** security.k8s.api
- **C.** batch
- **D.** kubeadm

## 07 . ADMINISTRAR EL ESTADO CON IMPLEMENTACIONES

# Objetivos de aprendizaje

Al final de este capítulo, debería poder:

- Analice los detalles de la configuración de deployment.
- Escale un deployment hacia arriba y hacia abajo.
- Implementar actualizaciones continuas y reversión.
- Utilice etiquetas para seleccionar varios objetos.

## Visión general

El controlador predeterminado para un contenedor implementado a través de `kubectl run` es un deployment. Si bien ya hemos estado trabajando con ellos, analizaremos más de cerca las opciones de configuración.

Al igual que con otros objetos, se puede realizar una implementación desde un archivo de especificaciones YAML o JSON. Cuando se agrega al clúster, el controlador creará un ReplicaSet y un Pod automáticamente. Los contenedores, su configuración y aplicaciones se pueden modificar mediante una actualización, que genera un nuevo ReplicaSet, que, a su vez, genera nuevos Pods.

Los objetos actualizados se pueden organizar para reemplazar los objetos anteriores como un bloque o como una actualización continua, que se determina como parte de la

especificación de implementación. La mayoría de las actualizaciones se pueden configurar editando un archivo YAML y ejecutando `kubectl apply`. También puede utilizar `kubectl edit` para modificar la configuración en uso. Las versiones anteriores de ReplicaSets se mantienen, lo que permite una reversión para volver a una configuración anterior.

También hablaremos más sobre etiquetas. Las etiquetas son esenciales para la administración en Kubernetes, pero no son un recurso de API. Son pares clave-valor definidos por el usuario que se pueden adjuntar a cualquier recurso y se almacenan en los metadatos. Las etiquetas se utilizan para consultar o seleccionar recursos en su clúster, lo que permite una gestión flexible y compleja del clúster.

Como una etiqueta es arbitraria, puede seleccionar todos los recursos utilizados por los desarrolladores, o pertenecientes a un usuario, o cualquier cadena adjunta, sin tener que averiguar qué tipo o cuántos de tales recursos existen.

## Deployments

**ReplicationControllers** (RC) aseguran que se esté ejecutando un número específico de réplicas de pod a la vez. ReplicationControllers también le brinda la capacidad de realizar actualizaciones continuas. Sin embargo, esas actualizaciones se administran en el lado del cliente. Esto es problemático si el cliente pierde la conectividad y puede dejar el clúster en un estado no planificado. Para evitar problemas al escalar los ReplicationControllers en el lado del cliente, se introdujo un nuevo recurso en el grupo de API de `apps/v1`: Deployments.

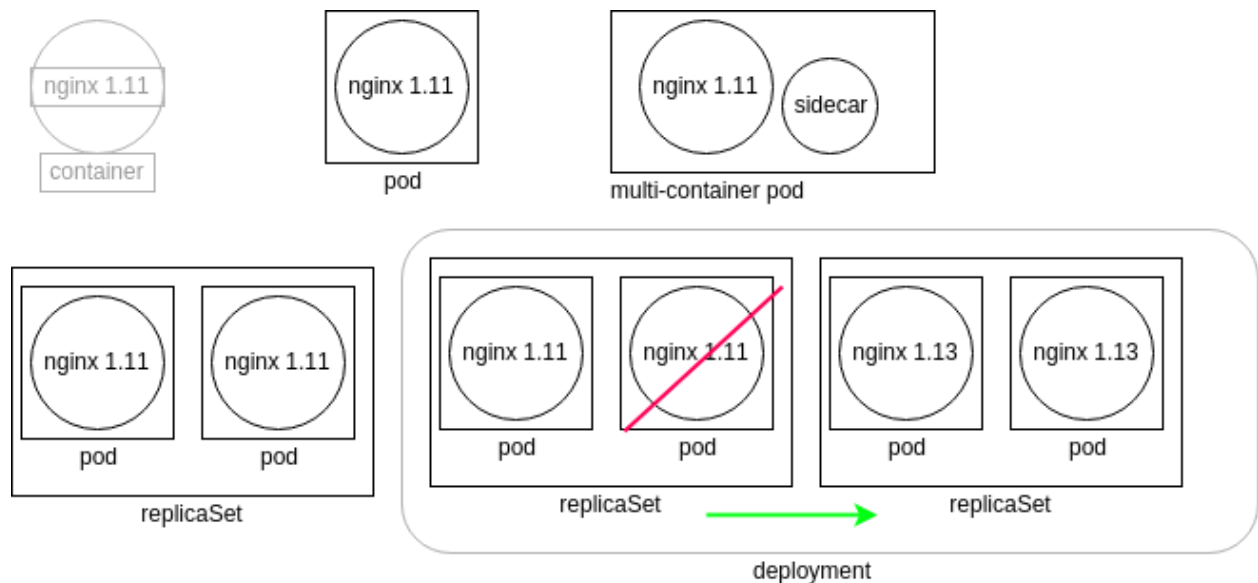
Los deployments permiten actualizaciones del lado del servidor a los pods a un ritmo específico. Se utilizan para patrones de implementación canary y otros. Las implementaciones generan ReplicaSets, que ofrecen más funciones de selección que ReplicationControllers, como `matchExpressions`.

```
$ kubectl create deployment dev-web --image=nginx:1.13.7-alpine
```

```
deployment "dev-web" created
```

## Object Relationship

Aquí puede ver la relación entre los objetos del contenedor, que Kubernetes no administra directamente, hasta la implementación.



### Nested Objects

Las cajas y las formas son lógicas, ya que representan los controladores, o watch loops, que se ejecutan como un hilo del kube-controller-manager. Cada controlador consulta al kube-apiserver sobre el estado actual del objeto que rastrea. El estado de cada objeto en un nodo worker se envía desde el kubelet local.

El gráfico en la parte superior izquierda representa un contenedor que ejecuta nginx 1.11. Kubernetes no administra directamente el contenedor. En cambio, el demonio de kubelet verifica las especificaciones del pod preguntando al motor del contenedor, que podría ser Docker o cri-o, por el estado actual. El gráfico a la derecha del contenedor muestra un pod que representa un watch loop que verifica el estado del contenedor. kubelet compara la especificación actual del pod con lo que responde el motor del contenedor y terminará y reiniciará el pod si es necesario.

A continuación, se muestra un pod de varios contenedores. Si bien se utilizan varios nombres, como *sidecar* o *ambassador*, todos estos son pods de varios contenedores. Los nombres se utilizan para indicar la razón particular para tener un segundo contenedor en el pod, en lugar de denotar un nuevo tipo de pod.

En la parte inferior izquierda vemos un **replicaSet**. Este controlador asegurará que tenga una cierta cantidad de pods en ejecución. Todos los pods se implementan con el mismo **podSpec**, por lo que se denominan réplicas. Si un pod termina o se encuentra uno nuevo, replicaSet creará o terminará los pods hasta que el número actual de pods en ejecución coincida con las especificaciones. Cualquiera de los pods actuales podría terminarse si la especificación exige que se ejecuten menos pods.

El gráfico de la parte inferior derecha muestra un despliegue. Este controlador nos permite gestionar las versiones de imágenes desplegadas en los pods. Si se realiza una edición en la implementación, se crea un nuevo **replicaSet**, que implementará pods utilizando la nueva **podSpec**. Luego, la implementación indicará al antiguo **replicaSet** que cierre los pods cuando los nuevos pods **replicaSet** estén disponibles. Una vez que se terminan todos los pods antiguos, la implementación finaliza el antiguo **replicaSet** y la implementación vuelve a tener solo un **replicaSet** en ejecución.

# Detalles de implementación

En la página anterior, creamos un nuevo deployment que ejecuta una versión particular del servidor web nginx.

Para generar el archivo YAML de los objetos recién creados, haga lo siguiente:

```
$ kubectl get deployments,rs,pods -o yaml
```

A veces, una salida JSON puede dejarlo más claro:

```
$ kubectl get deployments,rs,pods -o json
```

Ahora veremos la salida YAML, que también muestra los valores predeterminados que no se pasaron al objeto cuando se creó:

```
apiVersion: v1

items:

- apiVersion: apps/v1

  kind: Deployment
```

## Explicación de objetos

**apiVersion:** Un valor de `v1` muestra que este objeto se considera un recurso estable. En este caso, no es el deployment. Es una referencia al tipo `List`.

**Items:** Como la línea anterior es un `list`, declara la lista de elementos que muestra el comando.

**-apiVersion:** El guión es una indicación YAML del primer elemento, que declara la **apiVersion** del objeto como `apps/v1`. Esto indica que el objeto se considera estable. En la mayoría de los casos, los despliegues se utilizan por controlador.

**Kind:** Aquí es donde se declara el tipo de objeto a crear, en este caso, un despliegue.

# Metadatos de configuración de implementación

Continuando con la salida YAML, vemos que el siguiente bloque general de salida se refiere a los metadatos de la implementación. Aquí es donde encontraríamos etiquetas, anotaciones y otra información que no sea de configuración. Tenga en cuenta que esta salida no mostrará todas las configuraciones posibles. Muchas configuraciones que están establecidas en falso por defecto no se muestran, como `podAffinity` o `nodeAffinity`.

```
metadata:

  annotations:

    deployment.kubernetes.io/revision: "1"

  creationTimestamp: 2017-12-21T13:57:07Z

  generation: 1

  labels:

    app: dev-web

  name: dev-web

  namespace: default

  resourceVersion: "774003"

  selfLink: /apis/apps/v1/namespaces/default/deployments/dev-web

  uid: d52d3a63-e656-11e7-9319-42010a800003
```

**Annotation:** Estos valores no configuran el objeto, pero proporcionan información adicional que podría ser útil para aplicaciones de terceros o seguimiento administrativo. A diferencia de las etiquetas, no se pueden usar para seleccionar un objeto con `kubectl`.

**creationTimestamp:** Muestra cuándo se creó originalmente el objeto. No se actualiza si se edita el objeto.

**Generation:** Cuántas veces se ha editado este objeto, como cambiar el número de réplicas, por ejemplo.

**Labels:** Cadenas arbitrarias que se utilizan para seleccionar o excluir objetos para su uso con **kubectl** u otras llamadas a la API. Útil para que los administradores seleccionen objetos fuera de los límites de objetos típicos.

**Name:** Esta es una cadena requerida , que pasamos desde la línea de comando. El nombre debe ser exclusivo del espacio de nombres.

**resourceVersion:** Un valor vinculado a la base de datos etcd para ayudar con la concurrencia de objetos. Cualquier cambio en la base de datos hará que este número cambie.

**selfLink:** Hace referencia a cómo kube-apiserver ingiere esta información en la API.

**Uid:** Sigue siendo una identificación única durante la vida útil del objeto.

## Especificaciones de configuración de implementación

Hay dos declaraciones de **spec** para la implementación. El primero modificará el ReplicaSet creado, mientras que el segundo pasará la configuración del Pod.

```
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: dev-web
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
```

**Spec:** Una declaración de que los siguientes elementos configurarán el objeto que se está creando.

**progressDeadlineSeconds:** Tiempo en segundos hasta que se informa un error de progreso durante un cambio. Las razones pueden ser cuotas, problemas de imagen o rangos de límites.

**Replicas:** Como el objeto que se crea es un ReplicaSet, este parámetro determina cuántos pods se deben crear. Si usaras kubectl edit y cambiaras este valor a dos, se generaría un segundo Pod.

**revisionHistoryLimit:** Cuántas especificaciones antiguas de ReplicaSet se deben conservar para la reversión.

**Selector:** Una colección de valores unidos por AND. Todo debe estar satisfecho para que la réplica coincida. No cree pods que coincidan con estos selectores, ya que el controlador de implementación puede intentar controlar el recurso y generar problemas.

**matchLabels:** Requisitos basados en conjuntos del selector de pod. A menudo se encuentra con la declaración matchExpressions , para designar aún más dónde se debe programar el recurso.

**Strategy:** Un encabezado para valores relacionados con la actualización de pods. Funciona con el tipo enumerado más tarde . También se podría establecer en Recrear , lo que eliminaría todos los pods existentes antes de que se creen nuevos pods. Con RollingUpdate , puede controlar cuántos pods se eliminan a la vez con los siguientes parámetros.

**Maxsource:** Cantidad máxima de pods sobre la cantidad deseada de pods para crear. Puede ser un porcentaje, predeterminado del 25% o un número absoluto. Esto crea una cierta cantidad de pods nuevos antes de eliminar los antiguos, para un acceso continuo.

**maxUnavailable:** Un número o porcentaje de pods que pueden estar en un estado diferente a Listo durante el proceso de actualización.

**Type:** Aunque se enumera en último lugar en la sección, debido al nivel de sangría de los espacios en blanco, se lee como el tipo de objeto que se está configurando. (por ejemplo, RollingUpdate ).

## Deployment Configuration Pod Template

A continuación, veremos una plantilla de configuración para los pods que se implementarán. Veremos algunos valores similares.

```
template:
  metadata:
    creationTimestamp: null
    labels:
      app: dev-web
  spec:
    containers:
      - image: nginx:1.13.7-alpine
```



```
imagePullPolicy: IfNotPresent
name: dev-web
resources: {}
terminationMessagePath: /dev/termination-log
terminationMessagePolicy: File
dnsPolicy: ClusterFirst
restartPolicy: Always
schedulerName: default-scheduler
securityContext: {}
terminationGracePeriodSeconds: 30
```



**Nota:** Si el significado es básicamente el mismo que se definió anteriormente, no repetiremos la definición.

**Template:** Los datos se pasan al ReplicaSet para determinar cómo implementar un objeto (en este caso, contenedores).

**Containers:** Palabra clave que indica que los siguientes elementos de esta sangría son para un contenedor.

**Image:** Este es el nombre de la imagen que se pasa al motor del contenedor, normalmente Docker. El motor extraerá la imagen y creará el Pod.

**imagePullPolicy:** La configuración de la política se transfirió al motor del contenedor, sobre cuándo y si una imagen debe descargarse o usarse desde un caché local.

**Name:** El talón principal de los nombres de los pods. Se agregará una cadena única.

**Resources:** Por defecto, vacío. Aquí es donde establecería restricciones y configuraciones de recursos, como un límite en la CPU o la memoria para los contenedores.

**terminationMessagePath:** Una ubicación personalizable de dónde generar información sobre el éxito o el fracaso de un contenedor.

**terminationMessagePolicy:** El valor predeterminado es Archivo , que contiene el método de terminación. También se podría establecer en FallbackToLogsOnError, que utilizará el último fragmento del registro del contenedor si el archivo de mensaje está vacío y el contenedor muestra un error.

**dnsPolicy:** Determina si las consultas de DNS deben dirigirse a núcleos o, si se establece en Predeterminado , utilizar la configuración de resolución de DNS del nodo.

**restartPolicy:** ¿Debe reiniciarse el contenedor si se mata? Los reinicios automáticos son parte de la fortaleza típica de Kubernetes.

**scheduleName:** Permite el uso de un programador personalizado, en lugar del predeterminado de Kubernetes.

**securityContext:** Configuración flexible para pasar una o más configuraciones de seguridad, como el contexto de SELinux, los valores de AppArmor, los usuarios y los UID para que los usen los contenedores.

**terminationGracePeriodSeconds:** La cantidad de tiempo que se debe esperar a que se ejecute un SIGTERM hasta que se use un SIGKILL para terminar el contenedor.

## Deployment Configuration Status

La salida de estado se genera cuando se solicita la información:

```
status:
  availableReplicas: 2
  conditions:
  - lastTransitionTime: 2017-12-21T13:57:07Z
    lastUpdateTime: 2017-12-21T13:57:07Z
    message: Deployment has minimum availability.
    reason: MinimumReplicasAvailable
    status: "True"
    type: Available
  observedGeneration: 2
  readyReplicas: 2
  replicas: 2
  updatedReplicas: 2
```

El resultado anterior muestra cómo se vería la misma implementación si el número de réplicas se incrementara a dos. Los tiempos son diferentes a cuando se generó la implementación por primera vez.

## Explicación de elementos adicionales

**availableReplicas**: Indica cuántos fueron configurados por ReplicaSet. Esto se compararía con el valor posterior de readyReplicas , que se utilizaría para determinar si todas las réplicas se han generado completamente y sin errores.

**observedGeneration**: Muestra la frecuencia con la que se ha actualizado la implementación. Esta información se puede utilizar para comprender la situación de implementación y reversión de la implementación.

## Scaling and Rolling Updates

El servidor de API permite actualizar los ajustes de configuración para la mayoría de los valores. Hay algunos valores inmutables, que pueden ser diferentes según la versión de Kubernetes que haya implementado.

Una actualización común es cambiar el número de réplicas en ejecución. Si este número se establece en cero, no habría contenedores, pero todavía habría un ReplicaSet y Deployment. Este es el proceso de backend cuando se elimina una implementación.

```
$ kubectl scale deploy/dev-web --replicas=4
```

```
deployment "dev-web" scaled
```

```
$ kubectl get deployments
```

| NAME    | READY | UP-TO-DATE | AVAILABLE | AGE |
|---------|-------|------------|-----------|-----|
| dev-web | 4/4   | 4          | 4         | 20s |

Los valores no inmutables también se pueden editar mediante un editor de texto. Utilice editar para activar una actualización. Por ejemplo, para cambiar la versión implementada del servidor web nginx a una versión anterior:

```
$ kubectl edit deployment nginx
```

```
....
containers:
- image: nginx:1.8 #<---Set to an older version
  imagePullPolicy: IfNotPresent
```

```
name: dev-web
....
```

Esto desencadenaría una actualización continua de la implementación. Si bien la implementación mostraría una edad más avanzada, una revisión de los Pods mostraría una actualización reciente y una versión anterior de la aplicación del servidor web implementada.

## Deployment Rollbacks

Si se mantienen algunos de los ReplicaSets anteriores de una implementación, también puede retroceder a una revisión anterior escalando hacia arriba y hacia abajo. El número de configuraciones anteriores que se conservan es configurable y ha cambiado de una versión a otra. A continuación, vamos a echar un vistazo más de cerca a reversiones, utilizando el `--record` opción del comando `kubectl create`, lo que permite la anotación en la definición de los recursos.

```
$ kubectl create deploy ghost --image=ghost --record
```

```
$ kubectl get deployments ghost -o yaml
```

```
deployment.kubernetes.io/revision: "1"
```

```
kubernetes.io/change-cause: kubectl create deploy ghost --image=ghost --record
```

Si una actualización falla, debido a una versión de imagen incorrecta, por ejemplo, puede revertir el cambio a una versión funcional con `kubectl rollout undo`:

```
$ kubectl set image deployment/ghost ghost=ghost:09 --all
```

```
$ kubectl rollout history deployment/ghost deployments "ghost":
```

```
REVISION    CHANGE-CAUSE
```

```
1           kubectl create deploy ghost --image=ghost --record
```

```
2           kubectl set image deployment/ghost ghost=ghost:09 --all
```

```
$ kubectl get pods
```

| NAME                   | READY | STATUS           | RESTARTS | AGE |
|------------------------|-------|------------------|----------|-----|
| ghost-2141819201-tcths | 0/1   | ImagePullBackOff | 0        | 1m  |

```
$ kubectl rollout undo deployment/ghost ; kubectl get pods
```

| NAME                   | READY | STATUS  | RESTARTS | AGE |
|------------------------|-------|---------|----------|-----|
| ghost-3378155678-eq5i6 | 1/1   | Running | 0        | 7s  |

Puede retroceder a una revisión específica con la opción `--to-revision=2` .

También puede editar una implementación con el comando `kubectl edit` .

También puede pausar una implementación y luego reanudarla.

```
$ kubectl rollout pause deployment/ghost
```

```
$ kubectl rollout resume deployment/ghost
```

Tenga en cuenta que aún puede hacer una actualización continua en ReplicationControllers con el comando `kubectl rolling-update` , pero esto se hace en el lado del cliente. Por lo tanto, si cierra su cliente, la actualización continua se detendrá.

## Usando DaemonSets

Un objeto más nuevo con el que trabajar es DaemonSet. Este controlador garantiza que exista un solo pod en cada nodo del clúster. Cada pod usa la misma imagen. Si se agrega un nuevo nodo, el controlador DaemonSet implementará un nuevo Pod en su nombre. Si se elimina un nodo, el controlador también eliminará el Pod.

El uso de un DaemonSet permite garantizar que un contenedor en particular esté siempre en ejecución. En un entorno grande y dinámico, puede resultar útil tener una aplicación de generación de registros o métricas en cada nodo sin que un administrador se acuerde de implementar esa aplicación.

```
kind: DaemonSet..
```

Hay formas de realizar kube-apischeduler de manera que algunos nodos no ejecuten un DaemonSet.

## Labels

Parte de los metadatos de un objeto es una etiqueta. Aunque las etiquetas no son objetos de API, son una herramienta importante para la administración de clústeres. Se pueden utilizar para seleccionar un objeto en función de una cadena arbitraria, independientemente del tipo de objeto. Las etiquetas son inmutables a partir de la versión de API `apps/v1`.

Cada recurso puede contener etiquetas en sus metadatos. De forma predeterminada, la creación de una implementación con `kubectl create` agrega una etiqueta, como vimos en:

```
....  
  
  labels:  
  
    pod-template-hash: "3378155678"  
  
  run: ghost ....
```

Luego, podría ver etiquetas en nuevas columnas:

```
$ kubectl get pods -l run=ghost
```

| NAME                   | READY | STATUS  | RESTARTS | AGE |
|------------------------|-------|---------|----------|-----|
| ghost-3378155678-eq5i6 | 1/1   | Running | 0        | 10m |

```
$ kubectl get pods -L run
```

| NAME                   | READY | STATUS  | RESTARTS | AGE | RUN   |
|------------------------|-------|---------|----------|-----|-------|
| ghost-3378155678-eq5i6 | 1/1   | Running | 0        | 10m | ghost |
| nginx-3771699605-4v27e | 1/1   | Running | 1        | 1h  | nginx |

Si bien normalmente define etiquetas en las plantillas de pod y en las especificaciones de las implementaciones, también puede agregar etiquetas sobre la marcha:

```
$ kubectl label pods ghost-3378155678-eq5i6 foo=bar
```

```
$ kubectl get pods --show-labels
```

| NAME                   | READY | STATUS  | RESTARTS | AGE | LABELS   |
|------------------------|-------|---------|----------|-----|--|
| ghost-3378155678-eq5i6 | 1/1   | Running | 0        | 11m | foo=bar,<br>pod-template-hash=3378155678,run=ghost |

Por ejemplo, si desea forzar la programación de un pod en un nodo específico, puede usar un nodeSelector en una definición de pod, agregar etiquetas específicas a ciertos nodos en su clúster y usar esas etiquetas en el pod.

```
....
```

```
spec:
```

```
  containers:
```

```
  - image: nginx
```

```
  nodeSelector:
```

```
    disktype: ssd
```

## Lab 7.1 - Working with ReplicaSets

## Lab 7.2 - Working with DaemonSets

## Lab 7.3 - Rolling Updates and Rollbacks

### Question 7.1

¿Qué valor de implementación determina la cantidad de pods duplicados implementados?

- **A.** label
- **B.** uid
- **C.** status
- **D.** replicas

### Question 7.2

¿Cuál de los siguientes es un valor de encabezado que tiene que ver con la actualización de pods?

- **A.** selector
- **B.** type
- **C.** strategy
- **D.** None of the above



## Question 7.3

¿Cuál de los siguientes metadatos se usa para seleccionar un objeto con **kubectl** , en función de una cadena arbitraria, independientemente del tipo de objeto?

- **A.** strategy
- **B.** uid
- **C.** replicas
- **D.** label

## Question 7.4

¿Cuál de los siguientes argumentos pasamos al comando **kubectl rollout** para ver las revisiones de los objetos?

- **A.** history
- **B.** rollout
- **C.** undo
- **D.** redo

## Question 7.5

¿Qué argumento pasamos al comando **kubectl rollout** para volver a una versión anterior?

- **A.** history
- **B.** rollout
- **C.** undo
- **D.** redo

## **08 . VOLÚMENES Y DATOS**

# Objetivos de aprendizaje

Al final de este capítulo, debería poder:

- Comprenda y cree volúmenes persistentes.
- Configure las notificaciones de volumen persistentes.
- Administre los modos de acceso por volumen.
- Implemente una aplicación con acceso a almacenamiento persistente.
- Analice el aprovisionamiento dinámico de almacenamiento.
- Configure secretos y ConfigMaps.

## Visión general

Los motores de contenedores tradicionalmente no han ofrecido un almacenamiento que sobreviva al contenedor. Dado que los contenedores se consideran transitorios, esto podría provocar una pérdida de datos u opciones complejas de almacenamiento exterior. Un volumen de Kubernetes comparte la vida útil del pod, no los contenedores que contiene. Si un contenedor termina, los datos seguirán estando disponibles para el nuevo contenedor.

Un volumen es un directorio, posiblemente relleno previamente, que se pone a disposición de los contenedores de un pod. La creación del directorio, el almacenamiento backend de los datos y el contenido dependen del tipo de volumen. A partir de la v1.13, había 27 tipos de volumen diferentes que iban desde rbd para obtener acceso a Ceph, a NFS, a volúmenes dinámicos de un proveedor en la nube

como `gcePersistentDisk` de Google. Cada uno tiene dependencias y opciones de configuración particulares.

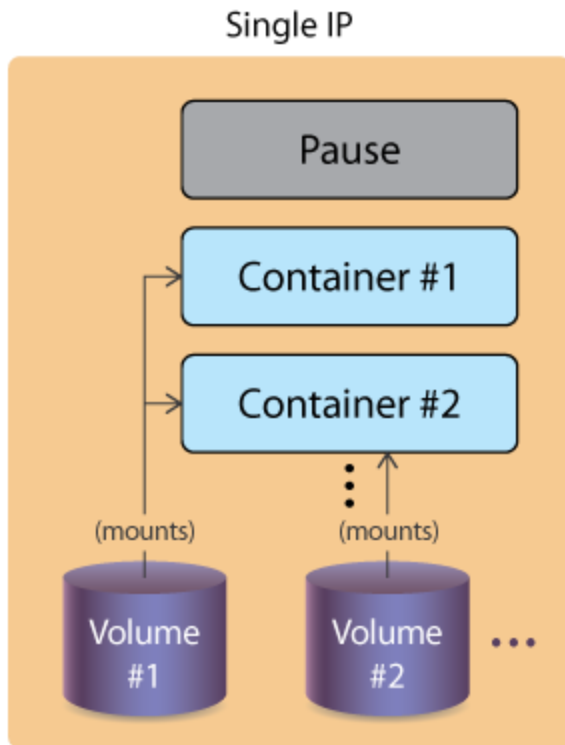
La adopción de Container Storage Interface (CSI) permite el objetivo de una interfaz estándar de la industria para la orquestación de contenedores para permitir el acceso a sistemas de almacenamiento arbitrarios. Actualmente, los complementos de volumen están "in-tree", lo que significa que se compilan y crean con los binarios principales de Kubernetes. Este objeto "out-of-tree" permitirá a los proveedores de almacenamiento desarrollar un solo controlador y permitir que el complemento sea contenedor. Esto reemplazará el complemento Flex existente que requiere un acceso elevado al nodo host, un gran problema de seguridad.

Si desea que su vida útil de almacenamiento sea distinta a la de un pod, puede usar volúmenes persistentes. Estos permiten que un pod reclame volúmenes vacíos o precargados mediante un reclamo de volumen persistente y luego sobreviva al pod. Los datos dentro del volumen podrían luego ser utilizados por otro Pod o como un medio para recuperar datos.

Hay dos objetos API que ya existen para proporcionar datos a un Pod. Los datos codificados se pueden pasar mediante un secreto y los datos no codificados se pueden pasar con un ConfigMap. Estos se pueden usar para pasar datos importantes como claves SSH, contraseñas o incluso un archivo de configuración como `/etc/hosts` .

## Presentación de volúmenes

Una especificación de Pod puede declarar uno o más volúmenes y dónde están disponibles. Cada uno requiere un nombre, un tipo y un punto de montaje. El mismo volumen puede estar disponible para varios contenedores dentro de un Pod, que puede ser un método de comunicación de contenedor a contenedor. Un volumen puede estar disponible para varios Pods, y cada uno tiene un modo de acceso para escribir. No hay verificación de concurrencia, lo que significa que es probable que se dañen los datos, a menos que se lleve a cabo un bloqueo externo.



### Volúmenes de pod de Kubernetes

Un modo de acceso particular es parte de una solicitud de Pod. Como solicitud, se puede otorgar al usuario más, pero no menos acceso, aunque primero se intenta una coincidencia directa. El clúster agrupa los volúmenes con el mismo modo y luego clasifica los volúmenes por tamaño, del más pequeño al más grande. La reclamación se compara con cada uno en ese grupo de modo de acceso, hasta que coincida un volumen de tamaño suficiente. Los tres modos de acceso son:

- ReadWriteOnce, que permite lectura-escritura por un solo nodo
- ReadOnlyMany, que permite solo lectura por varios nodos
- ReadWriteMany, que permite la lectura y escritura de muchos nodos.

Por lo tanto, dos pods en el mismo nodo pueden escribir en un ReadWriteOnce, pero un tercer pod en un nodo diferente no estaría listo debido a un error FailedAttachVolume.

Cuando se solicita un volumen, el kubelet local usa el script **kubelet\_pods.go** para mapear los dispositivos sin procesar, determinar y hacer el punto de montaje para el contenedor, luego crear el enlace simbólico en el sistema de archivos del nodo host para asociar el almacenamiento al contenedor. El servidor de API realiza una solicitud de almacenamiento al complemento **StorageClass** , pero los detalles de las solicitudes al almacenamiento de backend dependen del complemento en uso.

Si no se realizó una solicitud para una **StorageClass** en particular , los únicos parámetros utilizados serán el modo de acceso y el tamaño. El volumen podría provenir de cualquiera de los tipos de almacenamiento disponibles y no existe una configuración para determinar cuál de los disponibles se utilizará.

## Especificaciones de volumen

Uno de los muchos tipos de almacenamiento disponibles es **emptyDir** . El kubelet creará el directorio en el contenedor, pero no montará ningún almacenamiento. Todos los datos creados se escriben en el espacio contenedor compartido. Como resultado, no sería un almacenamiento persistente. Cuando se destruye el Pod, el directorio se eliminará junto con el contenedor.

```
apiVersion: v1

kind: Pod

metadata:

  name: ford Pinto

  namespace: default

spec:

  containers:

    - image: simpleapp

      name: gastank

      command:
```

```

    - sleep

    - "3600"

  volumeMounts:

    - mountPath: /scratch

      name: scratch-volume

  volumes:

    - name: scratch-volume

      emptyDir: {}

```

El archivo YAML anterior crearía un Pod con un solo contenedor con un volumen llamado `scratch-volume` creado, que crearía el directorio **/scratch** dentro del contenedor.

## Tipos de volumen

Hay varios tipos que puede utilizar para definir volúmenes, cada uno con sus pros y sus contras. Algunos son locales y muchos utilizan recursos basados en la red.

En GCE o AWS, puede usar volúmenes de tipo **GCEpersistentDisk** o **awsElasticBlockStore**, lo que le permite montar discos GCE y EBS en sus pods, suponiendo que ya haya configurado cuentas y privilegios.

Los volúmenes **emptyDir** y **hostPath** son fáciles de usar. Como se mencionó, **emptyDir** es un directorio vacío que se borra cuando el Pod muere, pero se vuelve a crear cuando el contenedor se reinicia. El volumen **hostPath** monta un recurso desde el sistema de archivos del nodo host. El recurso puede ser un directorio, un file socket, un carácter o un dispositivo de bloque. Estos recursos ya deben existir en el host que se utilizarán. Hay dos tipos, **DirectoryOrCreate** y **FileOrCreate**, que crean los recursos en el host y los utilizan si aún no existen.

NFS (Network File System) e iSCSI (Internet Small Computer System Interface) son opciones sencillas para escenarios de múltiples lectores.

rbd para almacenamiento en bloque o CephFS y GlusterFS, si están disponibles en su clúster de Kubernetes, pueden ser una buena opción para las necesidades de varios escritores.

Además de los tipos de volumen que acabamos de mencionar, hay muchos otros posibles, y se agregan más: **azureDisk** , **azureFile** , **csi** , **downwardAPI** , **fc** (fibre channel), **flocker** , **gitRepo** , **local** , **projected** , **portworxVolume** , **quobyte** , **scaleIO** , **secret** , **storageos** , **vsphereVolume** , **persistentVolumeClaim** , **CSIPersistentVolumeSource** , etc.

CSI permite aún más flexibilidad y complementos de desacoplamiento sin la necesidad de editar el código central de Kubernetes. Fue desarrollado como un estándar para exponer complementos arbitrarios en el futuro.

## Ejemplo de volumen compartido

El siguiente archivo YAML crea un pod, **exampleA** , con dos contenedores, ambos con acceso a un volumen compartido:

```
....

  containers:

    - name: alphacont

      image: busybox

      volumeMounts:

        - mountPath: /alphadir

          name: sharevol

    - name: betacont
```

```

    image: busybox

    volumeMounts:

    - mountPath: /betadir

      name: sharevol

  volumes:

  - name: sharevol

    emptyDir: {}

$ kubectl exec -ti exampleA -c betacont -- touch /betadir/foobar

$ kubectl exec -ti exampleA -c alphacont -- ls -l /alphadir

total 0

-rw-r--r-- 1 root root 0 Nov 19 16:26 foobar

```

Puede usar `emptyDir` o `hostPath` fácilmente, ya que esos tipos no requieren ninguna configuración adicional y funcionarán en su clúster de Kubernetes.

Tenga en cuenta que un contenedor (`betacont`) escribió y el otro contenedor (`alphacont`) tuvo acceso inmediato a los datos. No hay nada que impida que los contenedores sobrescriban los datos del otro. Las consideraciones de bloqueo o control de versiones deben ser parte de la aplicación en contenedor para evitar daños.

## Volúmenes y reclamos persistentes

Un **volumen persistente** (pv) es una abstracción de almacenamiento que se usa para retener datos durante más tiempo que el Pod que los usa. Los pods definen un volumen de tipo `persistentVolumeClaim` (pvc) con varios parámetros de tamaño y posiblemente el tipo de almacenamiento de backend conocido como `StorageClass`. Luego, el clúster adjunta `persistentVolume`.



Kubernetes utilizará dinámicamente los volúmenes que estén disponibles, independientemente de su tipo de almacenamiento, lo que permitirá reclamar cualquier almacenamiento de backend.

## Fases de almacenamiento persistentes

Provisión: **El aprovisionamiento** puede provenir de PV creados de antemano por el administrador del clúster o solicitarse de una fuente dinámica, como el proveedor de la nube

Bind: **La vinculación** ocurre cuando un bucle de control en el master nota el PVC, que contiene una cantidad de almacenamiento, solicitud de acceso y, opcionalmente, una `StorageClass` particular. El observador localiza un PV coincidente o espera a que el aprovisionador `StorageClass` cree uno. El PV debe coincidir al menos con la cantidad de almacenamiento solicitada, pero puede proporcionar más.

Use: La fase de **uso** comienza cuando se monta el volumen enlazado para que lo use el Pod, que continúa mientras el Pod lo requiera.

Release: **La liberación** ocurre cuando el Pod termina con el volumen y se envía una solicitud de API, eliminando el PVC. El volumen permanece en el estado desde que se elimina la reclamación hasta que está disponible para una nueva reclamación. Los datos residentes permanecen dependiendo de `persistentVolumeReclaimPolicy`.

Reclaim: La fase de **recuperación** tiene tres opciones:

- **Retain**, que mantiene los datos intactos, lo que permite que un administrador maneje el almacenamiento y los datos.
- **Delete** le dice al complemento de volumen que elimine el objeto API, así como el almacenamiento detrás de él.
- La opción **Recycle** ejecuta un `rm -rf /mountpoint` y luego lo pone a disposición de un nuevo reclamo. Con la estabilidad del aprovisionamiento dinámico, se prevé que la opción Recycle quede obsoleta.

```
$ kubectl get pv
```

```
$ kubectl get pvc
```

## Volumen persistente

El siguiente ejemplo muestra una declaración básica de un volumen persistente utilizando el tipo `hostPath`.

```
kind: PersistentVolume

apiVersion: v1

metadata:

  name: 10Gpv01

  labels:

    type: local

spec:

  capacity:

    storage: 10Gi

  accessModes:

    - ReadWriteOnce

  hostPath:

    path: "/somepath/data01"
```

Cada tipo tendrá sus propios ajustes de configuración. Por ejemplo, un Ceph o GCE Persistent Disk ya creado no necesitaría configurarse, pero podría reclamarse al proveedor.

Los volúmenes persistentes no son un objeto de namespace, pero las reclamaciones de volumen persistentes sí lo son. Una función beta de v1.13 permite el

aprovisionamiento estático de Raw Block Volumes, que actualmente admiten el complemento Fibre Channel, AWS EBS, Azure Disk y complementos RBD, entre otros.

El uso de almacenamiento adjunto localmente se ha graduado a una característica estable. Esta función se utiliza a menudo como parte de sistemas de archivos y bases de datos distribuidos.

## Reclamación de volumen persistente

Con un volumen persistente creado en su clúster, puede escribir un manifiesto para un reclamo y usar ese reclamo en su definición de pod. En el Pod, el volumen usa `persistentVolumeClaim`.

```
kind: PersistentVolumeClaim

apiVersion: v1

metadata:

  name: myclaim

spec:

  accessModes:

    - ReadWriteOnce

  resources:

    requests:

      storage: 8Gi
```

En el pod:

```
spec:

  containers:

  ....

  volumes:

    - name: test-volume

      persistentVolumeClaim:

        claimName: myclaim
```

La configuración del Pod también podría ser tan compleja como esta:

```
volumeMounts:
  - name: Cephpd
    mountPath: /data/rbd
volumes:
  - name: rbdpd
    rbd:
      monitors:
        - '10.19.14.22:6789'
        - '10.19.14.23:6789'
        - '10.19.14.24:6789'
      pool: k8s
      image: client
      fsType: ext4
      readOnly: true
      user: admin
      keyring: /etc/ceph/keyring
      imageformat: "2"
      imagefeatures: "layering"
```

## Aprovisionamiento dinámico

Si bien el manejo de volúmenes con una definición de volumen persistente y la abstracción del proveedor de almacenamiento mediante un reclamo es poderoso, un administrador de clúster aún necesita crear esos volúmenes en primer lugar. A partir de Kubernetes v1.4, el aprovisionamiento dinámico permitió que el clúster solicitara almacenamiento desde una fuente exterior preconfigurada. Las llamadas a la API

realizadas por el complemento adecuado permiten una amplia gama de uso de almacenamiento dinámico.

El recurso de la API `StorageClass` permite que un administrador defina un aprovisionador de volumen persistente de un cierto tipo, pasando parámetros específicos de almacenamiento.

Con una `StorageClass` creada, un usuario puede solicitar una reclamación, que el servidor API llena mediante el aprovisionamiento automático. El recurso también será reclamado según lo configurado por el proveedor. AWS y GCE son opciones comunes para el almacenamiento dinámico, pero existen otras opciones, como un clúster Ceph o iSCSI. La clase única por defecto es posible mediante anotaciones.

A continuación, se muestra un ejemplo de **StorageClass** que utiliza GCE:

```
apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

  name: fast          # Could be any name

provisioner: kubernetes.io/gce-pd

parameters:

  type: pd-ssd
```

## Usando Rook para Storage Orchestration

De acuerdo con la naturaleza disociada y distribuida de la tecnología en la nube, el proyecto **Rook** <https://rook.io/> permite la orquestación del almacenamiento utilizando múltiples proveedores de almacenamiento.

Al igual que con otros agentes del clúster, Rook usa definiciones de recursos personalizadas (CRD) y un operador personalizado para aprovisionar almacenamiento de acuerdo con el tipo de almacenamiento de backend, en la llamada API.

Se admiten varios proveedores de almacenamiento:

- Ceph
- Cassandra
- CockroachDB
- EdgeFS Geo-Transparent Storage
- Minio Object Store
- Network File System (NFS)
- YugabyteDB.

## Secrets

Los pods pueden acceder a datos locales utilizando volúmenes, pero hay algunos datos que no desea que se puedan leer a simple vista. Las contraseñas pueden ser un ejemplo. Con el recurso de la API secreta, se puede codificar o cifrar la misma contraseña.

Puede crear, obtener o eliminar secretos:

```
$ kubectl get secrets
```

Los secretos se pueden codificar manualmente o mediante `kubectl create secret` :

```
$ kubectl create secret generic --help
```

```
$ kubectl create secret generic mysql --from-literal=password=root
```

Un secret no está encriptado, solo codificado en base64, de forma predeterminada. Debe crear una configuración de cifrado con una clave y una identidad adecuada. Luego, **kube-apiserver** necesita que el indicador **--encryption-provider-config** se establezca en un proveedor configurado previamente, como **aescbc** o **ksm** . Una vez que esto está habilitado, debe volver a crear cada secret, ya que están encriptados al escribir.

Son posibles varias claves. Cada clave de un proveedor se prueba durante el descifrado. La primera clave del primer proveedor se utiliza para el cifrado. Para rotar claves, primero cree una nueva clave, reinicie (todos) los procesos de kube-apiserver y luego vuelva a crear cada secret.

Puede ver la cadena codificada dentro del secret con `kubectl`. El secret se decodificará y se presentará como una cadena guardada en un archivo. El archivo se puede utilizar como variable de entorno o en un nuevo directorio, similar a la presentación de un volumen.

También se puede crear un secret manualmente y luego insertarlo en un archivo YAML:

```
$ echo LFTr@1n | base64
```

```
TEZUckAxbgo=
```

```
$ vim secret.yaml
```

```
apiVersion: v1
```

```
kind: Secret
```

```
metadata:
```

```
  name: lf-secret
```

```
data:
```

```
  password: TEZUckAxbgo=
```

# Usar secret a través de variables de entorno

Un secret se puede utilizar como variable de entorno en un Pod. Puede ver que se está configurando uno en el siguiente ejemplo:

```
...  
spec:  
  containers:  
    - image: mysql:5.5  
      env:  
        - name: MYSQL_ROOT_PASSWORD  
          valueFrom:  
            secretKeyRef:  
              name: mysql  
              key: password  
      name: mysql
```

No hay límite para la cantidad de secrets usados, pero hay un límite de 1 MB para su tamaño. Cada secret ocupa memoria, junto con otros objetos API, por lo que una gran cantidad de secrets podría agotar la memoria de un host.

Se almacenan en el almacenamiento `tmpfs` en el nodo del host y solo se envían al pod que ejecuta el host. Todos los volúmenes solicitados por un Pod deben montarse antes de que se inicien los contenedores dentro del Pod. Entonces, debe existir un secret antes de ser solicitado.



# Montaje de secretos como volúmenes

También puede montar secrets como archivos mediante una definición de volumen en un manifiesto de pod. La ruta de montaje contendrá un archivo cuyo nombre será la clave del secret creado con el paso anterior de `kubectl create secret`.

...

```
spec:
  containers:
  - image: busybox

    command:
    - sleep
    - "3600"

    volumeMounts:
    - mountPath: /mysqlpassword

      name: mysql

    name: busy

  volumes:
  - name: mysql

    secret:

      secretName: mysql
```

Una vez que el pod se está ejecutando, puede verificar que el secret sea realmente accesible en el contenedor:

```
$ kubectl exec -ti busybox -- cat /mysqlpassword/password
```

```
LFTr@1n
```

# Datos portátiles con ConfigMaps

Un recurso de API similar a Secrets es ConfigMap, excepto que los datos no están codificados. De acuerdo con el concepto de desacoplamiento en Kubernetes, el uso de un ConfigMap desacopla una imagen de contenedor de los artefactos de configuración.

Almacenan datos como conjuntos de pares clave-valor o archivos de configuración simples en cualquier formato. Los datos pueden provenir de una colección de archivos o de todos los archivos de un directorio. También se puede completar a partir de un valor literal.

Un ConfigMap se puede utilizar de varias formas diferentes. Un contenedor puede usar los datos como variables ambientales de una o más fuentes. Los valores contenidos en el interior se pueden pasar a los comandos dentro del pod. Se puede crear un volumen o un archivo en un volumen, incluidos diferentes nombres y modos de acceso particulares. Además, los componentes del clúster, como los controladores, pueden utilizar los datos.

Digamos que tiene un archivo en su sistema de archivos local llamado config.js . Puede crear un ConfigMap que contenga este archivo. El objeto configmap tendrá una sección de datos que contiene el contenido del archivo:

```
$ kubectl get configmap foobar -o yaml
```

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: foobar
data:
  config.js: |
    {
```

...

ConfigMaps se puede consumir de varias formas:

- Variables ambientales de pod de ConfigMaps únicos o múltiples
- Usa los valores de ConfigMap en los comandos de pod
- Rellenar volumen desde ConfigMap
- Agregar datos de ConfigMap a una ruta específica en el volumen
- Establecer los nombres de archivo y el modo de acceso en Volumen desde los datos de ConfigMap
- Puede ser utilizado por controladores y componentes del sistema.

## Usando ConfigMaps

Al igual que los secrets, puede usar ConfigMaps como variables de entorno o usar un montaje de volumen. Deben existir antes de ser utilizados por un Pod, a menos que estén marcados como opcionales. También residen en un namespace específico.

En el caso de las variables de entorno, su manifiesto de pod utilizará la clave **valueFrom** y el valor **configMapKeyRef** para leer los valores. Por ejemplo:

```
env:
```

```
- name: SPECIAL_LEVEL_KEY
```

```
  valueFrom:
```

```
    configMapKeyRef:
```

```
      name: special-config
```

```
      key: special.how
```

Con los volúmenes, usted define un volumen con el tipo configMap en su pod y lo monta donde debe usarse.

```
volumes:  
  
  - name: config-volume  
  
    configMap:  
  
      name: special-config
```

## Lab 8.1 - Create a ConfigMap

## Lab 8.2 - Create a Persistent NFS Volume (PV)

## Lab 8.3 - Creating a Persistent Volume Claim (PVC)

## Lab 8.4 - Use a ResourceQuota to Limit PVC Count and Usage

## Question 8.1

Las aplicaciones deben utilizar almacenamiento persistente. ¿Verdadero o falso?

- A. True
- B. False

## Question 8.2

¿Una implementación utiliza un volumen persistente o una reclamación de volumen persistente?

- **A.** Persistent Volume
- **B.** Persistent Volume Claim

## Question 8.3

¿Cuál de las siguientes configuraciones determina qué sucede con el almacenamiento persistente al momento del lanzamiento?

- **A.** persistentVolumeReclaimPolicy
- **B.** releasePolicy
- **C.** StorageSettingK8S
- **D.** None of the above

## Question 8.4

Un secret contiene datos cifrados. ¿Verdadero o falso?

- **A.** True
- **B.** False

## Question 8.5

Los ConfigMaps se pueden crear desde \_\_\_\_\_.

- **A. Valores literales**
- **B. Archivos individuales**
- **C. Varios archivos en el mismo directorio**
- **D. Todas las anteriores**

## 09. SERVICES

### Objetivos de aprendizaje

Al final de este capítulo, debería poder:

- Explica los servicios de Kubernetes.
- Exponer una aplicación.
- Analice los tipos de servicios disponibles.
- Inicie un proxy local.
- Utilice el DNS del clúster.

### Visión general

Como se mencionó anteriormente, la arquitectura de Kubernetes se basa en el concepto de objetos transitorios desacoplados conectados entre sí. Los servicios son los agentes que conectan los pods entre sí, o brindan acceso fuera del clúster, con la idea de que cualquier pod en particular podría terminarse y reconstruirse.

Normalmente, al usar etiquetas, el pod actualizado está conectado y el microservicio continúa proporcionando el recurso esperado a través de un objeto de **Endpoint**.

Google ha estado trabajando en Extensible Service Proxy (ESP), basado en el servidor proxy inverso HTTP nginx, para proporcionar un objeto más flexible y potente que los

Endpoints, pero el ESP no se ha adoptado mucho fuera de los entornos de Google App Engine o GKE.

Hay varios tipos de servicios diferentes, con la flexibilidad de agregar más, según sea necesario. Cada servicio puede exponerse interna o externamente al clúster. Un servicio también puede conectar recursos internos a un recurso externo, como una base de datos de terceros.

El agente de proxy de kube observa la API de Kubernetes en busca de nuevos servicios y endpoints que se creen en cada nodo. Abre puertos aleatorios y escucha el tráfico al ClusterIP: Port y redirige el tráfico a los puntos finales de servicio generados aleatoriamente.

Los servicios proporcionan equilibrio de carga automático, haciendo coincidir una consulta de etiqueta. Si bien no existe configuración de esta opción, existe la posibilidad de afinidad de sesión vía IP. Además, se puede configurar un servicio headless, uno sin IP fija ni balanceo de carga.

Las direcciones IP únicas se asignan y configuran a través de la base de datos etcd, de modo que los Servicios implementan iptables para enrutar el tráfico, pero podrían aprovechar otras tecnologías para proporcionar acceso a los recursos en el futuro.

## Patrón de actualización de servicio

Las etiquetas se utilizan para determinar qué pods deben recibir tráfico de un servicio. Como hemos aprendido, las etiquetas se pueden actualizar dinámicamente para un objeto, lo que puede afectar qué Pods continúan conectándose a un servicio.

El patrón de actualización predeterminado es para una implementación continua, donde se agregan nuevos pods, con diferentes versiones de una aplicación y, debido al balanceo de carga automático, reciben tráfico junto con versiones anteriores de la aplicación.

Si hubiera una diferencia en las aplicaciones implementadas, de modo que los clientes tuvieran problemas para comunicarse con diferentes versiones, puede considerar una etiqueta más específica para la implementación, que incluye un número de versión. Cuando la implementación crea un nuevo controlador de replicación para la actualización, la etiqueta no coincidirá. Una vez que se hayan creado los nuevos Pods, y tal vez se haya permitido que se inicialicen por completo, editaríamos las etiquetas para las que se conecta el Servicio. El tráfico cambiaría a la versión nueva y lista, minimizando la confusión de la versión del cliente.

## Acceder a una aplicación con un servicio

El paso básico para acceder a un nuevo servicio es utilizar `kubectl`.

```
$ kubectl expose deployment/nginx --port=80 --type=NodePort
```

```
$ kubectl get svc
```

| NAME       | TYPE      | CLUSTER-IP | EXTERNAL-IP | PORT(S)      | AGE |
|------------|-----------|------------|-------------|--------------|-----|
| kubernetes | ClusterIP | 10.0.0.1   | <none>      | 443/TCP      | 18h |
| nginx      | NodePort  | 10.0.0.112 | <none>      | 80:31230/TCP | 5s  |

```
$ kubectl get svc nginx -o yaml
```

```
apiVersion: v1
```

```
kind: Service
```

```
...
```

```
spec:
```

```
  clusterIP: 10.0.0.112
```

```
  ports:
```

```
    - nodePort: 31230
```

```
...
```

Abra el navegador `http://Public-IP:31230`.



El comando `kubectl expose` creó un servicio para la implementación de `nginx`. Este servicio usó el puerto 80 y generó un puerto aleatorio en todos los nodos. También se puede pasar un `puerto` y `targetPort` en particular durante la creación del objeto para evitar valores aleatorios. El `targetPort` tiene como valor predeterminado el puerto, pero se puede establecer en cualquier valor, incluida una cadena que hace referencia a un puerto en un pod de backend. Cada pod puede tener un puerto diferente, pero el tráfico aún se transmite a través del nombre. Cambiar el tráfico a un puerto diferente mantendría una conexión de cliente, mientras se cambiaban las versiones del software, por ejemplo.

El comando `kubectl get svc` le proporcionó una lista de todos los servicios existentes y vimos el servicio `nginx`, que se creó con una IP de clúster interna.

El rango de IP de clúster y el rango de puertos usados para el NodePort aleatorio se pueden configurar en las opciones de inicio del servidor API.

Los servicios también se pueden usar para apuntar a un servicio en un namespace diferente, o incluso a un recurso fuera del clúster, como una aplicación heredada que aún no está en Kubernetes.

## Tipos de servicios

**Clusterip:** El tipo de servicio ClusterIP es el predeterminado y solo proporciona acceso internamente (excepto si se crea manualmente un punto final externo). El rango de ClusterIP utilizado se define mediante una opción de inicio del servidor API.

**NodePort:** El tipo NodePort es ideal para depurar o cuando se necesita una dirección IP estática, como abrir una dirección en particular a través de un firewall. El rango de NodePort se define en la configuración del clúster.

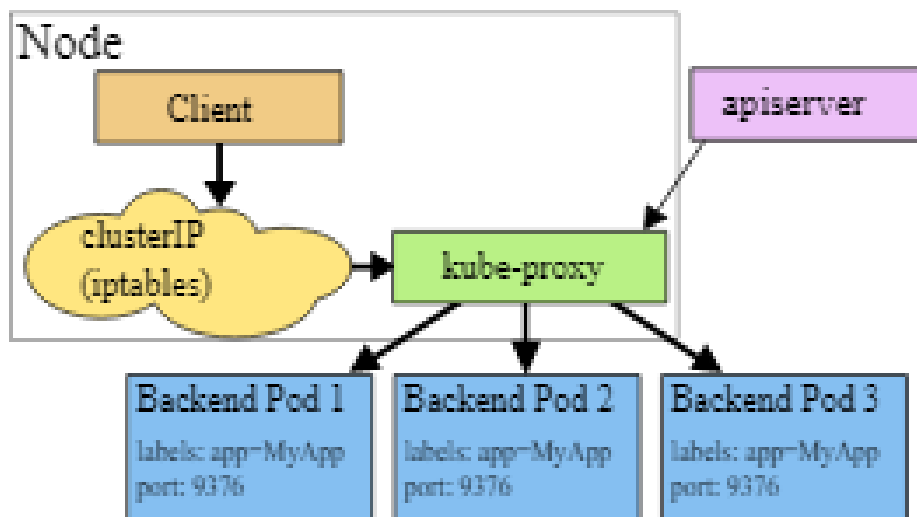
**LoadBalancer:** El servicio LoadBalancer se creó para pasar solicitudes a un proveedor de la nube como GKE o AWS. Las soluciones de nube privada también pueden

implementar este tipo de servicio si hay un complemento de proveedor de nube, como con CloudStack y OpenStack. Incluso sin un proveedor de nube, la dirección se pone a disposición del tráfico público y los paquetes se distribuyen automáticamente entre los Pods en la implementación.

**ExternalName:** Un servicio más nuevo es ExternalName, que es un poco diferente. No tiene selectores, ni define puertos ni puntos finales. Permite la devolución de un alias a un servicio externo. La redirección ocurre a nivel de DNS, no a través de un proxy o reenvío. Este objeto puede resultar útil para los servicios que aún no se han incorporado al clúster de Kubernetes. Un simple cambio del tipo en el futuro redirigiría el tráfico a los objetos internos.

El comando `kubectl proxy` crea un servicio local para acceder a ClusterIP. Esto puede resultar útil para la resolución de problemas o el trabajo de desarrollo.

## Diagrama de servicios



Tráfico de ClusterIP a Pod

El kube-proxy que se ejecuta en los nodos del clúster observa los recursos del servicio del servidor API. Presenta un tipo de dirección IP virtual para servicios distintos a **ExternalName** . El modo de este proceso ha cambiado en las versiones de Kubernetes.

En v1.0, los servicios se ejecutaban en modo de espacio de usuario como TCP/UDP sobre IP o Layer 4. En la versión v1.1, se agregó el proxy iptables y se convirtió en el modo predeterminado a partir de v1.2.

En el modo de proxy de iptables, kube-proxy continúa monitoreando el servidor API en busca de cambios en los objetos de servicio y punto final, y actualiza las reglas para cada objeto cuando se crea o elimina. Una limitación del nuevo modo es la imposibilidad de conectarse a un Pod en caso de que falle la solicitud original, por lo que utiliza una sonda de preparación para garantizar que todos los contenedores funcionen antes de la conexión. Este modo permite hasta aproximadamente 5000 nodos. Suponiendo que haya varios servicios y pods por nodo, esto conduce a un cuello de botella en el kernel.

Otro modo que comienza en v1.9 es ipvs. Mientras está en versión beta, y se espera que cambie, funciona en el espacio del kernel para una mayor velocidad y permite un algoritmo de equilibrio de carga configurable, como round-robin, retraso esperado más corto, conexión mínima y varios otros. Esto puede ser útil para clústeres grandes, mucho más allá de la limitación anterior de 5000 nodos. Este modo asume que los módulos del kernel de IPVS están instalados y ejecutándose antes que kube-proxy.

El modo kube-proxy se configura mediante un indicador enviado durante la inicialización, como **mode=iptables** y también podría ser IPVS o userspace.

# Proxy local para el desarrollo

Al desarrollar una aplicación o servicio, una forma rápida de verificar su servicio es ejecutar un proxy local con **kubectl** . Capturará el shell, a menos que lo coloque en segundo plano. Cuando se ejecuta, puede realizar llamadas a la API de Kubernetes en localhost y también acceder a los servicios ClusterIP en su URL de API. La IP y el puerto donde escucha el proxy se pueden configurar con argumentos de comando.

Ejecute un proxy:

```
$ kubectl proxy
```

```
Starting to serve on 127.0.0.1:8001
```

A continuación, para acceder a un servicio **ghost** usando el proxy local, podríamos usar la siguiente URL, por ejemplo, en

```
http://localhost:8001/api/v1/namespaces/default/services/ghost.
```

Si el puerto de servicio tiene un nombre, la ruta será

```
http://localhost:8001/api/v1/namespaces/default/services/ghost:<port_name>.
```

## DNS

El DNS se ha proporcionado como CoreDNS de forma predeterminada a partir de v1.13. El uso de CoreDNS permite una gran flexibilidad. Una vez que el contenedor se inicia, ejecutará un servidor para las zonas para las que se ha configurado. Luego, cada servidor puede cargar una o más cadenas de complementos para proporcionar otra funcionalidad. Al igual que con otros microservicios, los clientes accederían mediante un servicio, **kube-dns** .

Los aproximadamente treinta complementos integrados en el árbol proporcionan la funcionalidad más común, con un proceso fácil de escribir y habilitar otros complementos según sea necesario.

Los complementos comunes pueden proporcionar métricas para el consumo de Prometheus, registro de errores, informes de estado y TLS para configurar certificados para servidores TLS y gRPC.

Se puede encontrar más en la [página web CoreDNS Plugins](https://coredns.io/plugins/) .  
<https://coredns.io/plugins/>

## Verificación del registro de DNS

Para asegurarse de que la configuración de su DNS funcione bien y que los servicios se registren, la forma más sencilla de hacerlo es ejecutar un pod con un shell y herramientas de red en el clúster, crear un servicio para conectarse al pod, luego ejecutar un servidor para hacer una búsqueda de DNS.

La resolución de problemas de DNS utiliza herramientas típicas como **nslookup** , **dig** , **nc** , **wirehark** y más. La diferencia es que aprovechamos un servicio para acceder al servidor DNS, por lo que necesitamos verificar las etiquetas y los selectores además de las preocupaciones de la red estándar.

Otros pasos, similares a cualquier solución de problemas de DNS, serían verificar el archivo `/etc/resolv.conf` del contenedor, así como las políticas de red y los firewalls. Cubriremos más sobre las políticas de red en el capítulo de *seguridad* .

### Lab 9.1 - Deploy a New Service

### Lab 9.2 - Configure a NodePort

### Lab 9.3 - Working with CoreDNS

### Lab 9.4 - Use Labels to Manage Resources

## Question 9.1

¿Cuáles de los siguientes son tipos de servicios de Kubernetes?

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

## Question 9.2

¿Qué agente de Kubernetes vigila el servidor de API en busca de cambios de configuración y actualizaciones de iptable?

- **A.** kube-proxy
- **B.** kubeadm
- **C.** kubectl
- **D.** kubernetes

## Question 9.3

¿Cuál de los siguientes tipos de servicios distribuye paquetes entre pods en una implementación automáticamente?

- **A.** NodePort
- **B.** LoadBalancer
- **C.** PacketSpreader
- **D.** None of the above

## Question 9.4

¿Cómo se puede iniciar un proxy local que sea útil para el desarrollo y las pruebas?

- **A.** kubectl proxy
- **B.** kubeadm proxy
- **C.** proxy-start
- **D.** None of the above

### **10. INGRESS**

## Objetivos de aprendizaje

Al final de este capítulo, debería poder:

- Analice la diferencia entre un controlador de ingress y un service.
- Obtenga más información sobre los controladores de ingress nginx y GCE.
- Implemente un controlador de ingress.
- Configure una regla de ingress.

## Visión general

En un capítulo anterior, aprendimos sobre el uso de un servicio para exponer una aplicación en contenedores fuera del clúster. Usamos Controladores y Reglas de Ingress para hacer la misma función. La diferencia es la eficiencia. En lugar de utilizar muchos servicios, como LoadBalancer, puede enrutar el tráfico según la ruta o el host de la solicitud. Esto permite la centralización de muchos servicios en un solo punto.

Un controlador de entrada es diferente a la mayoría de los controladores, ya que no se ejecuta como parte del binario kube-controller-manager. Puede implementar varios controladores, cada uno con configuraciones únicas. Un controlador usa reglas de ingreso para manejar el tráfico hacia y desde fuera del clúster.

Hay muchos controladores de entrada como GKE, nginx, Traefik, Contour y Envoy, por nombrar algunos. Cualquier herramienta capaz de utilizar un proxy inverso debería funcionar. Estos agentes consumen reglas y escuchan el tráfico asociado. Una regla de entrada es un recurso de API que puede crear con **kubectl** . Cuando crea ese recurso, reprograma y reconfigura su Ingress Controller para permitir que el tráfico fluya desde el exterior hacia un servicio interno. Puede dejar un servicio como un tipo ClusterIP y definir cómo se enruta el tráfico a ese servicio interno mediante una regla de ingreso.

## Ingress Controller

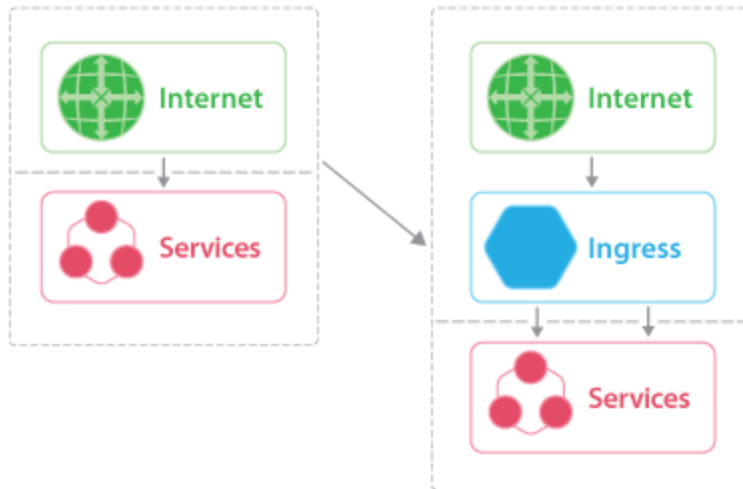
Un controlador de entrada es un demonio que se ejecuta en un pod que observa el endpoint `/ingresses` en el servidor API, que se encuentra en el grupo `networking.k8s.io/v1beta1` para nuevos objetos. Cuando se crea un nuevo endpoint, el demonio usa el conjunto de reglas configurado para permitir la conexión entrante a un servicio, generalmente tráfico HTTP. Esto permite un fácil acceso a un servicio a través de un enrutador de borde a los pods, independientemente de dónde se implemente el pod.

Se pueden implementar varios controladores de entrada. Traffic debe usar anotaciones para seleccionar el controlador adecuado. La falta de una anotación coincidente hará que cada controlador intente satisfacer el tráfico de entrada.



# The Ingress

is a collection of rules that allow inbound connections to reach the cluster services.



El ingress controller para conexiones entrantes

## Nginx

La implementación de un controlador nginx se ha simplificado mediante el uso de los archivos YAML proporcionados, que se pueden encontrar en el [repositorio ingress-nginx / deploy GitHub](#).

<https://github.com/kubernetes/ingress-nginx/tree/master/deploy>

Esta página tiene archivos de configuración para configurar nginx en varias plataformas, como AWS, GKE, Azure y bare metal, entre otras.

Al igual que con cualquier controlador Ingress, existen algunos requisitos de configuración para una implementación adecuada. La personalización se puede realizar a través de un ConfigMap, Anotaciones o, para una configuración detallada, una plantilla personalizada:

- Fácil integración con RBAC
- Utiliza la anotación `kubernetes.io/ingress.class: "nginx"`
- El tráfico L7 requiere la configuración de `proxy-real-ip-cidr`
- Omite kube-proxy para permitir la afinidad de sesiones
- No utiliza entradas de `conntrack` para iptables DNAT
- TLS requiere que se defina el campo de host.

## Google Load Balancer Controller (GLBC)

Hay varios objetos que deben crearse para implementar el controlador de ingreso de GCE. Los archivos YAML están disponibles para facilitar el proceso. Tenga en cuenta que se crearían varios objetos para cada servicio y, actualmente, las cuotas no se evalúan antes de la creación.

El controlador GLBC debe crearse e iniciarse primero. Además, debe crear un ReplicationController con una sola réplica, tres servicios para el pod de la aplicación y un Ingress con dos nombres de host y tres extremos para cada servicio. El backend es un grupo de instancias de máquinas virtuales, Grupo de instancias.

Cada ruta de tráfico utiliza un grupo de objetos similares denominados pool. Cada pool comprueba periódicamente el siguiente salto para garantizar la conectividad.

El camino de multi-pool es:

**Regla de reenvío global -> Proxy HTTP de destino -> Mapa de URL  
-> Servicio de backend -> Grupo de instancias**

Actualmente, TLS Ingress solo admite el puerto 443 y asume la terminación TLS. No es compatible con SNI, solo usa el primer certificado. El secret de TLS debe contener claves denominadas `tls.crt` y `tls.key`.

# Ingress API Resources

Los objetos Ingress ahora forman parte de la API `networking.k8s.io`, pero siguen siendo un objeto beta. Un objeto típico de Ingress que puede POSTAR al servidor API es:

```
apiVersion: networking.k8s.io/v1beta1

kind: Ingress

metadata:

  name: ghost

spec:

  rules:

    - host: ghost.192.168.99.100.nip.io

  http:

  paths:

    - backend:

        serviceName: ghost

        servicePort: 2368
```

Puede administrar los recursos de ingreso como lo hace con los pods, implementaciones, servicios, etc.

```
$ kubectl get ingress
```

```
$ kubectl delete ingress <ingress_name>
```

```
$ kubectl edit ingress <ingress_name>
```

# Implementar Ingress Controller

Para implementar un controlador de entrada, puede ser tan simple como crearlo con **kubectl** . La fuente para una implementación de controlador de muestra está disponible en [GitHub](https://github.com/kubernetes/ingress-nginx/tree/master/deploy) . <https://github.com/kubernetes/ingress-nginx/tree/master/deploy>

```
$ kubectl create -f backend.yaml
```

El resultado será un conjunto de pods administrados por un controlador de replicación y algunos servicios internos. Notará un backend HTTP predeterminado que sirve páginas 404.

```
$ kubectl get pods,rc,svc
```

| NAME                              | READY | STATUS  | RESTARTS | AGE |
|-----------------------------------|-------|---------|----------|-----|
| po/default-http-backend-xvcp8     | 1/1   | Running | 0        | 4m  |
| po/nginx-ingress-controller-fkshn | 1/1   | Running | 0        | 4m  |

| NAME                    | DESIRED | CURRENT | READY | AGE |
|-------------------------|---------|---------|-------|-----|
| rc/default-http-backend | 1       | 1       | 0     | 4m  |

| NAME                     | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
|--------------------------|------------|-------------|---------|-----|
| svc/default-http-backend | 10.0.0.212 | <none>      | 80/TCP  | 4m  |
| svc/kubernetes           | 10.0.0.1   | <none>      | 443/TCP | 77d |

# Creando un Ingress Rule

Para exponerse rápidamente al ingress, puede seguir adelante e intentar crear una regla similar a la mencionada en la página anterior. Primero, inicie una implementación `ghost` y expóngala con un servicio ClusterIP interno:

```
$ kubectl run ghost --image=ghost
```

```
$ kubectl expose deployments ghost --port=2368
```

Con la implementación expuesta y las reglas de Ingress implementadas, debería poder acceder a la aplicación desde fuera del clúster.

```
apiVersion: networking.k8s.io/v1beta1

kind: Ingress

metadata:

  name: ghost

spec:

  rules:

  - host: ghost.192.168.99.100.nip.io

    http:

      paths:

      - backend:

          serviceName: ghost

          servicePort: 2368
```

# Multiple Rules

En la página anterior, definimos una sola regla. Si tiene varios servicios, puede definir varias reglas en el mismo Ingress, y cada regla reenvía el tráfico a un servicio específico.

```
rules:

- host: ghost.192.168.99.100.nip.io

  http:

    paths:

      - backend:

          serviceName: ghost

          servicePort: 2368

- host: nginx.192.168.99.100.nip.io

  http:

    paths:

      - backend:

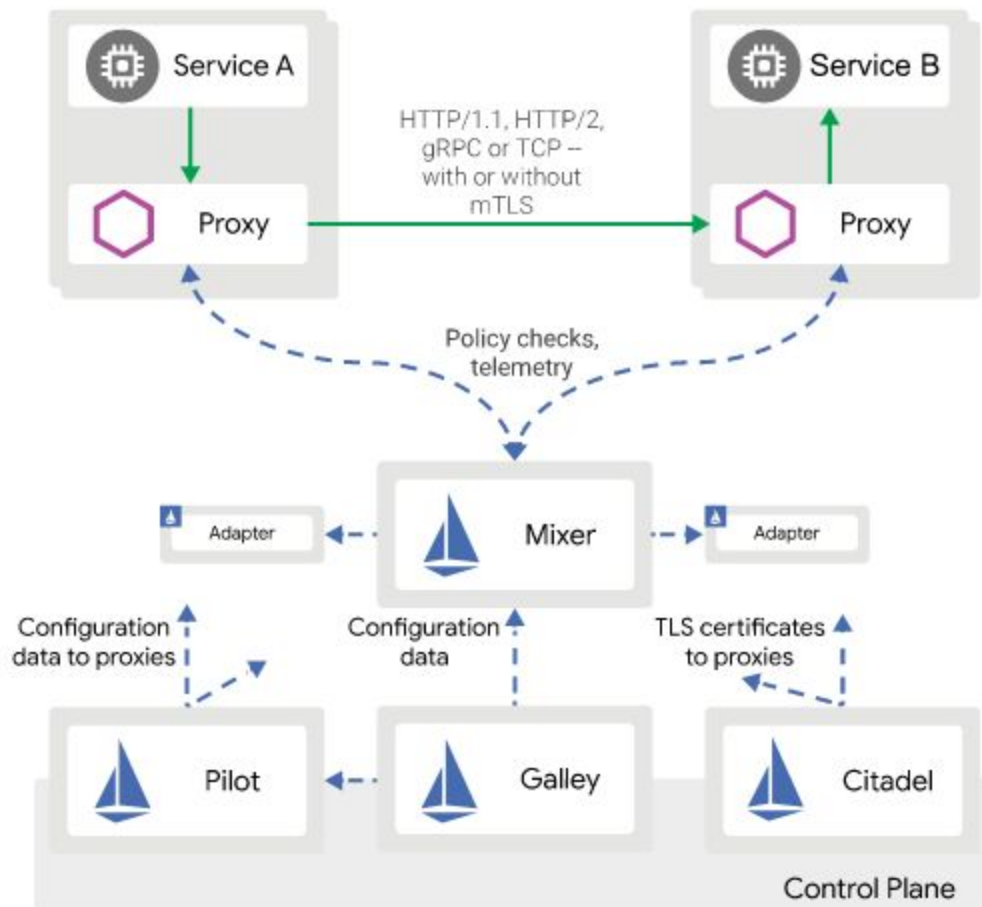
          serviceName: nginx

          servicePort: 80
```

# Proxies inteligentes conectados

Para conexiones o recursos más complejos, como service discovery, rate limiting, gestión del tráfico y métricas avanzadas, es posible que desee implementar un service mesh.

Un service mesh consta de servidores proxy integrados y de borde que se comunican entre sí y manejan el tráfico según las reglas de un control plane. Hay varias opciones disponibles, incluidas Envoy, Istio y Linkerd.



## Istio Service Mesh

<https://istio.io/latest/docs/concepts/what-is-istio/>

## Service Mesh Options:

**Envoy:** Envoy es un proxy modular y extensible favorecido debido a su construcción modular, arquitectura abierta y dedicación a permanecer sin monetizar. A menudo se utiliza como plano de datos en otras herramientas de una malla de servicios.

<https://www.envoyproxy.io/>

**Istio:** Istio es un poderoso conjunto de herramientas que aprovecha los proxies Envoy a través de un control-plan de varios componentes. Está diseñado para ser

independiente de la plataforma y se puede utilizar para hacer que el service mesh sea flexible y esté llena de funciones.

**Linkerd:** [linkerd](#) es otro service mesh, [diseñada expresamente](#) para ser fácil de implementar, rápida y ultraligera. <https://linkerd.io/>

## Lab 10.1 - Advanced Service Exposure

### Question 10.1

¿Cuántos controladores de entrada se admiten actualmente?

- **A.** uno
- **B.** dos
- **C.** tres

### Question 10.2

¿Cuál es la razón principal para utilizar un controlador de entrada en lugar de varios servicios?

- **A.** Es divertido
- **B.** Eficiencia
- **C.** Es requerido
- **D.** Todas las anteriores



## Question 10.3

Se puede configurar tanto el tráfico L4 como el L7. ¿Verdadero o falso?

- **A** . Cierto
- **B** . Falso

## 11. SCHEDULING

### Objetivos de aprendizaje

Al final de este capítulo, debería poder:

- Descubra cómo el programador de kube programa la colocación de pod
- Usa etiquetas para administrar la programación de pods.
- Configure taints y tolerancias.
- Utilice `podAffinity` y `podAntiAffinity` .
- Comprenda cómo ejecutar varios programadores.

## kube-scheduler

Cuanto más grande y diversa se vuelva una implementación de Kubernetes, más importante será la administración de la programación. El kube-scheduler determina qué nodos ejecutarán un Pod mediante un algoritmo con reconocimiento de topología.

Los usuarios pueden establecer la prioridad de un grupo, lo que permitirá la preferencia de grupos de menor prioridad. El desalojo de los grupos de menor prioridad permitirá programar el grupo de mayor prioridad.

El programador rastrea el conjunto de nodos en su clúster, los filtra en función de un conjunto de predicados y luego usa funciones de prioridad para determinar en qué

nodo se debe programar cada pod. La especificación del Pod como parte de una solicitud se envía al kubelet en el nodo para su creación.

La decisión de programación predeterminada puede verse afectada mediante el uso de etiquetas en nodos o pods. Las etiquetas de podAffinity, taints y enlaces de pod permiten la configuración desde el pod o la perspectiva del nodo. Algunas, como las tolerancias, permiten que un Pod funcione con un nodo, incluso cuando el nodo tiene una mancha que, de otro modo, impediría programar un Pod.

No todas las etiquetas son drásticas. La configuración de afinidad puede alentar la implementación de un pod en un nodo, pero lo implementaría en otro lugar si el nodo no estuviera disponible. A veces, la documentación puede usar el término *requerir*, pero la práctica muestra que la configuración es más una solicitud. Como características beta, espere que cambien los detalles. Algunas configuraciones desalojarán los pods de un nodo si la condición requerida ya no se cumple, como `requiredDuringScheduling`, `RequiredDuringExecution`.

Otras opciones, como un programador personalizado, deben programarse e implementarse en su clúster de Kubernetes.

## Predicates

El programador pasa por un conjunto de filtros, o predicados, para encontrar los nodos disponibles, luego clasifica cada nodo usando funciones de prioridad. Se selecciona el nodo con el rango más alto para ejecutar el Pod.

```
predicatesOrdering = []string{CheckNodeConditionPred,  
GeneralPred, HostNamePred,  
PodFitsHostPortsPred, MatchNodeSelectorPred,  
PodFitsResourcesPred,  
NoDiskConflictPred, PodToleratesNodeTaintsPred,  
PodToleratesNodeNoExecuteTaintsPred, CheckNodeLabelPresencePred,  
checkServiceAffinityPred, MaxEBSVolumeCountPred,
```

```
MaxGCEPDVolumeCountPred,MaxAzureDiskVolumeCountPred,  
CheckVolumeBindingPred,NoVolumeZoneConflictPred,  
CheckNodeMemoryPressurePred,CheckNodeDiskPressurePred,  
MatchInterPodAffinityPred}
```

Los predicados, como `PodFitsHost` o `NoDiskConflict`, se evalúan en un orden particular y configurable. De esta manera, un nodo tiene la menor cantidad de comprobaciones para la implementación de un nuevo Pod, lo que puede ser útil para excluir un nodo de comprobaciones innecesarias si el nodo no está en las condiciones adecuadas.

Por ejemplo, hay un filtro llamado `HostNamePred`, que también se conoce como `HostName`, que filtra los nodos que no coinciden con el nombre de nodo especificado en la especificación del pod. Otro predicado es `PodFitsResources` para asegurarse de que la CPU y la memoria disponibles puedan ajustarse a los recursos requeridos por el Pod.

El programador se puede actualizar pasando una configuración del `kind: Policy`, que puede ordenar predicados, dar pesos especiales a las prioridades e incluso `hardPodAffinitySymmetricWeight`, que implementa Pods de manera que si configuramos el Pod A para que se ejecute con el Pod B, entonces el Pod B debería automáticamente ser ejecutado con Pod A.

## Priorities

**Las prioridades** son funciones que se utilizan para ponderar los recursos. A menos que la afinidad de pods y nodos se haya configurado en la configuración `SelectorSpreadPriority`, que clasifica los nodos según la cantidad de pods en ejecución existentes, seleccionarán el nodo con la menor cantidad de pods. Esta es una forma básica de distribuir pods en el clúster.

Se pueden utilizar otras prioridades para necesidades particulares de los clústeres. El **ImageLocalityPriorityMap** favorece nodos que ya se han descargado las imágenes de contenedores. La suma total del tamaño de la imagen se compara con la más grande que tiene la prioridad más alta, pero no verifica la imagen que se va a utilizar.

Actualmente, hay más de diez prioridades incluidas, que van desde verificar la existencia de una etiqueta hasta elegir un nodo con el uso de CPU y memoria más solicitado. Puede ver una lista de prioridades en **master/pkg/scheduler/algorithm/priorities**.

Una característica estable a partir de la v1.14 permite configurar una **PriorityClass** y asignar pods mediante el uso de la configuración de **PriorityClassName**. Esto permite a los usuarios adelantarse o desalojar los pods de menor prioridad para que se puedan programar sus pods de mayor prioridad. El programador de kube determina un nodo donde el pod pendiente podría ejecutarse si uno o más pods existentes fueran desalojados. Si se encuentra un nodo, los pod (s) de baja prioridad se desalojan y se programa el pod de mayor prioridad. El uso de un presupuesto de interrupción de pods (PDB) es una forma de limitar la cantidad de desalojos preventivos de pods para garantizar que se sigan ejecutando suficientes pods. El programador eliminará los pods incluso si se infringe la PDB si no hay otras opciones disponibles.

## Políticas de programación

El planificador predeterminado contiene una serie de predicados y prioridades; sin embargo, estos se pueden cambiar mediante un archivo de política del programador.

A continuación se muestra una versión corta:

```
"kind" : "Policy",
"apiVersion" : "v1",
"predicates" : [
    {"name" : "MatchNodeSelector", "order": 6},
```

```

        {"name" : "PodFitsHostPorts", "order": 2},
        {"name" : "PodFitsResources", "order": 3},
        {"name" : "NoDiskConflict", "order": 4},
        {"name" : "PodToleratesNodeTaints", "order": 5},
        {"name" : "PodFitsHost", "order": 1}
    ],
    "priorities" : [
        {"name" : "LeastRequestedPriority", "weight" : 1},
        {"name" : "BalancedResourceAllocation", "weight" : 1},
        {"name" : "ServiceSpreadingPriority", "weight" : 2},
        {"name" : "EqualPriority", "weight" : 1}
    ],
    "hardPodAffinitySymmetricWeight" : 10
}

```

Por lo general, configurará un programador con esta política mediante el **parámetro** `--policy-config-file` y definirá un nombre para este programador mediante el **parámetro** `--scheduler-name` . Luego, tendrá dos programadores en ejecución y podrá especificar qué programador usar en la especificación del pod.

Con varios programadores, podría haber un conflicto en la asignación de Pod. Cada pod debe declarar qué programador se debe utilizar. Pero, si los programadores independientes determinan que un nodo es elegible debido a los recursos disponibles y ambos intentan implementar, lo que hace que el recurso ya no esté disponible, se produciría un conflicto. La solución actual es que el kubelet local devuelva los pods al programador para su reasignación. Eventualmente, un Pod tendrá éxito y el otro se programará en otro lugar.

# Pod Specification

La mayoría de las decisiones de programación se pueden tomar como parte de la especificación del Pod. Una especificación de pod contiene varios campos que informan la programación, a saber:

- `nodeName`
- `nodeSelector`
- `affinity`
- `schedulerName`
- `tolerations`

**nodeName and nodeSelector:** Las opciones `nodeName` y `nodeSelector` permiten que un Pod se asigne a un solo nodo o un grupo de nodos con etiquetas particulares.

**affinity and anti-affinity:** La afinidad y la antiafinidad pueden usarse para requerir o preferir qué nodo usa el programador. Si se usa una preferencia en su lugar, se elige primero un nodo coincidente, pero se usarán otros nodos si no hay coincidencia.

**taints and tolerations:** El uso de taints permite etiquetar un nodo de modo que los pods no se programen por algún motivo, como el nodo principal después de la inicialización. Una tolerancia permite que un Pod ignore la contaminación y se programe asumiendo que se cumplen otros requisitos.

**schedulerName:** Si ninguna de las opciones anteriores satisface las necesidades del clúster, también existe la posibilidad de implementar un programador personalizado. Cada Pod podría incluir un `schedulerName` para elegir qué programa usar.

# Especificación de la etiqueta de nodo

El campo `nodeSelector` en una especificación de pod proporciona una forma sencilla de apuntar a un nodo o un conjunto de nodos, utilizando uno o más pares clave-valor.

**spec:**

```
containers:
  - name: redis
    image: redis
nodeSelector:
  net: fast
```

La configuración de `nodeSelector` le dice al programador que coloque el pod en un nodo que coincida con las etiquetas. Se deben cumplir todos los selectores enumerados, pero el nodo podría tener más etiquetas. En el ejemplo anterior, cualquier nodo con una clave de `net` establecida en `fast` sería un candidato para la programación. Recuerde que las etiquetas son etiquetas creadas por el administrador, sin relación con los recursos reales. Este nodo podría tener una red lenta.

El pod permanecerá pendiente hasta que se encuentre un nodo con las etiquetas coincidentes.

El uso de affinity/anti-affinity debería poder expresar cada característica como `nodeSelector`.

## Pod Affinity Rules

Los pods que pueden comunicarse mucho o compartir datos pueden funcionar mejor si se ubican juntos, lo que sería una forma de afinidad. Para una mayor tolerancia a fallas, es posible que desee que los pods estén lo más separados posible, lo que sería antiafinidad. El programador utiliza estas configuraciones en función de las etiquetas de los pods que ya se están ejecutando. Como resultado, el programador debe interrogar

a cada nodo y rastrear las etiquetas de los pods en ejecución. Los clústeres de más de varios cientos de nodos pueden sufrir una pérdida significativa de rendimiento. Las reglas de afinidad de pod utilizan los operadores `In` , `NotIn` , `Exists` y `DoesNotExist` .

**requiredDuringSchedulingIgnoredDuringExecution:** El uso de `requiredDuringSchedulingIgnoredDuringExecution` significa que el Pod no se programará en un nodo a menos que el siguiente operador sea verdadero. Si el operador cambia para convertirse en falso en el futuro, el Pod continuará ejecutándose. Esto podría verse como una regla estricta.

**preferredDuringSchedulingIgnoredDuringExecution:** Del mismo modo, `favoriteDuringSchedulingIgnoredDuringExecution` elegirá un nodo con la configuración deseada antes que los que no lo tengan. Si no hay disponibles nodos etiquetados correctamente, el Pod se ejecutará de todos modos. Esta es más una configuración suave, que declara una preferencia en lugar de un requisito.

**podAffinity:** Con el uso de `podAffinity` , el programador intentará programar Pods juntos.

**podAntiAffinity:** El uso de `podAntiAffinity` haría que el programador mantuviera los pods en diferentes nodos.

**topologyKey:** El `topologyKey` permite una agrupación general de Pod deployments. Affinity (o la antiafinidad inversa) intentará ejecutarse en nodos con la clave de topología declarada y ejecutar Pods con una etiqueta en particular. La `topologyKey` puede ser cualquier clave legal, con algunas consideraciones importantes.

- Si usa `requiredDuringScheduling` y la configuración del controlador de admisión `LimitPodHardAntiAffinityTopology` , la `topologyKey` debe establecerse en `kubernetes.io/hostname` .
- Si usa `PreferredDuringScheduling` , se supone que una `topologyKey` vacía es all, o la combinación de `kubernetes.io/hostname` , `topology.kubernetes.io/zone` y `topology.kubernetes.io/region` .



## Ejemplo de podAffinity

A continuación se muestra un ejemplo de configuración de **affinity** y **podAffinity**. Esto también requiere que se haga coincidir una etiqueta en particular cuando se inicia el Pod, pero no es obligatorio si la etiqueta se elimina más tarde.

**spec:**

```
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
              - key: security
                operator: In
                values:
                  - S1
          topologyKey: topology.kubernetes.io/zone
```

Dentro de la zona de topología declarada, el Pod se puede programar en un nodo que ejecuta un Pod con una etiqueta clave de **security** y un valor de **S1**. Si no se cumple este requisito, el Pod permanecerá en estado **Pendiente**.

## Ejemplo de podAntiAffinity

Con **podAntiAffinity**, podemos preferir evitar los nodos con una etiqueta en particular. En este caso, el planificador preferirá evitar un nodo con una clave configurada para la **seguridad** y el valor de **S2**.

**podAntiAffinity:**

```
  preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 100
```

```

podAffinityTerm:
  labelSelector:
    matchExpressions:
      - key: security
        operator: In
        values:
          - S2
  topologyKey: kubernetes.io/hostname

```

En un entorno amplio y variado, puede haber varias situaciones que deben evitarse. Como preferencia, esta configuración intenta evitar ciertas etiquetas, pero aún programará el Pod en algún nodo. Como el Pod aún se ejecutará, podemos dar un peso a una regla en particular. Los pesos se pueden declarar como un valor de 1 a 100. El planificador entonces intenta elegir o evitar el nodo con el mayor valor combinado.

## Reglas de afinidad de nodo

Cuando la afinidad / **antiafinidad** de **pod** tiene que ver con otros **pods** , el uso de **nodeAffinity** permite la programación de **pods** según las etiquetas de los nodos. Esto es similar y algún día reemplazará el uso de la configuración **nodeSelector** . El programador no mirará otros Pods en el sistema, sino las etiquetas de los nodos. Esto debería tener un impacto mucho menor en el rendimiento del clúster, incluso con una gran cantidad de nodos.

- Utiliza los operadores **In** , **NotIn** , **Exists** , **DoesNotExist**
- **requiredDuringSchedulingIgnoredDuringExecution**
- **preferredDuringSchedulingIgnoredDuringExecution**
- Planeado para el futuro:  
**requiredDuringSchedulingRequiredDuringExecution** .

Hasta que `nodeSelector` esté completamente obsoleto, tanto el selector como las etiquetas obligatorias deben cumplirse para que se programe un pod.

## Ejemplo de Node Affinity

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: kubernetes.io/colo-tx-name
                operator: In
                values:
                  - tx-aus
                  - tx-dal
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 1
          preference:
            matchExpressions:
              - key: disk-speed
                operator: In
                values:
                  - fast
                  - quick
```

La primera regla `nodeAffinity` requiere un nodo con una clave de `kubernetes.io/colo-tx-name` que tiene uno de dos valores posibles: `tx-aus` o `tx-dal`.

La segunda regla otorga un peso adicional a los nodos con una clave de `disk-speed` de `disco` con un valor de `fast` o `quick`. El Pod se programará en algún nodo; en cualquier caso, esto solo prefiere una etiqueta en particular.

## Taints

Un nodo con una mancha en particular repelerá los Pods sin tolerancia para esa mancha. Una mancha se expresa como `key=value:effect`. El administrador crea la clave y el valor.

La clave y el valor utilizados pueden ser cualquier cadena legal, y esto permite flexibilidad para evitar que los pods se ejecuten en nodos en función de cualquier necesidad. Si un Pod no tiene una tolerancia existente, el programador no considerará el nodo contaminado.

### Formas de gestionar la programación de pods

**NoSchedule:** El programador no programará un Pod en este nodo, a menos que el Pod tenga esta tolerancia. Los pods existentes continúan ejecutándose, independientemente de la tolerancia.

**PreferNoSchedule:** El programador evitará el uso de este nodo, a menos que no haya nodos no contaminados para la tolerancia de Pods. Los Pods existentes no se ven afectados.

**NoExecute:** Esta contaminación hará que los Pods existentes sean evacuados y no se programen Pods futuros. Si un pod existente tiene tolerancia, continuará ejecutándose. Si se configura el Pod `tolerationSeconds`, permanecerán durante esa cantidad de segundos y luego serán desalojados. Ciertos problemas de nodos harán que kubelet agregue tolerancias de 300 segundos para evitar desalojos innecesarios.

Si un nodo tiene múltiples taints, el programador ignora aquellos con tolerancias coincidentes. Las manchas restantes no ignoradas tienen su efecto típico.

El uso de `TaintBasedEvictions` sigue siendo una función alfa. El kubelet usa taints para limitar la tasa de desalojos cuando el nodo tiene problemas.

## Tolerations

La configuración de tolerancias en un nodo se usa para programar pods en nodos contaminados. Esto proporciona una manera fácil de evitar que los pods usen el nodo. Solo se programarían aquellos con una tolerancia particular.

Se puede incluir un operador en una especificación de Pod, con el valor predeterminado `Equal` si no se declara. El uso del operador `Equal` requiere un valor que coincida. El `Exists` operador no debe ser especificado. Si una clave vacía utiliza el operador `Exists`, tolerará todas las manchas. Si no hay ningún efecto, pero se declaran una clave y un operador, todos los efectos se emparejan con la clave declarada.

```
tolerations:
- key: "server"
  operator: "Equal"
  value: "ap-east"
  effect: "NoExecute"
  tolerationSeconds: 3600
```

En el ejemplo anterior, el Pod permanecerá en el servidor con una clave de `server` y un valor de `ap-east` durante 3600 segundos después de que el nodo se haya contaminado con `NoExecute`. Cuando se acabe el tiempo, el Pod será desalojado.

## Custom Scheduler

Si los mecanismos de programación predeterminados (afinidad, taints, políticas) no son lo suficientemente flexibles para sus necesidades, puede escribir su propio programador. La programación de un programador personalizado está fuera del alcance de este curso, pero es posible que desee comenzar con el código del programador existente, que se puede encontrar en el [repositorio](https://github.com/kubernetes/kubernetes/tree/master/pkg/scheduler) del [Programador en GitHub](#) <https://github.com/kubernetes/kubernetes/tree/master/pkg/scheduler>

Si una especificación de Pod no declara qué programador usar, el programador estándar se usa de forma predeterminada. Si el Pod declara un programador y ese contenedor no se está ejecutando, el Pod permanecerá en estado **Pendiente** para siempre.

El resultado final del proceso de programación es que un pod obtiene un enlace que especifica en qué nodo debe ejecutarse. Un enlace es una primitiva de la API de Kubernetes en el grupo `api/v1`. Técnicamente, sin ningún programador en ejecución, aún puede programar un pod en un nodo, especificando un enlace para ese pod.

También puede ejecutar varios programadores simultáneamente.

Puede ver el planificador y otra información con:

```
$ kubectl get events
```

## Lab 11.1 - Assign Pods Using Labels

## Lab 11.2 - Using Taints to Control Pod Deployment

### Question 11.1

Se pueden implementar varios programadores al mismo tiempo. ¿Verdadero o falso?

- **A.** Cierto
- **B.** Falso

## Question 11.2

Las etiquetas y las anotaciones se utilizan con el mismo propósito. ¿Verdadero o falso?

- **A.** Cierto
- **B.** Falso

## Question 11.3

Cuando un nodo ha sido contaminado, ¿qué requiere un Pod para implementarse en ese nodo?

- **A.** Annotation
- **B.** Permission
- **C.** Ability
- **D.** Toleration

## Question 11.4

Todas las contaminaciones hacen que los pods dejen de ejecutarse en un nodo. ¿Verdadero o falso?

- **A.** Cierto
- **B.** Falso

## 12. LOGGING AND TROUBLESHOOTING

### Objetivos de aprendizaje

Al final de este capítulo, debería poder:

- Comprenda que Kubernetes aún no tiene registros integrados.
- Descubra qué productos externos se utilizan a menudo para agregar registros.
- Examine el flujo básico de resolución de problemas.
- Analice el uso de un sidecar para registros en pod.

### Visión general

Kubernetes se basa en llamadas a la API y es sensible a los problemas de red. Las herramientas y procesos estándar de Linux son el mejor método para solucionar problemas de su clúster. Si un shell, como bash, no está disponible en un pod afectado, considere implementar otro pod similar con un shell, como **busybox** . Los archivos de configuración de DNS y las herramientas como **dig** son un buen lugar para comenzar. Para desafíos más difíciles, es posible que deba instalar otras herramientas, como **tcpdump** .

Las cargas de trabajo grandes y diversas pueden ser difíciles de rastrear, por lo que el monitoreo del uso es esencial. El monitoreo consiste en recopilar métricas clave, como el uso de CPU, memoria y disco, y el ancho de banda de la red en sus nodos, así como también monitorear métricas clave en sus aplicaciones. Estas funciones se están incorporando a Kubernetes con Metric Server, que es una versión reducida del ahora obsoleto Heapster. Una vez instalado, Metrics Server expone una API estándar que



pueden consumir otros agentes, como los escaladores automáticos. Una vez instalado, este punto final se puede encontrar aquí en el servidor maestro:

`/apis/metrics/k8s.io/` .

La actividad de registro en todos los nodos es otra característica que no forma parte de Kubernetes. El uso de Fluentd puede ser un recolector de datos útil para una capa de registro unificada. Tener registros agregados puede ayudar a visualizar los problemas y brinda la capacidad de buscar todos los registros. Es un buen lugar para comenzar cuando la resolución de problemas de la red local no expone la causa raíz. Puede descargarse del [sitio web de Fluentd](https://www.fluentd.org/) <https://www.fluentd.org/>

Otro proyecto de CNCF combina registro, monitoreo y alerta y se llama Prometheus; puede obtener más información en el [sitio web de Prometheus](https://prometheus.io/) <https://prometheus.io/> . Proporciona una base de datos de series de tiempo, así como integración con Grafana para visualización y cuadros de mando.

Vamos a revisar algunos de los comandos básicos de **kubectl** que puede usar para depurar lo que está sucediendo, y lo **guiaremos** a través de los pasos básicos para poder depurar sus contenedores, sus contenedores pendientes y también los sistemas en Kubernetes.

## Basic Troubleshooting Steps

El flujo de solución de problemas debe comenzar con lo obvio. Si hay errores en la línea de comandos, investigue primero. Los síntomas del problema probablemente determinarán el siguiente paso a verificar. Trabajar desde la aplicación que se ejecuta dentro de un contenedor hasta el clúster como un todo puede ser una buena idea. La aplicación puede tener un shell que puede usar, por ejemplo:

```
$ kubectl create deploy busybox --image=busybox --command sleep 3600
```

```
$ kubectl exec -ti <busybox_pod> -- /bin/sh
```

Si el pod se está ejecutando, use `kubectl logs pod-name` para ver el estándar fuera del contenedor. Sin registros, puede considerar implementar un contenedor de sidecar en el Pod para generar y manejar el registro. El siguiente lugar para verificar es la red, incluidos DNS, firewalls y conectividad general, utilizando comandos y herramientas estándar de Linux.

La configuración de seguridad también puede ser un desafío. RBAC, cubierto en el capítulo de seguridad, proporciona control de acceso obligatorio o discrecional de manera granular. SELinux y AppArmor también son problemas comunes, especialmente con aplicaciones centradas en la red.

Una característica más nueva de Kubernetes es la capacidad de habilitar la auditoría para kube-apiserver, que puede permitir ver las acciones después de que se haya aceptado la llamada a la API.

Los problemas encontrados con un sistema desacoplado como Kubernetes son similares a los de un centro de datos tradicional, además de las capas adicionales de los controladores de Kubernetes:

- Errores de la línea de comando
- Registros de pods y estado de los pods
- Utilice el shell para solucionar problemas de red y DNS del Pod
- Verifique los registros de nodos para ver si hay errores, asegúrese de que haya suficientes recursos asignados
- RBAC, SELinux o AppArmor para configuraciones de seguridad
- Llamadas a API desde y hacia controladores a kube-apiserver
- Habilitar la auditoría
- Problemas de red entre nodos, DNS y firewall
- Controladores del servidor maestro (pods de control en estado pendiente o de error, errores en los archivos de registro, recursos suficientes, etc.).

## Contenedores efímeros

Una característica nueva de la versión 1.16 es la capacidad de agregar un contenedor a un pod en ejecución. Esto permitiría agregar un contenedor lleno de funciones a un pod existente sin tener que terminar y volver a crear. Los problemas intermitentes y difíciles de determinar pueden tardar un tiempo en reproducirse o no existir con la adición de otro contenedor.

Como característica de estabilidad alfa, puede cambiar o eliminarse en cualquier momento. Además, no se reiniciarán automáticamente y no se permiten varios recursos, como puertos o recursos.

Estos contenedores se agregan mediante el controlador `ephemeralcontainers` mediante una llamada a la API, no mediante `podSpec`. Como resultado, el uso de `kubectl edit` no es posible.

Es posible que pueda usar el comando `kubectl attach` para unirse a un proceso existente dentro del contenedor. Esto puede resultar útil en lugar de `kubectl exec`, que ejecuta un nuevo proceso. La funcionalidad del proceso adjunto depende completamente de a qué se adjunta.

```
kubectl debug buggypod --image debian --attach
```

## Secuencia de inicio de clúster

La secuencia de inicio del clúster comienza con `systemd` si creó el clúster con `kubeadm`. Otras herramientas pueden aprovechar un método diferente. Utilice `systemctl status kubelet.service` para ver el estado actual y los archivos de configuración utilizados para ejecutar el binario de `kubelet`.

- Utiliza `/etc/systemd/system/kubelet.service.d/10-kubeadm.conf`

Dentro del archivo `config.yaml` encontrará varias configuraciones para el binario, incluido el `staticPodPath` que indica el directorio donde `kubelet` leerá cada archivo

yaml e iniciará cada pod. Si coloca un archivo yaml en este directorio, es una forma de solucionar problemas del programador, ya que el pod se crea con cualquier solicitud al programador.

- Utiliza el archivo de configuración `/var/lib/kubelet/config.yaml`
- `staticPodPath` está configurado en `/etc/kubernetes/manifests/`

Los cuatro archivos yaml predeterminados iniciarán los pods básicos necesarios para ejecutar el clúster:

- kubelet crea todos los pods desde \*.yaml en el directorio: kube-apiserver, etcd, kube-controller-manager, kube-Scheduler.

Una vez que los bucles de vigilancia y los controladores de kube-controller-manager se ejecuten utilizando datos etcd, se creará el resto de los objetos configurados.

## Monitoring

El monitoreo consiste en recopilar métricas de la infraestructura, así como de aplicaciones.

El Heapster, usado durante mucho tiempo y ahora en desuso, ha sido reemplazado por un servidor de métricas integrado. Una vez instalado y configurado, el servidor expone una API estándar que otros agentes pueden utilizar para determinar el uso. También se puede configurar para exponer métricas personalizadas, que luego los escaladores automáticos también podrían usar para determinar si se debe realizar una acción.



Prometheus es parte de Cloud Native Computing Foundation (CNCF). Como complemento de Kubernetes, permite extraer métricas de uso de recursos de los objetos de Kubernetes en todo el clúster. También tiene varias bibliotecas de cliente que le permiten instrumentar el código de su aplicación para recopilar métricas a nivel de la aplicación.

# Usando krew

Hemos estado usando el comando `kubectl` durante todo el curso. Los comandos básicos se pueden usar juntos de una manera más compleja ampliando lo que se puede hacer. Hay más de setenta y cada vez más complementos disponibles para interactuar con los objetos y componentes de Kubernetes.

En el momento en que se escribió este curso, los complementos no pueden sobrescribir los comandos `kubectl` existentes , ni pueden agregar subcomandos a los comandos existentes. La escritura de nuevos complementos debe tener en cuenta el paquete de tiempo de ejecución de la línea de comandos y una biblioteca Go para los autores de complementos.

Como complemento, la declaración de opciones como el espacio de nombres o el contenedor a utilizar debe ir después del comando.

```
$ kubectl sniff bigpod-abcd-123 -c mainapp -n accounting
```

Los complementos se pueden distribuir de muchas formas. El uso de `krew` (el **administrador de complementos de `kubectl`** ) permite el empaquetado multiplataforma y un índice de complementos útil, lo que facilita la búsqueda de nuevos complementos.

Instale el software siguiendo los pasos disponibles en [el repositorio GitHub de krew](https://github.com/kubernetes-sigs/krew) <https://github.com/kubernetes-sigs/krew/> .

```
$ kubectl krew help
```

Puede invocar `krew` a través de `kubectl`:

```
kubectl krew [command]...
```

Uso:

```
Krew [command]
```

## Comandos disponibles

```

Help Ayuda sobre cualquier comando
Info Mostrar información sobre el complemento kubectl
Install Instalar complementos de kubectl
List Lista de complementos de kubectl instalados
Search Descubra los complementos de kubectl
Uninstall Desinstalar complementos
Update Actualizar la copia local del índice del complemento
Upgrade Actualice los complementos instalados a versiones más nuevas
Version Mostrar versión y diagnóstico de krew

```

Puede encontrar más información en la Documentación de Kubernetes, "Extender kubectl con complementos" .

<https://kubernetes.io/docs/tasks/extend-kubectl/kubectl-plugins/>

## Administrar complementos

La opción de ayuda explica el funcionamiento básico. Después de la instalación, asegúrese de que \$ PATH incluya los complementos. krew debería permitir una fácil instalación y uso después de eso.

```
$ export PATH="$ {KREW_ROOT:-$HOME/.krew}/bin:$PATH"
```

```
$ kubectl krew search
```

| NAME          | DESCRIPTION                                     |
|---------------|---|
| access-matrix | Show an RBAC access matrix for server resources |
| advice-psp    | Suggests PodSecurityPolicies for cluster.       |
| .....         |   |

```
$ kubectl krew install tail
```

Updated the local copy of plugin index.

Installing plugin: tail

Installed plugin: tail

\

| Use this plugin:

....

| | Usage:

| |

| | # match all pods

| | \$ kubectl tail

| |

| | # match pods in the 'frontend' namespace

| | \$ kubectl tail --ns staging

....

Para ver los complementos actuales, use:

```
kubectl plugin list
```

Para encontrar nuevos complementos, use:

```
kubectl krew search
```

Para instalar use:

```
kubectl krew install new-plugin
```

Una vez instalado, utilícelo como subcomando `kubectl` . También puede actualizar y desinstalar.

## Sniffing Traffic With Wireshark

El tráfico de la red del clúster está cifrado, lo que hace que la resolución de posibles problemas de red sea más compleja. Con el complemento sniff puede ver el tráfico desde dentro. sniff requiere Wireshark y la capacidad de exportar pantallas gráficas.

El comando sniff usará el primer contenedor encontrado a menos que pase la opción `-c` para declarar qué contenedor en el pod usar para monitorear el tráfico.

```
$ kubectl krew install sniff nginx-123456-abcd -c webcont
```

## Herramientas de registro

El registro, como el monitoreo, es un tema muy amplio en TI. Tiene muchas herramientas que puede utilizar como parte de su arsenal.

Por lo general, los registros se recopilan localmente y se agregan antes de ser ingeridos por un motor de búsqueda y mostrados a través de un panel que puede usar la sintaxis de búsqueda. Si bien hay muchas pilas de software que puede usar para el registro , [Elasticsearch](#), [Logstash](#) y [Kibana Stack](#) (ELK) se han vuelto bastante comunes.

En Kubernetes, kubelet escribe registros de contenedores en archivos locales (a través del controlador de registro de Docker). El comando `kubectl logs` le permite recuperar estos registros.



En todo el clúster, puede usar [Fluentd](#) para agregar registros. Consulte los [conceptos de registro de administración de clústeres](#) para obtener una descripción detallada.

<https://kubernetes.io/docs/concepts/cluster-administration/logging/>

Fluentd es parte de Cloud Native Computing Foundation y, junto con Prometheus, hacen una buena combinación para monitorear y registrar. Puede encontrar un [tutorial detallado sobre cómo ejecutar Fluentd en Kubernetes](#) en la documentación de Kubernetes.

<https://kubernetes.io/docs/tasks/debug-application-cluster/logging-elasticsearch-kibana/>



Configurar Fluentd para el registro de Kubernetes es un buen ejercicio para comprender DaemonSets. Los agentes de Fluentd se ejecutan en cada nodo a través de un DaemonSet, agregan los registros y los alimentan a una instancia de Elasticsearch antes de visualizarlos en un panel de Kibana.

## Más recursos

Hay varias cosas que puede hacer para diagnosticar rápidamente problemas potenciales con su aplicación y / o clúster. La documentación oficial ofrece materiales adicionales para ayudarlo a familiarizarse con la resolución de problemas:

- [Pautas e instrucciones generales para la solución de problemas](#)
- [Aplicaciones de resolución de problemas](#)
- [Solución de problemas de clústeres](#)
- [Pods de depuración](#)
- [Servicios de depuración](#)
- [Sitio web de GitHub para problemas y seguimiento de errores](#)
- [Canal Slack de Kubernetes](#) .

## Lab 12.1 - Review Log File Locations

## Lab 12.2 - Viewing Log Outputs

## Lab 12.3 - Adding Tools for Monitoring and Metrics

### Question 12.1

Kubernetes tiene un registro integrado en todo el clúster. ¿Verdadero o falso?

- **A.** Cierto
- **B.** Falso

### Question 12.2

Si un contenedor no proporciona registro, ¿qué se podría usar para generar y manejar el registro en el Pod?

- **A.** Deployment
  - **B.** Controller
  - **C.** Kubelet
  - **D.** Sidecar container
- 

## 13. CUSTOM RESOURCE DEFINITIONS

## Objetivos de aprendizaje

Al final de este capítulo, debería poder:

- Haga crecer los objetos de Kubernetes disponibles.
- Implemente una nueva definición de recurso personalizada.

- Implemente un nuevo recurso y un punto final de API.
- Analice las API agregadas.

## Custom Resources

Hemos estado trabajando con recursos integrados o puntos finales de API. La flexibilidad de Kubernetes también permite la adición dinámica de nuevos recursos. Una vez que se han agregado estos recursos personalizados, los objetos se pueden crear y acceder a ellos mediante llamadas y comandos estándar, como **kubectl**. La creación de un nuevo objeto almacena nuevos datos estructurados en la base de datos etcd y permite el acceso a través de kube-apiserver.

Para que un nuevo recurso personalizado forme parte de una API declarativa, debe haber un controlador que recupere los datos estructurados de forma continua y actúe para cumplir y mantener el estado declarado. Este controlador u operador es un agente que crea y administra una o más instancias de una aplicación con estado específica. Hemos trabajado con controladores integrados como Deployments, DaemonSets y otros recursos.

Las funciones codificadas en un operador personalizado deberían ser todas las tareas que un humano necesitaría realizar si implementara la aplicación fuera de Kubernetes. Los detalles de la construcción de un controlador personalizado están fuera del alcance de este curso y, por lo tanto, no están incluidos.

Hay dos formas de agregar recursos personalizados a su clúster de Kubernetes. La forma más fácil, pero menos flexible, es agregar una definición de recurso personalizada (CRD) al clúster. La segunda forma, que es más flexible, es el uso de API agregadas (AA), que requiere que se escriba y se agregue un nuevo servidor de API al clúster.

Cualquiera de las formas de agregar un nuevo objeto al clúster, a diferencia de un recurso integrado, se denomina recurso personalizado.

Si está utilizando RBAC para la autorización, probablemente necesitará otorgar acceso al nuevo recurso y controlador CRD. Si usa una API agregada, puede usar el mismo proceso de autenticación o uno diferente.

## Definiciones de recursos personalizadas

Como ya hemos aprendido, la naturaleza desacoplada de Kubernetes depende de una colección de bucles de observadores, o controladores, que interrogan al kube-apiserver para determinar si una configuración particular es verdadera. Si el estado actual no coincide con el estado declarado, el controlador realiza llamadas a la API para modificar el estado hasta que coincidan. Si agrega un nuevo objeto y controlador de API, puede usar el kube-apiserver existente para monitorear y controlar el objeto. La adición de una definición de recurso personalizada se agregará a la ruta de la API del clúster, actualmente en `apiextensions.k8s.io/v1`.

Si bien esta es la forma más sencilla de agregar un nuevo objeto al clúster, es posible que no sea lo suficientemente flexible para sus necesidades. Solo se puede utilizar la funcionalidad API existente. Los objetos deben responder a las solicitudes REST y tener su estado de configuración validado y almacenado de la misma manera que los objetos integrados. También tendrían que existir con las reglas de protección de los objetos integrados.

Un CRD permite que el recurso se implemente en un espacio de nombres o esté disponible en todo el clúster. El archivo YAML establece esto con el parámetro `scope:`, que se puede establecer en `Namespaced` o `Cluster`.

Antes de la versión 1.8, existía un tipo de recurso llamado `ThirdPartyResource` (TPR). Esto ha quedado obsoleto y ya no está disponible. Todos los recursos deberán reconstruirse como CRD. Después de la actualización, los TPR existentes deberán eliminarse y reemplazarse por CRD de manera que la URL de la API apunte a objetos funcionales.

## Ejemplo de configuración

```
apiVersion: apiextensions.k8s.io/v1
```

```
kind: CustomResourceDefinition
```

```
metadata:
```

```
  name: backups.stable.linux.com
```

```
spec:
```

```
  group: stable.linux.com
```

```
  version: v1
```

```
  scope: Namespaced
```

```
  names:
```

```
    plural: backups
```

```
    singular: backup
```

```
    shortNames:
```

```
      - bks
```

```
  kind: BackUp
```

**apiVersion:** Debe coincidir con el nivel actual de estabilidad, que es `apiextensions.k8s.io/v1`.

**kind: CustomResourceDefinition:** El tipo de objeto que está insertando kube-apiserver.

**name: backups.stable.linux.com**: El nombre debe coincidir con el campo de **especificación** declarado más adelante. La sintaxis debe ser **<plural name>**. **<Grupo>**.

**group: stable.linux.com**: El nombre del grupo pasará a formar parte de la API REST en **/apis/<group>/<version>** o **/apis/stable/v1** en este caso con la versión establecida en v1.

**scope**: Determina si el objeto existe en un solo namespace o es para todo el clúster.

**plural**: Define la última parte de la URL de la API, como **apis/stable/v1/backups**

**singular and shortNames**: Representan el nombre que se muestra y facilitan el uso de CLI.

**kind**: Un tipo singular basado en Camel utilizado en los manifiestos de recursos.

## Configuración de nuevo objeto

```
apiVersion: "stable.linux.com/v1"
kind: BackUp
metadata:
  name: a-backup-object
spec:
  timeSpec: "* * * * */5"
  image: linux-backup-image
replicas: 5
```

Tenga en cuenta que **apiVersion** y **kind** coinciden con el CRD que creamos en un paso anterior. Los parámetros de **especificaciones** dependen del controlador.

El objeto será evaluado por el controlador. Si la sintaxis, como **timeSpec**, no coincide con el valor esperado, recibirá un error, si se configura la validación. Sin validación, solo se verifica la existencia de la variable, no sus detalles.

## Optional Hooks

Al igual que con los objetos integrados, puede utilizar un gancho de eliminación previa asíncrono conocido como **Finalizer**. Si se recibe una solicitud de **delete de API**,

se actualiza el campo de `metadatos` del objeto `metadata.deletionTimestamp`. A continuación, el controlador activa el finalizador que se haya configurado. Cuando finaliza el finalizador, se elimina de la lista. El controlador continúa completando y eliminando finalizadores hasta que la cadena está vacía. Luego, el objeto en sí se elimina.

Finalizer:

`metadata:`

`finalizers:`

`- finalizer.stable.linux.com`

Validation:

`validation:`

`openAPIV3Schema:`

`properties:`

`spec:`

`properties:`

`timeSpec:`

`type: string`

`pattern: '^(\\d+|\\*) (/\\d+)? (\\s+ (\\d+|\\*) (/\\d+)?) {4}$'`

`replicas:`

`type: integer`

`minimum: 1`

`maximum: 10`

Una función en beta que comienza con v1.9 permite la validación de objetos personalizados a través del esquema OpenAPI v3. Esto comprobará varias propiedades de la configuración del objeto que pasa el servidor API. En el ejemplo

anterior, `timeSpec` debe ser una cadena que coincida con un patrón particular y el número de réplicas permitidas está entre 1 y 10. Si la validación no coincide, el error devuelto es la línea de validación fallida.

## Comprensión de las API agregadas

El uso de API agregadas permite agregar servidores API de tipo Kubernetes adicionales al clúster. El servidor agregado actúa como subordinado de kube-apiserver, que, a partir de la v1.7, ejecuta la capa de agregación en proceso. Cuando se registra un recurso de extensión, la capa de agregación observa una ruta de URL pasada y envía cualquier solicitud al servicio de API recién registrado.

La capa de agregación es fácil de habilitar. Edite las marcas pasadas durante el inicio de kube-apiserver para incluir `--enable-aggregator-routing = true`. Algunos proveedores habilitan esta función de forma predeterminada.

La creación del exterior se puede realizar a través de archivos de configuración YAML o API. También se requiere configurar la autorización TLS entre componentes y reglas RBAC para varios objetos nuevos. Hay un [servidor de API de muestra](#) disponible en GitHub. Un proyecto que se encuentra actualmente en la etapa de incubación es un [constructor de servidores API](#) que debería manejar gran parte de la configuración de seguridad y conexión.

## Lab 13.1 - Create a Custom Resource Definition



## Question 13.1

¿Qué usamos cuando agregamos un nuevo objeto API a kube-apiserver?

- **A.** Exterior object
- **B.** Custom Resource Definition
- **C.** Aggregated APIs
- **D.** ExtendedAPI

## Question 13.2

¿Qué usamos para agregar un nuevo servidor API al clúster que actúa como subordinado de kube-apiserver?

- **A.** Exterior object
- **B.** Custom Resource Definition
- **C.** Aggregated APIs
- **D.** ExtendedAPI

## Question 13.3

Todas las definiciones de recursos personalizadas deben existir en un namespaces.  
¿Verdadero o falso?

- **A.** Cierto
- **B.** Falso

## **14. HELM**

### **Objetivos de aprendizaje**

Al final de este capítulo, debería poder:

- Examine las implementaciones sencillas de Kubernetes con el administrador de paquetes Helm.
- Comprenda la plantilla de gráfico que se utiliza para describir qué aplicación implementar.
- Analice cómo Tiller crea la implementación basada en el gráfico.
- Inicialice Helm en un clúster.

### **Implementación de aplicaciones complejas**

Hemos utilizado herramientas de Kubernetes para implementar aplicaciones Docker simples. A partir de la versión v1.4, el objetivo era tener una ubicación canónica para el software. Helm es similar a un administrador de paquetes como **yum** o **apt** , con un gráfico similar a un paquete. Helm v3 es significativamente diferente a v2.

Una aplicación típica en contenedores tendrá varios manifiestos. Manifiestos para implementaciones, servicios y ConfigMaps. Probablemente también creará algunos secretos, Ingress y otros objetos. Cada uno de estos necesitará un manifiesto.

Con Helm, puede empaquetar todos esos manifiestos y hacerlos disponibles como un solo tarball. Puede poner el tarball en un repositorio, buscar en ese repositorio, descubrir una aplicación y luego, con un solo comando, implementar e iniciar toda la aplicación.

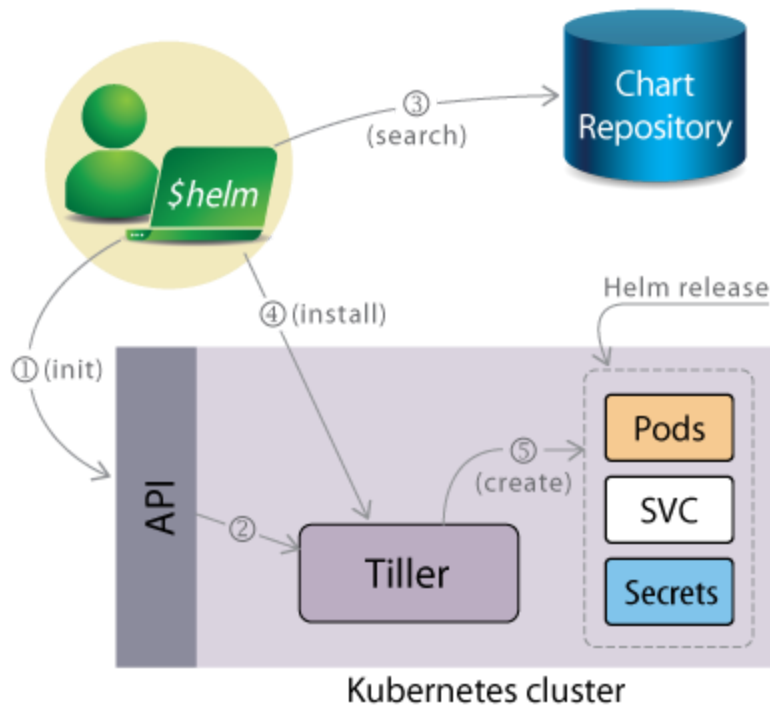
El servidor se ejecuta en su clúster de Kubernetes y su cliente es local, incluso una computadora portátil local. Con su cliente, puede conectarse a múltiples repositorios de aplicaciones.

También podrá actualizar o revertir una aplicación fácilmente desde la línea de comandos.

## Helm v2 y Tiller

La herramienta helm empaqueta una aplicación de Kubernetes mediante una serie de archivos YAML en un gráfico o paquete. Esto permite compartir de forma sencilla entre usuarios, sintonizar mediante un esquema de plantillas, así como el seguimiento de procedencia, entre otras cosas.

# HELM



## Basic Helm and Tiller Flow

Helm v2 consta de dos componentes:

- Un servidor llamado Tiller, que se ejecuta dentro de su clúster de Kubernetes.
- Un cliente llamado Helm, que se ejecuta en su máquina local.

Helm versión 2 usa un pod Tiller para implementar en el clúster. Esto ha provocado muchos problemas con la seguridad y los permisos del clúster. El nuevo Helm v3 no implementa un pod.

Con el cliente Helm, puede buscar repositorios de paquetes (que contienen Charts publicados) e implementar esos Charts en su clúster de Kubernetes. Helm descargará el chart y enviará una solicitud a Tiller para crear una versión, también conocida como instancia de chart. El lanzamiento se realizará con varios recursos que se ejecutan en el clúster de Kubernetes.

# Helm v3

Con la revisión casi completa de Helm, los procesos y comandos han cambiado bastante. Espere dedicar algo de tiempo a actualizar e integrar estos cambios si actualmente está utilizando Helm v2.

Uno de los cambios más notables es la eliminación del módulo Tiller. Este era un problema de seguridad continuo, ya que el pod necesitaba permisos elevados para implementar gráficos. La funcionalidad está solo en el comando y ya no requiere inicialización para su uso.

En la versión 2, una actualización de un chart y deployment utilizaron una combinación estratégica bidireccional para el parcheo. Esto comparó el manifiesto anterior con el manifiesto previsto, pero no las posibles ediciones realizadas fuera de los comandos de `helm`. La tercera forma ahora comprobada es el estado vivo de los objetos.

Entre otros cambios, la instalación del software ya no genera un nombre automáticamente. Se debe proporcionar uno o se debe pasar la opción `--generated-name`.

## Chart Contents

Un **chart** es un conjunto archivado de manifiestos de recursos de Kubernetes que componen una aplicación distribuida. Puede obtener más información en la [documentación de Helm 3](#). Otros existen y pueden crearse fácilmente, por ejemplo, por un proveedor que proporcione software. Los gráficos son similares al uso de repositorios YUM independientes.

```
|— Chart.yaml
|— README.md
|— templates
|   |— NOTES.txt
|   |— _helpers.tpl
|   |— configmap.yaml
|   |— deployment.yaml
```

```
|   ├── pvc.yaml
|   ├── secrets.yaml
|   └── svc.yaml
└── values.yaml
```

**chart.yaml:** El archivo `chart.yaml` contiene algunos metadatos sobre el Chart, como su nombre, versión, palabras clave, etc., en este caso, para MariaDB.

**values.yaml:** El archivo `values.yaml` contiene claves y valores que se utilizan para generar la versión en su clúster. Estos valores se reemplazan en los manifiestos de recursos mediante la sintaxis de plantillas Go.

**templates:** El directorio de `templates` contiene los manifiestos de recursos que componen esta aplicación MariaDB.

## Templates

Las plantillas son manifiestos de recursos que utilizan la sintaxis de plantillas de Go. Las variables definidas en el archivo de `values`, por ejemplo, se inyectan en la plantilla cuando se crea una versión. En el ejemplo de MariaDB que proporcionamos, las contraseñas de la base de datos se almacenan en un secret de Kubernetes y la configuración de la base de datos se almacena en un ConfigMap de Kubernetes.

Podemos ver que un conjunto de etiquetas se definen en los metadatos secretos utilizando el nombre del chart, Release name, etc. Los valores reales de las contraseñas se leen del archivo `values.yaml`.

```
apiVersion: v1
kind: Secret
metadata:
  name: {{ template "fullname" . }}
  labels:
    app: {{ template "fullname" . }}
    chart: "{{ .Chart.Name }}" - "{{ .Chart.Version }}"
    release: "{{ .Release.Name }}"
    heritage: "{{ .Release.Service }}"
type: Opaque
data:
```

```
    mariadb-root-password: {{ default ""  
.Values.mariadbRootPassword | b64enc | quote }}  
    mariadb-password: {{ default "" .Values.mariadbPassword |  
b64enc | quote }}
```

## Inicializando Helm v2

No es necesario inicializar Helm v3.

Como siempre, puede compilar Helm desde el código fuente o descargar un tarball. Esperamos ver pronto los paquetes de Linux para la versión estable. Los requisitos de seguridad actuales de RBAC para implementar helm requieren la creación de un nuevo `serviceaccount` y la asignación de permisos y roles. Hay varias configuraciones opcionales que se pueden pasar al comando `helm init`, generalmente para problemas de seguridad particulares, opciones de almacenamiento y también una opción de ejecución en seco.

```
$ helm init
```

```
...
```

```
Tiller (the helm server side component) has been installed into your  
Kubernetes Cluster.
```

```
Happy Helming!
```

```
$ kubectl get deployments --namespace=kube-system
```

| NAMESPACE   | NAME          | READY | UP-TO-DATE | AVAILABLE | AGE |
|-------------|---------------|-------|------------|-----------|-----|
| kube-system | tiller-deploy | 1/1   | 1          | 1         | 15s |

La inicialización de helm v2 debería haber creado un nuevo pod de `tiller-deploy` en su clúster. Tenga en cuenta que esto creará una implementación en el namespace del `kube-system`.

El cliente podrá comunicarse con el helm mediante el reenvío de puertos. Por lo tanto, no verá ningún servicio que exponga el tiller.

# Chart Repositories

Se incluye un repositorio predeterminado al inicializar helm, pero es común agregar otros repositorios. Los repositorios son actualmente servidores HTTP simples que contienen un archivo de índice y un tarball de todos los gráficos presentes.

Puede interactuar con un repositorio utilizando los comandos de `helm repo`.

```
$ helm repo add testing
http://storage.googleapis.com/kubernetes-charts-testing
```

```
$ helm repo list
```

```
NAME URL
```

```
stable http://storage.googleapis.com/kubernetes-charts
```

```
local http://localhost:8879/charts
```

```
testing http://storage.googleapis.com/kubernetes-charts...
```

Una vez que tenga un repositorio disponible, puede buscar charts basados en palabras clave. A continuación, busquemos un chart redis :

```
$ helm search redis
```

```
WARNING: Deprecated index file format. Try 'helm repo update'
```

| NAME                     | VERSION | DESCRIPTION  |
|--------------------------|---------|--|
| testing/redis-cluster    | 0.0.5   | Highly available Redis cluster with multiple se... |
| testing/redis-standalone | 0.0.1   | Standalone Redis Master                            |
| testing/...              |         |  |



Una vez que encuentre el gráfico dentro de un repositorio, puede implementarlo en su clúster.

## Implementar un chart

Para implementar un chart, puede usar el comando `helm install`. Puede haber varios recursos necesarios para que la instalación sea exitosa, como PV disponibles para coincidir con el PVC del chart. Actualmente, la única forma de descubrir qué recursos deben existir es leyendo los archivos `READMEs` de cada gráfico:

```
$ helm install testing/redis-standalone
```

```
Fetches testing/redis-standalone to redis-standalone-0.0.1.tgz
```

```
amber-eel
```

```
Last Deployed: Fri Oct 21 12:24:01 2016
```

```
Namespace: default
```

```
Status: DEPLOYED
```

```
Resources:
```

```
==> v1/ReplicationController
```

| NAME             | DESIRED | CURRENT | READY | AGE |
|------------------|---------|---------|-------|-----|
| redis-standalone | 1       | 1       | 0     | 1s  |

```
==> v1/Service
```

| NAME  | CLUSTER-IP | EXTERNAL-IP | PORT(S)  | AGE |
|-------|------------|-------------|----------|-----|
| redis | 10.0.81.67 | <none>      | 6379/TCP | 0s  |

Podrá enumerar la versión, eliminarla, incluso actualizarla y revertirla.

```
$ helm list
```

| NAME             | REVISION | UPDATED                  | STATUS   | CHART |
|------------------|----------|--------------------------|----------|-------|
| amber-eel        | 1        | Fri Oct 21 12:24:01 2016 | DEPLOYED |       |
| redis-standalone | 0.0.1    |                          |          |       |

Se creará un nombre único y colorido para cada instancia de helm implementada. También puede usar **kubectl** para ver los nuevos recursos que Helm creó en su clúster.

El resultado de la implementación debe revisarse cuidadosamente. A menudo incluye información sobre el acceso a las aplicaciones que contiene. Si su clúster no tenía un recurso de clúster necesario, el resultado suele ser el primer lugar para comenzar a solucionar problemas.

## Lab 14.1 - Working with Helm and Charts

### Question 14.1

¿Cuál de las siguientes es la plantilla que describe la aplicación a implementar, las configuraciones y las dependencias?

- **A.** Chart
- **B.** Tiller
- **C.** Helm
- **D.** Repository

### Question 14.2

Un resultado de implementación de chart nos informa sobre las dependencias faltantes. ¿Verdadero o falso?

- **A.** Cierto
- **B.** Falso

## Question 14.3

¿Cuál de los siguientes es el agente que implementa objetos según un chart?

- **A.** Chart
- **B.** Tiller
- **C.** Helm
- **D.** Repository

## Question 14.4

¿Cómo se llama una colección de gráficos?

- **A.** Chart
- **B.** Tiller
- **C.** Helm
- **D.** Repository

## **15. SECURITY**

## Objetivos de aprendizaje

Al final de este capítulo, debería poder:

- Explique el flujo de solicitudes de API.
- Configure las reglas de autorización.
- Examine las políticas de autenticación.
- Restrinja el tráfico de la red con políticas de red.

## Visión general

La seguridad es un tema grande y complejo, especialmente en un sistema distribuido como Kubernetes. Por lo tanto, solo vamos a cubrir algunos de los conceptos que tratan con la seguridad en el contexto de Kubernetes.

Luego, nos centraremos en el aspecto de autenticación del servidor de API y profundizaremos en la autorización, analizando cosas como ABAC y RBAC, que ahora es la configuración predeterminada cuando arranca un clúster de **Kubernetes** con **kubeadm**.

Vamos a ver el sistema de **control de admisión**, que le permite ver y posiblemente modificar las solicitudes que están entrando, y hacer una denegación o aceptación final de esas solicitudes.

Después de eso, veremos algunos otros conceptos, incluido cómo puede proteger sus pods de manera más estricta utilizando contextos de seguridad y políticas de seguridad de pod, que son objetos API completos en Kubernetes.

Finalmente, veremos las políticas de red. De forma predeterminada, tendemos a no activar las políticas de red, que permiten que el tráfico fluya a través de todos nuestros pods, en todos los diferentes namespaces. Usando políticas de red, podemos definir reglas de Ingress para que podamos restringir el tráfico de Ingress entre los diferentes namespaces. La herramienta de red en uso, como Flannel o Calico, determinará si se puede implementar una política de red. A medida que Kubernetes se vuelva más maduro, esta se convertirá en una configuración muy sugerida.

# Accediendo a la API

Para realizar cualquier acción en un clúster de Kubernetes, debe acceder a la API y seguir tres pasos principales:

- Autenticación:
- Autorización (ABAC o RBAC):
- Control de admisión.

Estos pasos se describen con más detalle en la documentación oficial sobre el [control del acceso a la API de Kubernetes](#) y se ilustran en el diagrama a continuación.

## Autenticación

Hay tres puntos principales para recordar con la autenticación en Kubernetes:

- En su forma sencilla, la autenticación se realiza con certificados, tokens o autenticación básica (es decir, nombre de usuario y contraseña).
- Los usuarios no son creados por la API, pero deben ser administrados por un sistema externo.
- Los procesos utilizan las cuentas del sistema para acceder a la API (para obtener más información, lea [Configurar cuentas de servicio para pods](#) ).

Si desea obtener más información sobre cómo los procesos utilizan las cuentas del sistema para acceder a la API:

Hay dos mecanismos de autenticación más avanzados. Los webhooks se pueden utilizar para verificar tokens de portador y la conexión con un proveedor OpenID externo.

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 la elección de mecanismo de autenticación que elija:

`--basic-auth-file`

`--oidc-issuer-url`

`--token-auth-file`

`--authorization-webhook-config-file`

Se utilizan uno o más módulos autenticadores:

- x509 Client Certs;
- static token, bearer or bootstrap token;
- static password file;
- service account;
- OpenID connect tokens.

Cada uno se prueba hasta que tiene éxito y el pedido no está garantizado. También se puede habilitar el acceso anónimo; de lo contrario, recibirá una respuesta 401. Los usuarios no son creados por la API y deben ser administrados por un sistema externo.

Para obtener más información sobre la autenticación, consulte la [documentación](#) oficial de [Kubernetes](#) .

## Autorización

Una vez que se autentica una solicitud, debe estar autorizada para poder continuar a través del sistema Kubernetes y realizar la acción prevista.

Hay tres modos de autorización principales y dos configuraciones globales de Denegar / Permitir. Los tres modos principales son:

- ABAC
- RBAC
- Webhook.

Se pueden configurar como opciones de inicio de kube-apiserver:

```
--authorization-mode=ABAC
```

```
--authorization-mode=RBAC
```

```
--authorization-mode=Webhook
```

```
--authorization-mode=AlwaysDeny
```

```
--authorization-mode=AlwaysAllow
```

Los modos de autorización implementan políticas para permitir solicitudes. Los atributos de las solicitudes se comparan con las políticas (por ejemplo, usuario, grupo, namespaces, verbo).

## Modos ABAC, RBAC y Webhook

**ABAC:** **ABAC** significa Control de acceso basado en atributos. Fue el primer modelo de autorización en Kubernetes que permitió a los administradores implementar las políticas adecuadas. Hoy, RBAC se está convirtiendo en el modo de autorización predeterminado.

Las políticas se definen en un archivo JSON y se hace referencia a ellas mediante una opción de inicio de kube-apiserver:

```
--authorization-policy-file=my_policy.json
```

Por ejemplo, el archivo de política que se muestra a continuación autoriza al usuario Bob a leer pods en el namespace **foobar** :

```
{  
  
  "apiVersion": "abac.authorization.kubernetes.io/v1beta1",  
  
  "kind": "Policy",
```

```
"spec": {  
  
  "user": "bob",  
  
  "namespace": "foobar",  
  
  "resource": "pods",  
  
  "readonly": true  
  
}
```

Puede consultar otros [ejemplos de políticas](#) en la documentación de Kubernetes.

**RBAC:** **RBAC** significa Control de acceso basado en roles.

Todos los recursos son objetos de 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. Las operaciones se denominan **verbs** dentro de los archivos YAML. Además de estos componentes básicos, agregaremos más elementos de la API, que luego se pueden administrar a través de RBAC.

Las reglas son operaciones que pueden actuar sobre un grupo de API. Los roles son un grupo de reglas que afectan, o **alcanzan** , un solo namespace, mientras que **ClusterRoles** tienen un alcance de todo el clúster.

Cada operación puede actuar sobre uno de los tres sujetos, que son **User Accounts** que no existen como objetos API, **Service Accounts** y **groups** que se conocen como **clusterrolebinding** cuando se usa kubectl.



Luego, RBAC está escribiendo reglas para permitir o denegar operaciones por parte de usuarios, roles o grupos sobre los recursos.

Si bien RBAC puede ser complejo, el flujo básico es crear un certificado para un usuario. Como un usuario no es un objeto de API de Kubernetes, requerimos autenticación externa, como certificados OpenSSL. Después de generar el certificado contra la autoridad de certificación del clúster, podemos configurar esa credencial para el usuario mediante un contexto.

Los roles se pueden usar para configurar una asociación de **apiGroups**, **resources** y los **verbs** que se les permiten. El usuario puede entonces estar vinculado a un rol que limite qué y dónde puede trabajar en el clúster.

A continuación, se muestra un resumen del proceso de RBAC:

- Determinar o crear un namespace
- Crear credenciales de certificado para el usuario
- Configure las credenciales del usuario en el namespace usando un contexto
- Cree un rol para el conjunto de tareas esperado
- Vincular al usuario al rol
- Verifique que el usuario tenga acceso limitado.

**Webhook:** Un Webhook es una devolución de llamada HTTP, una POST HTTP que se produce cuando sucede algo; una simple notificación de eventos a través de HTTP POST. Una aplicación web que implemente Webhooks PUBLICARÁ un mensaje a una URL cuando sucedan ciertas cosas.

Para obtener más información sobre el uso del modo Webhook, consulte la sección [Modo Webhook](#) de la documentación de Kubernetes.

## Controlador 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 agregaron controladores a medida que Kubernetes maduraba. A partir de la versión **1.13.1 de kube-apiserver**, los controladores de admisión ahora se compilan en el binario, en lugar de una lista pasada durante la ejecución. Para habilitar o deshabilitar, puede pasar las siguientes opciones, cambiando los complementos que desea habilitar o deshabilitar:

```
--enable-admission-plugins=Initializers,NamespaceLifecycle,Limit  
Ranger
```

```
--disable-admission-plugins=PodNodeSelector
```

El primer controlador es **Initializers** que permitirá la modificación dinámica de la solicitud de API, proporcionando 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.

## Contextos de seguridad

Los pods y los contenedores dentro de los pods pueden recibir restricciones de seguridad específicas para limitar lo que pueden hacer los procesos que se ejecutan en los contenedores. Por ejemplo, el UID del proceso, las capacidades de Linux y el grupo del sistema de archivos pueden ser limitados.

Esta limitación de seguridad se denomina contexto de seguridad. Se puede definir para todo el pod o por contenedor, y se representa como secciones adicionales en los manifiestos de recursos. La diferencia notable es que las capacidades de Linux se establecen a nivel de contenedor.

Por ejemplo, si desea aplicar una política de que los contenedores no pueden ejecutar su proceso como usuario raíz, puede agregar un contexto de seguridad de pod como el siguiente:

```
apiVersion: v1

kind: Pod

metadata:

  name: nginx

spec:

  securityContext:

    runAsNonRoot: true

  containers:

  - image: nginx

    name: nginx
```

Luego, cuando cree este Pod, verá una advertencia de que el contenedor está intentando ejecutarse como root y que no está permitido. Por lo tanto, el Pod nunca se ejecutará:

```
$ kubectl get pods
```

| NAME     | READY | STATUS   |
|----------|-------|--|
| RESTARTS | AGE   |  |
| nginx    | 0/1   | container has runAsNonRoot and image will run as |
| root     | 0     | 10s  |

Para obtener más información, lea la sección [Configurar un contexto de seguridad para un pod o contenedor](#) en la documentación de Kubernetes.

## Políticas de seguridad de pod

Para automatizar la aplicación de contextos de seguridad, puede definir [PodSecurityPolicies](#) (PSP). Un PSP se define a través de un manifiesto de Kubernetes estándar siguiendo el esquema de API de PSP. A continuación se presenta un ejemplo.

Estas políticas son reglas a nivel de clúster que gobiernan lo que puede hacer un pod, a qué puede acceder, con qué usuario se ejecuta, etc.

Por ejemplo, si no desea que ninguno de los contenedores de su clúster se ejecute como usuario root, puede definir un PSP a tal efecto. También puede evitar que los contenedores tengan privilegios o utilizar el namespace de la red del host o el namespace del PID del host.

Puede ver un ejemplo de una PSP a continuación:

```
apiVersion: policy/v1beta1

kind: PodSecurityPolicy

metadata:

  name: restricted

spec:

  seLinux:

    rule: RunAsAny

  supplementalGroups:

    rule: RunAsAny
```

```
runAsUser:  
  
  rule: MustRunAsNonRoot  
  
fsGroup:  
  
  rule: RunAsAny
```

Para que las políticas de seguridad de pod estén habilitadas, debe configurar el controlador de admisión del `controller-manager` para que contenga `PodSecurityPolicy`. Estas políticas tienen aún más sentido cuando se combinan con la configuración de RBAC en su clúster. Esto le permitirá ajustar con precisión lo que sus usuarios pueden ejecutar y qué capacidades y privilegios de bajo nivel tendrán sus contenedores.

Consulte el [ejemplo de PSP RBAC](#) en GitHub para obtener más detalles.

## Políticas de seguridad de la red

De forma predeterminada, todos los pods pueden comunicarse entre sí; Se permite todo el tráfico de entrada y salida. Este ha sido un requisito de red de alto nivel en Kubernetes. Sin embargo, se puede configurar el aislamiento de la red y se puede bloquear el tráfico a los pods. En las versiones más recientes de Kubernetes, el tráfico de salida también se puede bloquear. Esto se hace configurando una `NetworkPolicy`. Como todo el tráfico está permitido, es posible que desee implementar una política que elimine todo el tráfico y luego otras políticas que permitan el tráfico de entrada y salida deseado.

La **especificación** de la política puede reducir el efecto a un namespace en particular, lo que puede ser útil. Otras configuraciones incluyen un `podSelector`, o

etiqueta, para **delimitar** qué Pods se ven afectados. Más configuraciones de entrada y salida declaran el tráfico hacia y desde direcciones IP y puertos.

No todos los proveedores de la red admiten el tipo **NetworkPolicies** . Una lista no exhaustiva de proveedores con soporte incluye Calico, Romana, Cilium, Kube-router y WeaveNet.

En versiones anteriores de Kubernetes, existía el requisito de anotar un namespace como parte del aislamiento de la red, específicamente el **net.beta.kubernetes.io/network-policy= value** . Es posible que algunos complementos de red aún requieran esta configuración.

En la página siguiente, puede encontrar un ejemplo de una receta de **NetworkPolicy** . Se pueden encontrar más [recetas de políticas de red](#) en GitHub.

## Ejemplo de política de seguridad de red

El uso de políticas se ha estabilizado, como se observa con la **apiVersion v1** . El siguiente ejemplo reduce la política para afectar el namespace predeterminado.

Solo los pods con la etiqueta de **rol: db** se verán afectados por esta política, y la política tiene configuraciones de entrada y salida.

La configuración de **ingress** incluye una red **172.17** , con un rango más pequeño de direcciones IP **172.17.1.0** excluidas de este tráfico.

```
apiVersion: networking.k8s.io/v1
```

```
kind: NetworkPolicy
```

```
metadata:

  name: ingress-egress-policy

  namespace: default

spec:

  podSelector:

    matchLabels:

      role: db

  policyTypes:

    - Ingress

    - Egress

  ingress:

    - from:

        - ipBlock:

            cidr: 172.17.0.0/16

            except:

              - 172.17.1.0/24

        - namespaceSelector:

            matchLabels:

              project: myproject

    - podSelector:

        matchLabels:

          role: frontend

  ports:

    - protocol: TCP

      port: 6379
```

```
egress:

- to:

  - ipBlock:

    cidr: 10.0.0.0/24

  ports:

    - protocol: TCP

      port: 5978
```

Estas reglas cambian el namespace para que las siguientes configuraciones se etiqueten como `project: myproject`. Los pods afectados también deberían coincidir con la etiqueta `role: frontend`. Finalmente, se permitiría el tráfico TCP en el puerto 6379 desde estos pods.

Las reglas de salida tienen la configuración `to`, en este caso el tráfico TCP de rango `10.0.0.0/24` al puerto 5978.

El uso de reglas de entrada o salida vacías niega todo tipo de tráfico para los pods incluidos, aunque esto no se sugiere. Utilice otra `NetworkPolicy` dedicada en su lugar.



Tenga en cuenta que también puede haber declaraciones complejas

`matchExpressions` en la especificación, pero esto puede cambiar a medida que `NetworkPolicy` madura.

```
podSelector:

  matchExpressions:

    - {key: inns, operator: In, values: ["yes"]}
```



# Ejemplo de política predeterminada

Las llaves vacías coincidirán con todos los pods no seleccionados por otra **NetworkPolicy** y no permitirán el tráfico de entrada. El tráfico de salida no se verá afectado por esta política.

```
apiVersion: networking.k8s.io/v1
```

```
kind: NetworkPolicy
```

```
metadata:
```

```
  name: default-deny
```

```
spec:
```

```
  podSelector: {}
```

```
  policyTypes:
```

```
    - Ingress
```

Con el potencial de reglas complejas de entrada y salida, puede ser útil crear varios objetos que incluyan reglas de aislamiento simples y utilicen nombres y etiquetas fáciles de entender.

Algunos complementos de red, como WeaveNet, pueden requerir una anotación del namespace. A continuación se muestra la configuración de **DefaultDeny** para el namespace **myns** :

```
kind: Namespace
```

```
apiVersion: v1
```

```
metadata:
```

```
  name: myns
```

```
  annotations:
```

```
    net.beta.kubernetes.io/network-policy: |
```

```
{  
  
  "ingress": {  
  
    "isolation": "DefaultDeny"  
  
  }  
  
}
```

## Lab 15.1 - Working with TLS

## Lab 15.2 - Authentication and Authorization

## Lab 15.3 - Admission Controllers

### Question 15.1

¿Cuál de las siguientes opciones no forma parte del acceso a la API de Kubernetes?

- **A.** Authentication
- **B.** Keystone
- **C.** Authorization
- **D.** Admission Controllers

### Question 15.2

¿Qué agente acepta la opción `--authorization-mode` para cambiar la herramienta de autorización en uso?

- **A.** kubelet
- **B.** kubeadm
- **C.** kubectl

- **D.** kube-apiserver

## Question 15.3

Podemos especificar reglas de salida con políticas de red. ¿Verdadero o falso?

- **A.** Cierto
- **B.** Falso

## 16. HIGH AVAILABILITY

### Objetivos de aprendizaje

Al final de este capítulo, debería poder:

- Analice la alta disponibilidad en Kubernetes.
- Discutir sobre bases de datos colocadas y no colocadas.
- Conozca los pasos para lograr una alta disponibilidad en Kubernetes.

### Clúster de alta disponibilidad

Una característica más nueva de **kubeadm** es la capacidad integrada para unir múltiples nodos maestros con bases de datos etcd colocadas. Esto permite una mayor redundancia y tolerancia a fallas. Mientras los servicios de la base de datos, el clúster

continuará ejecutándose y se pondrá al día con la información de kubelet en caso de que el nodo principal se caiga y vuelva a estar en línea.

Se requieren tres instancias para que etcd pueda determinar el quórum si los datos son precisos, o si los datos están corruptos, la base de datos podría dejar de estar disponible. Una vez que etcd pueda determinar el quórum, elegirá un líder y volverá a funcionar como lo hacía antes del fracaso.

Se puede colocar la base de datos con planos de control o utilizar un clúster de base de datos externo etcd. El comando **kubeadm** facilita el uso de la implementación compartida.

Para garantizar que los workers y otros control planes sigan teniendo acceso, es una buena idea utilizar un equilibrador de carga. La configuración predeterminada aprovecha SSL, por lo que es posible que deba configurar el equilibrador de carga como un paso de TCP a menos que desee el trabajo adicional de la configuración del certificado. Como los certificados se decodificarán solo para nombres de nodos en particular, es una buena idea usar un FQDN en lugar de una dirección IP, aunque hay muchas formas posibles de manejar el acceso.

## Bases de datos colocadas

La forma más sencilla de obtener una mayor disponibilidad es utilizar el comando **kubeadm** y unir al menos dos servidores master más al clúster. El comando es casi lo mismo que una unión de worker, excepto un indicador **--control-plane** adicional y un **certificate-key**. Es probable que sea necesario generar la clave a menos que los otros nodos maestros se agreguen dentro de las dos horas posteriores a la inicialización del clúster.

Si un nodo falla, perdería tanto un control plane como una base de datos. Como la base de datos es el único objeto que no se puede reconstruir, esto puede no ser un problema importante.

# Bases de datos no colocadas

El uso de un clúster externo de etcd permite menos interrupciones si falla un nodo. Crear un clúster de esta manera requiere mucho más equipo para distribuir adecuadamente los servicios y requiere más trabajo para configurarlo.

El clúster etcd externo debe configurarse primero. El comando **kubeadm** tiene opciones para configurar este clúster, o hay otras opciones disponibles. Una vez que se está ejecutando el clúster etcd, los certificados deben copiarse manualmente en el primer nodo del control plane previsto.

El archivo **kubeadm-config.yaml** debe rellenarse con etcd configurado en externos, endpoints y ubicaciones de certificados. Una vez que el primer control plane está completamente inicializado, los control planes redundantes deben agregarse uno a la vez, cada uno completamente inicializado antes de agregar el siguiente.

## Lab 16.1 - High Availability Steps

## Lab 16.2 - Detailed Steps

