Capítulo I

Introducción

Con las imágenes de contenedor, limitamos el código de la aplicación, su tiempo de ejecución y todas sus dependencias en un formato predefinido. Y, con tiempos de ejecución de contenedores como runC, containerd o rkt, podemos usar esas imágenes preempaquetadas para crear uno o más contenedores.

Todos estos tiempos de ejecución son buenos para ejecutar contenedores en un solo host. Pero, en la práctica, nos gustaría tener una solución tolerante a fallos y escalable, que se puede lograr mediante la creación de un único controlador / unidad de gestión, después de conectar varios nodos. Este controlador / unidad de gestión generalmente se denomina orquestador de contenedores.

Definir el concepto de orquestación de contenedores.

Explica las razones para hacer la orquestación de contenedores.

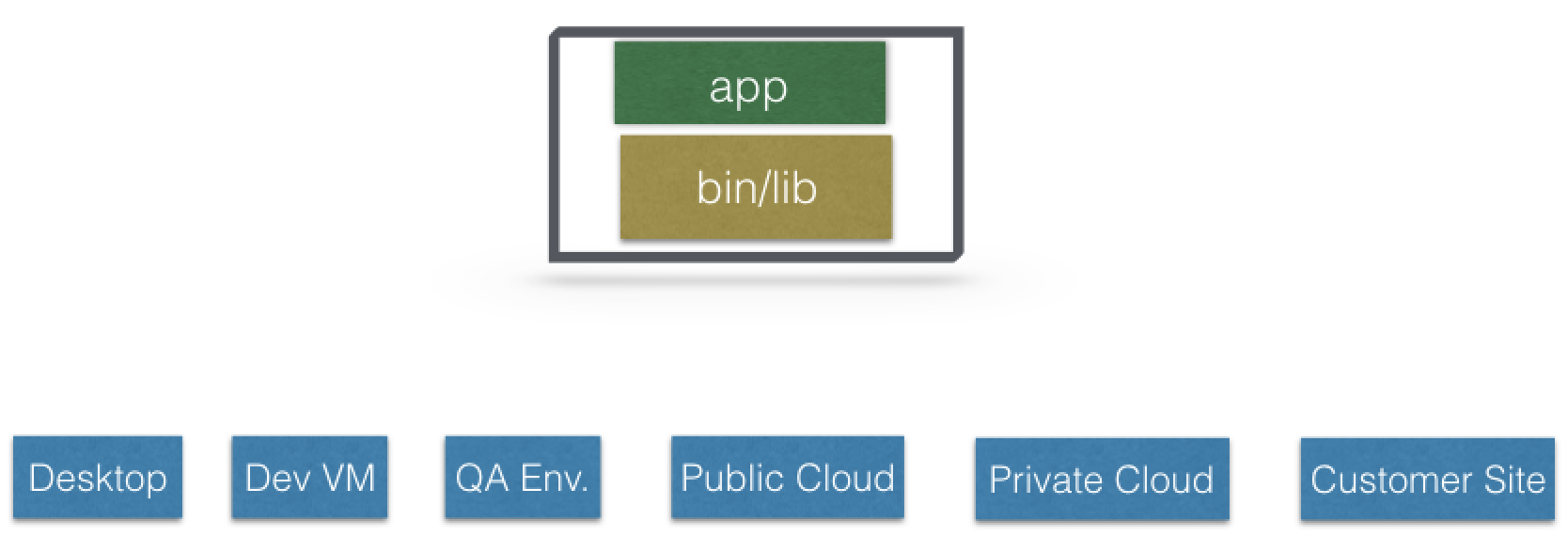
Discuta diferentes opciones de orquestación de contenedores.

Discuta diferentes opciones de implementación de orquestación de contenedores.

¿Qué son los contenedores?

Antes de profundizar en la orquestación de contenedores, revisemos primero lo que son contenedores.

Los contenedores son una forma centrada en aplicaciones para entregar aplicaciones escalables y de alto rendimiento en la infraestructura de su elección.



Contenedores

Con una imagen de contenedor, agrupamos la aplicación junto con su runtime y dependencias. Usamos esa imagen para crear un entorno ejecutable aislado, también conocido como contenedor. Podemos implementar contenedores desde una imagen determinada en la plataforma que elijamos, como escritorios, máquinas virtuales, nubes, etc.

¿Qué es Orquestación de contenedores?

En los entornos de aseguramiento de la calidad (QA), es suficiente con ejecutar los contenedores en un solo host para desarrollar y probar aplicaciones. Sin embargo, cuando vamos a producción, no tenemos la misma libertad, ya que necesitamos asegurarnos de que nuestras aplicaciones:

* Sean tolerantes a fallas
* Puedan ser escaladas, y hacer esto bajo demanda
* Usan los recursos de manera óptima
* Puedan descubrir otras aplicaciones automáticamente y comunicarse entre sí
* Sean accesibles desde el mundo externo
* Puedan ser actualizadas o realizar un rollback sobre ellas sin ningún tiempo de inactividad.

Los orquestadores de contenedores son las herramientas que agrupan hosts para formar un clúster y nos ayudan a cumplir los requisitos mencionados anteriormente.

Orquestadores de contenedores

Hoy en día, hay muchos orquestadores de contenedores disponibles, como:

Docker Swarm

[Docker Swarm](https://docs.docker.com/engine/swarm/) es un orquestador de contenedores proporcionado por [Docker, Inc](https://www.docker.com/). Es parte de [Docker Engine](https://docs.docker.com/engine/).

Kubernetes

[Kubernetes](https://kubernetes.io/) fue iniciado por Google, pero ahora es parte del proyecto [Cloud Native Computing Foundation](https://www.cncf.io/).

Mesos Marathon

[Marathon](https://github.com/mesosphere/marathon) es uno de los marcos para ejecutar contenedores a escala en [Apache Mesos](http://mesos.apache.org/).

Amazon ECS

El servicio de contenedor de Amazon EC2 (ECS) es un servicio hospedado proporcionado por AWS para ejecutar contenedores Docker a escala en su infraestructura.

Hashicorp Nomad

[Nomad](https://www.nomadproject.io/) es el orquestador de contenedores provisto por [HashiCorp](https://www.hashicorp.com/).

¿Por qué usar orquestadores de contenedores?

Aunque podemos argumentar que los contenedores a escala se pueden mantener manualmente, o con la ayuda de algunos scripts, los orquestadores de contenedores pueden facilitar las cosas a los operadores.

Los orquestadores de contenedores pueden:

* Reunir múltiples hosts y hacerlos parte de un clúster
* Programar contenedores para ejecutarse en diferentes hosts
* Ayuda a los contenedores que se ejecutan en un host alcanzar a los contenedores que se ejecutan en otros hosts del clúster
* Enlazar contenedores y almacenamiento
* Enlazar contenedores de tipo similar a un constructor de nivel superior, como servicios, para que no tengamos que ocuparnos de contenedores individuales
* Mantener el uso de recursos bajo control y optimizarlos cuando sea necesario
* Permitir el acceso seguro a las aplicaciones que se ejecutan dentro de los contenedores.

Con todos estos beneficios incorporados, tiene sentido usar orquestadores de contenedores para administrar contenedores.

¿Dónde se pueden desplegar los Orquestadores de Contenedores?

La mayoría de los orquestadores de contenedores se pueden implementar en la infraestructura que elijamos. Podemos implementarlos en bare metal, VM, on-premise o en una nube de nuestra elección. Por ejemplo, Kubernetes se puede implementar en nuestra laptop o workstation, dentro del centro de datos de una compañía, en AWS, en OpenStack, etc. Incluso hay instaladores de one-click disponibles para configurar Kubernetes en la nube, como Google Kubernetes Engine en Google Cloud, o Azure Container Service en Microsoft Azure. También hay soluciones similares disponibles para otros orquestadores de contenedores.

Hay empresas que ofrecen Container Orchestration as a Service. Los exploraremos para Kubernetes más adelante.

Capítulo II Kubernetes

Introducción

En este capítulo, explicaremos qué es Kubernetes, sus características y las razones por las que uno debería usarlo. Exploraremos la evolución de Kubernetes de Borg, que es un administrador de clúster creado por Google.

También hablaremos de la Cloud Native Computing Foundation (CNCF), que actualmente alberga el proyecto Kubernetes, junto con otros proyectos nativos de la nube, como Prometheus, Fluentd, rkt, containerd, etc.

Objetivos de aprendizaje

Al final de este capítulo, deberías ser capaz de:

* Definir Kubernetes.
* Explicar las razones para usar de Kubernetes.
* Comentar las características de Kubernetes.
* Comentar la evolución de Kubernetes desde Borg.
* Explicar lo que hace la Cloud Native Computing Foundation.

¿Qué es Kubernetes?

De acuerdo con el sitio web de Kubernetes,

"Kubernetes es un sistema de código abierto para automatizar la implementación, la ampliación y la administración de aplicaciones en contenedores".

Kubernetes proviene de la palabra griega κυβερνήτης, que significa timonel o piloto de barco. Con esta analogía en mente, podemos pensar en Kubernetes como el administrador del transporte de contenedores.

Kubernetes también se conoce como k8s, ya que hay 8 caracteres entre k y s.

Kubernetes está inspirado en el sistema Google Borg, que exploraremos en este capítulo. Es un proyecto de código abierto escrito en el idioma Go y con licencia bajo la versión 2.0 de Apache License.

Kubernetes fue iniciado por Google y, con su versión v1.0 liberada en julio de 2015, Google lo donó a la Cloud Native Computing Foundation (CNCF). Hablaremos más sobre CNCF más adelante en este capítulo.

Generalmente, Kubernetes tiene nuevos lanzamientos cada tres meses. La versión estable actual es 1.9 (a partir de febrero de 2018).

De Borg a Kubernetes

De acuerdo con el resumen del documento Borg de Google, publicado en 2015,

"El sistema Borg de Google es un administrador de clústeres que ejecuta cientos de miles de trabajos, desde miles de aplicaciones diferentes, a través de una serie de clusters, cada uno con hasta decenas de miles de máquinas".

Durante más de una década, Borg fue el secreto de Google para ejecutar cargas de trabajo en contenedores en producción. Independientemente de los servicios que utilicemos de Google, como Gmail, Drive, etc., todos reciben servicios con Borg.

Algunos de los autores iniciales de Kubernetes eran empleados de Google que usaron Borg y lo desarrollaron en el pasado. Ellos aportaron su valioso conocimiento y experiencia al diseñar Kubernetes. Algunas de las características u objetos de Kubernetes que se remontan a Borg, o las lecciones aprendidas de ella, son:

* Servidores API
* Vainas
* IP-per-Pod
* Servicios
* Etiquetas.

Exploraremos todos, y más, en este curso.

Características de Kubernetes I

Kubernetes ofrece un conjunto muy amplio de características para la orquestación de contenedores. Algunas de sus características totalmente compatibles son:

**Automatic binpacking**

Kubernetes programa automáticamente los contenedores según el uso y las restricciones de los recursos, sin sacrificar la disponibilidad.

**Self-healing**

Kubernetes reemplaza y reprograma automáticamente los contenedores de nodos fallidos. También mata y reinicia los contenedores que no responden a los controles de estado, de acuerdo con las reglas o políticas existentes.

**Escalamiento horizontal**

Kubernetes puede escalar automáticamente aplicaciones basadas en el uso de recursos como CPU y memoria. En algunos casos, también es compatible con la escala dinámica basada en las métricas de los clientes.

**Service discovery and Load balancing**

Kubernetes agrupa conjuntos de contenedores y se refiere a ellos a través de un Sistema de nombres de dominio (DNS). Este DNS también se llama *Kubernetes service*. Kubernetes puede descubrir estos servicios automáticamente y balancear las solicitudes entre los contenedores de un servicio determinado.

Características de Kubernetes I

Algunas otras características de Kubernetes totalmente compatibles son:

**Rollouts y rollbacks automatizados**

Kubernetes puede implementar y deshacer nuevas versiones o configuraciones de una aplicación, sin introducir ningún tiempo de inactividad.

**Secrets y gestión de la configuración**

Kubernetes puede administrar secrets y detalles de configuración para una aplicación sin necesidad de reconstruir las imágenes respectivas. Con secret, podemos compartir información confidencial con nuestra aplicación sin exponerla a la configuración del stack, como en GitHub.

**Orquestación de almacenamiento**

Con Kubernetes y sus complementos, podemos montar automáticamente soluciones locales, externas y de almacenamiento en los contenedores de forma transparente, en función del almacenamiento definido por software (software-defined storage SDS).

**Ejecución por lotes**

Además de los trabajos de larga ejecución, Kubernetes también admite la ejecución por lotes.

Hay muchas otras características además de las que acabamos de mencionar, y actualmente se encuentran en fase alfa / beta. Agregarán un gran valor a cualquier implementación de Kubernetes una vez que se conviertan en características estables. Por ejemplo, la compatibilidad con el control de acceso basado en roles (RBAC) es estable a partir de la versión 1.8 de Kubernetes.

¿Por qué usar Kubernetes?

Acabamos de ver algunas de las funciones de Kubernetes totalmente compatibles. También debemos mencionar que Kubernetes es portable y extensible. Kubernetes se puede implementar en el entorno de nuestra elección, ya sean VM, bare metal o configuraciones públicas, privadas, híbridas, multi-nube. Además, Kubernetes tiene una arquitectura modular y conectable. Podemos escribir APIs personalizadas o complementos para ampliar sus funcionalidades.

Para un proyecto de código abierto exitoso, la comunidad es tan importante como tener un gran código. Kubernetes tiene una comunidad muy próspera en todo el mundo. Tiene más de 1600 colaboradores, quienes, con el tiempo, han realizado más de 62,000 commits. Hay grupos de reunión en diferentes ciudades que se reúnen regularmente para hablar sobre Kubernetes y su ecosistema. Existen grupos de interés especial (Special Interest Groups SIG), que se enfocan en intereses especiales, tales como escalar, bare metal, redes, etc. Hablaremos más sobre ellos en nuestro último capítulo, Comunidades de Kubernetes.

Usuarios de Kubernetes

Con solo unos pocos años desde su debut, muchas compañías están ejecutando cargas de trabajo usando Kubernetes. Podemos encontrar numerosos estudios de casos de usuarios en el sitio web de Kubernetes:

* [Pearson](https://kubernetes.io/case-studies/pearson/)
* [Box](https://blog.box.com/blog/kubernetes-box-microservices-maximum-velocity/)
* [eBay](https://www.nextplatform.com/2015/11/12/inside-ebays-shift-to-kubernetes-and-containers-atop-openstack/)
* [Wikimedia](https://www.youtube.com/watch?v=6XGUTu3WhBw)
* [Huawei](https://kubernetes.io/case-studies/huawei/)
* [Haufe Group](https://kubernetes.io/case-studies/haufegroup/)
* [BlackRock](https://kubernetes.io/case-studies/blackrock/)
* [BlaBlaCar](https://kubernetes.io/case-studies/blablacar/)
* Y muchos más…

Cloud Native Computing Foundation (CNCF)

La [Cloud Native Computing Foundation](https://www.cncf.io/) (CNCF) es uno de los proyectos alojados por [The Linux Foundation](https://www.linuxfoundation.org/). CNCF tiene como objetivo acelerar la adopción de contenedores, microservicios y aplicaciones nativas de la nube.

**Logotipo de CNCF**



CNCF alberga un conjunto de proyectos, con más que se agregarán en el futuro. CNCF proporciona recursos para cada uno de los proyectos, pero, al mismo tiempo, cada proyecto continúa operando de manera independiente bajo su estructura de gobierno preexistente y con sus mantenedores existentes. En el momento en que se creó este curso, los siguientes proyectos formaban parte de CNCF:

* [containerd](http://containerd.io/) para container runtime
* [rkt](https://github.com/rkt/rkt) para container runtime
* [Kubernetes](https://kubernetes.io/) para container orchestration
* [Linkerd](https://linkerd.io/) para service mesh
* [Envoy](https://github.com/envoyproxy/envoy) para service mesh
* [gRPC](http://www.grpc.io/) para remote procedure call (RPC)
* [Container Network Interface](https://github.com/containernetworking/cni) (CNI) para networking API
* [CoreDNS](https://coredns.io/) para service discovery
* [Rook](https://github.com/rook/rook) para cloud-native storage
* [Notary](https://github.com/theupdateframework/notary) para security
* [The Update Framework](https://github.com/theupdateframework/specification) (TUF) para software updates
* [Prometheus](https://prometheus.io/) para monitoring
* [OpenTracing](http://opentracing.io/) para tracing
* [Jaeger](https://github.com/jaegertracing/jaeger) para distributed tracing
* [Fluentd](http://www.fluentd.org/) para logging
* [Vitess](http://vitess.io/) para storage.

Como podemos ver, este conjunto de proyectos de CNCF puede abarcar todo el ciclo de vida de una aplicación, desde su ejecución utilizando tiempos de ejecución del contenedor hasta su supervisión y registro. Esto es muy importante para cumplir con el objetivo de CNCF.

CNCF y Kubernetes

Para Kubernetes, la Cloud Native Computing Foundation:

* Proporciona un host neutral para la marca Kubernetes e impone el uso adecuado
* Proporciona escaneo de licencia de código principal y de código
* Ofrece orientación legal sobre patentes y derechos de autor
* Crea plan de [estudios de código abierto](https://github.com/cncf/curriculum), [capacitación](https://training.linuxfoundation.org/linux-courses/system-administration-training/kubernetes-fundamentals) y [certificación](https://www.cncf.io/certification/expert/)
* Gestiona un [grupo de trabajo](https://ponymail.cncf.io/thread.html/Zaw9xi4cg7fx9v6) de conformidad de software
* Activamente comercializa Kubernetes
* Organiza y financia actividades de marketing para desarrolladores como [K8Sport](http://k8sport.org/)
* Admite actividades ad hoc
* Conferencias de fondos y eventos de encuentro.

Capítulo III Arquitectura de Kubernetes

Introducción

En este capítulo, exploraremos la arquitectura de Kubernetes, los diferentes componentes de los nodos master y worker, el cluster state management con etcd y los requisitos de configuración de la red. También hablaremos sobre la especificación de red llamada Container Network Interface (CNI), que es utilizada por Kubernetes.

Objetivos de aprendizaje

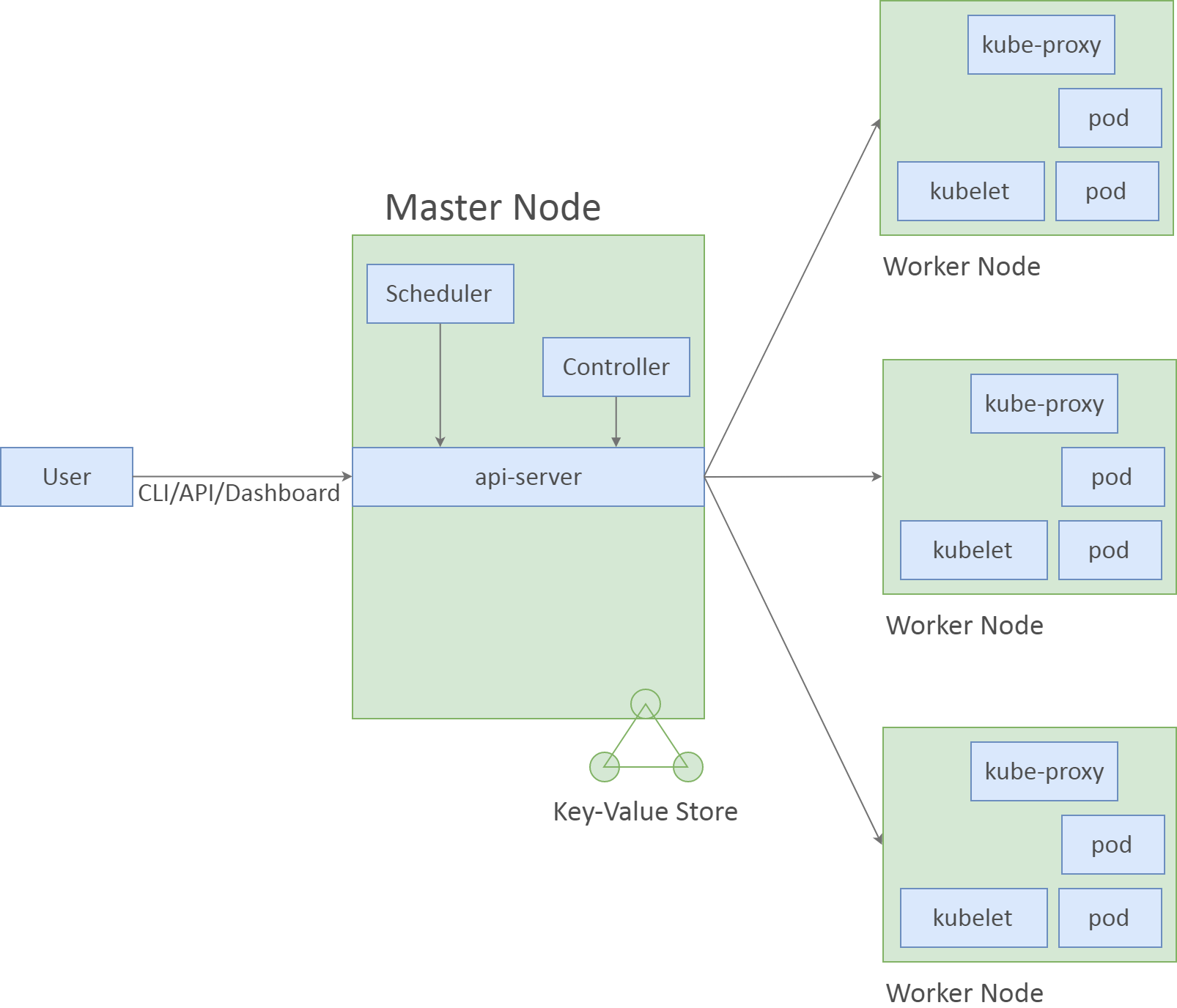
Al final de este capítulo, usted debería ser capaz de:

* Hablar de la arquitectura de Kubernetes.
* Explicar los diferentes componentes para los nodos master y worker.
* Hablar de cluster state management con etcd.
* Revisar los requisitos de configuración de red de Kubernetes.

Arquitectura de Kubernetes

En un nivel muy alto, Kubernetes tiene los siguientes componentes principales:

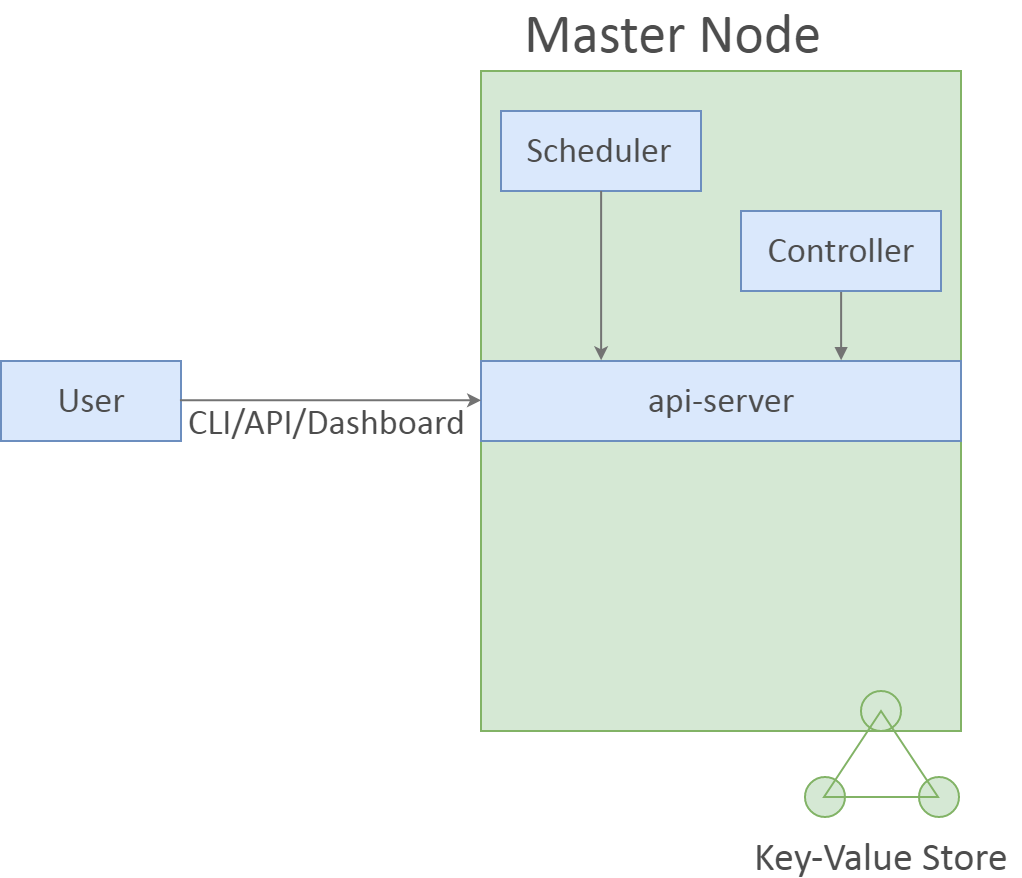
* Uno o más nodos maestros
* Uno o más nodos de trabajador
* Almacén de valores clave distribuidos, como etcd.



A continuación, exploraremos la arquitectura de Kubernetes con más detalle.

Nodo master

El nodo master es responsable de administrar el clúster de Kubernetes y es el punto de entrada para todas las tareas administrativas. Podemos comunicarnos con el nodo maestro a través de la CLI, la GUI (Dashboard) o mediante API.



Para propósitos de tolerancia a fallas, puede haber más de un nodo master en el clúster. Si tenemos más de un nodo master, estarían en modo HA (Alta disponibilidad), y solo uno de ellos será el leader, realizando todas las operaciones. El resto de los nodos maestros serían followers.

Para administrar el estado del clúster, Kubernetes usa [etcd](https://coreos.com/etcd/) y todos los nodos master se conectan a él. etcd es un store distribuido de key-values, que discutiremos en un momento. El store key-value puede ser parte del nodo maestro. También se puede configurar externamente, en cuyo caso, los nodos maestros se conectarían a él.

Componentes del nodo master

Un nodo master tiene los siguientes componentes:

* API server
* Scheduler
* Controller manager
* etcd.

En la siguiente sección, hablaremos de ellos a mayor detalle.

Componentes del nodo master: API Server

Todas las tareas administrativas se realizan a través del servidor API dentro del nodo master. Un usuario u operador envía comandos REST al servidor API, que luego valida y procesa las solicitudes. Después de ejecutar las solicitudes, el estado resultante del clúster se almacena en el store de key-value distribuidos.

Componentes del nodo master: Scheduler

Como su nombre indica, el Scheduler programa el trabajo para diferentes nodos workers. El planificador tiene la información de uso de recursos para cada nodo worker. También conoce las restricciones que los usuarios u operadores pueden haber establecido, como la programación del trabajo en un nodo que tiene el disco disk == ssd set. Antes de programar el trabajo, el planificador también tiene en cuenta la calidad de los requisitos del servicio, la ubicación de los datos, la afinidad, la antiafinidad, etc. El scheduler programa el trabajo en términos de Pods y Servicios.

Master Node Components: Controller Manager

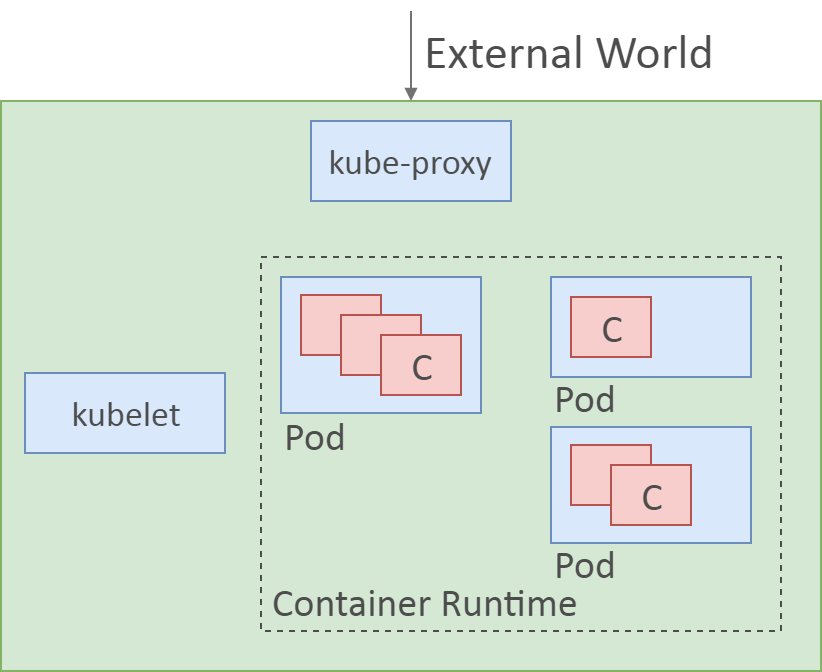
El controller manager gestiona diferentes control loops que no terminan, que regulan el estado del clúster de Kubernetes. Cada uno de estos control loops conoce el estado deseado de los objetos que administra y observa su estado actual a través del servidor API. En un bucle de control, si el estado actual de los objetos que maneja no cumple con el estado deseado, entonces el bucle de control toma medidas correctivas para asegurarse de que el estado actual sea el mismo que el estado deseado.

Master Node Components: etcd

Como se discutió anteriormente, etcd es un store de key-value distribuido que se usa para almacenar el estado del clúster. Puede ser parte del Kubernetes Master, o puede configurarse externamente, en cuyo caso, los nodos master se conectarían a él.

Nodo Worker

Un nodo worker es una máquina (VM, servidor físico, etc.) que ejecuta las aplicaciones utilizando Pods y está controlado por el nodo master. Los pods están programados en los nodos de trabajadores, que tienen las herramientas necesarias para ejecutarlos y conectarlos. Un Pod es la unidad de programación en Kubernetes. Es una colección lógica de uno o más contenedores que siempre se programan juntos. Los exploraremos más a fondo en capítulos posteriores.



Además, para acceder a las aplicaciones desde el mundo externo, nos conectamos a los nodos de trabajadores y no a los nodos maestros. Nos adentraremos más en esto en capítulos futuros.

Componentes del nodo worker

Un nodo worker tiene los siguientes componentes

* Container runtime
* kubelet
* kube-proxy.

En las siguientes secciones, hablaremos con más detalle de ellos.

Componentes del nodo worker: Container Runtime

Para ejecutar y administrar el ciclo de vida de un contenedor, necesitamos un runtime del contenedor en el nodo worker. Algunos ejemplos de tiempos de ejecución de contenedor son:

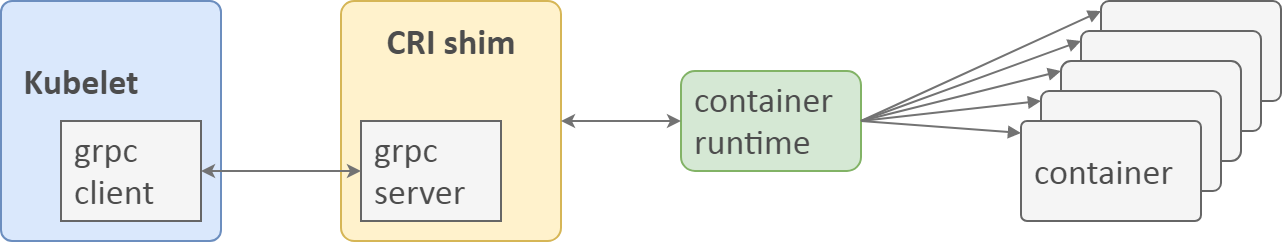
* [containerd](https://containerd.io/)
* [rkt](https://coreos.com/rkt/)
* [lxd](https://linuxcontainers.org/lxd/).

A veces, Docker también se conoce como runtime del contenedor, pero para ser precisos, Docker es una plataforma que usa containerd como runtime del contenedor.

Componentes del nodo worker: kubelet

El kubelet es un agente que se ejecuta en cada nodo de worker y se comunica con el nodo master. Recibe la definición de Pod por varios medios (principalmente, a través del API Server) y ejecuta los contenedores asociados con el Pod. También se asegura de que los contenedores que forman parte de los Pods estén sanos en todo momento.

El kubelet se conecta al runtime del contenedor utilizando Container Runtime Interface (CRI). La interfaz Container Runtime consiste en búferes de protocolo, gRPC API y bibliotecas.



Como se muestra arriba, el kubelet (cliente grpc) se conecta a la CRI shim (servidor grpc) para realizar operaciones de contenedores e imágenes. CRI implementa dos servicios: ImageService y RuntimeService. ImageService es responsable de todas las operaciones relacionadas con la imagen, mientras que RuntimeService es responsable de todas las operaciones relacionadas con el Pod y el contenedor.

Los runtimes de los contenedores solían estar codificados en Kubernetes, pero con el desarrollo de CRI, Kubernetes ahora puede usar diferentes runtimes de contenedores sin la necesidad de recompilarlos. Kubernetes puede usar cualquier runtimes de contenedor que implemente CRI para administrar Pods, contenedores e imágenes de contenedores.

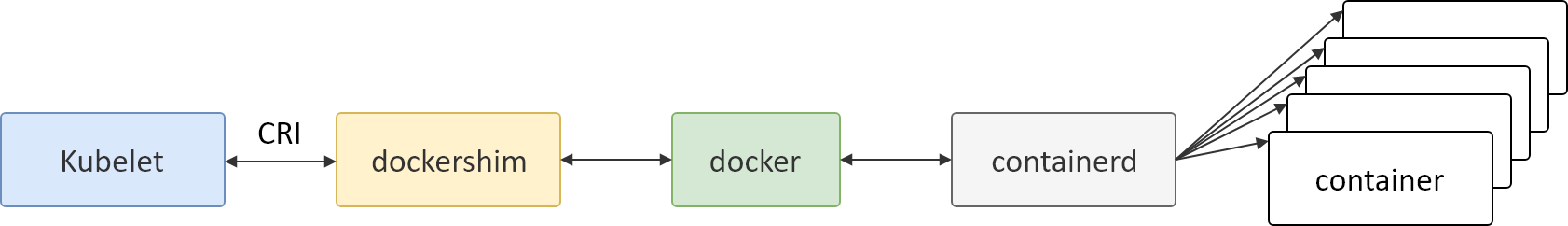
En la siguiente sección, hablaremos de algunos CRI shims.

Componentes del nodo worker: CRI shims

A continuación, encontrarás algunos ejemplos de CRI shims:

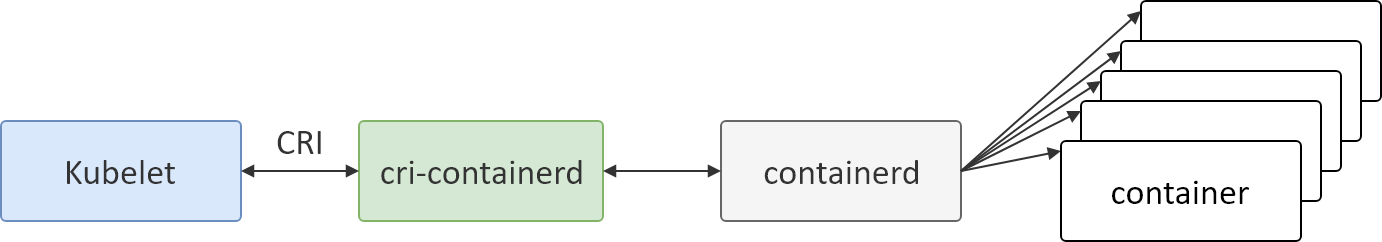
**dockershim**

Con dockershim, los contenedores se crean usando Docker instalado en los nodos worker. Internamente, Docker usa containerd para crear y administrar los contenedores.



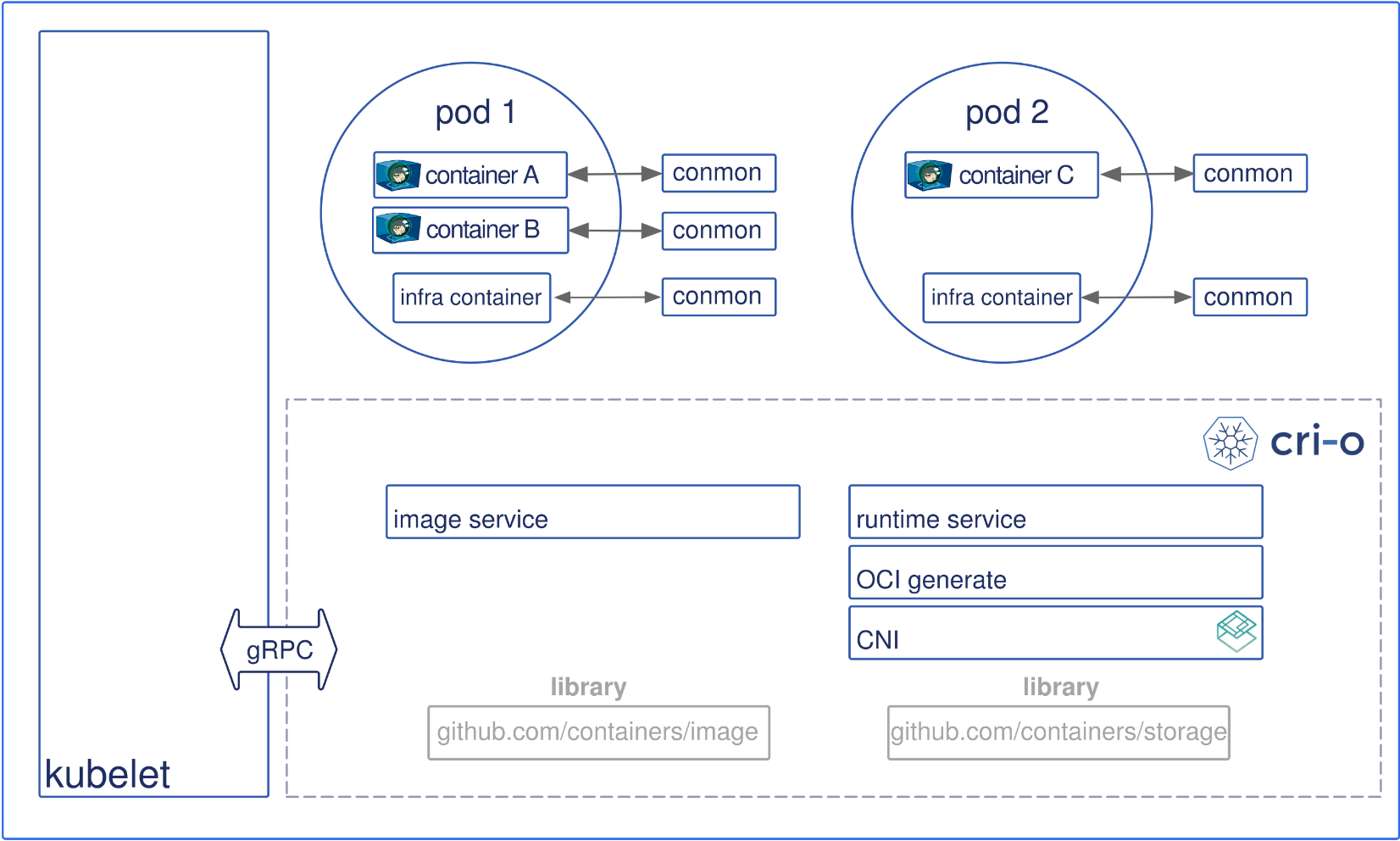
**cri-containerd**

Con cri-containerd, podemos usar directamente el contenedor descendiente más pequeño de Docker para crear y administrar contenedores.



**CRI-O**

CRI-O permite utilizar cualquier runtime compatible con Open Container Initiative (OCI) con Kubernetes. En el momento en que se creó este curso, CRI-O admitía RunC y Clear Containers como runtime del contenedor. Sin embargo, en principio, cualquier runtime compatible con OCI se puede conectar.



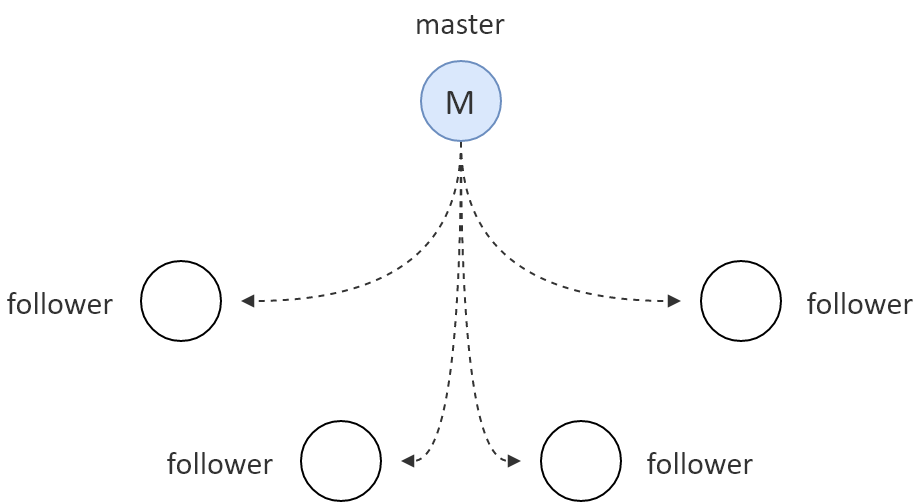
Componentes del nodo worker: kube-proxy

En lugar de conectarse directamente a Pods para acceder a las aplicaciones, usamos una construcción lógica llamada Service como punto final de conexión. Un Service agrupa los Pods relacionados y, cuando se accede a ellos, los balancea. Hablaremos más sobre Servicios en capítulos posteriores.

kube-proxy es el proxy de red que se ejecuta en cada nodo worker y escucha al API server para cada creación o eliminación de los service endpoints. Para cada Service endpoint, kube-proxy configura las rutas para que pueda alcanzarlo. También exploraremos esto con más detalle en capítulos posteriores.

State Management con etcd

Como mencionamos anteriormente, Kubernetes usa etcd para almacenar el estado del clúster. etcd es un almacén de key-value distribuido basado en el Raft Consensus Algorithm. Raft permite que una colección de máquinas funcione como un grupo coherente que puede sobrevivir a las fallas de algunos de sus miembros. En un momento dado, uno de los nodos en el grupo será el master, y el resto de ellos serán los followers. Cualquier nodo puede tratarse como un master.



etcd está escrito en el lenguaje de programación Go. En Kubernetes, además de almacenar el estado del clúster, etcd también se usa para almacenar detalles de configuración como subredes, ConfigMaps, Secrets, etc.

Desafíos de la configuración de red

Para tener un clúster de Kubernetes completamente funcional, debemos asegurarnos de lo siguiente:

* Se asigna una IP única a cada Pod
* Los contenedores en un Pod pueden comunicarse entre sí
* El Pod puede comunicarse con otros Pods en el clúster
* Si se configura, la aplicación desplegada dentro de un Pod es accesible desde el mundo externo.

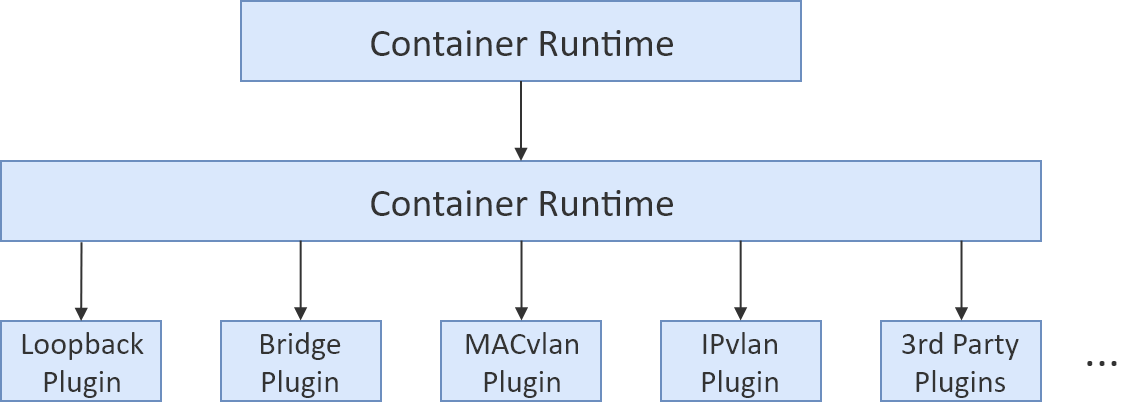
Todo lo anterior son desafíos de red que deben abordarse antes de implementar el clúster de Kubernetes. A continuación, veremos cómo podemos resolver estos desafíos.

Asignación de una dirección IP única a cada Pod

En Kubernetes, cada Pod obtiene una dirección de IP única. Para cada red de contenedores, hay dos especificaciones principales:

* **Container Network Model (CNM)**, propuesto por Docker
* **Container Network Interface (CNI)**, propuesto por CoreOS.

Kubernetes usa CNI para asignar la dirección IP a cada Pod.



El runtime del contenedor trapasa la asignación de IP a CNI, que se conecta al complemento configurado subyacente, como Bridge o MACvlan, para obtener la dirección IP. Una vez que la dirección IP viene dada por el plugin respectivo, CNI lo reenvía al runtime del contenedor solicitado.

Contenedor a contenedor comunicación dentro de un Pod

Con la ayuda del sistema operativo del host subyacente, todos los runtimes del contenedor generalmente crean una entidad de red aislada para cada contenedor que se inicia. En Linux, esa entidad se conoce como network namespace. Estos network namespaces se pueden compartir entre contenedores o con el sistema operativo del host.

Dentro de un Pod, los contenedores comparten los network namespace, de modo que puedan comunicarse entre sí a través de localhost.

Pod a pod comunicación a través de nodos

En un entorno de un cluster, los Pods pueden ser programados en cualquier nodo. Necesitamos asegurarnos de que los Pods puedan comunicarse a través de los nodos, y todos los nodos deberían poder alcanzar a cualquier Pod. Kubernetes también establece la condición de que no debe haber ninguna traducción de direcciones de red (NAT) mientras se realiza la comunicación de Pod a Pod en los hosts. Podemos lograr esto a través de:

Pods y nodos enrutables, usando la infraestructura física subyacente, como Google Kubernetes Engine

Uso de redes definidas por software, como [Flannel](https://coreos.com/flannel/docs/latest/), [Weave](https://www.weave.works/oss/net/" \t "_blank), [Calico](https://www.projectcalico.org/" \t "_blank), etc.

Para más detalles, puede echar un vistazo a la [documentación disponible de Kubernetes](https://kubernetes.io/docs/concepts/cluster-administration/networking/).

Comunicación entre el mundo exterior y los Pods

Exponiendo nuestros servicios al mundo exterior con kube-proxy, podemos acceder a nuestras aplicaciones desde fuera del cluster. Tendremos un capitulo completo dedicado a esto, entonces ahondaremos en esto después.