Lo que pasa es que administrar Kubernetes

Guía completa de administración de Kubernetes en lenguaje sencillo - Muestra gratuita (Capítulo 1)



Luis Felipe Ariza Vesga

Lo que pasa es que administrar Kubernetes

Lo que pasa es que administrar Kubernetes

Guía completa de administración de Kubernetes en lenguaje sencillo - Muestra gratuita (Capítulo 1)

Luis Felipe Ariza Vesga

Lo que pasa es que administrar Kubernetes
Guía completa de administración de Kubernetes en lenguaje sencillo - Muestra gratuita (*Capítulo 1*)

Copyright © Luis Felipe Ariza Vesga.
Todos los derechos reservados.

La información de este libro se distribuye "tal cual", sin garantía. Aunque se han tomado todas las precauciones en su preparación, ni el autor ni la editorial asumen responsabilidad alguna por pérdidas o daños derivados directa o indirectamente de su contenido.

ISBN: 9798178786840

Para el rinoceronte negro del oeste (Diceros bicornis longipes), que vivió en las sabanas de Camerún, Chad, la República Centroafricana y el noreste de Nigeria, y que sostuvo esa sabana entera con solo pastar y caminar por ella. La caza furtiva de sus cuernos y los agricultores que lo mataban para proteger sus cultivos borraron el 96 por ciento de su población entre 1970 y 1992; el último rastro confirmado se registró en 2006, en la provincia Norte de Camerún, y la UICN lo declaró extinto en 2011. Y para todo el que administra un clúster: el trabajo más importante es casi siempre el que nadie nota hasta que falta.

Índice general

Convenciones de este libro **1**
Tipografía y referencias 2
Las cuatro cajas de aviso 2
Créditos y licencias de terceros 4

Prefacio **5**
De qué se trata este libro 5
De qué no se trata este libro 6
Cómo está organizado 7
Sobre los comandos y ejemplos de código 8
Cómo contactarnos 10
Agradecimientos 10

I Arquitectura del clúster, instalación y configuración (25%) **13**

1 Los cimientos de la casa: quién hace qué en el clúster **15**
1.1 El reparto de oficios antes de poner un solo ladrillo 15
1.2 La ventanilla única: por qué todo pasa por el kube-apiserver . 17
1.3 El cerebro de la obra: el cuaderno, el maestro y los capataces . 20
1.4 Los oficios de cada frente de obra: el obrero de planta, el cartero y el contratista 23
1.5 Dos casas en vez de una: el sobrecosto que evita la ruina 25
1.6 Una obra con más de una oficina central: pods estáticos y alta disponibilidad 26
1.7 Practique en el laboratorio 28
1.8 Resumen 28

Bibliografía 29

Convenciones de este libro

> *El pequeño sistema visual de este libro, demostrado en vivo, no solo descrito.*

Antes de arrancar, media página sobre cómo está escrito este libro. Se lee en dos minutos y le ahorra confusiones después.

Este libro le habla de **usted**. No es formalidad ni distancia: es la manera respetuosa de dirigirse a alguien con quien se trabaja hombro a hombro.

Cada capítulo abre con una escena cotidiana. Esa escena no es adorno ni relleno para hacer bulto: **es** el concepto técnico contado de otra manera, y va a volver al final del capítulo, en el resumen, ya explicada con las palabras correctas. Si en algún momento no entiende una explicación técnica, devuélvase a la escena del comienzo: casi siempre la respuesta está ahí en versión sencilla. Y esas escenas no andan sueltas: forman una sola historia, la de la finca cuya casa se construye en el capítulo 1 y que se sigue cuidando hasta el capítulo 14, así que cada capítulo recoge lo que dejaron los anteriores.

Este libro no busca hacerlo reír a propósito. Lo que hace es traducir el lenguaje técnico a lenguaje cotidiano, con historias de la vida real que todos hemos vivido, familiares y fáciles de reconocer, usadas como analogía para entender rápido un concepto de Kubernetes que tradicionalmente se explica en lenguaje directamente técnico, lo cual dificulta el estudio. Eso incluye datos como que `kubectl cordon` deja un nodo marcado para no recibir nada nuevo mientras los Pods que ya estaban ahí siguen corriendo tan campantes, o que un `NetworkPolicy` mal escrito puede dejar mudo un namespace entero sin que salte ni una alarma: si eso le saca una sonrisa, bienvenida sea, pero la meta nunca es el chiste, es que usted entienda. Por eso la analogía nunca reemplaza el rigor: después de ella viene siempre la definición oficial, citada tal cual, con su enlace a la fuente.

Tipografía y referencias

Tipografía monoespaciada se usa para todo lo que usted teclearía o leería en una pantalla: nombres de objetos como `Pod` y `Deployment`, campos de YAML como `restartPolicy`, comandos como `kubectl describe pod`, y rutas como `/etc/kubernetes/manifests`.

Negrita marca la primera aparición de un término importante, y también las palabras que suelen aparecer tal cual en el enunciado de las tareas del examen. Cuando usted vea una palabra en negrita en medio de una definición, es porque esa palabra precisa importa.

Los **paréntesis angulares** marcan lo que usted debe reemplazar: en `kubectl logs <nombre-del-pod>`, usted escribe el nombre real y no escribe los paréntesis.

Las **citas de la documentación oficial** aparecen integradas en el texto, no como bloques aparte, y siempre llevan un enlace a la fuente la primera vez que se usan en el capítulo. Eso es intencional: usted debe poder verificar cualquier afirmación de este libro sin buscarla, haciendo clic.

Cada capítulo cierra con una **bibliografía** de al menos tres fuentes, todas oficiales y todas con enlace.

Las **figuras** se referencian en el texto y llevan pie explicativo. Si una figura aparece cerca de una explicación, es porque esa explicación se entiende mejor mirándola.

Las cuatro cajas de aviso

A lo largo del libro va a encontrar cuatro tipos de caja. Cada una significa algo distinto y aporta algo que la prosa no dijo: un comando para verificar, un atajo para la terminal del examen, un comportamiento observado en un clúster real o un riesgo de producción. Ninguna le advierte sobre preguntas capciosas, porque el CKA no las tiene: es un examen práctico que le asigna tareas y califica el estado final del clúster. Lo que sí hacen las cajas es ayudarle a dejar ese estado bien hecho y comprobado. Conviene reconocerlas de un vistazo, porque no todas tienen la misma urgencia.

Nota
Información complementaria que amplía lo que acaba de leer, casi siempre con la forma de comprobarlo usted mismo. Se puede saltar sin perder el hilo. Ejemplo: los componentes del plano de control que instala kubeadm son Pods estáticos, definidos como archivos en `/etc/kubernetes/manifests` y gestionados directamente por el kubelet del nodo, sin que el kube-apiserver los vigile ni el scheduler los asigne.

Consejo
Un atajo para la terminal del examen, una regla mnemotécnica o una forma más rápida de hacer algo. Ejemplo: `kubectl create deployment cuchara-de-oro --image=nginx --replicas=2 --dry-run=client -o yaml` le escribe el esqueleto del manifiesto en un segundo, sin crear nada en el clúster; usted lo redirige a un archivo y edita solo lo que la tarea pide.

Advertencia
Un error frecuente o un comportamiento que sorprende en un clúster real, de esos que dejan una tarea a medio resolver. Si usted solo va a leer un tipo de caja antes de presentar el examen, que sean estas. Ejemplo: `kubectl cordon` marca el nodo como no planificable, pero no saca los Pods que ya estaban corriendo en él; para vaciarlo hace falta `kubectl drain`.

Precaución
Algo que puede causar daño real en un clúster de producción: pérdida de datos, caída de servicio o un hueco de seguridad. Son pocas y van en serio. Ejemplo: un NetworkPolicy con `podSelector: {}`, `policyTypes: [Ingress]` y ninguna regla de entrada deja sin tráfico entrante a todos los Pods de su namespace, siempre que el plugin de red del clúster aplique NetworkPolicy.

Créditos y licencias de terceros

Este libro reproduce y traduce fragmentos de la documentación oficial de Kubernetes (https://kubernetes.io/docs/), publicada por The Kubernetes Authors bajo licencia Creative Commons Attribution 4.0 International (CC BY 4.0). El texto de la licencia está disponible en https://creativecommons.org/licenses/by/4.0/. Los cambios respecto del original son de dos clases: la traducción al español y el relato cotidiano con que cada concepto se acompaña; la definición oficial aparece siempre aparte, sin alterar, con su enlace a la fuente.

Las definiciones citadas textualmente aparecen entre comillas y enlazadas a la página de la que provienen. Las traducciones al español son propias de este libro y no han sido revisadas ni avaladas por The Kubernetes Authors. Los enlaces están ahí para que usted pueda comprobar cada afirmación en su origen, y si una página oficial cambia después de la publicación del libro, la fuente vigente es la que conviene consultar.

Kubernetes es una marca registrada de The Linux Foundation. Los nombres de las certificaciones citadas pertenecen a la Linux Foundation y a la Cloud Native Computing Foundation. Este libro es material de estudio independiente y no está afiliado a esas organizaciones, ni cuenta con su respaldo o patrocinio. Los nombres de los demás proyectos y productos que aparecen a lo largo del libro, como kind, etcd o CoreDNS, pertenecen a sus respectivos titulares y se mencionan solo para identificar el software con el que se practica.

Prefacio

Cómo está organizado este libro, con qué voz le explica el examen CKA, y qué esperar de las cinco partes que vienen.

Este libro nace de la misma pregunta que le dio origen a toda la serie "Lo que pasa es que": ¿y si el examen de turno se explicara con la misma voz con la que se cuenta la vida en una finca o en un barrio cualquiera? Al examen de certificación de administración de Kubernetes, el CKA (Certified Kubernetes Administrator), lo pintan como un examen árido: kube-apiserver, etcd, kubelet, RBAC (Role-Based Access Control), taints y tolerations, palabras que suenan a sala de servidores y a nada más. Pero lo que pasa es que, debajo de ese vocabulario, administrar un clúster de Kubernetes se parece muchísimo a administrar una finca grande: si queremos construir una casa nueva, hay que levantar sus cimientos, repartir las llaves, contratar los oficios, organizar los turnos del restaurante La Cuchara de Oro, vigilar quién entra por la puerta de la calle y, sobre todo, saber qué hacer cuando algo se daña y nadie sabe por qué. Ese es el hilo que atraviesa los catorce capítulos: detrás de cada objeto de la API (Application Programming Interface) de Kubernetes hay una tarea doméstica que usted ya entiende de memoria.

De qué se trata este libro

Este es un libro de preparación para el examen de certificación CKA, Certified Kubernetes Administrator, de la Cloud Native Computing Foundation (CNCF) y la Linux Foundation. Sigue la versión del temario v1.35, organizado en cinco partes que siguen los cinco dominios oficiales vigentes del examen: Arquitectura del clúster, instalación y configuración (25%), Cargas de trabajo y planificación (15%), Servicios y redes (20%), Almacenamiento (10%) y Diagnóstico (30%). Son catorce capítulos, cada uno abierto con una

escena cotidiana que se convierte en la explicación técnica precisa de un mecanismo real de Kubernetes, y cerrado con un resumen que recoge esa misma escena. Cada una de las cinco partes tiene su propio laboratorio, ejecutado sobre un clúster kind real con kubectl y kubeadm, no un ejercicio de relleno. El objetivo no es que usted memorice cinco porcentajes, sino que termine el libro entendiendo por qué el examen les da ese peso: el Diagnóstico pesa el 30%, más que cualquier otro dominio, porque un administrador que sabe instalar un clúster pero no sabe por qué un Pod quedó en CrashLoopBackOff no sirve el día que algo se daña en serio.

Un aviso sobre la versión, porque aquí es fácil confundirse. ¿Por qué v1.35, si Kubernetes ya va en la 1.37? Porque este libro sigue el temario oficial del examen, y no la última versión de Kubernetes. Al escribir esta edición, el temario publicado por la CNCF es el v1.35, y la Linux Foundation indica que el examen corre sobre Kubernetes v1.35. Kubernetes 1.37 salió el 26 de agosto de 2026, y la Linux Foundation alinea el examen con la versión menor más reciente entre cuatro y ocho semanas después de cada lanzamiento. Los laboratorios fijan Kubernetes v1.36, que está dentro de las versiones con soporte. Antes de agendar su examen, revise la página oficial del CKA y el repositorio github.com/cncf/curriculum: si el temario ya cambió de versión, esa página manda sobre este libro.

De qué no se trata este libro

Este libro no es un manual de referencia exhaustivo de cada campo, cada bandera ni cada objeto de la API de Kubernetes; para eso está la documentación oficial en kubernetes.io/docs, que se cita en cada capítulo. Tampoco es un curso de Linux desde cero, ni de contenedores, ni de Docker: se asume que usted ya sabe leer un archivo YAML, escribir un Dockerfile, construir imágenes, moverse en una terminal y reconocer qué es un contenedor sin que se lo expliquen aquí. Ninguna distribución específica de Kubernetes ni ningún proveedor de nube en particular recibe un capítulo propio: el libro se queda en lo que kubeadm instala y en lo que el examen CKA evalúa, sobre un clúster genérico, no en cómo configurar el servicio administrado de tal o cual proveedor. Y definitivamente no reproduce preguntas de exámenes reales ni de simuladores como killer.sh: eso violaría tanto la ética como el acuerdo

de confidencialidad del examen, y además sería mal negocio, porque memorizar una tarea no le sirve el día que llega la misma con otro nombre de namespace.

Si usted apenas está comenzando con Kubernetes, empiece por el libro anterior de esta serie: **Lo que pasa es que Kubernetes**, *Guía completa de Kubernetes en lenguaje sencillo*, ya publicado en Amazon. Ese libro explica desde cero qué es un contenedor, qué es un Pod, cómo está armado un clúster y por qué el sistema se comporta como se comporta, con situaciones de la vida real y analogías cotidianas, y cubre el temario del examen KCNA (Kubernetes and Cloud Native Associate). Este libro arranca donde ese termina: da por entendidos esos conceptos y se dedica a administrarlos. Leídos en ese orden, los dos se entienden mucho más fácil.

Cómo está organizado

El libro sigue el orden de los cinco dominios oficiales del examen, agrupados en cinco partes con su peso declarado, y cuenta una sola historia de principio a fin: la de una finca donde usted construye, reparte, contrata, atiende y cuida. En la primera parte (arquitectura del clúster) se levantan la casa principal y su casa de control en el capítulo 1, se reparten las llaves con RBAC en el capítulo 2 y se contrata la cuadrilla que instala, actualiza y respalda todo con kubeadm, Helm, Kustomize, CRDs (Custom Resource Definitions) y operadores en el capítulo 3. En la segunda parte (cargas de trabajo y planificación) abre sus puertas el restaurante La Cuchara de Oro: en el capítulo 4 se organizan sus turnos con Deployments, rollouts y autoescalado, y en el capítulo 5 cada empleado recibe su receta, su ingrediente secreto y su lugar de trabajo con ConfigMaps, Secrets y las reglas del scheduler. La tercera parte (servicios y redes) sale al barrio: el chisme que permite encontrar a cada vecino aunque las personas se muden muy a menudo (capítulo 6), Justo Segura, el celador que decide qué tráfico pasa (capítulo 7), y la puerta de la calle por donde entran los visitantes con Ingress y la Gateway API (capítulo 8). La cuarta parte (almacenamiento) baja a la bodega: el depósito compartido con sus volúmenes y modos de acceso en el capítulo 9, y el espacio que se reserva y crece a pedido con storage classes en el capítulo 10. La quinta y última parte, la más pesada del examen con un 30%, se dedica al diagnóstico

con cuatro figuras conocidas: el doctor de cabecera y su método (capítulo 11), el mecánico que revisa el motor del control plane (capítulo 12), el medidor de gasolina de los recursos (capítulo 13) y el teléfono cuando la llamada no entra porque la red falla (capítulo 14).

Para que la historia no se enrede, la analogía base es fija en todo el libro:

Kubernetes	En la finca
Clúster	La finca completa
Namespace	Una casa de la finca: la principal, y luego las casas de verano de los hijos y los nietos
Control plane	La casa de control, la oficina central de la obra
Nodo de trabajo	Un frente de obra, una cuadrilla que puede trabajar en cualquier casa
Pod	Quien vive y trabaja adentro: habitantes, trabajadores, cocineros

Dentro de cada parte, los capítulos avanzan de lo más fundamental a lo más específico, y cada uno arranca recordando lo que construyeron los anteriores. El apéndice de glosario reúne los términos clave del libro, con esta misma analogía, para repasar antes del examen, y el laboratorio de cada parte asume que usted ya leyó los capítulos de esa parte, no los de partes futuras.

Sobre los comandos y ejemplos de código

Todos los ejemplos de YAML y de comandos kubectl de este libro son reales y verificados contra la documentación oficial de kubernetes.io/docs, con los campos que la API de Kubernetes espera y con los nombres de recurso que define el propio esquema OpenAPI del clúster, no pseudocódigo inventado para que se vea bonito. Cuando un ejemplo usa kubectl, el libro le indica el verbo preciso (get, describe, apply, rollout) y el recurso sobre el que actúa,

porque confundir `kubectl delete pod` con `kubectl delete deployment` es un error común de examen que borra cosas distintas. Cuando un comportamiento depende de la versión instalada, el libro le enseña a consultarla con `kubectl version` y en las notas de lanzamiento de kubernetes.io, en lugar de fijar un número que quede desactualizado el día que usted lea esto. Solo los laboratorios fijan una versión concreta del clúster, por digest de imagen de kind, porque un laboratorio sí necesita ser reproducible de principio a fin, sin sorpresas de una versión nueva que cambió un campo por debajo.

Todo el código de los laboratorios está publicado en https://github.com/lfarizav/lo-que-pasa-es-que-administrar-kubernetes. El repositorio tiene dos carpetas: `es/` para la edición en español y `en/` para la edición en inglés. Dentro de cada una hay un laboratorio por parte, de `lab-part-01` a `lab-part-05`, con sus comandos, sus comprobaciones y las versiones con las que se probó, listo para clonar y ejecutar en una terminal de Linux:

```
git clone https://github.com/lfarizav/lo-que-pasa-\
es-que-administrar-kubernetes.git
cd lo-que-pasa-es-que-administrar-kubernetes/es
```

Cada comando de esos laboratorios se ejecutó en un clúster antes de publicarse, y cada laboratorio levanta su propio clúster kind aislado sin tocar su kubeconfig de siempre, así que nunca lo corra contra un clúster de producción. Se parecen a lo que le van a pedir, porque el CKA no es de opción múltiple: según la página oficial de la certificación, es una prueba práctica de dos horas en la línea de comandos que asigna tareas y califica el estado que usted deja en el clúster. Aun así no son un simulador ni reproducen preguntas del examen; sirven para que cada afirmación del libro se convierta en algo que usted vio pasar en su propia terminal.

Cada capítulo cierra, justo antes de su resumen, con una sección **Practique en el laboratorio** que le dice qué laboratorio le corresponde y qué temas de Kubernetes va a trabajar allí; los pasos, los comandos y las comprobaciones están solo en el repositorio. Si un comando del libro no le funciona, revise primero ese repositorio: cuando la documentación oficial cambia, la corrección llega ahí antes que a la siguiente edición impresa.

Cómo contactarnos

Este libro es parte del catálogo de Luis Felipe Ariza Vesga. Si encuentra un ejemplo que no aplica como el libro dice, un campo que cambió de nombre en una versión reciente de Kubernetes, un comportamiento del scheduler que se actualizó después de escribir este capítulo, o una afirmación que no coincide con la documentación oficial vigente, esa retroalimentación es bienvenida y se toma en serio: un libro técnico que no se corrige deja de ser confiable. Kubernetes es un proyecto que evoluciona rápido, con nuevas versiones cada pocos meses y objetos de la API que se gradúan de beta a estable sin previo aviso, así que un libro sobre el tema tiene que aceptar que ninguna edición es la última palabra, y que la corrección hecha a tiempo vale más que la promesa de que nunca hará falta. Este libro prefiere quedar corregido a quedar bonito, igual que un administrador de clúster prefiere una alerta molesta de la liveness probe a un silencio cómodo que esconde un Pod que ya se cayó.

Agradecimientos

Este libro no inventó nada desde cero, y eso es justamente lo que quiere agradecer. Kubernetes existe porque miles de personas decidieron no reinventar la rueda cada una por su lado, sino construir una sola rueda entre todas, a la vista de todos.

A la Cloud Native Computing Foundation (CNCF) y a la Linux Foundation, por darle una casa neutral a Kubernetes y a los proyectos que lo rodean, por publicar abierto el temario del examen CKA, y por sostener la certificación que este libro prepara.

A las comunidades y a las empresas que donaron proyectos completos a la CNCF. Entregar un proyecto entero a una fundación, con su código, su nombre y su gobierno, en lugar de guardarlo como ventaja propia, es lo que permite que un administrador en cualquier parte del mundo use las mismas herramientas sin pedirle permiso a nadie. Cada capítulo de este libro se apoya en una de esas donaciones.

A los mantenedores de Kubernetes, kubeadm, etcd, CoreDNS, Helm, Kustomize, la Gateway API y los plugins de red y almacenamiento que aparecen

en estas páginas. Revisar código ajeno, responder issues a medianoche y decir que no a una función mal pensada es un trabajo que casi nadie ve y del que todo el mundo depende.

A todos los contribuidores: a quienes escriben código, pero también a quienes escriben y traducen la documentación de kubernetes.io, reportan errores, prueban versiones beta y responden preguntas en los foros. La documentación oficial que este libro cita en cada capítulo existe gracias a ellos, abierta y gratis para cualquiera.

A las comunidades locales de código abierto: a los miembros de Cloud Native Colombia, de KCD Colombia (Kubernetes Community Days) y del Colombia OpenInfra User Group. Organizan meetups, talleres y eventos para fomentar el uso del código abierto sin cobrar por ello, y muchas veces ponen de su propio bolsillo el refrigerio, el transporte o el alquiler del lugar para que cada encuentro se haga realidad. Buena parte de lo que aquí se cuenta se aprendió en esas reuniones.

Y gracias, por último, al rinoceronte negro del oeste, cuya historia le presta a la portada su lección más dura: que el trabajo que sostiene todo un sistema rara vez se nota mientras existe, y que cuando ese trabajo callado desaparece, lo que se cae no es una sola pieza sino la sabana entera.

PART I

Arquitectura del clúster, instalación y configuración (25%)

1

Los cimientos de la casa: quién hace qué en el clúster

Por qué un clúster de Kubernetes se parece a levantar una casa desde los cimientos, y quién hace cada oficio en esa obra.

1.1 El reparto de oficios antes de poner un solo ladrillo

Imagínese que usted decide, por fin, construir la casa de la finca que lleva años prometiéndose. Si leyó el libro anterior de esta serie, *Lo que pasa es que Kubernetes*, ya conoce el terreno desde la cerca; ahora le toca ponerse el casco y administrar la obra por dentro. No empieza por el techo ni por escoger el color de la fachada: empieza por los cimientos. Si los cimientos quedan mal, lo demás se viene abajo tarde o temprano, y usted lo sabe porque vio caerse la casa del vecino, con los ahorros de media vida adentro.

Una obra seria reparte los oficios, y cada quien hace lo suyo sin meterse en lo ajeno. Hay un maestro de obra, que decide en qué frente trabaja cada cuadrilla. Hay capataces, que recorren el terreno revisando que todo siga como manda el plano. Hay un cuaderno donde se anota cada decisión, para que nadie la olvide ni la invente después. Y en cada frente hay un obrero de planta que recibe las órdenes y llama a un contratista de confianza: ese contratista

es el que pone ladrillo sobre ladrillo, levanta paredes y columnas, y tiende las tuberías de agua, luz y gas.

En la repisa de la oficina central, encima de los planos enrollados, el maestro tiene una figura de madera ya despintada: un animal grande y macizo, de dos cuernos, uno largo adelante y uno corto atrás. Nadie sabe de dónde salió y nadie le pregunta. Lo único que dice él, de pasada y sin voltearse, cuando alguien anda de afán por ver paredes levantadas, es esto: que el trabajo callado es el que sostiene la obra, y que a ese trabajo nadie lo nota hasta que falta.

Lo que pasa es que un clúster de Kubernetes se organiza con ese mismo reparto de oficios. La finca entera es el clúster. Cada casa de la finca es un namespace: hoy levantamos la casa principal, y con los años llegarán las casas de verano de los hijos y de los nietos. Quienes viven y trabajan dentro de esas casas son los Pods, con sus contenedores. Y hay una diferencia que conviene ver desde la primera página: antes de la casa principal, Kubernetes le pide levantar una segunda construcción, la casa de control, que funciona como oficina central de la obra.

Esa oficina central es el control plane, y allí trabajan cuatro oficios que nunca se confunden entre sí:

- el kube-apiserver, la única ventanilla por donde entra cualquier instrucción;
- etcd, el cuaderno de obra donde queda anotado el estado de todo;
- el kube-scheduler, el maestro de obra, que decide en qué frente (en qué nodo) se hace cada trabajo nuevo;
- el kube-controller-manager, que reúne a los capataces que vigilan sin pausa que la obra siga pareciéndose al plano.

En cada frente de obra, es decir, en cada nodo de trabajo, vive el kubelet, el obrero de planta que recibe las órdenes y se encarga de que se cumplan. A su lado están el kube-proxy, que lleva el correo al lugar correcto, y el contratista, el container runtime, que es quien pone cada ladrillo. Un frente de obra no está amarrado a una sola casa: su cuadrilla puede trabajar hoy en la casa principal y mañana en la de los nietos.

Este capítulo recorre la obra oficio por oficio: quién manda, quién anota,

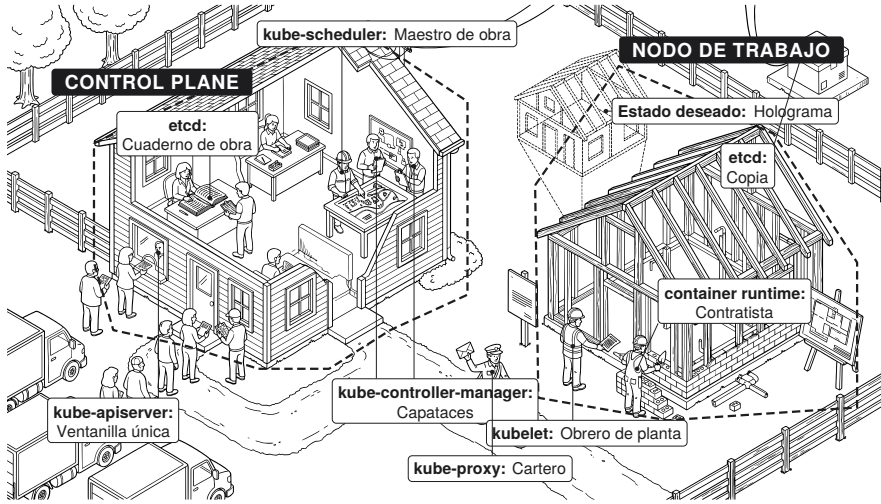


Figura 1.1. La finca completa. A la izquierda la casa de control: la ventanilla única (kube-apiserver), el cuaderno de obra (etcd), el maestro de obra (kube-scheduler) y los capataces (kube-controller-manager). A la derecha un frente de obra: el obrero de planta (kubelet) lee la orden y el contratista (container runtime) es quien pone los ladrillos.

quién decide, quién ejecuta, por qué la casa de control se construye antes que todo lo demás, y qué pasa cuando la finca crece tanto que necesita más de una oficina central.

En la Figura 1.1 aparece la finca entera de un solo vistazo, con cada oficio en su puesto: vale la pena volver a ella cada vez que un componente nuevo entre en escena.

1.2 La ventanilla única: por qué todo pasa por el kube-apiserver

En una obra seria no se le recibe material a cualquiera que llegue con un camión: hay una sola ventanilla, con un solo funcionario, que recibe todo pedido, lo revisa y solo después lo deja pasar al resto de la obra. Nadie, ni el maestro, ni los capataces, ni el obrero de planta del frente tres, le habla directamente al cuaderno de obra: todos le hablan a la ventanilla, y la ventanilla es la única autorizada para escribir en él. La Figura 1.2 resume el recorrido completo de una petición, desde su `kubectl apply` hasta que la decisión

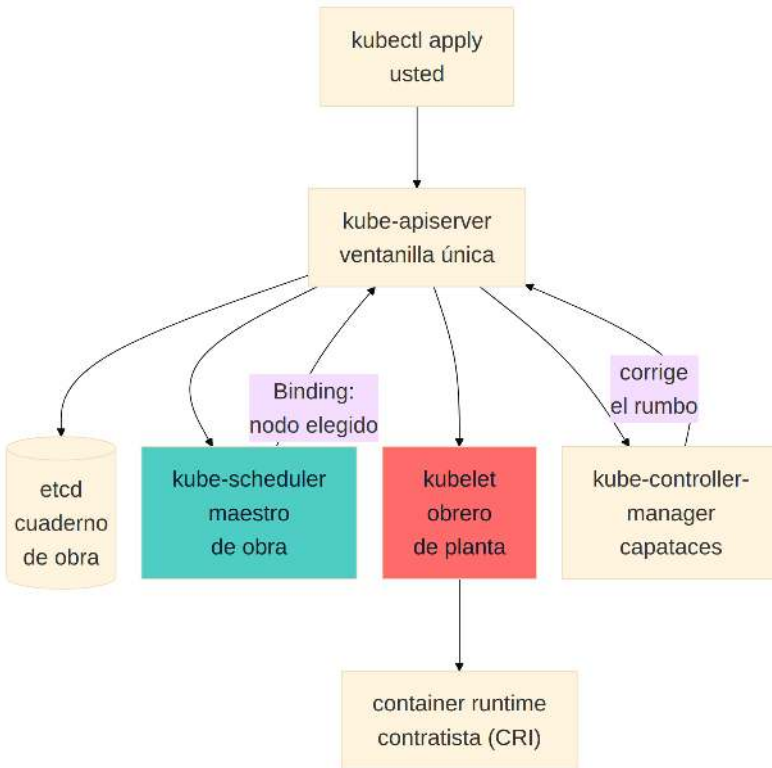


Figura 1.2. El camino de una petición en la obra: la ventanilla única, el cuaderno, el maestro y los capataces.

queda escrita en el cuaderno.

En Kubernetes esa ventanilla se llama kube-apiserver (como muestra la Figura 1.3), y la documentación oficial lo describe sin rodeos como el componente central que expone la API HTTP de Kubernetes.

Veamos el pedido más sencillo que usted le puede llevar a esa ventanilla. El archivo `casa.yaml` declara la casa principal (un namespace) y a quienes van a vivir en ella (un Deployment con dos Pods):

```
apiVersion: v1
kind: Namespace
```



Figura 1.3. El kube-apiserver, la ventanilla única de la casa de control: recibe, revisa y deja pasar cada petición, y es el único que escribe en etcd.

```yaml
metadata:
  name: casa-principal
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: habitantes
  namespace: casa-principal
spec:
  replicas: 2
  selector:
    matchLabels:
      app: habitantes
  template:
    metadata:
      labels:
        app: habitantes
    spec:
      containers:
      - name: habitante
        image: nginx:1.29
```

```
kubectl apply -f casa.yaml
```

```
namespace/casa-principal created
deployment.apps/habitantes created
```

Cuando usted corre `kubectl apply -f casa.yaml`, esa instrucción llega directo al kube-apiserver, y solo el kube-apiserver decide si la deja pasar y la escribe en etcd. Usted no le habla al scheduler ni a ningún nodo: entrega el plano en la ventanilla, y el resto de la obra se entera por ella.

El kube-scheduler tampoco coloca un Pod conversando con el kubelet del nodo elegido: le comunica su decisión al kube-apiserver, que la deja anotada. El kubelet, del otro lado, no recibe órdenes por debajo de cuerda: vigila al kube-apiserver preguntando qué le corresponde en su frente. Todos los oficios de la obra son, en el fondo, clientes de la misma ventanilla, y ninguno tiene permiso de saltársela.

1.3 El cerebro de la obra: el cuaderno, el maestro y los capataces

Detrás de la ventanilla única trabajan tres oficios que nunca se mezclan, aunque comparten la misma oficina central.

El primero es el cuaderno de obra, y en Kubernetes se llama etcd (la Figura 1.4 lo muestra en su escritorio). La documentación oficial lo describe como un almacén de clave-valor consistente y de alta disponibilidad para todos los datos del kube-apiserver, y esa frase merece leerse al pie de la letra: etcd no guarda registros de aplicación, ni bitácoras de lo que hicieron sus contenedores, ni nada parecido a un diario de obra completo. Guarda el estado deseado (la casa de líneas punteadas que flota sobre la obra en la Figura 1.7) y el estado conocido del clúster, ni una línea más, y lo guarda de forma consistente: si dos capataces preguntan al mismo tiempo cuántos obreros hay trabajando, los dos reciben la misma respuesta, nunca dos versiones distintas de la realidad.

El segundo oficio es el maestro de obra, el kube-scheduler (en la Figura 1.5 aparece señalando el mapa de la finca), y su tarea es más angosta de lo que mucha gente cree: la documentación dice que busca los Pods que todavía no



Figura 1.4. etcd, el cuaderno de obra: la archivista anota en el libro el estado deseado y el estado conocido de la finca, siempre de forma consistente.



Figura 1.5. El kube-scheduler, el maestro de obra: señala en el mapa de la finca a qué nodo le toca cada Pod que todavía no tiene dónde ir, y anota su decisión a través de la ventanilla.

están asignados a ningún nodo y le asigna a cada uno un nodo adecuado. Eso es todo. El maestro no construye nada con sus manos, no habla con el obrero de planta y ni siquiera se queda a vigilar que la obra se termine: decide, anota su decisión en el cuaderno a través de la ventanilla, y sigue con el siguiente Pod de la fila.

Aquí conviene detenerse un momento. Es muy común, sobre todo entre quienes preparan el examen, imaginarse que el kube-scheduler llama por radio al kubelet del nodo ganador y le dice "arranque este Pod ya". Eso no ocurre, y creerlo lleva a buscar la falla en el lugar equivocado cuando un Pod no arranca. El scheduler jamás se comunica con un kubelet: escribe un objeto Binding en el kube-apiserver, que a su vez queda reflejado en etcd, y es el kubelet de ese nodo el que, vigilando la ventanilla, descubre que le asignaron trabajo nuevo y lo pone en marcha. El maestro decide; el obrero de planta se entera solo, sin que nadie lo llame en persona.



Figura 1.6. El kube-controller-manager, los capataces: con la tabla en la mano comparan lo que quedó en el cuaderno con lo que hay en el terreno y corrigen la diferencia.

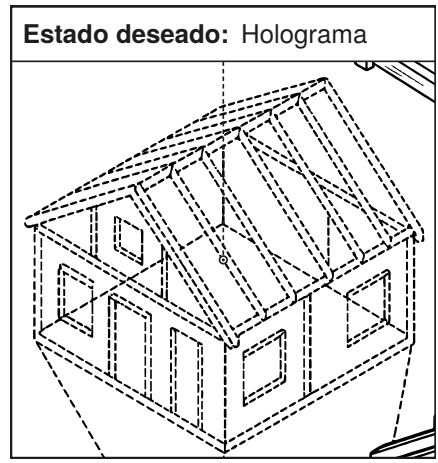


Figura 1.7. El estado deseado: el holograma de la casa terminada flota sobre la obra y es la meta contra la que todos comparan; en Kubernetes es lo que declara el manifiesto y guarda etcd.

El tercer oficio es el equipo de capataces, el kube-controller-manager (la Figura 1.6 los muestra mirando ese mismo mapa), y ahí la palabra clave es vigilancia continua. La documentación oficial dice que ejecuta los controladores que implementan el comportamiento de la API de Kubernetes: un capataz cuenta cuántos habitantes hay en cada casa y pide más si faltan, otro revisa que cada oficio tenga su cuenta de servicio, otro vigila los nodos y avisa si alguno dejó de responder. Ninguno construye nada tampoco: todos comparan lo que anotó el cuaderno contra lo que hay en el terreno, y cuando no coinciden, corrigen el rumbo escribiendo una nueva instrucción, otra vez a través de la ventanilla única.

> **Nota**
> Un Pod que se queda en `Pending` con `<none>` en la columna NODE de `kubectl get pod <nombre> -o wide` nunca recibió su Binding: el problema está del lado del scheduler (falta de recursos, un nodeSelector imposible, un taint), y `kubectl describe pod <nombre>` lo confirma con un evento `FailedScheduling`. Si la columna NODE ya muestra un nodo y el Pod igual no arranca, el maestro cumplió su parte y toca revisar al kubelet y al container runtime de ese frente.

1.4 Los oficios de cada frente de obra: el obrero de planta, el cartero y el contratista

Bajemos de la oficina central a cada frente de obra, es decir, a cada nodo. Ahí trabajan tres oficios más, y estos sí tienen las manos metidas en el barro todos los días.

El primero es el kubelet, que la documentación describe sencillamente como el agente que se asegura de que los Pods estén corriendo, incluidos sus contenedores. Es el obrero de planta de ese frente específico: vigila al kube-apiserver preguntando qué Pods le tocaron a su nodo, y cuando encuentra trabajo nuevo, lo ejecuta. Pero el kubelet tampoco pone ladrillo con sus propias manos: para de verdad crear y correr un contenedor, le pasa la orden a un contratista de confianza, el container runtime (containerd, por ejemplo; en la Figura 1.8 se ve al obrero leyendo la orden junto al contratista que levanta el muro), a través de un contrato estándar llamado CRI (Container Runtime Interface), la interfaz de tiempo de ejecución de contenedores. Ese contrato es lo que le permite a Kubernetes cambiar de contratista sin tener que reescribir al obrero de planta: cualquier runtime que hable CRI sirve.

Quede claro, entonces, quién pone cada ladrillo. El obrero de planta recibe la orden, manda al contratista y después revisa que lo construido siga en pie. El contratista es el que trabaja con la cuchara y la plomada: descarga la imagen del contenedor, levanta sus paredes de aislamiento, le conecta las tuberías de recursos y lo arranca. Si un contenedor se cae, el kubelet lo nota y vuelve a mandar al contratista, según la política de reinicio que traiga el Pod.

El tercer oficio de cada frente es el kube-proxy (como muestra la Figura 1.9),

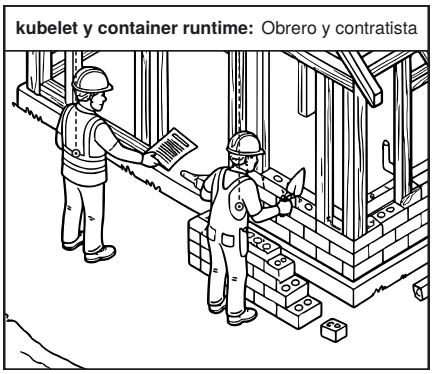


Figura 1.8. El obrero de planta (kubelet) recibe la orden de trabajo y se la pasa al contratista; no toca un ladrillo. Quien los pone, uno por uno, es el contratista (container runtime), al que el kubelet encarga la obra mediante el contrato CRI.



Figura 1.9. El kube-proxy, el cartero de cada nodo de trabajo: mantiene las reglas de red que llevan el tráfico de cada Service a una de sus réplicas.

y la documentación lo resume como el componente que mantiene las reglas de red en los nodos para implementar los Services. Piénselo como el cartero del barrio: cuando alguien manda una carta dirigida al Service del frente tres, no importa cuál de las réplicas la reciba en ese momento, el cartero se encarga de que llegue a una de ellas, siguiendo reglas de red que él mismo mantiene al día en cada nodo.

Tres oficios por frente, y una regla que no cambia: el kubelet hace cumplir lo que la oficina central decidió, el container runtime pone cada ladrillo siguiendo el contrato CRI, y el kube-proxy asegura que el correo de red llegue a la puerta correcta. Ninguno decide qué se construye; eso ya se resolvió arriba, en la ventanilla única y en el cuaderno de obra.

1.5 Dos casas en vez de una: el sobrecosto que evita la ruina

Hay una pregunta que toda esta historia deja pendiente: ¿quién construye la oficina central misma? Porque el kube-apiserver, etcd, el scheduler y el controller-manager también son procesos que tienen que arrancar en alguna parte, y no pueden depender de un scheduler que todavía no existe para que los coloque.

Aquí aparece una complejidad que no se ve cuando uno despliega con Docker Compose. Con Compose usted levanta una sola casa: un archivo, una máquina y sus contenedores adentro. Compose sí sabe reiniciar un contenedor que se cae, con una política de reinicio, y puede marcarlo como enfermo con un healthcheck, pero todo eso pasa dentro de esa misma máquina. No hay oficina central que lleve la cuenta del estado deseado de toda la finca, no hay maestro de obra que reparta el trabajo entre varias máquinas, y si la máquina entera se apaga, nadie muda a sus habitantes a otro frente de obra. Kubernetes, en cambio, le exige construir dos casas: la casa principal, donde viven sus aplicaciones, y la casa de control, donde trabajan la ventanilla, el cuaderno, el maestro y los capataces.

Y el orden no se negocia: la casa de control se construye primero. Sin ella nadie sabría quién hace qué en la obra de la casa principal. Es un sobrecosto, sin duda, y se paga para que la casa no se caiga como la del vecino, llevándose una inversión que costó años de trabajo reunir.

Una comparación ayuda a ver qué se compra con ese sobrecosto. En 2023, Seymour, Masic y sus colegas publicaron en *Science Advances* un estudio sobre la durabilidad del concreto de la antigua Roma, titulado en inglés «Hot mixing». Analizaron muestras del sitio arqueológico de Privernum y propusieron que los pequeños grumos de cal de esa mezcla sirven como reservas de calcio reactivo, capaces de ir rellenando poros y grietas con el paso del tiempo. Después fabricaron mezclas modernas con esos mismos grumos y demostraron su potencial de autorreparación. Una casa moderna de la finca no hace eso: si una pared se cuartea, toca llamar a un maestro para que encuentre el daño y lo arregle. Kubernetes se parece más al concreto romano. Mire lo que pasa si alguien saca a uno de los habitantes de la casa principal (los nombres de sus Pods serán otros):

```
kubectl -n casa-principal delete pod
 ↪  habitantes-86564cc6b9-4jzmk
kubectl -n casa-principal get pods
```

```
pod "habitantes-86564cc6b9-4jzmk" deleted from
 ↪  casa-principal namespace
NAME                           READY   STATUS
 ↪  RESTARTS   AGE
habitantes-86564cc6b9-mrxkf    1/1     Running   0
 ↪  3s
habitantes-86564cc6b9-vnw2h    1/1     Running   0
 ↪  149m
```

Nadie llamó al maestro: el capataz del Deployment vio dos habitantes en el plano y uno solo en la casa, y repuso al que faltaba.

> **Precaución**
> La autorreparación tiene dueño. Un Pod creado a mano con `kubectl run visitante --image=nginx:1.29` no pertenece a ningún controlador: si se borra, o si su nodo desaparece, nadie lo repone. En un clúster kind, `kubectl delete pod visitante` lo dejó borrado para siempre, mientras los Pods del Deployment de al lado volvían solos. En producción, lo que deba sobrevivir se declara con un Deployment, un StatefulSet o un DaemonSet, nunca como Pod suelto.

1.6 Una obra con más de una oficina central: pods estáticos y alta disponibilidad

Volvamos a la pregunta pendiente: si la casa de control va primero, ¿quién la levanta? kubeadm resuelve ese dilema del huevo y la gallina con pods estáticos. Un pod estático, según la documentación oficial de Kubernetes, es un Pod que el kubelet de un nodo específico gestiona directamente, leyendo manifiestos de una carpeta fija del sistema de archivos (normalmente `/etc/kubernetes/manifests/`), sin que el kube-apiserver ni el scheduler intervengan en la decisión. Es la única obra que el obrero de planta encarga

por su cuenta, sin esperar instrucciones de la oficina central, precisamente porque lo que está levantando es esa oficina. Así se ve la carpeta en el nodo de control de un clúster creado con kubeadm:

```
ls /etc/kubernetes/manifests
```

```
etcd.yaml
kube-apiserver.yaml
kube-controller-manager.yaml
kube-scheduler.yaml
```

Cuatro archivos, cuatro oficios de la casa de control: el kubelet los lee al arrancar y los mantiene vivos mientras sigan en el disco.

Lo que usted ve con `kubectl get pods -n kube-system` es apenas un reflejo de esos archivos, un Pod espejo que el kubelet publica para que la ventanilla sepa qué hay corriendo. Si usted lo borra con `kubectl delete`, el kubelet lo vuelve a levantar enseguida, con el mismo nombre, porque el manifiesto sigue en su carpeta. Un Pod de un Deployment también regresa, como vimos en la sección anterior, pero por otro camino: lo repone un capataz del controller-manager, a través de la ventanilla y con un nombre nuevo. El pod estático, en cambio, lo repone el kubelet local sin consultarle a nadie.

Y cuando la finca crece y una sola oficina central ya no basta, hay que decidir cómo repartir el cuaderno de obra entre varias. La documentación oficial de kubeadm sobre topologías de alta disponibilidad describe dos formas de hacerlo. En la topología apilada (stacked etcd), cada nodo de control plane guarda su propio miembro de etcd en la misma oficina donde vive el kube-apiserver: es más sencilla de montar, pero si esa oficina se cae, se pierden juntos el control plane y la copia del cuaderno de ese nodo. En la topología externa (external etcd), el cuaderno se muda a una bodega aparte, en nodos dedicados solo a etcd: cuesta más máquinas, pero un incendio en la oficina central ya no se lleva el cuaderno con ella. Ninguna de las dos es "la correcta": la apilada es la que kubeadm monta por defecto, y la externa es la que se escoge cuando perder las dos cosas a la vez no es una opción.

> **Consejo**
> Para saber en segundos si un Pod es estático, pregúntele por su dueño: `kubectl -n kube-system get pod kube-scheduler-controlplane -o jsonpath='{.metadata.ownerReferences[0].kind}'` responde `Node`, mientras que un Pod de Deployment responde `ReplicaSet`. El nombre también lo delata, porque termina con el nombre del nodo (aquí, `controlplane`). Para cambiar o retirar un pod estático hay que editar o sacar su archivo de `/etc/kubernetes/manifests` en ese nodo: `kubectl delete` solo borra el reflejo, y el kubelet lo vuelve a publicar.

1.7 Practique en el laboratorio

Temas del laboratorio `lab-part-01` (Parte I) para este capítulo:

- Listar los Pods del namespace `kube-system` y reconocer los cuatro oficios de la casa de control.
- Entrar al nodo de control y contar los manifiestos de la carpeta de pods estáticos.
- Averiguar quién figura como dueño del Pod del kube-apiserver y de dónde salió su configuración.
- Comparar ese resultado con un Pod de kube-proxy, que sí reparte un controlador del clúster (un DaemonSet).

1.8 Resumen

Volvamos a la finca. Los cimientos de la casa no los pone un solo oficio. Hay una ventanilla única, el kube-apiserver, por donde pasa cualquier instrucción antes de quedar escrita, empezando por su `kubectl apply -f casa.yaml`. Hay un cuaderno de obra, etcd, que guarda el estado del clúster y nada más que eso. Hay un maestro de obra, el kube-scheduler, que decide qué nodo recibe cada Pod nuevo sin hablar nunca en persona con quien lo va a construir. Y hay capataces, el kube-controller-manager, que no dejan de comparar lo que anota el cuaderno contra lo que existe en el terreno, y corrigen cuando no coinciden.

En cada frente de obra, el kubelet recibe lo que le asignaron y manda al con-

tratista, el container runtime, que es quien pone cada ladrillo a través del contrato CRI; el kube-proxy, cartero del barrio, reparte el tráfico de red hacia la puerta correcta. A diferencia de Docker Compose, que levanta una sola casa en una sola máquina, Kubernetes le pide dos: la casa principal y la casa de control. La de control se construye primero, con pods estáticos que el kubelet de cada nodo de control plane levanta sin depender del scheduler, y ese sobrecosto compra algo parecido al concreto romano: una obra que tapa sus propias grietas sin que haya que llamar al maestro.

Cuando la obra crece, toca decidir dónde vive el cuaderno: apilado junto al control plane, más simple pero más frágil, o en una bodega externa, más costosa pero más segura. Con la casa ya en pie, el próximo capítulo saca el llavero: quién recibe copia de cuál llave de esta misma casa, y por qué nadie debería andar con la maestra.

Bibliografía

- Kubernetes documentation, Kubernetes Components https://kubernetes.io/docs/concepts/overview/components/
- Kubernetes documentation, Static Pods https://kubernetes.io/docs/tasks/configure-pod-container/static-pod/
- Kubernetes documentation, Options for Highly Available Topology https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/
- Seymour, L. M., Maragh, J., Sabatini, P., Di Tommaso, M., Weaver, J. C., Masic, A. (2023). Hot mixing: Mechanistic insights into the durability of ancient Roman concrete. Science Advances 9(1), eadd1602 https://doi.org/10.1126/sciadv.add1602
- Docker documentation, Compose file reference: services, restart https://docs.docker.com/reference/compose-file/services/#restart