# **Guía de estudio sobre Kubernetes y Cloud Native**

## **Componentes del plano de control**

Administran el estado general del clúster:

### **kube-apiserver**

El servidor de componentes principal que expone la API HTTP de Kubernetes.

### **etcd**

Almacén de valores clave coherente y de alta disponibilidad para todos los datos de los servidores de API.

En Kubernetes, etcd es la «fuente de la verdad» y almacena todos los datos de estado y configuración del clúster, incluidos los objetos, los secretos y los ajustes de configuración.

### **kube-scheduler**

Busca los pods que aún no estén enlazados a un nodo y asigna cada pod a un nodo adecuado.

El kube-scheduler se activa para asignar un pod a un nodo cuando el pod se crea sin un nombre de nodo específico, y determina cuál es el mejor nodo en función de las necesidades y restricciones de recursos.

### **kube-controller-manager**

Ejecuta controladores para implementar el comportamiento de la API de Kubernetes.

### **Administrador de controladores en la nube (opcional)**

Se integra con los proveedores de nube subyacentes.

## **Componentes del nodo**

Ejecutan en todos los nodos, mantienen los pods en ejecución y proporcionan el entorno de ejecución de Kubernetes:

### **kubelet**

Garantiza que los pods estén funcionando, incluidos sus contenedores, y es el principal responsable de garantizar que los contenedores funcionen según lo definido en las especificaciones de los pods.

En Kubernetes, el componente de un nodo responsable de ejecutar los contenedores, tal y como se especifica en las definiciones de los pods, es el kubelet. El kubelet monitorea continuamente el estado del nodo y se asegura de que los contenedores descritos en las especificaciones del pod funcionen según lo esperado. Se comunica con el tiempo de ejecución del contenedor (por ejemplo, Docker o containerd) para lanzar y administrar los contenedores.

### **kube-proxy (opcional)**

Responsable de enrutar el tráfico de los servicios y administrar las reglas de IP. Mantiene las reglas de red en los nodos para implementar los servicios.

### **Tiempo de ejecución del contenedor**

Software responsable de ejecutar los contenedores. Este elemento enlaza con un proyecto o producto de terceros que no forma parte de Kubernetes en sí. Es posible que su clúster necesite software adicional en cada nodo; por ejemplo, también puede ejecutar systemd en un nodo de Linux para supervisar los componentes locales.

## **Complementos**

* **DNS**: Complemento para la resolución de DNS en todo el clúster.
* **Interfaz de usuario web (panel de control)**: Complemento para la administración de clústeres a través de una interfaz web.
* **Monitorización de recursos de contenedores**: Complemento para recopilar y almacenar las métricas de los contenedores.
* **Registro a nivel de clúster**: Complemento para guardar los registros de los contenedores en un almacén de registros central.

## **Cumplimiento de la Iniciativa de Contenedores Abiertos (OCI)**

RunC es el motor de ejecución nativo que cumple con los estándares de la Open Container Initiative (OCI) y sirve como implementación de referencia para sus estándares. Garantiza la compatibilidad de los contenedores en varias plataformas.

## **Ejecución de aplicaciones sin estado en Kubernetes**

La **Implementación** es el objeto de API recomendado para ejecutar aplicaciones escalables y sin estado en un clúster de Kubernetes. Gestiona el estado deseado de las aplicaciones y gestiona las actualizaciones y el escalado de forma eficiente.

## **Ejecución de CronJob en Kubernetes**

**Controlador CronJob**: En Kubernetes, el controlador CronJob es responsable de crear un trabajo a una hora programada, lo que a su vez crea un pod para ejecutar las tareas. El pod se ejecuta y completa las tareas asignadas de acuerdo con el cronograma.

## **El papel de Kubelet en Kubernetes**

kubelet es un componente esencial de Kubernetes. Se ejecuta en cada nodo del clúster y garantiza que los contenedores se ejecuten dentro de los Pods según lo previsto. El kubelet es responsable de ejecutar los contenedores en un nodo de acuerdo con las especificaciones de los pods.

## **Tipos de servicio de Kubernetes**

**Ingress** no se considera un tipo de servicio. Los servicios de Kubernetes incluyen:

* **ClusterIP**
* **NodePort**
* **LoadBalancer**
* **ExternalName**

Estos servicios exponen las aplicaciones que se encuentran dentro o fuera del clúster.

## **Estándar de comunicación Kubelet**

**Interfaz de ejecución de contenedores (CRI)**: Kubelet usa la CRI para comunicarse con los tiempos de ejecución de contenedores, como containerd o CRI-O.

## **Funcionalidad de Cgroups**

Los **Cgroups** (grupos de control) permiten limitar los recursos dentro de un sistema, lo que garantiza que los contenedores no superen la CPU, la memoria u otros recursos asignados.

## **Escalar aplicaciones en la nube**

El **escalado horizontal** es el método más común para escalar las aplicaciones en la nube, donde se agregan más réplicas de una aplicación o un servidor para gestionar la carga adicional.

## **Ventajas de los microservicios nativos de la nube**

Los microservicios ofrecen una mayor flexibilidad y escalabilidad en comparación con las aplicaciones monolíticas, lo que facilita la escalabilidad y la realización de actualizaciones en componentes específicos sin afectar a todo el sistema.

## **Escalado vertical**

El **escalado vertical** implica la asignación de recursos adicionales (como memoria o CPU) a las instancias existentes de una aplicación para aumentar su capacidad.

## **Componentes del plano de control de Kubernetes**

kube-proxy **no** forma parte del plano de control, pero funciona como un proxy de red en cada nodo. Los componentes del plano de control incluyen:

* kube-apiserver
* kube-scheduler
* etcd

## **Factores en la programación de nodos**

Kubernetes tiene en cuenta los requisitos de recursos al seleccionar un nodo para programar un pod. Se tienen en cuenta factores como la CPU, la memoria y las reglas de afinidad.

## **Uso de etcd en Kubernetes**

etcd es el almacenamiento de objetos de fondo de la API de Kubernetes, que almacena todos los datos del clúster, incluidas las configuraciones, los secretos y la información de estado. En Kubernetes, etcd es la «fuente de la verdad», ya que almacena todos los datos de estado y configuración del clúster, incluidos los objetos, los secretos y los ajustes de configuración.

## **Tiempos de ejecución de contenedores con Kubernetes**

Kubernetes admite varios tiempos de ejecución de contenedores, como CRI-O, containerd y el ahora obsoleto Dockershim.

## **Creación de trabajos en Kubernetes**

Un recurso de **CronJob** se usa para crear trabajos de Kubernetes a intervalos programados, por ejemplo, cada hora o todos los días.

## **Características de los StatefulSets**

Los **StatefulSets** ofrecen una implementación y un escalado ordenados y se utilizan a menudo para aplicaciones con estado. También confían en los servicios independientes para mantener una identidad de red estable.

Para lograr una nomenclatura de DNS uniforme para los pods administrados por un StatefulSet en Kubernetes, debes usar un **servicio independiente**. Esto permite que cada pod tenga una entrada de DNS estable, normalmente con el formato pod-name.service-name.namespace.svc.cluster.local, lo que garantiza identidades de red coherentes para los pods de StatefulSet.

## **Agregar nuevos tipos de recursos**

Las **Definiciones de Recursos Personalizadas (CRD)** permiten a los usuarios ampliar la API de Kubernetes añadiendo nuevos tipos de recursos específicos para sus aplicaciones.

## **Escalado horizontal**

El **escalado horizontal** es el proceso de agregar réplicas adicionales de una aplicación o un servidor para gestionar más tráfico o carga.

## **Edición de recursos en Kubernetes**

kubectl edit es el comando que se usa para modificar los recursos directamente en el servidor de Kubernetes.

## **Modelo de computación sin servidor**

El modelo **sin servidor** es un modelo de computación en la nube en el que el aprovisionamiento y la administración de la infraestructura no recaen en el usuario. Los desarrolladores se centran únicamente en el código.

## **Ejecutar pods en nodos específicos**

Un **DaemonSet** garantiza que un pod se ejecute en todos o en algunos nodos específicos, por ejemplo, para supervisar o registrar aplicaciones.

## **Construcciones de pods de Kubernetes**

Un pod de Kubernetes contiene uno o más contenedores, lo que lo convierte en la unidad desplegable más pequeña de Kubernetes.

## **La unidad más pequeña de Kubernetes**

El **pod** es la unidad más pequeña posible en Kubernetes que puede ejecutar un contenedor.

## **Obtención de documentación con kubectl**

kubectl explain proporciona documentación detallada para un tipo de recurso específico directamente desde la línea de comandos.

## **Escalador automático de clústeres**

Ajusta automáticamente la cantidad de nodos de un clúster en función de la demanda de recursos.

## **Protección de secretos**

Kubernetes almacena los secretos como valores codificados en Base64, lo que garantiza la confidencialidad básica. Los datos no se cifran, sino que se codifican mediante Base64, que es un mecanismo de codificación reversible simple.

Kubernetes almacena los secretos en etcd, el almacén de valores clave del clúster, que normalmente se cifra en reposo cuando se configura correctamente.

## **kube-proxy**

Gestiona el reenvío de la red a los puntos finales de servicio correctos en un clúster de Kubernetes.

En Kubernetes, un **punto final** se usa principalmente para representar las direcciones IP y los puertos de los pods que están asociados a un servicio. Realiza un seguimiento de las ubicaciones de red de los pods reales que respaldan el servicio, lo que permite la comunicación entre los servicios y los pods.

## **Programación dinámica**

Kubernetes programa las cargas de trabajo de forma dinámica en función de la disponibilidad de los recursos y las características de los nodos.

## **Helm**

**Helm**, un administrador de paquetes para Kubernetes, simplifica la implementación y la administración de las aplicaciones.

## **Herramientas de GitOps**

Herramientas como **Argo CD** y **Flux** comparan de forma continua el estado deseado en Git con el estado real de producción, teniendo en cuenta las diferencias.

## **Definición de recursos personalizada (CRD)**

Una forma de ampliar la API de Kubernetes mediante la definición de nuevos tipos de recursos. Los CRD permiten crear objetos de recursos personalizados específicos para una aplicación, lo que amplía la funcionalidad de Kubernetes sin alterar la API principal. Se administran mediante comandos estándar de kubectl.

## **Interfaz Service Mesh (SMI)**

Un estándar común para las mallas de servicios, que ayuda a la comunicación entre microservicios.

En el contexto de las aplicaciones nativas de la nube, la **Interfaz Service Mesh (SMI)** se reconoce como la especificación de interfaz estandarizada para las mallas de servicios.

Entre las principales características de las interfaces de malla de servicio se incluyen las siguientes:

* Proporciona una API común para las mallas de servicios en Kubernetes.
* Define las interfaces de administración del tráfico, seguridad y observabilidad en diferentes implementaciones de redes de servicios.

## **Ingress**

Administra el acceso externo a los servicios dentro de un clúster de Kubernetes al exponer las rutas HTTP a los servicios. **Ingress** dirige el tráfico HTTP y HTTPS externo a los servicios del clúster.

## **Descubrimiento de servicios**

Ayuda a que las aplicaciones y los microservicios se localicen entre sí en una red, algo crucial para las aplicaciones nativas de la nube.

## **Arquitectura de Kubernetes**

### **Componentes del nodo maestro:**

* **kube-apiserver**: administra las solicitudes de API y sirve como puerta de entrada al clúster.
* **etcd**: almacena el estado del clúster de Kubernetes. En Kubernetes, etcd es la «fuente de la verdad» y almacena todos los datos de configuración y estado del clúster, incluidos los objetos, los secretos y los ajustes de configuración.
* **kube-scheduler**: decide en qué nodos se ejecutan nuevos pods en función de la disponibilidad de recursos. El kube-scheduler asigna un pod a un nodo cuando se crea sin un nodeName, en función de los requisitos y las restricciones de los recursos.
* **kube-controller-manager**: administra los controladores principales, como el Node Controller, el Replication Controller y otros.

### **Componentes del nodo de trabajo:**

* **kubelet**: garantiza que los contenedores se ejecuten en Pods.
* **kube-proxy**: gestiona el proxy de red para los servicios y dirige el tráfico al contenedor apropiado.
* **Tiempo de ejecución del contenedor**: software responsable de ejecutar los contenedores, por ejemplo, containerd, CRI-O.

## **Redes de Kubernetes**

### **Tipos de servicio:**

* **ClusterIP**: expone los servicios solo dentro del clúster.
* **NodePort**: expone los servicios en un puerto estático en la IP de cada nodo.
* **LoadBalancer**: expone los servicios de forma externa mediante el balanceador de cargas de un proveedor de nube.
* **ExternalName**: asigna un servicio a un nombre DNS externo.

### **Ingress**

Proporciona acceso externo a los servicios a través de HTTP y HTTPS. Gestiona el enrutamiento en función de las rutas URL y los nombres de host.

### **Políticas de red**

Define cómo los pods pueden comunicarse entre sí y con otros puntos finales de la red.

En Kubernetes, las políticas de red son **aditivas**. Esto significa que cuando se aplican varias políticas de red al mismo conjunto de pods, las políticas se combinan (o se intersecan) y el tráfico solo está permitido si cumple con las reglas de todas las políticas aplicables. Básicamente, cada política añade restricciones y solo se permitirá el tráfico permitido de forma explícita en todas las políticas.

## **Objetos de Kubernetes**

* **Pod**: la unidad desplegable más pequeña, que contiene uno o más contenedores.
* **Implementación**: administra los pods sin estado, lo que permite actualizarlos y escalarlos de forma continua.
* **ReplicaSet**: garantiza que se esté ejecutando una cantidad específica de pods en un momento dado.
* **DaemonSet**: garantiza que un pod se ejecute en todos (o algunos) de los nodos del clúster.
* **StatefulSet**: administra las aplicaciones con estado, lo que garantiza que los pods tengan identidades estables.
* **Job/CronJob**: ejecuta los pods hasta completarlos. **CronJobs** se ejecuta de forma programada.
* **PersistentVolume (PV)** y **PersistentVolumeClaim (PVC)**: recursos de almacenamiento abstractos en Kubernetes.

## **Orquestación de contenedores**

### **Programación de contenedores**

Kubernetes usa el planificador para decidir qué nodos deben ejecutarse en función de los requisitos y restricciones de recursos.

### **Afinidad o antiafinidad**

Define reglas para determinar cómo se distribuyen los pods entre los nodos (por ejemplo, colocar ciertos pods juntos o separados).

### **Contaminaciones y tolerancias**

Se utilizan para impedir o permitir que se programen pods en determinados nodos. Las contaminaciones se establecen en los nodos y los pods deben tener una tolerancia equivalente para poder programarse en nodos contaminados.

## **Conceptos nativos de la nube**

### **Aplicación de 12 factores**

Conjunto de directrices para ayudar a diseñar aplicaciones nativas de la nube escalables y fáciles de mantener, incluida la contenedorización, la configuración del entorno y la gestión de dependencias.

Los cuatro pilares de la arquitectura nativa de la nube son:

1. **Microservicios**: descomposición de las aplicaciones en servicios independientes y poco acoplados que se pueden desarrollar e implementar por separado.
2. **Contenedores**: empaquetan las aplicaciones y sus dependencias en unidades livianas y portátiles que pueden funcionar de manera uniforme en varios entornos.
3. **DevOps**: fomentar la colaboración entre los equipos de desarrollo y operaciones, haciendo hincapié en la automatización y mejorando el proceso de entrega.
4. **Integración continua/implementación continua (CI/CD)**: automatización de la integración, las pruebas y la implementación del código para permitir lanzamientos de software rápidos y confiables.

### **Microservicios**

Un estilo de arquitectura en el que las aplicaciones se componen de servicios pequeños e independientes que se comunican a través de API.

### **Service Mesh**

Una capa de infraestructura dedicada para administrar la comunicación por microservicios.

Una malla de servicios facilita la distribución y replicación de datos entre los servicios al:

* **División del tráfico**: dirige el tráfico a diferentes versiones del servicio para lograr actualizaciones graduales y lograr la coherencia de los datos.
* **Equilibrio de carga global**: distribuye las solicitudes entre las regiones para lograr una alta disponibilidad y replicación.
* **Comunicación entre servicios**: garantiza una comunicación segura y confiable para la sincronización de datos.

Además, proporciona administración del tráfico, detección de servicios, seguridad (TLS mutuo, cifrado), observabilidad (monitoreo, registro, rastreo) y resiliencia (reintentos, tiempos de espera, interrupciones del circuito).

Entre las herramientas de red de servicios más comunes se incluyen **Istio** y **Linkerd**.

Por lo general, una malla de servicios incluye:

* **Proxy**: los proxies de Sidecar (por ejemplo, Envoy) administran el tráfico de cada servicio.
* **Plano de control**: configura y administra los proxies, gestionando las políticas de tráfico y la seguridad (por ejemplo, Istio).
* **Plano de datos**: los proxies gestionan el tráfico de red entre los servicios.
* **Seguridad**: garantiza el cifrado (mTLS), la autenticación y la autorización.
* **Observabilidad**: recopila métricas, rastreos y registros para su monitoreo.
* **Administración del tráfico**: administra el equilibrio de carga, los reintentos y el enrutamiento del tráfico.

### **DevOps**

Una práctica cultural y técnica que une a los equipos de desarrollo y operaciones para automatizar y optimizar la entrega de software.

### **CI/CD**

La integración continua (CI) y la entrega e implementación continuas (CD) son prácticas esenciales para realizar cambios de código de manera frecuente y confiable.

En el proceso tradicional de CI/CD:

* **Integración continua (CI)**: implica automatizar la integración de los cambios de código de varios colaboradores, por lo general mediante procesos de creación y prueba automatizados.
* **Despliegue continuo (CD)**: implementa automáticamente todos los cambios que pasan por la fase de prueba hasta la fase de producción sin intervención manual.

Este flujo de trabajo hace hincapié en que, una vez que el código se ha integrado y probado, se implementa directamente en el entorno de producción. Esto contrasta con la entrega continua, en la que el código se prepara automáticamente para su implementación en producción, pero es posible que aún se requiera una aprobación manual para su lanzamiento actual.

### **Infraestructura como código (IaC)**

Administración de la infraestructura mediante archivos de configuración legibles por máquina en lugar de mediante una configuración de hardware físico.

Los servicios independientes de Kubernetes no equilibran la carga, sino que proporcionan acceso directo a las IP de cada pod, lo que permite la comunicación con instancias de pod específicas. Es compatible con la detección de servicios y con nombres DNS estables para los pods de StatefulSet, y es ideal para cargas de trabajo con estado continuo, como las bases de datos, que necesitan un almacenamiento persistente e identidades de red coherentes.

Para lograr una nomenclatura de DNS uniforme para los pods administrados por un StatefulSet en Kubernetes, debes usar un **servicio independiente**. Esto permite que cada pod tenga una entrada de DNS estable, normalmente con el formato pod-name.service-name.namespace.svc.cluster.local, lo que garantiza identidades de red coherentes para los pods de StatefulSet.

## **Observabilidad en Kubernetes**

### **Registros**

Consulta los registros de los pods en ejecución mediante kubectl logs. Para ver los registros en tiempo real, usa la opción -f.

El comando kubectl logs con el indicador --previous se usa para ver los registros de un contenedor terminado en un pod con varios contenedores en Kubernetes.

Ejemplo:

bash

Copy code

kubectl logs -c <container-name> --previous

La marca --previous se puede abreviar como --p.

### **Métricas**

Recopila datos de rendimiento con herramientas como **Prometheus** o el **servidor de métricas de Kubernetes**.

### **Supervisión**

Se utilizan herramientas como **Grafana**, **Prometheus** y **Jaeger** para supervisar y rastrear las cargas de trabajo de Kubernetes.

#### **Prometheus**

Una solución de monitoreo de código abierto que recopila y almacena métricas como datos de series temporales, con un potente lenguaje de consulta para el análisis de datos. Se usa comúnmente junto con Grafana para visualizar las métricas.

Prometheus admite cuatro tipos de métricas:

* **Contador**: acumulativo, solo aumenta (p. ej., el recuento de solicitudes).
* **Indicador**: puede aumentar o disminuir (por ejemplo, el uso de memoria).
* **Histograma**: registra las observaciones en grupos (por ejemplo, la duración de las solicitudes).
* **Resumen**: es similar al histograma, pero también proporciona cuantiles (por ejemplo, la mediana de la duración de las solicitudes).

#### **Grafana**

Una aplicación web versátil de análisis y visualización de código abierto que admite varias fuentes de datos, incluido Prometheus. Es conocida por su capacidad para crear paneles detallados que proporcionan información sobre las métricas y los registros.

#### **Fluentd**

Un recopilador de datos de código abierto diseñado para una capa de registro unificada, que permite integrar la recopilación y el consumo de datos para mejorar la comprensión de los datos. Su vasto ecosistema de complementos admite conexiones de datos con múltiples fuentes y destinos.

#### **KubeCost**

Se centra en la administración de costos de Kubernetes, lo que permite a los equipos monitorear, analizar y optimizar sus gastos en Kubernetes de manera eficiente.

**OpenTracing** y **OpenTelemetry** se centran principalmente en la capa de aplicación de una arquitectura de software para la supervisión y la observabilidad. Estas herramientas se centran en el rastreo, las métricas y el registro distribuidos dentro de las aplicaciones, lo que permite rastrear las solicitudes a medida que pasan por los diferentes servicios y componentes de una arquitectura de microservicios. Al instrumentar la capa de aplicación, proporcionan información sobre el rendimiento, la latencia y las dependencias entre los servicios, lo que ayuda a los desarrolladores a supervisar y solucionar problemas en sistemas distribuidos complejos.

## **Service Mesh**

Las mallas de servicios ofrecen una capa de infraestructura sólida que facilita la comunicación segura y confiable de servicio a servicio dentro de las arquitecturas de microservicios, lo que mejora la observabilidad, la confiabilidad y la seguridad.

* **Cilium**: utiliza la tecnología eBPF para proporcionar funciones de seguridad avanzadas, conectividad de red y equilibrio de carga para los clústeres de Kubernetes. Es fundamental para proteger la comunicación de la red y hacer cumplir las políticas de seguridad.
* **Linkerd**: una red de servicios ligera y de alto rendimiento que simplifica la comunicación entre servicios con observabilidad, rastreo y seguridad integrados, lo que requiere una configuración mínima.
* **Istio**: ofrece una solución integral de red de servicios que permite un control detallado de la comunicación entre servicios con funciones como la comunicación segura entre servicios, la gestión del tráfico y una amplia capacidad de observación. Istio usa **Envoy** como su proxy de servicio predeterminado.
* **Envoy**: Diseñado para aplicaciones modernas nativas de la nube, este proxy periférico y de servicio es la base de varias implementaciones de redes de servicios, incluida Istio, que proporciona detección dinámica de servicios, equilibrio de carga y terminación de TLS.

## **Tiempo de ejecución y ejecución del contenedor**

En esta sección se describen las herramientas esenciales para la administración de contenedores y se destaca la distinción entre los tiempos de ejecución de los contenedores y las soluciones de administración de Kubernetes.

* **Docker**: la plataforma de contenedores más popular, que permite empaquetar el software en unidades estandarizadas para su desarrollo, envío e implementación.
* **Kata Containers**: combina la velocidad de los contenedores con la seguridad de las máquinas virtuales, ofreciendo un entorno aislado para la ejecución de contenedores.
* **containerd**: un tiempo de ejecución estándar del sector centrado en la simplicidad, la solidez y la portabilidad, utilizado por Docker y Kubernetes.
* **CRI-O**: diseñado para Kubernetes, proporciona un tiempo de ejecución de contenedor ligero que cumple totalmente con la interfaz de ejecución de contenedores de Kubernetes.
* **runc**: una herramienta de CLI para generar y ejecutar contenedores de acuerdo con la especificación de la Open Container Initiative (OCI).
* **gVisor**: un motor de ejecución de contenedores de código abierto, diseñado para proporcionar un entorno aislado para ejecutar contenedores, lo que mejora la seguridad al aislar la carga de trabajo del kernel del host.

## **Administración y distribuciones de Kubernetes**

Herramientas y plataformas diseñadas para simplificar la implementación, la administración y el funcionamiento de los clústeres de Kubernetes.

* **k3s**: una distribución de Kubernetes ligera y fácil de instalar, ideal para entornos de computación periférica, IoT y CI/CD debido a sus requisitos mínimos de recursos.
* **minikube**: permite el funcionamiento de un clúster de Kubernetes en ordenadores personales, lo que resulta ideal para fines de desarrollo y pruebas.
* **EKS (Amazon Elastic Kubernetes Service)**: un servicio gestionado que simplifica la ejecución de Kubernetes en AWS y en las instalaciones, proporcionando una integración perfecta con los servicios de AWS.

## **Integración continua/implementación continua (CI/CD) y GitOps**

Estas herramientas automatizan los pasos del proceso de entrega de software, como iniciar compilaciones, pruebas e implementaciones automáticas para agilizar el ciclo de vida del desarrollo.

* **Jenkins**: un servidor de automatización de código abierto versátil, facilita la creación, las pruebas y la implementación del software, ya que admite una amplia gama de complementos para fines de CI/CD.
* **Argo CD**: se especializa en flujos de trabajo nativos de Kubernetes, lo que permite la entrega continua y automatiza los procesos de implementación directamente desde los repositorios de Git.
* **Argo Workflows**: una extensión de Argo como motor de flujo de trabajo para organizar trabajos paralelos en Kubernetes.
* **Flux**: adopta los principios de GitOps al aplicar automáticamente los cambios de Git a Kubernetes, lo que garantiza que el estado de los clústeres coincide con la configuración almacenada en el control de versiones.

Flux aprovecha el kit de herramientas de GitOps para administrar los clústeres y las aplicaciones de Kubernetes. El kit de herramientas de GitOps proporciona un conjunto de API componibles y herramientas especializadas que permiten aplicar los principios de GitOps de forma continua, como la implementación de aplicaciones mediante cambios automatizados en los repositorios de Git.

## **Seguridad y cumplimiento**

Herramientas diseñadas para reforzar la seguridad y el cumplimiento en los entornos de Kubernetes, desde la implementación hasta el tiempo de ejecución.

* **Falco**: detecta el comportamiento anormal de las aplicaciones y las posibles amenazas de seguridad en tiempo real mediante la supervisión de las llamadas al sistema y los registros de auditoría de Kubernetes.
* **Kubescape**: evalúa los clústeres de Kubernetes comparándolos con los estándares de seguridad conocidos y las mejores prácticas, e identifica los problemas de cumplimiento y las vulnerabilidades de acuerdo con las directrices de la NSA y la CISA.
* **OPA (Open Policy Agent)**: un motor de políticas versátil que hace cumplir las políticas en todos los ámbitos, garantizando el cumplimiento de las políticas y normativas de seguridad.

### **Estructura de las 4C**

En el contexto de la seguridad nativa de la nube, el orden correcto para la estructura de las 4C es:

1. **Nube**: la seguridad comienza con la infraestructura de la nube, que proporciona las políticas y los controles de seguridad fundamentales.
2. **Clústeres**: se centra en proteger los clústeres de Kubernetes o de orquestación de contenedores que ejecutan cargas de trabajo.
3. **Contenedores**: implica proteger el tiempo de ejecución de los contenedores y garantizar que estén aislados y protegidos.
4. **Código**: garantiza la seguridad del código de la aplicación, incluidas las prácticas de codificación seguras y la gestión de vulnerabilidades.

### **Contextos de seguridad**

Se definen a nivel de pod o contenedor y especifican la configuración de seguridad para los pods o contenedores individuales. Controlan aspectos como los privilegios de usuario, el acceso al sistema de archivos y las capacidades de un pod o contenedor específico. Los contextos de seguridad se centran en configurar la seguridad de los pods o contenedores individuales, especificando cómo deben ejecutarse.

### **Políticas de seguridad**

* **Pod Security Policies (PSP, Kyverno)**: funcionan a nivel de todo el clúster o del espacio de nombres y aplican los estándares de seguridad en varios pods, lo que garantiza que solo se puedan implementar los pods que cumplan con requisitos de seguridad específicos. Las políticas de seguridad se centran en hacer cumplir las reglas y políticas de seguridad para evitar que, en primer lugar, se programen o creen pods que no cumplan con los requisitos.

## **Almacenamiento en Kubernetes**

Las soluciones de almacenamiento en Kubernetes son esenciales para la persistencia y la administración de los datos en entornos en contenedores, ya que permiten el aprovisionamiento y el escalado dinámicos.

### **Volúmenes**

Almacene datos para contenedores, lo que permite el almacenamiento efímero y persistente.

* **emptyDir**: almacenamiento temporal que se borra cuando se elimina un pod.
* **hostPath**: monta un directorio desde el nodo host en un pod.
* **ConfigMap**: proporciona datos de configuración para los pods. En Kubernetes, el ConfigMap puede utilizar el atributo immutable: true para garantizar que sus datos no se puedan modificar después de la creación. Esto es útil para optimizar el rendimiento y evitar actualizaciones accidentales.
* **Secret**: almacena datos confidenciales como contraseñas, tokens de API o claves SSH. En Kubernetes, las entidades secretas pueden utilizar el atributo immutable: true para garantizar que sus datos no puedan modificarse después de su creación. Esto es útil para optimizar el rendimiento y evitar actualizaciones accidentales.

### **etcd**

Fundamental para Kubernetes, ya que actúa como almacén principal del estado y los metadatos del clúster, lo que garantiza una alta disponibilidad y coherencia en un entorno distribuido.

### **Rook**

Un orquestador de almacenamiento nativo de la nube de código abierto para Kubernetes, que proporciona la plataforma, el marco y el soporte para que un conjunto diverso de soluciones de almacenamiento se integren de forma nativa con los entornos nativos de la nube.

### **Ceph**

Un sistema de almacenamiento distribuido y unificado diseñado para ofrecer un rendimiento, una fiabilidad y una escalabilidad excelentes.

En Kubernetes, el **almacenamiento efímero** es un almacenamiento temporal vinculado al ciclo de vida de un pod. Se usa para datos no persistentes, como registros o archivos temporales, y se elimina cuando se cierra el pod.

En Kubernetes, el **almacenamiento persistente** se agrega a un pod mediante un **PersistentVolume (PV)** y un **PersistentVolumeClaim (PVC)**. El PVC solicita almacenamiento y lo conectas al pod definiendo un volumen en el YAML del pod, que monta el almacenamiento en el contenedor. Este almacenamiento se mantiene mientras se reinicia el Pod.

## **Arquitecturas sin servidor y basadas en eventos**

Estas herramientas ayudan a crear aplicaciones que pueden escalar de cero a escala planetaria sin administrar la infraestructura, centrándose en paradigmas basados en eventos y sin servidor.

* **KEDA (Escalado automático basado en eventos de Kubernetes)**: escala dinámicamente las aplicaciones de Kubernetes en respuesta a eventos de diversas fuentes, lo que optimiza la utilización de los recursos.
* **Knative**: permite crear e implementar aplicaciones sin servidor y basadas en eventos en Kubernetes, lo que agiliza el desarrollo de aplicaciones nativas de la nube.

## **Administración de paquetes e implementación de aplicaciones**

Simplificar la implementación y la administración de aplicaciones en Kubernetes con herramientas que administran paquetes y dependencias.

* **Helm**: Helm, el gestor de paquetes de Kubernetes, agiliza el despliegue de aplicaciones mediante gráficos de Helm, que definen, instalan y actualizan incluso las aplicaciones de Kubernetes más complejas.

## **Mejores prácticas de Kubernetes**

### **Uso del espacio de nombres**

Utilice los espacios de nombres para separar de forma lógica los recursos y las políticas de los diferentes entornos (por ejemplo, desarrollo, prueba o producción).

### **Solicitudes y límites de recursos**

Define las solicitudes y los límites de CPU y memoria para garantizar que los pods obtengan los recursos que necesitan sin sobreaprovisionar.

### **Uso de etiquetas y selectores**

Usa etiquetas para organizar y seleccionar grupos de objetos en Kubernetes.

### **Administración de la configuración**

Almacene la configuración por separado del código de la aplicación mediante **ConfigMaps** y **Secrets**.

## **Seguridad en Kubernetes**

### **Control de acceso basado en funciones (RBAC)**

Controla el acceso a la API de Kubernetes en función de las funciones y los permisos.

### **Políticas de seguridad de los pods (PSP)**

Controlan el contexto de seguridad en el que funcionan los pods, incluidos los privilegios y el acceso al sistema de archivos.

Los contextos de seguridad de Kubernetes se asocian principalmente a la definición de la configuración de seguridad de los pods o los contenedores. Estas configuraciones controlan aspectos como los privilegios de los usuarios, los permisos del sistema de archivos y las capacidades, y garantizan que los contenedores funcionen con el nivel de seguridad adecuado.

### **Políticas de red**

Define qué pods pueden comunicarse entre sí y con terminales externos.

En Kubernetes, las políticas de red son **aditivas**. Cuando varias políticas se dirigen a los mismos pods, solo se permite el tráfico permitido por todas las políticas, lo que genera reglas cada vez más restrictivas.

### **Administración de secretos**

Usa Kubernetes **Secrets** para almacenar datos confidenciales, pero considera la posibilidad de usar herramientas externas como **HashiCorp Vault** para tus necesidades de seguridad avanzadas.

### **Seguridad de la imagen**

Asegúrese de que los contenedores se construyan a partir de fuentes confiables y se escaneen para detectar vulnerabilidades (por ejemplo, con herramientas como **Clair** o **Trivy**).

## **Solución de problemas de Kubernetes**

### **Comandos de kubectl**

* kubectl get: muestra los recursos, como los pods, las implementaciones y los servicios.
* kubectl describe: proporciona información detallada sobre un recurso.
* kubectl logs: obtiene los registros de un contenedor en ejecución.
* kubectl exec: ejecuta un comando dentro de un contenedor.
* kubectl top: muestra el uso de recursos de los nodos y pods.

### **Errores comunes**

* **ImagePullBackOff**: Kubernetes no puede extraer la imagen del contenedor especificada. Esto suele ocurrir debido a un nombre de imagen incorrecto, a que falta una imagen en el registro o a problemas de autenticación con un registro privado.
* **CrashLoopBackOff**: el contenedor del pod no se inicia repetidamente. Esto puede deberse a bloqueos de la aplicación, a una configuración incorrecta o a la falta de dependencias.
* **ErrImagePull**: Kubernetes no pudo extraer la imagen del contenedor del registro. Por lo general, esto se debe a un nombre de imagen incorrecto, a que la imagen no existe o a un problema de red.
* **ImageInspectError**: Kubernetes no puede inspeccionar la imagen después de extraerla, a menudo debido a problemas de corrupción o compatibilidad.
* **ErrImageNeverPull**: la ImagePullPolicy está configurada como Never, pero la imagen no está presente en el nodo. Kubernetes está configurado para no extraer la imagen, pero no puede encontrarla localmente.
* **CreateContainerConfigError**: Kubernetes no puede crear la configuración del contenedor. Esto puede deberse a una configuración no válida, como variables de entorno incorrectas, montajes de volumen o falta de secretos.
* **CreateContainerError**: el contenedor no se puede crear, a menudo debido a problemas con la configuración subyacente de Docker o a que falta una imagen.
* **InvalidImageName**: el nombre de la imagen del contenedor especificado en la especificación del pod no es válido. Esto puede deberse a errores tipográficos o a un formato incorrecto.
* **Conflictos de afinidad o antiafinidad entre nodos o pods**: la programación del pod falla debido a conflictos en las reglas de afinidad o antiafinidad de los nodos o los pods. Esto evita que el pod se programe en cualquier nodo disponible.
* **NodeNotReady**: el nodo en el que está programado el pod no está listo, posiblemente debido a un problema con la configuración o la conectividad del nodo.
* **Pendiente**: el pod está atascado en estado pendiente, a menudo porque no hay suficientes recursos (CPU, memoria) en los nodos o porque ningún nodo cumple con los requisitos de programación del pod.
* **ContainerCannotRun**: el contenedor se inicia pero falla inmediatamente. Esto puede deberse a un comando o punto de entrada no válido, a variables de entorno incorrectas o a la falta de dependencias en el contenedor.
* **OOMKilled**: el eliminador de la falta de memoria (OOM) de Kubernetes canceló el contenedor porque superaba sus límites de memoria.
* **Fecha límite superada**: la tarea o el pod no se completaron en el límite de tiempo especificado. Esto suele ocurrir con Kubernetes Jobs o CronJobs que tienen un límite de tiempo.
* **Desalojado**: el pod se desalojó del nodo debido a limitaciones de recursos, como la presión del disco, la presión de la memoria o los límites de la CPU.
* **No se puede programar**: el pod no se puede programar en ningún nodo. Esto suele ocurrir debido a la falta de recursos, a problemas con la selección de nodos o a problemas con la selección de nodos.
* **DNSResolutionError**: el pod no puede resolver los nombres de DNS, a menudo debido a problemas con el servicio DNS de Kubernetes o con las políticas de red que bloquean el tráfico de DNS.
* **VolumeMountError**: Kubernetes no puede montar el volumen especificado en la especificación del pod, a menudo debido a una configuración incorrecta, a la falta de PersistentVolumeClaims (PVC) o a problemas de permisos.
* **VolumeNotFound**: no se puede encontrar el volumen especificado en la configuración del pod, a menudo porque falta un PersistentVolume (PV) o PersistentVolumeClaim (PVC).
* **Fallo en la sonda de disponibilidad o disponibilidad**: el pod no pasa las sondas de disponibilidad o disponibilidad, lo que hace que Kubernetes lo marque como no listo o lo reinicie.
* **No autorizado**: Kubernetes no puede autenticarse en el servidor API ni en un registro de contenedores privado porque las credenciales no son válidas o los tokens han caducado.
* **Prohibido**: una acción está bloqueada debido a la falta de permisos, a menudo relacionados con las políticas de control de acceso basado en roles (RBAC).
* **ConfigMapKeyRefError**: Kubernetes no puede encontrar una clave específica en el ConfigMap al que hace referencia el pod o la implementación, lo que provoca un error cuando el pod intenta iniciarse.
* **SecretKeyRefError**: al igual que ConfigMapKeyRefError, esto ocurre cuando Kubernetes no puede encontrar una clave específica en el secreto al que hace referencia el pod.
* **ServiceUnavailable**: el servicio Kubernetes no está disponible, a menudo debido a problemas con los pods de servicio o la red subyacentes.
* **Pod CrashLoopBackOff**: el pod se bloquea repetidamente; investiga los registros con kubectl logs.
* **Pods pendientes**: los pods no se pueden programar, a menudo debido a la falta de recursos o a reglas de afinidad.

## **Comandos comunes e importantes de Kubernetes**

### **Comandos básicos**

kubectl version: Muestra la versión de Kubernetes del cliente y del servidor.  
bash  
Copy code  
kubectl version

kubectl get [recurso]: Muestra uno o más recursos de un tipo específico, como pods, despliegues o servicios.  
bash  
Copy code  
kubectl get pods

kubectl get services

kubectl get deployments

kubectl describe [recurso] [nombre]: Muestra información detallada sobre un recurso específico.  
bash  
Copy code  
kubectl describe pod [nombre-del-pod]

kubectl describe service [nombre-del-servicio]

kubectl logs [nombre-del-pod]: Obtiene los registros de un pod. Añade -f para seguir los registros en tiempo real.  
bash  
Copy code  
kubectl logs [nombre-del-pod]

kubectl logs -f [nombre-del-pod]

* kubectl logs -c <container-name> --previous: Ver los registros de un contenedor terminado en un pod con varios contenedores.

kubectl exec [nombre-del-pod] -- [comando]: Ejecuta un comando dentro de un contenedor en ejecución de un pod.  
bash  
Copy code  
kubectl exec [nombre-del-pod] -- [comando]

kubectl exec -it [nombre-del-pod] -- /bin/sh

kubectl delete [recurso] [nombre]: Elimina un recurso específico.  
bash  
Copy code  
kubectl delete pod [nombre-del-pod]

kubectl delete deployment [nombre-del-despliegue]

* kubectl api-resources: Enumera todos los recursos de API disponibles en un clúster de Kubernetes.

### **Creación y exposición de recursos**

kubectl create -f [archivo.yaml]: Crea un recurso mediante un archivo de configuración YAML o JSON.  
bash  
Copy code  
kubectl create -f [archivo.yaml]

kubectl expose [recurso] --port=[puerto] --target-port=[puerto-objetivo]: Expone un recurso, como un pod o una implementación, como un servicio.  
bash  
Copy code  
kubectl expose pod [nombre-del-pod] --port=80 --target-port=8080

kubectl expose deployment [nombre-del-despliegue] --type=NodePort --port=80

kubectl apply -f [archivo.yaml]: Aplica una configuración a un recurso por nombre de archivo o stdin.  
bash  
Copy code  
kubectl apply -f [archivo.yaml]

kubectl scale [recurso] --replicas=[num]: Amplía una implementación o un ReplicaSet al número deseado de réplicas.  
bash  
Copy code  
kubectl scale deployment [nombre-del-despliegue] --replicas=5

# **Visualización del estado de los recursos**

## **kubectl get pods -o wide**

Muestra todos los pods con detalles adicionales, como el estado del nodo, la IP y el contenedor.

**Ejemplo:**

bash

Copy code

kubectl get pods -o wide

## **kubectl top nodes/pods**

Muestra el uso de recursos (CPU/memoria) de los nodos o pods.

**Ejemplo:**

bash

Copy code

# Para nodos

kubectl top nodes

# Para pods

kubectl top pods

## **kubectl get events**

Muestra los eventos recientes del clúster.

**Ejemplo:**

bash

Copy code

kubectl get events

# **Administración del espacio de nombres**

## **kubectl get namespaces**

Muestra todos los espacios de nombres del clúster.

**Ejemplo:**

bash

Copy code

kubectl get namespaces

## **kubectl create namespace [nombre-del-namespace]**

Crea un nuevo espacio de nombres.

**Ejemplo:**

bash

Copy code

kubectl create namespace mi-namespace

## **kubectl delete namespace [nombre-del-namespace]**

Elimina un espacio de nombres y todos los recursos que contiene.

**Ejemplo:**

bash

Copy code

kubectl delete namespace mi-namespace

## **kubectl config set-context --current --namespace=[nombre-del-namespace]**

Cambia el contexto actual al espacio de nombres especificado.

**Ejemplo:**

bash

Copy code

kubectl config set-context --current --namespace=mi-namespace

# **Recursos de edición y aplicación de parches**

## **kubectl edit [recurso] [nombre]**

Abre un editor para modificar la configuración de un recurso.

**Ejemplo:**

bash

Copy code

kubectl edit pod mi-pod

## **kubectl patch [recurso] [nombre] --patch [json|yaml]**

Actualiza un recurso mediante un parche.

**Ejemplo:**

bash

Copy code

kubectl patch deployment mi-despliegue --patch '{"spec": {"replicas": 3}}'

# **Depuración y solución de problemas**

## **kubectl describe [recurso] [nombre]**

Proporciona información detallada sobre un recurso, incluidos los eventos recientes y los cambios de estado.

**Ejemplo:**

bash

Copy code

kubectl describe pod mi-pod

## **kubectl get pod [nombre-del-pod] -o yaml**

Genera la configuración completa de YAML de un recurso.

**Ejemplo:**

bash

Copy code

kubectl get pod mi-pod -o yaml

## **kubectl port-forward [nombre-del-pod] [puerto-local]:[puerto-remoto]**

Reenvía un puerto local a un puerto de un pod para su depuración.

**Ejemplo:**

bash

Copy code

kubectl port-forward pod/mi-pod 8080:80

# **Gestión de Jobs y CronJobs**

## **kubectl create job [nombre-del-job] --image=[imagen]**

Crea un Job que ejecuta una imagen específica.

**Ejemplo:**

bash

Copy code

kubectl create job mi-job --image=busybox

## **kubectl get jobs**

Muestra todos los Jobs.

**Ejemplo:**

bash

Copy code

kubectl get jobs

## **kubectl create cronjob [nombre] --image=[imagen] --schedule="[cron-schedule]"**

Crea un CronJob que se ejecuta de forma programada.

**Ejemplo:**

bash

Copy code

kubectl create cronjob mi-cronjob --image=busybox --schedule="\*/5 \* \* \* \*"

# **Control de acceso basado en roles y cuentas de servicio (RBAC)**

## **kubectl create serviceaccount [nombre-de-la-cuenta]**

Crea una nueva cuenta de servicio.

**Ejemplo:**

bash

Copy code

kubectl create serviceaccount mi-cuenta

## **kubectl get serviceaccounts**

Muestra todas las cuentas de servicio.

**Ejemplo:**

bash

Copy code

kubectl get serviceaccounts

## **kubectl create rolebinding [nombre-del-binding] --clusterrole=[rol] --serviceaccount=[namespace]:[cuenta] --namespace=[namespace]**

Vincula un rol a una cuenta de servicio en un espacio de nombres específico.

**Ejemplo:**

bash

Copy code

kubectl create rolebinding mi-binding --clusterrole=view --serviceaccount=default:mi-cuenta --namespace=default

# **Gestión del contexto y la configuración**

## **kubectl config view**

Muestra la configuración actual del contexto de Kubernetes.

**Ejemplo:**

bash

Copy code

kubectl config view

## **kubectl config get-contexts**

Muestra todos los contextos disponibles.

**Ejemplo:**

bash

Copy code

kubectl config get-contexts

## **kubectl config use-context [nombre-del-contexto]**

Cambia a un contexto de Kubernetes diferente.

**Ejemplo:**

bash

Copy code

kubectl config use-context mi-contexto

# **Certificados y seguridad**

## **Certificados TLS en Kubernetes**

* **TLS (Transport Layer Security)** garantiza una comunicación segura entre los componentes de Kubernetes, como el servidor API, kubelet, etc.
* Los certificados TLS normalmente se generan automáticamente cuando se usa kubeadm o se pueden administrar manualmente.

### **Componentes clave que utilizan certificados TLS:**

* **kube-apiserver**: Debe tener un certificado para asegurar sus comunicaciones.
* **etcd**: Asegura la comunicación entre los miembros y los clientes de etcd.
* **kubelet**: Cada kubelet tiene un certificado de cliente para autenticarse en el kube-apiserver.

### **Rotación de certificados:**

* Kubernetes gestiona automáticamente la rotación de los certificados mediante kube-controller-manager.
* Los administradores deben asegurarse de que la caducidad del certificado no afecte al clúster.

### **Comando para comprobar la caducidad de los certificados:**

bash

Copy code

sudo openssl x509 -noout -enddate -in /etc/kubernetes/pki/apiserver.crt

## **Solicitudes de firma de certificados (CSR)**

* Kubernetes puede gestionar las solicitudes de firma de certificados, donde un kubelet u otro componente solicita un certificado al clúster.
* Los administradores pueden aprobar, denegar o firmar manualmente las CSR mediante el comando kubectl certificate.

**Ejemplos de comandos:**

bash

Copy code

kubectl get csr

kubectl certificate approve <nombre-de-csr>

## **KubeConfig**

* **KubeConfig** es el archivo de configuración que define cómo conectarse a los clústeres de Kubernetes.
* Almacena información como clústeres, usuarios y contextos para interactuar con varios clústeres.

### **Campos clave del archivo ~/.kube/config:**

* **clusters**: Define las direcciones y los certificados de los clústeres.
* **users**: Almacena información de autenticación, como tokens o certificados.
* **contexts**: Asocia usuarios con clústeres.

### **Cambio de contextos:**

bash

Copy code

kubectl config use-context <nombre-del-contexto>

## **Asegurar el servidor API**

* El servidor API es el principal punto de entrada al clúster y necesita controles de seguridad adecuados.
* El modo --authorization-mode predeterminado en el servidor API de Kubernetes es Node, RBAC y AlwaysAllow.

### **Medidas de seguridad clave:**

* **Control de acceso basado en roles (RBAC)**: Controla quién puede acceder a qué recursos del clúster.
* **Controladores de admisión**: Módulos que rigen el comportamiento de las solicitudes. Importantes:
  + **NodeRestriction**: Evita que los nodos modifiquen los recursos de otros nodos.
  + **PodSecurityPolicies (obsoleto)**: Regula la configuración de seguridad a nivel de pod. Use **Open Policy Agent (OPA)** o **Kyverno** como sustitutos.

# **Añadidos de red**

## **CNI (Interfaz de Red de Contenedores)**

* Los complementos del CNI administran las configuraciones de red y la conectividad entre los pods en un clúster de Kubernetes.
* Kubernetes admite varios complementos de CNI:
  + **Calico**: Proporciona políticas de red y enrutamiento basado en BGP.
  + **Flannel**: Proveedor de redes superpuestas simple que asigna subredes a los nodos.
  + **Weave Net**: Solución de red basada en mallas.
  + **Cilium**: Utiliza eBPF para proporcionar redes, seguridad y observabilidad avanzadas.

### **Instalación de complementos de CNI:**

bash

Copy code

kubectl apply -f <plugin.yaml>

## **Configuración de CoreDNS**

* **CoreDNS** es el servicio de DNS de Kubernetes que proporciona resolución interna de nombres para servicios y pods.
* La configuración de CoreDNS implica actualizar el ConfigMap de CoreDNS en el espacio de nombres kube-system.

**Ejemplo:**

bash

Copy code

kubectl edit configmap coredns -n kube-system

### **Depuración de DNS:**

bash

Copy code

kubectl get pods -n kube-system -l k8s-app=kube-dns

kubectl logs <coredns-pod> -n kube-system

# **Políticas de red**

* Las políticas de red definen cómo los pods se comunican entre sí y con sistemas externos.
* En Kubernetes, las políticas de red son **aditivas**: solo se permite el tráfico que es permitido por todas las políticas aplicadas.

## **Ejemplo de una política de red que solo permite el tráfico desde pods del mismo espacio de nombres:**

yaml

Copy code

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-same-namespace

namespace: default

spec:

podSelector: {}

policyTypes:

- Ingress

ingress:

- from:

- podSelector: {}

# **Programación avanzada**

## **Afinidad y antiafinidad de pods**

* **Afinidad de pods**: Garantiza que los pods se programen en nodos cercanos a otros pods específicos.
* **Antiafinidad de pods**: Garantiza que los pods se programen lejos de otros pods específicos.

**Ejemplo de PodAffinity:**

yaml

Copy code

affinity:

podAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

- labelSelector:

matchExpressions:

- key: app

operator: In

values:

- frontend

topologyKey: "kubernetes.io/hostname"

## **Afinidad de nodos y taints/tolerations**

* **Afinidad de nodos**: Garantiza que los pods se programen en nodos que coinciden con ciertas etiquetas.

**Ejemplo de NodeAffinity:**

yaml

Copy code

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: "kubernetes.io/e2e-az-name"

operator: In

values:

- "e2e-az1"

- "e2e-az2"

* **Taints**: Se utilizan para hacer que un nodo sea "no programable" a menos que un pod tenga una tolerancia coincidente.

**Aplicar un taint a un nodo:**

bash

Copy code

kubectl taint nodes node1 key=value:NoSchedule

# **Actualizaciones de Kubernetes**

## **Política de versiones**

* Kubernetes admite **version skew**, donde kubelet puede estar una versión menor por detrás del plano de control.
* Asegúrese de actualizar primero los componentes del plano de control y luego los nodos de trabajo.

## **Actualización de un clúster de Kubernetes**

* Actualización de un clúster administrado por kubeadm:

bash

Copy code

kubeadm upgrade plan

kubeadm upgrade apply <versión>

* Actualice kubelet y kubectl después de actualizar el plano de control:

bash

Copy code

sudo apt-get update && sudo apt-get install -y kubelet kubectl

# **Clústeres de alta disponibilidad (HA)**

## **Configuración del plano de control HA de Kubernetes**

* Un plano de control de alta disponibilidad incluye múltiples instancias de servidor API, etcd y un balanceador de carga.
* Debe configurar un balanceador de carga para distribuir el tráfico entre los servidores API.

## **Copia de seguridad y restauración de etcd**

**Realizar una copia de seguridad de etcd:**

bash

Copy code

etcdctl snapshot save /ruta/a/backup.db

**Restaurar etcd:**

bash

Copy code

etcdctl snapshot restore /ruta/a/backup.db

## **Estrategias de recuperación ante desastres**

* Mantenga siempre instantáneas de etcd actualizadas.
* Mantenga copias de seguridad de los manifiestos del clúster y utilice GitOps para gestionar el estado de la infraestructura.

# **Operadores de Helm y Kubernetes**

## **Conceptos básicos de Helm**

* **Helm** es el gestor de paquetes de Kubernetes.
* Los **charts de Helm** definen, instalan y actualizan los recursos de Kubernetes.

**Ejemplo de estructura de un chart de Helm:**

bash

Copy code

mychart/

Chart.yaml # Metadatos sobre el chart

values.yaml # Valores predeterminados para las plantillas

templates/ # Los recursos de Kubernetes se definen aquí

## **Operadores de Kubernetes**

* Los operadores extienden la funcionalidad de Kubernetes automatizando el ciclo de vida de las aplicaciones mediante controladores personalizados.

# **Dominio: Entrega de aplicaciones nativas de la nube**

Recuerda siempre mantener las mejores prácticas y estar al día con las actualizaciones y cambios en Kubernetes y el ecosistema de aplicaciones nativas de la nube.

# **Reclamaciones de Volumen Persistentes**

## **Proceso de Enlace**

Los **PersistentVolumeClaims (PVC)** son solicitudes de almacenamiento realizadas por los usuarios y se vinculan a un **PersistentVolume (PV)** que coincide con el tamaño, el modo de acceso y la clase de almacenamiento solicitados. Una vez que un PVC se vincula a un PV, la relación es exclusiva hasta que se elimina el PVC.

## **Clases de Almacenamiento**

Los PVC pueden especificar una clase de almacenamiento para determinar el tipo de almacenamiento que necesitan (por ejemplo, clases de SSD, HDD o clases específicas de la nube). Si no se especifica ninguna clase, se utilizará la clase de almacenamiento predeterminada del clúster.

## **Modos de Acceso**

* **ReadWriteOnce (RWO)**: Un solo nodo puede montar el volumen en modo de lectura-escritura.
* **ReadOnlyMany (ROX)**: Muchos nodos pueden montar el volumen como de solo lectura.
* **ReadWriteMany (RWX)**: Muchos nodos pueden montar el volumen en modo de lectura-escritura.

## **Aprovisionamiento Dinámico vs. Estático**

* **Aprovisionamiento Dinámico**: Kubernetes crea automáticamente un PV cuando se solicita un PVC, en función de la clase de almacenamiento especificada.
* **Aprovisionamiento Estático**: Un administrador crea previamente los PV y los asocia manualmente con los PVC según los requisitos de almacenamiento.

## **Políticas de Reclamación**

* **Conservar**: Conserva el PV y requiere una intervención manual para recuperar o limpiar el almacenamiento.
* **Eliminar**: Elimina el almacenamiento subyacente cuando se elimina el PVC, comúnmente utilizado con volúmenes aprovisionados dinámicamente.
* **Reciclar**: Limpia el almacenamiento (elimina los datos), pero está obsoleto en favor del aprovisionamiento dinámico.

## **Redimensionamiento del Volumen**

Los PVC se pueden redimensionar dinámicamente si el almacenamiento subyacente admite el cambio de tamaño y el PVC tiene habilitada la configuración correcta (por ejemplo, habilitar la expansión del volumen en StorageClass).

## **Ejemplo de PVC**

yaml

Copy code

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: mi-pvc

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 10Gi

storageClassName: mi-clase-de-almacenamiento

# **Descubrimiento de Servicios**

## **Modos Principales de Descubrimiento de Servicios**

1. **Basado en DNS**: Los servicios se descubren mediante nombres DNS creados automáticamente.
2. **Variables de Entorno**: Los pods obtienen información del servicio mediante variables de entorno predefinidas en el momento de la creación.

## **Ejemplo de Descubrimiento Basado en DNS**

Cuando se crea un servicio en Kubernetes, se asigna automáticamente un nombre DNS que los pods pueden utilizar para acceder al servicio.

**Ejemplo:**

yaml

Copy code

apiVersion: v1

kind: Service

metadata:

name: mi-servicio

spec:

selector:

app: mi-aplicacion

ports:

- protocol: TCP

port: 80

targetPort: 8080

Los pods pueden acceder al servicio utilizando mi-servicio.default.svc.cluster.local.

# **Puntos Finales de Servicio**

Los **Endpoints** de servicio en Kubernetes representan las ubicaciones de red (direcciones IP y puertos) de los pods que respaldan un servicio específico. Permiten que el servicio dirija el tráfico a las instancias de pod correctas.

## **Asignación de Servicios a los Pods**

Los puntos finales proporcionan las direcciones IP y los puertos reales de los pods que están detrás de un servicio. Cuando se crea un servicio, Kubernetes genera automáticamente un objeto de punto final con las direcciones de todos los pods coincidentes.

## **Actualizaciones Dinámicas**

A medida que se crean o destruyen los pods, el objeto Endpoint se actualiza dinámicamente. Esto garantiza que el servicio siempre sepa a dónde enviar el tráfico, reflejando el estado actual de los pods.

## **Servicios Independientes**

En el caso de un servicio independiente (ClusterIP: none), las consultas DNS del servicio devuelven las direcciones IP de los pods individuales a través del objeto Endpoints, permitiendo la comunicación directa entre pods.

## **Ejemplo de Endpoints**

yaml

Copy code

apiVersion: v1

kind: Endpoints

metadata:

name: mi-servicio

subsets:

- addresses:

- ip: 10.0.0.1

- ip: 10.0.0.2

ports:

- port: 80

# **Programación Avanzada**

## **Afinidad y Antiafinidad de Pods**

### **Afinidad de Pods**

La **afinidad de pods** garantiza que los pods se programen en nodos cercanos a otros pods específicos.

**Ejemplo de PodAffinity:**

yaml

Copy code

affinity:

podAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

- labelSelector:

matchExpressions:

- key: app

operator: In

values:

- frontend

topologyKey: "kubernetes.io/hostname"

### **Antiafinidad de Pods**

La **antiafinidad de pods** garantiza que los pods se programen lejos de otros pods específicos.

**Ejemplo de PodAntiAffinity:**

yaml

Copy code

affinity:

podAntiAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

- labelSelector:

matchExpressions:

- key: app

operator: In

values:

- backend

topologyKey: "kubernetes.io/hostname"

## **Afinidad de Nodos y Taints/Tolerations**

### **Afinidad de Nodos**

La **afinidad de nodos** garantiza que los pods se programen en nodos que coinciden con determinadas etiquetas.

**Ejemplo de NodeAffinity:**

yaml

Copy code

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: "kubernetes.io/e2e-az-name"

operator: In

values:

- "e2e-az1"

- "e2e-az2"

### **Taints y Tolerations**

Las **taints** se utilizan para hacer que un nodo sea "no programable" a menos que un pod tenga una **toleration** coincidente.

**Aplicar un taint a un nodo:**

bash

Copy code

kubectl taint nodes node1 key=value:NoSchedule

**Ejemplo de Toleration en un Pod:**

yaml

Copy code

tolerations:

- key: "key"

operator: "Equal"

value: "value"

effect: "NoSchedule"

# **Actualizaciones de Kubernetes**

## **Política de Versiones**

Kubernetes admite el **version skew**, donde el kubelet puede estar una versión menor por detrás del plano de control. Es importante seguir la política de versiones para evitar incompatibilidades.

* **Actualice primero los componentes del plano de control**.
* **Luego, actualice los nodos de trabajo**.

## **Actualización de un Clúster de Kubernetes**

### **Actualización con kubeadm**

**Plan de actualización:**bash  
Copy code  
kubeadm upgrade plan

**Aplicar la actualización:**bash  
Copy code  
kubeadm upgrade apply <versión>

**Actualizar kubelet y kubectl:**bash  
Copy code  
sudo apt-get update && sudo apt-get install -y kubelet kubectl

# **Clústeres de Alta Disponibilidad (HA)**

## **Configuración del Plano de Control HA de Kubernetes**

Un plano de control de alta disponibilidad incluye múltiples instancias de servidores API, instancias de etcd y un balanceador de carga. Debes configurar un balanceador de carga para distribuir el tráfico entre los servidores API.

## **Copia de Seguridad y Restauración de Etcd**

### **Realizar una Copia de Seguridad de Etcd**

bash

Copy code

etcdctl snapshot save /ruta/a/backup.db

### **Restaurar Etcd**

bash

Copy code

etcdctl snapshot restore /ruta/a/backup.db

# **Recuperación ante Desastres**

## **Estrategias de Recuperación ante Desastres**

* **Mantén siempre instantáneas de etcd**.
* **Mantén copias de seguridad de los manifiestos del clúster** y utiliza **GitOps** para gestionar el estado de la infraestructura.

# **Operadores de Helm y Kubernetes**

## **Conceptos Básicos de Helm**

**Helm** es el gestor de paquetes de Kubernetes. Los **charts de Helm** definen, instalan y actualizan los recursos de Kubernetes.

**Ejemplo de estructura de un chart de Helm:**

bash

Copy code

mychart/

Chart.yaml # Metadatos sobre el chart

values.yaml # Valores predeterminados para las plantillas

templates/ # Los recursos de Kubernetes se definen aquí

## **Operadores de Kubernetes**

Los **operadores** amplían la funcionalidad de Kubernetes automatizando el ciclo de vida de las aplicaciones mediante controladores personalizados.

# **Espacios de Nombres**

Los **espacios de nombres** proporcionan una forma de dividir los recursos del clúster entre varios usuarios o equipos. Ayudan a organizar y gestionar los objetos de Kubernetes, como los pods, los servicios y las implementaciones, de forma lógica.

## **Puntos Clave**

* **Por defecto**, Kubernetes viene con cuatro espacios de nombres:
  + **default**: El espacio de nombres predeterminado para objetos sin otro espacio de nombres especificado.
  + **kube-system**: Contiene los componentes del sistema Kubernetes (por ejemplo, kube-dns).
  + **kube-public**: De acceso público, normalmente se usa para recursos que deberían estar visibles en todo el clúster.
  + **kube-node-lease**: Se usa para los latidos del corazón de los nodos.

## **Comandos de Espacio de Nombres**

### **Listar Todos los Espacios de Nombres**

bash

Copy code

kubectl get namespaces

### **Crear un Nuevo Espacio de Nombres**

bash

Copy code

kubectl create namespace <nombre-del-namespace>

**Ejemplo:**

bash

Copy code

kubectl create namespace mi-namespace

### **Eliminar un Espacio de Nombres**

bash

Copy code

kubectl delete namespace <nombre-del-namespace>

**Ejemplo:**

bash

Copy code

kubectl delete namespace mi-namespace

### **Cambiar a un Espacio de Nombres Específico**

bash

Copy code

kubectl config set-context --current --namespace=<nombre-del-namespace>

**Ejemplo:**

bash

Copy code

kubectl config set-context --current --namespace=mi-namespace

## **Mejores Prácticas**

* **Segregar entornos y aplicaciones de equipo** utilizando espacios de nombres.
* **Utilizar RBAC** para administrar el acceso por espacio de nombres.
* **Supervisar cuotas de recursos y límites** dentro de cada espacio de nombres para evitar el agotamiento de recursos.

# **Objetos**

Los **objetos de Kubernetes** son entidades persistentes en el sistema Kubernetes. Representan el estado deseado del clúster (por ejemplo, aplicaciones en ejecución, configuraciones de carga de trabajo).

## **Objetos Comunes**

* **Pod**: La unidad desplegable más pequeña, que representa uno o más contenedores.
* **Service**: Expone una aplicación como un servicio de red.
* **Deployment**: Administra las aplicaciones sin estado y gestiona las actualizaciones y el escalado.
* **ReplicaSet**: Garantiza que se esté ejecutando una cantidad específica de pods idénticos en un momento dado.
* **StatefulSet**: Administra las aplicaciones con estado.
* **Job**: Administra los trabajos por lotes, garantizando que los pods se ejecuten hasta su finalización.
* **ConfigMap**: Contiene los datos de configuración como pares clave-valor.
* **Secret**: Contiene datos confidenciales como contraseñas o claves de API.

# **API de Kubernetes**

La **API de Kubernetes** es la interfaz de administración central para interactuar con los componentes de Kubernetes. Expone todas las operaciones del clúster a través de puntos finales RESTful.

* **El servidor API** es la interfaz del plano de control, que procesa las solicitudes de API REST.

## **Recursos de API Comunes**

* **/api/v1**: Accede a los recursos principales de Kubernetes (pods, servicios, nodos, etc.).
* **/apis/apps/v1**: Accede a recursos adicionales (deployments, statefulsets, etc.).

## **Ejemplo de Llamada a la API con curl**

bash

Copy code

curl https://<api-server>/api/v1/namespaces/default/pods

Tanto kubectl como los componentes internos utilizan la API de Kubernetes para interactuar con el clúster.

# **Control de Acceso Basado en Roles (RBAC)**

El **RBAC** es un mecanismo de Kubernetes para controlar quién puede acceder a qué recursos del clúster.

## **Componentes Clave**

* **Role**: Otorga permisos dentro de un espacio de nombres específico.
* **ClusterRole**: Otorga permisos en todo el clúster.
* **RoleBinding**: Asocia un rol con usuarios o cuentas de servicio dentro de un espacio de nombres.
* **ClusterRoleBinding**: Asocia un ClusterRole a los usuarios de todo el clúster.

## **Ejemplo: Creación de un Rol y un RoleBinding**

yaml

Copy code

---

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

namespace: default

name: pod-reader

rules:

- apiGroups: [""]

resources: ["pods"]

verbs: ["get", "watch", "list"]

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: read-pods

namespace: default

subjects:

- kind: User

name: "jane"

apiGroup: rbac.authorization.k8s.io

roleRef:

kind: Role

name: pod-reader

apiGroup: rbac.authorization.k8s.io

# **Cápsulas (Pods)**

Un **pod** es la unidad desplegable más pequeña de Kubernetes y representa una única instancia de una aplicación que se ejecuta en un contenedor.

## **Conceptos Clave**

* Un pod puede ejecutar uno o más contenedores, que están estrechamente acoplados y comparten la misma red y almacenamiento.
* Los pods son efímeros por naturaleza, lo que significa que se crean y destruyen de forma dinámica.

## **Ejemplo de Configuración de un Pod**

yaml

Copy code

apiVersion: v1

kind: Pod

metadata:

name: nginx

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

# **Implementaciones**

Una **implementación** es un objeto de Kubernetes de nivel superior que se utiliza para administrar aplicaciones sin estado. Define el estado deseado de la aplicación (por ejemplo, cuántas réplicas de un pod deben estar ejecutándose).

## **Características Principales**

* **Actualizaciones continuas**: Garantiza que no haya ningún tiempo de inactividad al actualizar las aplicaciones. Las implementaciones de Kubernetes utilizan la estrategia RollingUpdate de forma predeterminada, que reemplaza gradualmente los pods antiguos por otros nuevos para minimizar el tiempo de inactividad. Controla el proceso de actualización mediante maxSurge (se permiten pods adicionales) y maxUnavailable (pods que pueden no estar disponibles).
* **Escalado**: Puedes escalar los pods hacia arriba o hacia abajo ajustando el número de réplicas.
* **Autocuración**: Reemplaza automáticamente a los pods enfermos.

## **Ejemplo de Configuración de Despliegue**

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.14.2

ports:

- containerPort: 80

# **Conjuntos de Réplicas (ReplicaSets)**

Un **ReplicaSet** garantiza que se esté ejecutando un número específico de réplicas de pods en todo momento.

## **Características Principales**

* Garantiza que se esté ejecutando la cantidad deseada de pods añadiendo o quitando pods según sea necesario.
* Las implementaciones usan ReplicaSets internamente para mantener las réplicas de los pods.

## **Ejemplo de Configuración de ReplicaSet**

yaml

Copy code

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: nginx-replicaset

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

# **Ingress**

**Ingress** es un objeto de API que administra el acceso externo a los servicios dentro de un clúster de Kubernetes, normalmente a través de rutas HTTP/HTTPS.

## **Características Principales**

* Puede definir reglas para enrutar el tráfico externo a los servicios internos.
* Soporta la terminación SSL/TLS para comunicaciones seguras.

## **Ejemplo de Configuración de Ingress**

yaml

Copy code

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: example-ingress

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

spec:

rules:

- host: example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: my-service

port:

number: 80

# **Escalado Automático Vertical**

El **escalado automático vertical** ajusta de forma dinámica las solicitudes de recursos y los límites de los pods en función de su uso. Esto es útil para aplicaciones que necesitan más CPU o memoria bajo carga, pero no requieren escalarse en términos del número de réplicas.

## **Características Principales**

* El **Vertical Pod Autoscaler (VPA)** ajusta automáticamente la CPU y la memoria de los pods.
* Ayuda a mejorar la utilización de los recursos y evita el sobreaprovisionamiento.

## **Escalado Automático de Clústeres**

El **Cluster Autoscaler** ajusta automáticamente la cantidad de nodos de un clúster en función de las demandas de recursos de las cargas de trabajo.

## **Características Principales**

* Añade nodos cuando los pods no se pueden programar por falta de recursos.
* Elimina los nodos infrautilizados cuando disminuyen las cargas de trabajo.

## **Activación del Cluster Autoscaler**

La mayoría de los servicios gestionados de Kubernetes (por ejemplo, AWS EKS, GCP GKE o Azure AKS) incluyen el Cluster Autoscaler como una función integrada. Requiere definir los recuentos mínimos y máximos de nodos para el clúster.

## **Mejores Prácticas**

* **Utiliza HPA** para aplicaciones con cargas de trabajo variables.
* **Combina HPA y VPA** para un escalado más dinámico.
* **Asegúrate de que el clúster tenga suficientes recursos** (ajuste de escala automático de nodos) para soportar el escalado de los pods.

# **CI/CD**

La **CI/CD** (Integración Continua/Entrega Continua) se refiere al proceso de crear, probar e implementar automáticamente los cambios de código en un entorno de producción.

## **Conceptos Clave**

* **Integración Continua (CI)**: Los desarrolladores combinan con frecuencia los cambios de código en un repositorio compartido y realizan pruebas automatizadas para garantizar la calidad.
* **Entrega Continua (CD)**: Automatiza la implementación de los cambios de código en los entornos de producción o ensayo.

## **Herramientas Comunes**

* **Jenkins**: Un servidor de automatización de código abierto que permite crear, probar e implementar código.
* **Argo CD**: Una herramienta de entrega continua nativa de Kubernetes que sigue el modelo de GitOps.
* **Flux**: Otra solución de entrega continua nativa de Kubernetes.

# **Metodología de 12 Factores para Crear Aplicaciones Nativas de la Nube**

La **metodología de aplicaciones de 12 factores** es un conjunto de mejores prácticas para crear aplicaciones nativas de la nube modernas, escalables y fáciles de mantener.

1. **Base de Código**: Se rastrea una base de código en el control de versiones, con múltiples implementaciones.
2. **Dependencias**: Declare y aísle explícitamente las dependencias.
3. **Configuración**: Almacene la configuración en el entorno.
4. **Servicios de Respaldo**: Trate los servicios de respaldo (como las bases de datos) como recursos adjuntos.
5. **Construir, Lanzar y Ejecutar**: Separe estrictamente las etapas de compilación y ejecución.
6. **Procesos**: Ejecuta la aplicación como uno o más procesos sin estado.
7. **Vinculación de Puertos**: Exporta servicios mediante el enlace de puertos.
8. **Concurrencia**: Escala mediante el modelo de proceso.
   * Las aplicaciones deben diseñarse para gestionar el escalamiento de la carga de trabajo mediante la ejecución de varios procesos (instancias) en contenedores o máquinas virtuales. Kubernetes se destaca en la gestión de este proceso a través de Pods y ReplicaSets.
9. **Desechabilidad**: Maximice la robustez con un arranque rápido y un apagado sin problemas.
   * Las aplicaciones deben diseñarse para que se inicien rápidamente y se cierren correctamente. Kubernetes ayuda en este sentido mediante sondeos de preparación y funcionamiento para gestionar el ciclo de vida de los contenedores.
10. **Paridad entre Desarrollo y Producción**: Mantenga el desarrollo, la puesta en escena y la producción lo más similares posible.
    * Al usar entornos en contenedores y herramientas de orquestación como Kubernetes, los equipos pueden mantener la paridad entre los entornos de desarrollo, preparación y producción, minimizando así los problemas de "todo funciona en mi máquina".
11. **Registros**: Trate los registros como transmisiones de eventos.
    * Las aplicaciones no deben administrar ni almacenar sus propios archivos de registro. En su lugar, los registros deben enviarse a stdout/stderr, donde pueden capturarse y agregarse mediante sistemas de registro centralizados como Fluentd, ELK (Elasticsearch, Logstash, Kibana) o Prometheus/Grafana.
12. **Procesos de Administración**: Ejecute las tareas de administración y gestión como procesos únicos.
    * Las tareas administrativas, como las migraciones de bases de datos, las copias de seguridad y la limpieza de datos, deben tratarse como tareas puntuales y efímeras, que a menudo se gestionan mediante Kubernetes Jobs o CronJobs.

# **Conceptos Adicionales a Tener en Cuenta**

## **Service Mesh (Más Allá de las Aplicaciones de 12 Factores)**

Una **malla de servicios** brinda soporte a nivel de infraestructura para administrar las comunicaciones de servicio a servicio dentro de una arquitectura de microservicios nativa de la nube.

### **Herramientas Populares**

* **Istio**: Proporciona administración del tráfico, observabilidad, seguridad y aplicación de políticas.
* **Linkerd**: Una malla de servicios ligera y de alto rendimiento centrada en la simplicidad y la seguridad.
* **Consul**: Una herramienta de red de servicios para el descubrimiento de servicios, la verificación del estado y el control del tráfico.

### **Principales Beneficios de una Malla de Servicios**

* **Observabilidad**: Proporciona métricas, registros y rastreo de los servicios.
* **Control del Tráfico**: Puede aplicar reglas para el enrutamiento del tráfico, el equilibrio de carga y los reintentos.
* **Seguridad**: Aplica el TLS mutuo (mTLS) entre los servicios, lo que garantiza una comunicación segura.

## **Observabilidad Nativa de la Nube**

La **observabilidad** se refiere a la capacidad de medir los estados internos de un sistema mediante el examen de sus resultados (métricas, registros, trazas).

### **Herramientas Clave**

* **Prometheus**: Una herramienta de monitoreo que recopila datos de series temporales y los expone a través de una API para generar alertas o visualizaciones.
* **Grafana**: Funciona con Prometheus para crear paneles y visualizar las métricas recopiladas.
* **Jaeger**: Una herramienta de rastreo distribuido para rastrear el flujo de solicitudes en arquitecturas de microservicios.

### **Componentes de Observabilidad en Kubernetes**

* **Registros**: Se pueden recopilar mediante herramientas como Fluentd o Elastic Stack.
* **Métricas**: Herramientas como Prometheus y Kubernetes Metrics Server recopilan métricas para escalarlas y supervisarlas automáticamente.
* **Rastreo**: Las herramientas de rastreo distribuidas, como Jaeger o Zipkin, proporcionan información sobre las comunicaciones entre servicios.

## **GitOps para Kubernetes**

**GitOps** es un conjunto de prácticas que utilizan Git como la única fuente confiable para administrar los clústeres de Kubernetes.

### **Herramientas Clave**

1. **Argo CD**: Una herramienta de entrega continua diseñada específicamente para Kubernetes que sigue la metodología GitOps.
2. **Flux**: Otra herramienta de GitOps que automatiza las implementaciones en función de los cambios introducidos en un repositorio de Git.

### **Flujo de Trabajo de GitOps**

1. **Definir el estado deseado** de los recursos de Kubernetes en un repositorio de Git.
2. Cuando se realizan cambios en el repositorio de Git (como configuraciones actualizadas o nuevas implementaciones), herramientas como Argo CD o Flux aplican automáticamente los cambios al clúster de Kubernetes.
3. Estas herramientas comparan continuamente el estado real del clúster con el estado deseado definido en Git y toman medidas correctivas si es necesario.

# **Los Microservicios Frente a las Aplicaciones Monolíticas en el Desarrollo Nativo de la Nube**

## **Arquitectura de Microservicios**

Divide las aplicaciones en servicios pequeños, poco acoplados e implementables de forma independiente que se comunican a través de API.

### **Ventajas de los Microservicios**

* **Escalabilidad**: Cada servicio se puede escalar de forma independiente.
* **Resiliencia**: Las fallas en un servicio no necesariamente afectan a todo el sistema.
* **Implementación más Rápida**: Cada servicio se puede desarrollar, probar e implementar de forma independiente, lo que permite ciclos de lanzamiento más rápidos.

## **Arquitectura Monolítica**

Una **arquitectura monolítica** es una aplicación única y grande en la que todos los componentes están estrechamente integrados.

### **Desafíos de la Arquitectura Monolítica**

* **Escalabilidad**: Requiere escalar toda la aplicación, incluso si solo una parte necesita más recursos.
* **Complejidad de la Implementación**: Los pequeños cambios pueden requerir la reimplementación de toda la aplicación.
* **Mantenimiento**: Se hace cada vez más difícil de mantener y probar a medida que la aplicación crece.

## **Kubernetes y Microservicios**

Kubernetes es ideal para la **arquitectura de microservicios**, ya que admite el escalado dinámico, el descubrimiento de servicios y la gestión del tráfico.

# **Sin Servidor en Kubernetes**

La **computación sin servidor** permite a los desarrolladores crear y ejecutar aplicaciones sin administrar la infraestructura. En Kubernetes, herramientas como **Knative** o **KEDA** permiten la funcionalidad sin servidor.

## **Knative**

* **Knative Serving**: Escala automáticamente las aplicaciones en función del tráfico. Puede reducirse a cero cuando no hay tráfico.
* **Knative Eventing**: Gestiona arquitecturas basadas en eventos, lo que permite que las funciones se activen en función de eventos externos.

## **KEDA (Escalado Automático Basado en Eventos de Kubernetes)**

Proporciona un escalado automático para aplicaciones basadas en eventos al escalar automáticamente los recursos en respuesta a eventos externos, como los mensajes en cola o las solicitudes HTTP.

### **Características Principales de Serverless**

* **Escalado Automático**: Escala hacia arriba y hacia abajo en función de la demanda.
* **Pago por Uso**: Los cargos se basan en el uso real de los recursos (CPU, memoria, tiempo de ejecución).
* **Gestión Cero**: Los desarrolladores se centran en el código, no en la infraestructura subyacente.

# **Comunicación entre Pods**

En Kubernetes, la **comunicación de pod a pod** dentro del mismo nodo tiene las siguientes características:

1. **Comunicación Directa**:
   * Los pods del mismo nodo pueden comunicarse directamente entre sí mediante las direcciones IP internas asignadas por la red del clúster.
2. **Superposición de Red Pod**:
   * Kubernetes usa una superposición de red (según el complemento de red, como Calico o Flannel) para garantizar que los pods puedan comunicarse entre sí, incluso dentro del mismo nodo.
3. **Sin NAT (Traducción de Direcciones de Red)**:
   * Para la comunicación de un pod a otro dentro del mismo nodo, Kubernetes normalmente no requiere la traducción de direcciones de red (NAT). Los pods se comunican directamente mediante sus direcciones IP sin necesidad de traducción.
4. **Mismo Espacio de Nombres de Red**:
   * Los pods que se encuentran dentro del mismo nodo forman parte del mismo espacio de nombres de red, lo que les permite comunicarse entre sí directamente a través de las IP asignadas sin necesidad de equilibrar la carga o enrutamiento externos.
5. **IPs Únicas de los Pods**:
   * Aunque los pods se ejecuten en el mismo nodo, cada pod tiene una dirección IP única dentro del clúster, lo que garantiza una comunicación clara y directa.
6. **Comunicación Rápida y Eficiente**:
   * Dado que la comunicación se produce dentro del mismo nodo, generalmente es más rápida y eficiente que la comunicación entre nodos porque evita los saltos de red entre los nodos.

# **Implementación de Aplicaciones Sin Estado**

En Kubernetes, el objeto de **Deployment** es ideal para implementar aplicaciones sin estado. Gestiona réplicas de pods, admite actualizaciones continuas para evitar tiempos de inactividad y ofrece capacidad de recuperación automática y escalabilidad.

## **Características Principales**

* **Sin Necesidad de Almacenamiento Permanente**: Las aplicaciones sin estado no necesitan almacenamiento permanente y los pods se pueden reemplazar o escalar fácilmente según sea necesario.
* **Gestión de Réplicas**: Define cuántas réplicas de un pod deben estar ejecutándose.
* **Actualizaciones Continuas**: Utiliza estrategias como RollingUpdate para implementar cambios sin tiempo de inactividad.
* **Autocuración**: Reemplaza automáticamente los pods que fallan o están en mal estado.

## **Ejemplo de Despliegue de Aplicación Sin Estado**

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: mi-deployment

spec:

replicas: 3

selector:

matchLabels:

app: mi-aplicacion

template:

metadata:

labels:

app: mi-aplicacion

spec:

containers:

- name: mi-contenedor

image: mi-imagen:latest

ports:

- containerPort: 80

# **Etiquetas**

En Kubernetes, las **etiquetas** son clave para administrar los costos mediante la categorización y la organización de los recursos. Puedes asignar etiquetas para agrupar los recursos por proyecto, equipo o entorno, lo que facilita el seguimiento del uso y la asignación de los costos. Los proveedores de servicios en la nube admiten ampliamente las etiquetas con fines de facturación y supervisión.

## **Ejemplo de Uso de Etiquetas**

yaml

Copy code

metadata:

labels:

proyecto: mi-proyecto

equipo: desarrollo

entorno: producción

## **Comandos para Gestionar Etiquetas**

### **Asignar Etiquetas a un Recurso**

bash

Copy code

kubectl label pods mi-pod proyecto=mi-proyecto equipo=desarrollo

### **Filtrar Recursos por Etiquetas**

bash

Copy code

kubectl get pods -l proyecto=mi-proyecto

# **Comando kubectl logs --previous**

El comando kubectl logs con el indicador --previous se usa para ver los registros de un contenedor terminado en un pod con varios contenedores en Kubernetes.

## **Ejemplo**

bash

Copy code

kubectl logs -c mi-contenedor --previous mi-pod

Esto se puede abreviar como --p:

bash

Copy code

kubectl logs -c mi-contenedor --p mi-pod

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

En un entorno nativo de la nube, el **Ingeniero de Confiabilidad del Sitio (SRE)** suele ser responsable de administrar los acuerdos de nivel de servicio (SLA), los indicadores de nivel de servicio (SLO) y los objetivos de nivel de servicio (SLO).

# **Implementaciones y StatefulSets**

En Kubernetes, las especificaciones de los contenedores de **Deployments** y **StatefulSets** son similares en los siguientes aspectos:

1. **Plantilla de Pod**: Ambos utilizan una plantilla de pod para definir las especificaciones del contenedor, incluida la imagen del contenedor, los recursos, los puertos, las variables de entorno y los montajes de volumen.
2. **Administración de Réplicas**: Ambas permiten definir la cantidad de réplicas (instancias) de los pods.
3. **Comportamiento de los Contenedores**: Ambos administran los contenedores dentro de los Pods de la misma manera, asegurándose de que los contenedores se ejecuten según la configuración especificada.

# **Presupuesto de Pod Disruption (PDB)**

El **Pod Disruption Budget (PDB)** en Kubernetes es un mecanismo que garantiza que haya un número mínimo de pods disponibles durante las interrupciones voluntarias, como las de mantenimiento o actualizaciones. Limita el número de pods que se pueden desalojar o interrumpir simultáneamente.

## **Beneficios**

* **Evita el Tiempo de Inactividad**: Ayuda a mantener la disponibilidad de las aplicaciones durante la actualización o desalojo de los nodos.
* **Disponibilidad Mínima/Máxima**: Puedes especificar la cantidad mínima de pods que deben estar disponibles o la cantidad máxima de pods que pueden interrumpirse a la vez.
* **Interrupciones Voluntarias**: Se aplica a acciones voluntarias, como drenar los nodos, no a las interrupciones involuntarias, como los bloqueos.

## **Ejemplo de PDB**

yaml

Copy code

apiVersion: policy/v1

kind: PodDisruptionBudget

metadata:

name: mi-pdb

spec:

minAvailable: 2

selector:

matchLabels:

app: mi-aplicacion

# **Patrón de Sidecar**

En Kubernetes, el **patrón sidecar** consiste en ubicar los contenedores en un pod para proporcionar funciones complementarias, como el registro, la supervisión, el uso de proxy o la seguridad. Los contenedores sidecar amplían o dan soporte al contenedor principal al gestionar tareas como el enrutamiento del tráfico, el procesamiento de datos o la observabilidad.

## **Ejemplo de Patrón Sidecar**

yaml

Copy code

apiVersion: v1

kind: Pod

metadata:

name: mi-pod

spec:

containers:

- name: contenedor-principal

image: mi-imagen:latest

ports:

- containerPort: 80

- name: sidecar-registro

image: fluentd:latest

ports:

- containerPort: 24224

# **Cerebro Dividido y Elección de Líderes**

La **división del cerebro** ocurre en los sistemas distribuidos cuando los problemas de red hacen que varios nodos actúen como líderes, generando acciones conflictivas e inconsistencias en los datos. Esto interrumpe la estabilidad y la coherencia del sistema.

## **Elección de Líderes**

**Leader Election** evita conflictos similares a los de "cerebros divididos" al garantizar que solo una instancia de una aplicación distribuida con estado actúe como líder a la vez.

# **Espacios de Nombres de Red**

En un pod, los contenedores suelen **compartir el espacio de nombres de la red**. Esto permite que los contenedores del mismo pod se comuniquen entre sí a través del host local y compartan la misma dirección IP y los mismos puertos. También suelen compartir otros espacios de nombres, como IPC (comunicación entre procesos) y UTS (nombre de host).

# **OpenID Connect (OIDC)**

En el contexto de la autenticación y la autorización, **OIDC** (OpenID Connect) es una capa de identidad construida sobre el protocolo OAuth 2.0, que se utiliza para verificar la identidad del usuario y permitir una autenticación segura.

# **Administración de Secretos**

La **administración de secretos** implica almacenar y acceder de forma segura a información confidencial, como las claves de API, las contraseñas y los certificados.

## **Secretos de Kubernetes**

Kubernetes proporciona un objeto **Secret** para almacenar información confidencial, como tokens y credenciales. Los secretos pueden montarse como volúmenes o exponerse como variables de entorno en pods.

### **Mejores Prácticas para Gestionar Secretos**

* **Utilizar el Cifrado**: Almacene siempre los secretos en un formato cifrado. Kubernetes admite el cifrado de los secretos en reposo.
* **Limitar el Acceso**: Utilice el control de acceso basado en roles (RBAC) para restringir el acceso a los secretos. Solo los servicios que necesitan el secreto deben tener acceso.
* **Evitar Codificar Secretos**: Nunca codifique datos confidenciales en el código de la aplicación o en los archivos de configuración.

### **Herramientas Clave para la Gestión de Secretos**

* **HashiCorp Vault**: Una herramienta para almacenar y administrar de forma segura el acceso a los secretos. Proporciona secretos dinámicos, rotación de claves y registros de auditoría.
* **AWS Secrets Manager y Azure Key Vault**: Servicios nativos de la nube que permiten el almacenamiento, la recuperación y la rotación seguros de los secretos.
* **Sealed Secrets**: Cifra los secretos de Kubernetes para garantizar que estén seguros cuando se almacenan en sistemas de control de versiones como Git.

### **Ejemplo de Uso de Sealed Secrets**

bash

Copy code

kubectl create secret generic mi-secreto --from-literal=clave=valor --dry-run=client -o json | \

kubeseal --cert mycert.pem -o yaml > mi-secreto-sealed.yaml

# **Estándares de Seguridad para Pods (PSS) en Kubernetes**

Los **Pod Security Standards (PSS)** definen la configuración de seguridad a nivel de pod para ayudar a aplicar las mejores prácticas de seguridad en los clústeres de Kubernetes. El PSS se introdujo como sustituto de las obsoletas **PodSecurityPolicies (PSP)**.

## **Admisión de Seguridad al Pod**

Kubernetes 1.22 presentó un nuevo controlador de admisión **Pod Security** para aplicar el PSS, que clasifica los pods en tres niveles:

1. **Privilegiado**: No se aplican restricciones. Se usa para pods confiables que requieren muchos permisos, como los componentes de infraestructura.
2. **Base de Referencia**: Aplica las mejores prácticas de seguridad básicas, como evitar la escalada de privilegios o actuar como usuario raíz.
3. **Restringido**: Aplica políticas de seguridad más estrictas, como prevenir los volúmenes de rutas de host y aplicar contextos de seguridad más estrictos.

## **Cómo Usarlo**

Las etiquetas de los espacios de nombres se utilizan para hacer cumplir las políticas de seguridad. Los pods de un espacio de nombres específico deben cumplir los estándares de seguridad aplicados.

Aplica políticas de seguridad en función de las cargas de trabajo y su nivel de sensibilidad (por ejemplo, las cargas de trabajo de producción deben estar en modo restringido).

## **Mejores Prácticas**

* **Utilice el Modo Restringido** para cargas de trabajo críticas a fin de garantizar el entorno más seguro.
* **Evite Utilizar Contenedores Privilegiados** a menos que sea absolutamente necesario.
* **Audite Regularmente** las configuraciones de los pods y cumpla con los estándares de seguridad para reducir las superficies de ataque.

# **Dominio: Entrega de Aplicaciones Nativas en la Nube**

Recuerda siempre mantener las mejores prácticas y estar al día con las actualizaciones y cambios en Kubernetes y el ecosistema de aplicaciones nativas de la nube. Utiliza este conjunto de notas como referencia para gestionar y operar eficazmente tus clústeres de Kubernetes, asegurando la seguridad, escalabilidad y fiabilidad de tus aplicaciones.

# **Dominio: Entrega de Aplicaciones Nativas en la Nube**

Recuerda siempre mantener las mejores prácticas y estar al día con las actualizaciones y cambios en Kubernetes y el ecosistema de aplicaciones nativas de la nube.