Universitat Politècnica de València

Escola Tècnica Superior d'Enginyeria Informàtica

Cloud Computing

Scheduler con kind

Pablo Gonzálbez Cabo

Christian David León Ilaño

George-Alexandru Dogarel

19 de diciembre de 2025

MUCNAP

1. Introducción y Objetivos

Este proyecto de laboratorio consistió en la implementación y despliegue de un scheduler personalizado para Kubernetes, desarrollado en Python. El objetivo principal es comprender el ciclo de vida de planificación (observar, decidir y actuar) y su interacción con el API Server.

El scheduler se diseñó para:

- Identificar Pods Pendientes.
- Elegir un Nodo basándose en una política de planificación.
- Realizar el Binding (vinculación) del Pod al Nodo mediante una llamada a la API de Kubernetes.

2. Lógica de Selección de Nodos (Política de Planificación)

La política inicial de selección de nodos implementada en la función *choose_node* sigue una estrategia sencilla de "menos Pods en ejecución" (least-busy o least-allocated).

2.1. Criterio principal

El scheduler itera sobre todos los Nodos disponibles y cuenta el número de Pods que tienen asignado a cada uno de ellos (*p.spec.node_name* == *n.metadata.name*). El Nodo elegido (*pick*) es aquel que tiene la menor cantidad de Pods actualmente asignados (excluyendo a los que aún están en estado *Pending* sin un *nodeName* asignado).

3. Implementación y Observaciones de Comportamiento

3.1. Proceso de Binding

El paso crucial de "Actuar" se realiza mediante la función *bind_pod*, que crea un objeto V1Binding y lo envía al API Server.

Importancia de la Vending API: Es vital que el scheduler escriba un objeto Binding en lugar de parchear directamente el Pod. Esto garantiza la coherencia transaccional y permite que el API Server se encargue de las validaciones necesarias antes de actualizar el estado de planificación del Pod, manteniendo una clara separación de responsabilidades.

3.2. Observaciones del Scheduler

Al desplegar un Pod de prueba con el *schedulerName: my-scheduler*, se observó el siguiente flujo:

- **Observar**: El scheduler encuentra el Pod pendiente con el selector *spec.nodeName=.*
- **Decidir**: Se aplica la lógica *choose_node* (ej. least-busy).
- **Actuar**: El scheduler registra en sus logs el mensaje de éxito, por ejemplo: *Bound default/test-pod -> kind-control-plane*.

El estado del Pod en el cluster pasa de Pending a Running una vez que el kubelet en el Nodo asignado inicia el contenedor, completando el ciclo de vida del Pod.

4. Polling vs. Watch

Característica	Modelo Polling	Modelo Watch
Mecanismo	Llama periódicamente a *list_pod_for_all_namespaces*.	Abre un stream continuo de eventos (*w.stream*).
Latencia	Alta, limitada por el intervalo de *time.sleep(2).*	Baja, reacciona casi instantáneamente al evento Pod ADDED.
Carga en API	Alta y constante, requiere llamadas LIST frecuentes.	Baja, solo una conexión de larga duración.
Eficiencia	Menos eficiente, procesa la lista completa cada ciclo.	Más eficiente, solo procesa el objeto que ha cambiado *(evt['object'])*

5. Respuestas a las cuestiones planteadas

¿Por qué es importante que el scheduler escriba un objeto Binding en lugar de parchear directamente un Pod?

Porque el Binding es el mecanismo oficial de Kubernetes para la asignación de Pods a Nodos. Al crear un objeto Binding, el scheduler delega en el API Server la validación y actualización del estado del Pod, garantizando coherencia y seguridad. Parchear directamente el campo spec.nodeName saltaría este flujo estándar, podría provocar estados inconsistentes y rompería la separación de responsabilidades entre el scheduler y el control plane.

¿Cuáles son los trade-offs entre los modelos polling y event-driven (watch)?

El polling es más sencillo de implementar, pero introduce mayor latencia y más carga sobre el API Server, ya que requiere llamadas periódicas para listar todos los Pods. En cambio, el modelo event-driven (watch) es más eficiente y reactivo,

ya que el scheduler responde casi inmediatamente a los eventos relevantes (por ejemplo, un Pod añadido en estado Pending), reduciendo tanto la latencia como el número de peticiones al API Server. El trade-off principal es que el modelo watch es algo más complejo de implementar y gestionar.

¿Cómo interactúan los taints y tolerations con tu lógica de planificación?

En este scheduler, la lógica de selección de nodos no evalúa explícitamente taints ni tolerations, por lo que la decisión se basa únicamente en el número de Pods por Nodo. Sin embargo, Kubernetes valida estos aspectos en el API Server durante el proceso de Binding. En un scheduler más completo, sería necesario filtrar previamente los Nodos que no sean tolerables por el Pod para evitar intentos de asignación inválidos.

¿Qué políticas del mundo real podrías implementar usando este framework?

Este framework permite implementar fácilmente políticas reales como:

- Balanceo de carga basado en CPU o memoria disponible.
- Afinidad y anti-afinidad entre Pods.
- Asignación de Pods a Nodos con hardware específico (por ejemplo, GPUs).
- Políticas cost-aware o energy-aware en entornos cloud.
- Priorización de Pods críticos frente a Pods de menor importancia.

6. Conclusiones

El modelo Polling es más simple de implementar, pero ineficiente y lento para reaccionar a nuevos Pods. El modelo Watch es el preferible por su eficiencia y sensibilidad/reactividad (baja latencia), ya que solo actúa cuando se detecta un Pod pendient.