Engineer

entorno de ejecución de contenedores

llevar una imagen de contenedor a un proceso aislado

Qué es

Un entorno de ejecución de contenedores es el software situado entre una imagen de contenedor en disco y un proceso aislado por el kernel. La expresión abarca dos capas, y no todo el mundo se refiere a la misma: el entorno de bajo nivel, también llamado entorno de ejecución OCI, y el de alto nivel, más conocido como motor de contenedores.

El entorno de bajo nivel tiene una tarea concreta. Lee un paquete, que la especificación de ejecución de Open Container Initiative define como un directorio con un archivo `config.json` y el sistema de archivos raíz del contenedor, solicita al kernel los espacios de nombres, cgroups y montajes descritos en la configuración e inicia el proceso dentro de ellos. La especificación enumera las operaciones que debe admitir toda implementación, state, create, start, kill y delete, y deja la línea de comandos a cada una. La versión 1.0 se publicó en julio de 2017. runc, mantenido por la propia OCI, se describe como «una herramienta de bajo nivel que no está diseñada pensando en el usuario final». crun realiza la misma tarea en C con una huella de memoria menor y es la opción predeterminada que indica la documentación de Podman. youki es una tercera opción, escrita en Rust.

El motor se sitúa por encima. containerd se describe como un demonio que gestiona todo el ciclo de vida de los contenedores del host: transferencia y almacenamiento de imágenes, ejecución y supervisión de contenedores y las conexiones de almacenamiento y red que los rodean. CRI-O hace un trabajo similar, diseñado exclusivamente para Kubernetes. Un motor descarga y descomprime la imagen, escribe el paquete, lo entrega al entorno de bajo nivel y sigue vigilando el proceso cuando ese entorno ya ha terminado. Docker Engine está un escalón más arriba: su documentación indica que utiliza containerd para el ciclo de vida del contenedor y que containerd usa runc de forma predeterminada. Podman toma otro camino y no mantiene un demonio de

larga duración. Cada comando prepara el paquete por sí mismo, llama a crun o runc y deja a un pequeño monitor llamado conmon a cargo del contenedor en ejecución.

Kubernetes añade otro término. El kubelet de cada nodo se comunica con el motor mediante Container Runtime Interface, CRI, un protocolo gRPC que implementan tanto containerd como CRI-O. Docker Engine es anterior a CRI, por lo que Kubernetes incluyó durante años un adaptador llamado dockershim y lo eliminó en v1.24.

Utilice el concepto en una decisión

Aplique «¿Qué es un entorno de ejecución de contenedores?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Software empaquetado para comportarse igual en un portátil, un servidor de pruebas y producción, y ampliar su capacidad cuando aumente la actividad.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. 20 aplicaciones Docker en Google Cloud y AWS — 20 aplicaciones en contenedores desplegadas y monitorizadas

20 aplicaciones Docker en Google Cloud y AWS Póngase en contacto →

Por qué importa

Elegir un entorno de ejecución de contenedores significa en realidad elegir qué capa cambiar, y las dos aportan cosas distintas.

El motor decide cómo funcionan los contenedores en el día a día: quién descarga y almacena las imágenes, quién reinicia un contenedor que se ha detenido por un fallo, adónde van sus registros, qué API utilizan las demás herramientas y quién controla el árbol de procesos. Sustituir Docker Engine por Podman en un servidor, o por containerd en un nodo Kubernetes, cambia todo eso y deja intactos los propios contenedores, porque el formato de imagen y el paquete subyacentes siguen los mismos estándares OCI. La documentación de Kubernetes sobre la retirada de dockershim lo dice con claridad: las imágenes creadas con `docker build` funcionan con todas las implementaciones de CRI.

El entorno de bajo nivel decide el grado de aislamiento del proceso. runc y crun inician un proceso normal sobre el kernel del host, algo rápido y plenamente compatible que también significa que todos los contenedores de la máquina comparten ese único kernel. El entorno de gVisor, runsc, coloca un kernel de aplicación escrito en Go entre el contenedor y el host y responde por sí mismo a

las llamadas al sistema del contenedor; su documentación señala como costes una menor compatibilidad con las aplicaciones y una sobrecarga mayor por llamada al sistema. Kata Containers arranca cada carga dentro de una máquina virtual ligera con un kernel propio, utiliza la virtualización por hardware como segunda capa de defensa y necesita un hipervisor en cada nodo que lo ejecute.

Kubernetes permite hacer esta segunda elección por pod en lugar de por clúster. Una RuntimeClass nombra un manejador configurado en el motor, y un pod que establece `runtimeClassName` se ejecuta con ese manejador, de modo que una carga que no sea de confianza puede usar gVisor o Kata mientras el resto sigue en runc. La documentación lo plantea como un equilibrio entre rendimiento y seguridad.

Cómo se aplica en la práctica

En el uso cotidiano, la expresión tiende a referirse a la capa del motor, y el trabajo de este sitio la utiliza con ese sentido.

Veinte aplicaciones empaquetadas con Docker, repartidas entre dos nubes y unos cuantos entornos de ejecución de contenedores, debían desplegarse de forma coherente y permitir un diagnóstico rápido cuando algo fallara. Docker servía para empaquetarlas, Podman para investigar los problemas y Kubernetes para depurar la orquestación. Trabajar así con distintos motores se apoya en las capas anteriores: los motores difieren, pero comparten los estándares OCI de imagen y ejecución que hay debajo.

En un host más reciente, la elección se hizo en la capa del motor. La capa de contenedores utiliza Podman y Quadlet en lugar de Docker, porque el motor predeterminado publica los puertos de los contenedores por delante de las reglas de cortafuegos que instala el rol de endurecimiento y añadiría un segundo supervisor junto a systemd. La elección cambió quién inicia cada contenedor y quién lo vigila después: los contenedores pasaron a ser servicios que arrancan como los demás servicios, bajo el supervisor en el que el host ya confiaba.

El mismo razonamiento se aplica a un host de 512 MB para los servicios de acceso público de la empresa, donde la memoria zanjó todos los debates. Su capa de contenedores consta de unidades Quadlet bajo systemd en lugar de un demonio con supervisor propio, una elección hecha en una máquina con unos 464 MB de memoria utilizables.

Dónde surgen los problemas

Los problemas de ejecución suelen aparecer en las uniones entre capas, donde cada lado da por supuesto que el otro está configurado como espera.

Las herramientas del nodo se quedan sin visibilidad tras actualizar el clúster. Desde Kubernetes v1.24, el kubelet habla directamente con containerd o CRI-O, así que los contenedores que programa el clúster nunca pasan por Docker Engine. `docker ps` en el nodo solo enumera lo que inició el propio Docker, y los agentes de monitorización, scripts de arranque y pods con privilegios que llaman a la CLI de Docker o montan `docker.sock` pierden la visibilidad. Las cargas siguen ejecutándose. Son las herramientas que las rodean las que deben pasar a `crictl` o a la API del motor.

El kubelet y el entorno de ejecución gestionan los cgroups de dos maneras distintas. Ambos deben usar el mismo controlador de cgroups, `cgroupfs` o `systemd`, y el kubelet utiliza `cgroupfs` de forma predeterminada. En un host cuyo sistema de inicio es systemd, eso deja dos gestores de cgroups con dos visiones de los mismos recursos, y la documentación de Kubernetes advierte que esos nodos pueden volverse inestables bajo presión de recursos. La solución es el controlador systemd en ambos lados, que también es el recomendado con cgroup v2. Para containerd 2.x con runc, la parte del entorno de ejecución queda así:

```
# /etc/containerd/config.toml, containerd 2.x
[plugins.'io.containerd.cri.v1.runtime'.containerd.runtimes.runc.options]
  SystemdCgroup = true
```

En el kubelet se configura `cgroupDriver: systemd`, y después hay que reiniciar containerd. containerd 1.x escribe la sección de otra forma, y la página de Kubernetes ofrece ambas. Cambiar el controlador en un nodo que ya se ha unido a un clúster es delicado: los pods creados con uno pueden sufrir errores al recrearse con el otro, por lo que la recomendación documentada es sustituir o reinstalar el nodo con la configuración nueva.

Un pod trasladado a un entorno con aislamiento adicional no logra arrancar. Una RuntimeClass cuyo manejador no está en el motor del nodo envía el pod directamente a la fase Failed y deja el motivo en un evento. Además, la carga debe ser compatible con el entorno aislado. gVisor responde a las llamadas al

sistema en su propio kernel, por lo que el software que utiliza funciones menos comunes del kernel puede comportarse de otra forma, y su documentación considera poco adecuadas las cargas que hacen muchas llamadas al sistema. Kata necesita virtualización por hardware en el nodo, lo que en una máquina virtual de nube significa virtualización anidada. Un selector de nodos `scheduling` en la RuntimeClass mantiene esos pods en los nodos capaces de ejecutarlos.

Una imagen importada a mano queda invisible para el kubelet. containerd conserva los nombres y metadatos de imágenes por espacio de nombres. Su cliente `ctr` usa el espacio `default` salvo que se indique otro, mientras que el plugin CRI con el que habla el kubelet utiliza `k8s.io` . Por eso, una imagen cargada con un simple `ctr images import` termina donde el kubelet no mira, y el pod vuelve al registro o se detiene con un error de descarga. Importar con `ctr -n k8s.io images import` y comprobar con `crictl images` , que ve lo mismo que el kubelet, evita esa separación.

Lecturas adicionales

- Especificación de ejecución OCI — el estándar que implementa todo entorno de bajo nivel: el paquete, `config.json` y las operaciones del ciclo de vida.
- runc — el entorno de ejecución de la propia OCI, con notas sobre cómo lo controlan las herramientas de nivel superior.
- Entornos de ejecución de contenedores — controladores de cgroups, versiones de CRI y los ajustes de containerd y CRI-O que espera Kubernetes.
- Container Runtime Interface (CRI) — el protocolo gRPC entre el kubelet y el motor.
- Runtime Class — elegir otro manejador de ejecución por pod y mantener esos pods en nodos que lo admitan.
- Entornos de ejecución alternativos para contenedores — cómo Docker Engine entrega contenedores a gVisor, Kata Containers, youki o Wasmtime.

Glosario

Elija según sus restricciones reales

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es un entorno de ejecución de contenedores?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/container-runtime/