Análisis de integración de Docker con CI/CD

Resumen de la integración de Docker en cada fase de CI/CD

Compilación

Se define un Dockerfile con los pasos para construir una nueva imágen cada vez que un desarrollador actualiza el repositorio base. Este Dcokerfile define los entornos necesarios, dependencias, y requerimientos para la ejecusión. La imagen generada es, como resultado, un entorno consistente y portable que puede ser reusado durante las siguientes fases.

Testing

La imagen construida en la fase de compilación se usa en esta fase para conducir tests unitarios, de integración. Esto permite mayor confianza en el funcionamiento correcto de la aplicación y estabilidad luego de aplicar los cambios de código asociados con la imágen.

Deploy

Una vez que una imágen pasó exitósamente por la fase de *testing* se le asigna un identificador y se almacenan en un *container registry* como DockerHub. El *Docker registry* funciona como repositorio centrlizada en el que todas las imágenes pueden ser recuperadas para la fase de deploy.

Por último, un servicio de orquestración como Kubernets maneja el proceso *deploy* de las imágenes generadas hacia contenedores en distintos ambientes, tomando en cuenta escalación y balanceo de cargas.

Beneficios y posibles retos

Beneficios

- **Aislamiento:** En la fase de testeo, por ejemplo, gracias a la creación de imágenes, cada conjunto de tests puede ser ejecutado en su propio contenedor. Con esto se aumenta la confiabilidad del proceso de testing.
- **Consistencia entre los distintos entornos:** El uso de Docker aseguro que en las diferentes fases, compilación, tests y deploy, la aplicación corre de manera consistente.
- **Estandarización de las dependencias:** Ya que Docker encapsula todas las dependencias, se evitan problemas de compatibilidad.
- **Eficiencia:** Como se vio en clase, los contenedores de Docker usan menos recursos y son más rápidos de configurar y levantar que las máquinas virtuales. Las tres fases descritas en el laboratorio, compilación, tests y deploy, ocurrirán de manera más rápida que si se usaran otras herramientas o se mantuvieran entornos distintos para cada fase.
- **Escalabilidad y orquestración:** Como además de Docker se están usando herramientas de orquestración, los deployes se pueden adaptar fácilmente a la demanda con las herramientas ofrecidasde sacaling y load balancing – lo que permite deploys más rápidos, seguros y que no afecten al usuario final.
- **Posibilidad de revertir cambios:** Como las imágenes son almacenadas en un Docker Registry y taggeadas, es posible revertir rápidamente a una imagen anterior en caso de que se encuentren fallos en producción.

Retos/vulnerabilidades

- El costo del almacenamiento de imágenes puede acumularse y resultar muy elevado.
- Es posible que las imágenes de Docker incluyan versiones desactualizadas o vulneradas de las dependencias.
- Los Dockerfiles podrían exponer informasión sensible como llaves, credenciales o secretos.

Posibles soluciones

- Para reducir costos de almacenamiento de imágenes, primero se deberían seguir en general las sugerencias vistas en clase para la creación óptima de imágenes. Además el Docker Registry debería ser periódicamente actualizado y limpiado de versiones obsoletas.
- Además, se debe hacer Multi-Stage builds, para separar entre entornos y que las imágenes de producción sean lo más ligeras posibles. También se debe evitar instalar paquetes que no sean utilizados.
- Para evitar problemas de seguridad, también se debe implementar un mecanismo para scanear las imágenes en busca de vulneralidades en las versiones de

paquetes de las dependencias. Asímismo, solo se deben usar imágenes de terceros que sean oficiales y confiables.

- Para evitar exponer información sensible en Dockerfiles se debe usar un manejador de secretos para no hardcodearlos en los archivos.