JENKINS PARTE II

Facilitador: Daniel Ojcius

• Seguridad en Jenkins
• Jenkins + Docker
• Pipelines
• Branching Model
• Gitflow
• CI/CD
• Practica.

Slides: Seguridad en Jenkins.

Jenkins es un motor de automatización. Como tal, es alojado por alguna entidad de cómputo que, como vimos anteriormente, puede ser accedido remotamente. Esta característica hace que Jenkins pueda ser vulnerable y exponer algunas cuestiones de seguridad. Es necesario aplicar lo que se llama “Buenas Prácticas de seguridad” en Jenkins. Si miramos atentamente, no es más que una replicación de buenas prácticas para alguna entidad de cómputo, pero con algunos otros puntos que deben ser tenidos en cuenta que respectan a Jenkins. A continuación, vamos a listar alguna de estas “Buenas Prácticas”:

• Proteger Jenkins de internet
• Establecer reglas de firewall
• Poner Jenkins detrás de una vpn
• Bloquear puertos para solo acceder por vpn de la empresa.
• Agregar whitelist (github, gitlab)
• Establecer autenticaciones y autorizaciones.
• Mantener Jenkins y plugins siempre actualizados

Slides: Jenkins + Docker

Como vimos anteriormente y tal como lo aplicamos la clase pasada, pusimos nuestro entorno de Jenkins dentro de un contenedor. ¿Pero Que sucede si queremos ejecutar comandos de Jenkins en nuestros “agentes”? ¿Como conviven?
Ya dijimos que nuestros agentes deben ser capaz de soportar los comandos que se programan desde las tareas. Para ello, en caso de necesitar correr comandos de docker, debemos hacer una especie de “instalación” de docker-in-docker. Lo haremos de la siguiente manera:

1. Accedemos a la carpeta Jenkins-training --> recursos --> pipelines del repositorio https://github.com/Daniel-MundosE/Jenkins-training
2. Listamos los archivos que aparecen adentro y ahí debemos ver dos archivos: Dockerfile y Docker-compose.yaml
3. Luego dentro de este directorio creamos una carpeta con el nombre “jenkins_home” y cambiamos el owner a esa carpeta (sudo chown jenkins_home)
4. Luego corremos el comando docker-compose up –d
5. Corremos el comando “cat jenkins_home/secrets/initialAdminPassword”
6. Instalamos los plugins sugeridos.
7. Creamos un usuario y tenemos listo jenkins con docker

Slides: Pipelines

Los equipos de desarrollo deben seguir ciertas etapas para poner a disposición un producto a los clientes. La etapa comienza desde la codificación de las funcionalidades, pasando por pruebas, y luego a los usuarios. Las etapas de CI/CD se adecuan bastante bien a este marco de trabajo, por lo cual esta es la práctica fundamental para poder entregar código con alta calidad a nivel operacional. ¿Como se traduce estas etapas en DevOps? Esto lo podemos lograr a través de pipelines, y se define como un conjunto de pasos a través de los cuales el código va desde las fuentes a ser un ejecutable en algún ambiente de desarrollo.

Slides: Pipeline

Un ejemplo de pipelines es el siguiente donde hemos segmentado en tres conjuntos de pasos, pero que se traducen como etapas de CI/CD. Cabe destacar que no existe un pipeline mandatorio, sino que los pipelines se adecuan a la realidad de cada proyecto. Es decir, hay proyectos que pueden obviar test de carga, o test de disponibilidad. El pipeline que se define a continuación debe ser tomado como un ejemplo de pipeline:

1. **Descargar Repositorio - DEV**
2. **Ejecutar pruebas de análisis de código estático - DEV**
3. **Verificar compilación - DEV**
4. **Ejecutar pruebas unitarias – DEV**
5. **Ejecutar pruebas de humo - DEV**
6. **Empaquetar - DEV**
7. **Instalar en ambiente QA**
8. **Ejecutar pruebas de humo - QA**
9. **Ejecutar pruebas de regresión - QA**
10. **Ejecutar pruebas de integración - QA**
11. **Ejecutar pruebas de seguridad - QA**
12. **Ejecutar pruebas de rendimiento/carga - QA**
13. **Instalar en ambiente STAGING**
14. **Ejecutar pruebas de aceptación - STAGING**
15. **Ejecutar pruebas de seguridad - STAGING**
16. **Ejecutar pruebas de rendimiento/carga – STAGING**
17. **Instalar en ambiente de producción: PROD**

18. **Ejecutar pruebas de humo – PROD**
19. **Ejecutar pruebas de seguridad - PROD**

Slides: Branching Model

Cuando el equipo de desarrollo genera nuevas funcionalidades o repara las funcionalidades existentes debe existir algún mecanismo que permita versionar el código fuente. Para ello es necesario seguir un modelo de ramas o branching model.
Branching model es un acuerdo el cual define como se va a manejar el codigo. Define el proceso de modificaciones y las distintas versiones de codigo. Existen varios estandares o modelos, donde los mas comunes son:

• Git Flow
• Trunked Based Development

A efectos prácticos de este curso nosotros utilizaremos GitFlow.

Slides: Gitflow

Git Flow es un flujo de trabajo de Git que favorece el desarrollo continuo de software y las prácticas de implementación de DevOps [1].
Gitflow define al menos tres ramas de trabajo:

• Master: que tendrá el código oficial, el que ira a producción, es decir, el que llegara al usuario.
• Develop: tiene el código en el cual los desarrolladores usaran para introducir nuevas características.
• Feature: Nace de la rama develop y es la que contiene las nuevas funcionalidades a ser introducidas.

A estas tres ramas se pueden introducir dos ramas más. La rama hotfix que tendrá arreglos necesarios para volver a salir a producción y la rama release, que es la que contiene código para pruebas de homologación.

Slides: Continuos Integration.

Un conjunto de pasos dentro del pipeline cuyo es objetivo es garantizar que el código sea funcional u operativo con respecto a unas determinadas pautas.
• El Código compila
• El Código pasa todas las pruebas unitarias

• El Código pasa las pruebas de sintaxis o estructura del texto
• El Código pasa las pruebas de regresión
• El Código pasa las pruebas de seguridad
Por lo general a esto le decimos que el código sea integrable

Slides: Continuous Delivery.

Un conjunto de pasos dentro del pipeline cuyo es objetivo es garantizar que el código sea entregable (instalable o deployable)
• El Código pasa las pruebas de integración en staging
• El Código pasa las pruebas de rendimiento/carga en staging
• El Código pasa las pruebas de seguridad en staging
• El Código se empaqueto y tiene una versión específica

Slides: Continuous Deployment.

Un conjunto de pasos dentro del pipeline cuyo es objetivo es garantizar que el código sea deployado (instalado correctamente)
• El paquete con el código nuevo es instalado en prod
• La nueva funcionalidad pasa las pruebas de validación en prod
• La nueva funcionalidad pasa las pruebas de seguridad en prod
• La nueva funcionalidad pasa las pruebas de rendimiento/carga en prod

Practica: Pipeline.

1. Corriendo un pipeline básico.

a. Vamos al panel izquierdo -> nueva Tarea -> completamos el nombre de nuestra tarea-> elegimos la opción pipeline.
b. Vamos a la sección “Pipeline” y en script copiamos el ejemplo del repo ubicado en: https://github.com/Daniel-MundosE/Jenkins-training/blob/develop/recursos/pipelines/pipeline-basic.groovy
c. Damos en guardar y pulsamos el boton ejecutar tarea planificada.
d. Luego podemos ver la salida de nuestro pipeline

2. Corriendo un pipeline Parametrizado.

a. Vamos al panel izquierdo -> nueva Tarea -> completamos el nombre de nuestra tarea-> elegimos la opción pipeline.
b. Tildamos el check “Esta ejecución debe parametrizarse”
c. Seleccionamos tipo cadena
d. Como nombre colocamos “PROFILE” y como valor por defecto colocamos “mundose”

e. Vamos a la sección “Pipeline” y en script copiamos el ejemplo del repo ubicado en: https://github.com/Daniel-MundosE/Jenkins-training/blob/develop/recursos/pipelines/pipeline-commands.groovy
f. Luego damos click en guardar
g. Por último, presionamos en construir ahora y nos pedirá el parámetro. Le damos en el botón construir y tendremos la salida del pipeline.

3. Corriendo un pipeline con tareas paralelas.

a. Vamos al panel izquierdo -> nueva Tarea -> completamos el nombre de nuestra tarea-> elegimos la opción pipeline.
b. Vamos a la sección “Pipeline” y en script copiamos el ejemplo del repo ubicado en: https://github.com/Daniel-MundosE/Jenkins-training/blob/develop/recursos/pipelines/pipeline-parallel.groovy
c. Damos en guardar y pulsamos el botón ejecutar tarea planificada.
d. Luego podemos ver la salida de nuestro pipeline

4. Corriendo pipelines de aplicacion.

1. Instalar el plugin de Docker:
a. - Panel de control -> Administrar Jenkins -> [System Configuration] Administrar Plugins ->
b. -> click en Todos los Plugins -> buscar 'Docker pipeline' -> click en install without restart
2. Añadir las credenciales de dockerhub:
c. Panel de control -> Administrar Jenkins -> [Security] Manage Credentials ->
d. -> click en el nombre del store (jenkins) -> click global credentials ->
e. click add credentials con la siguiente configuracion:
f. - Kind: Username with password
g. - Username: Tu username de dockerhub
h. - Password: tu password
i. - Id: para este ejemplo rellenarlo con 'dockerhub_id'
j. - Description: La descripcion que tu quieras
5. Bibliografia.
[1] GitFlow Attlasian:
https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow