Práctica 4

En esta ejercitación realizaremos las configuraciones necesarias para que el servidor Jenkins pueda orquestar un build automático de la aplicación y ejecutar las pruebas unitarias desarrolladas. Este es ciclo es el más básico recomendable para comenzar a practicar Integración Continua. Es altamente recomendable que luego se genere un deploy en algún ambiente, se ejecuten analizadores de código estático para detectar su "salud" y otro tipo de métricas y análisis

1. Abrir una terminal de comandos (presionar las tecla "Windows" + r, escribir "cmd" y luego hacer clic en "Ok"). Luego de cada comando debe presionar la tecla "Enter" para que se ejecute"

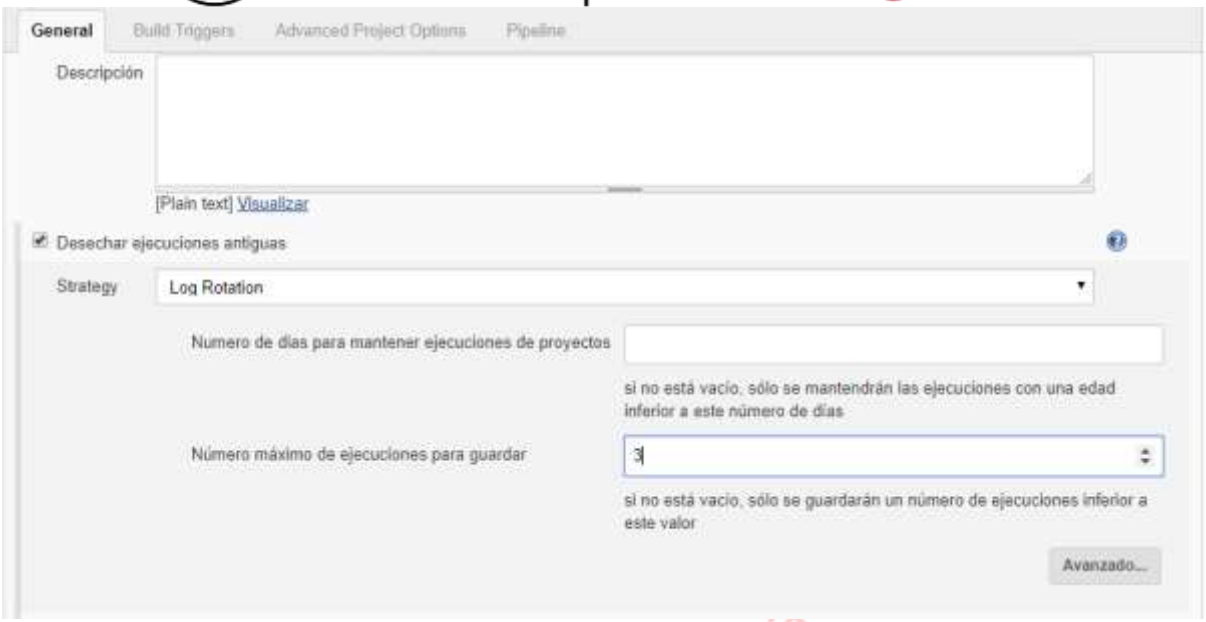
2. cd UTN-DevOps/utn-devops
3. git pull
4. git checkout unidad-4-jenkins
5. vagrant up --provision
6. set PATH=%PATH%;C:\Program Files\Git\usr\bin

7. vagrant ssh
8. Ingresar en un navegador web a: http://127.0.0.1:8082 para ingresar a Jenkins
    a. En la práctica anterior creamos el usuario admin con clave utndevops
9. sudo puppet agent -tv
    a. Creación de los certificados
10. sudo puppet cert sign utn-devops.localhost
    a. Firmo los certificados creados
11. sudo puppet agent -tv
    a. Aplica los cambios de Puppet, en este caso lo agregado es un cambio al archivo /etc/sudoers para que el usuario jenkins tenga todos los permisos
    b. Este comando suele demorar unos minutos.

12. Ingresando a la aplicación de Jenkins, vamos a hacer click en el menú lateral izquierdo "Nueva Tarea" de Jenkins

13. Ingresamos el nombre del nuevo Job que vamos a crear, “DevOpsAppTest”
    a. Hacer clic en la opción “Pipeline”

    b. Hacer clic en el botón “Ok”
14. Tildar la opción “Desechar ejecuciones antiguas”
    a. Número máximo de ejecuciones a guardar: 3



15. Tildar “Do not allow concurrent builds”
16. Tildar “GitHub Project”
    a. Project url: https://github.com/Fichen/utn-devops-app.git

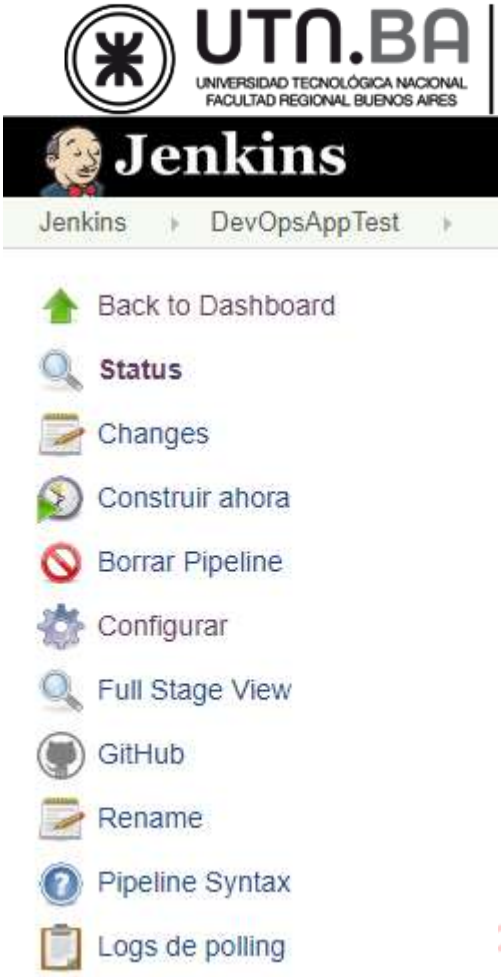
17. Tildar “Consultar repositorio (SCM)”
    a. Si bien esto no se utilizará para esta práctica es una configuración mínima que se requiere para el servidor. Lo que hace es buscar cambios en el repositorio de la aplicación. Si encuentra alguno, realiza la ejecución de la tarea. De esta manera cuando un desarrollador sube un cambio el servidor de integración continua realiza la tarea automáticamente. Hay otras opciones más performantes que funcionan para un pull-request o mediante notificación de cambios, en lugar de hacer peticiones todos los minutos al sistema de versionado.
    b. Completar con: * * * * *
        i. Hacer clic fuera del campo de texto para que detecte cambio

18. Pipeline: definition = pipeline script from SCM
    a. SCM: Git
    b. Repository URL: https://github.com/Fichen/utn-devops.git
    c. Credentials: -none-
    d. Branch Specifier: */unidad-4-jenkins
    e. Script Path: hostConfigs/jenkins/Jenkinsfile
    f. Tildar “Lightweight checkout”

19. Hacer clic en “Apply” y luego en “Guardar”

20. Ingresar a la siguiente URL http://127.0.0.1:8082/job/DevOpsAppTest/

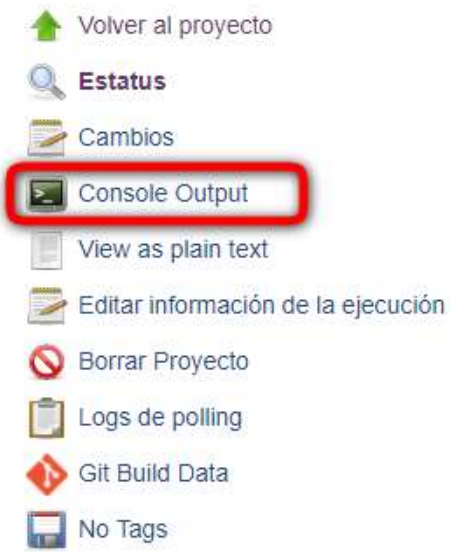
21. Hacer clic en Construir ahora



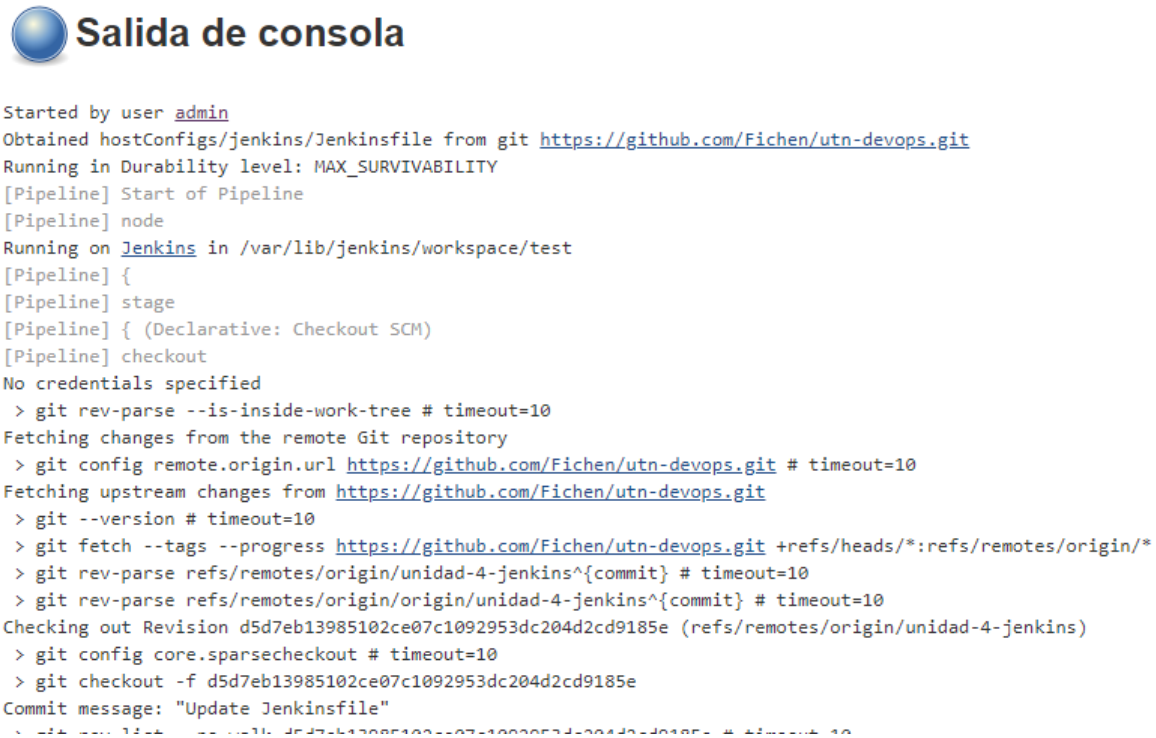
22. Esto ejecutará manualmente la tarea de Jenkins en la cual realizará un build y la ejecución de las pruebas unitarias (configuraciones declaradas en el archivo phpunit.xml que se encuentra en la raíz de la aplicación)
23. Hacer clic en la última tarea ejecutada: por ejemplo "#5"

24. Hacer clic en "console output" para verificar los registros de la ejecución del

build.



25. La salida debería mostrar un resultado similar a estas capturas



Y similar al final de la salida de la ejecución:

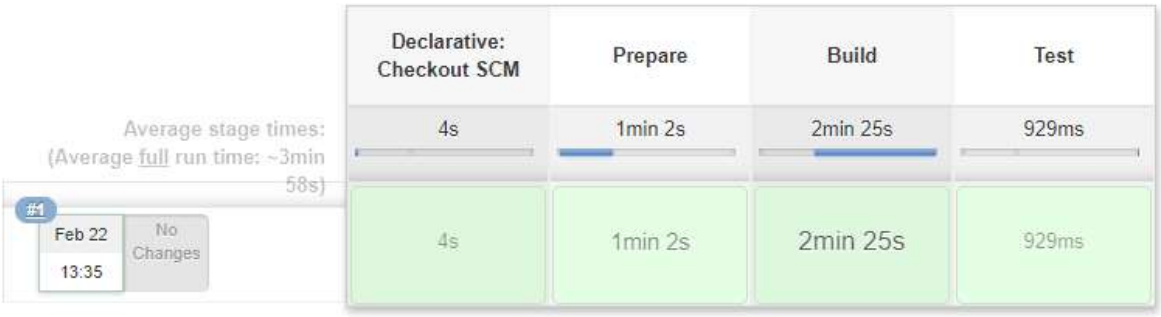
```
phpunit:
  [phpunit] PHPUnit 5.7.27 by Sebastian Bergmann and contributors.
  [phpunit]
  [phpunit] ..                                                        2 / 2 (100%)
  [phpunit]
  [phpunit] Time: 160 ms, Memory: 10.00MB
  [phpunit]
  [phpunit] OK (2 tests, 2 assertions)

quick-build:

BUILD SUCCESSFUL
Total time: 1 minute 44 seconds
Finished: SUCCESS
```

26. Ingresar a http://127.0.0.1:8082/job/DevOpsAppTest/ y verificar que el pipeline se haya ejecutado correctamente

Stage View



27. vagrant halt
    a. Solo cuando se desea finalizar la práctica
28. El entregable de la práctica es una captura de lo obtenido en el punto #25 y #26

Con estos últimos resultados hemos finalizado la práctica. Resumiendo brevemente lo que se hizo fue crear una tarea en Jenkins que verifique sólo el branch unidad-2 (esto se encuentra especificado en el archivo del pipeline) de un repositorio de código y configurar dicha tarea para que haga una análisis de sintaxis de código fuente y ejecución de pruebas unitarias al encontrar un cambio (encuentra el cambio revisando el repositorio cada minuto). Dado que no se van a realizar modificaciones en la rama de la aplicación se procedió a ejecutar manualmente la tarea para que puedan observar el resultado. Por otro lado, hemos configurado el plugin “Pipeline” de Jenkins para definir los pasos de nuestro ciclo (https://github.com/Fichen/utn-devops/blob/unidad-4-jenkins/hostConfigs/jenkins/Jenkinsfile) mediante versionado de código. Estos pasos también se verán reflejados en dashboard principal del Job “DevOpsAppTest” (http://127.0.0.1:8082/job/DevOpsAppTest/)

Lo recomendable es que luego que haya pasado exitosamente el pipeline, al menos hasta las pruebas unitarias, se realice el merge correspondiente de código. Para ello lo conveniente es realizar un análisis de la estrategia de versionado que utiliza el proyecto y determinar los pasos del pipeline. Por otro lado, por el plugin Pipeline de Jenkins ofrece la posibilidad de manejar imágenes de Docker, con lo cual lo que también recomendamos es que si tienen una arquitectura de microservicios se aproveche esta versatilidad y se realice una instancia de la aplicación en los containers desde la definición de Jenkins. Esto último lo dejamos fuera del alcance del curso por lo extenso y variabilidad de casos.

La ventaja del plugin de Jenkins es que tenemos definidos los pasos de nuestro pipeline en una herramienta de versionado (el lenguaje de programación que utiliza es Groovy), al igual que la instalación de Jenkins, obteniendo como ventaja lo siguiente:

- Se puede definir toda la configuración manual de Jenkins mediante código
- Levantar la infraestructura de la aplicación y otros ambientes como desarrollo, pruebas y Jenkins mediante infraestructura como código.
- Documentación del proyecto siempre actualizada: landscape de infraestructura, pipeline, repositorios, branches. Todo es accesible a los miembros del proyecto haciendo una mejor capitalización y distribución del conocimiento. Los desarrolladores y testers comprenden qué implica la arquitectura e infraestructura y los perfiles de operaciones y seguridad conocen más a detalle las aplicaciones que se implementan.
- Recovery: no importa que suceda, si lo que está en el código es el estado deseado del pipeline se podrá recuperar en pocos minutos/horas ejecutando

El Jenkins también posee la versatilidad de encadenar la ejecución de Jobs, por lo cual se pueden especificar complejos pipelines de integración y despliegues continuo.