Guión de prácticas

# Laboratorio 1

Vamos a realizar una reunión de retrospectiva con los alumnos para ver qué experiencias han tenido en anteriores proyectos y cómo se podría canalizar su atención hacia las nuevas prácticas que propone DevOps.

Cada alumno responderá a las siguientes cuestiones:

Presentación personal, roles realizados durante el último proyecto (o del que quiera hablar) y las tres preguntas:

* Qué fue mal, o no tan bien como se esperaba?
* Qué podía haber hecho para resolverlo?
* Por último, qué fue bien, de qué se siente orgulloso?

Esta práctica ayuda a conocer la experiencia de cada alumno, así como ayuda a exponer casos prácticos en los que un vistazo hacia atrás puede dar mucha información, no solo al que expone, sino a sus compañeros.

# Laboratorios 2

Este laboratorio, el más grande de todos, implica la creación de un toolchain típico de devops para una aplicación de ejemplo. Además lo usaremos de base para los laboratorios de los módulos siguientes.

Vamos a usar un código fuente distribuido en un fichero zip llamado myapp.zip. Desde este punto, vamos a iniciar el proceso de construcción de una cadena de montaje automatizada de integración y despliegue continuo.

1. Versionar el código
2. Integrar maven en la aplicación
3. Configurar un repositorio de artefactos
4. Configurar un gestor de tareas automáticas
5. Configurar tarea de construcción maven en Jenkins
6. Configurar tarea de tests
7. Configurar tarea de creación y despliegue de SNAPSHOTS a Nexus

Despliegue en Amazon (días 1 y 2)

1. Configurar tarea de despliegue en entorno de desarrollo
2. Configurar tarea de creación y despliegue de RELEASE CANDIDATE a Nexus
3. Configurar tarea de despliegue en entorno de qa
4. Configurar tarea de ejecución de tests
5. Configurar Acción Manual Aceptación de RELEASE CANDIDATE
6. Configurar tarea de despliegue en entorno de staging
7. Configurar tarea de testing continuo en staging
8. Configurar monitorización de staging y producción
9. Configurar tarea de Aceptación de RELEASE
10. Configurar tarea de despliegue en producción

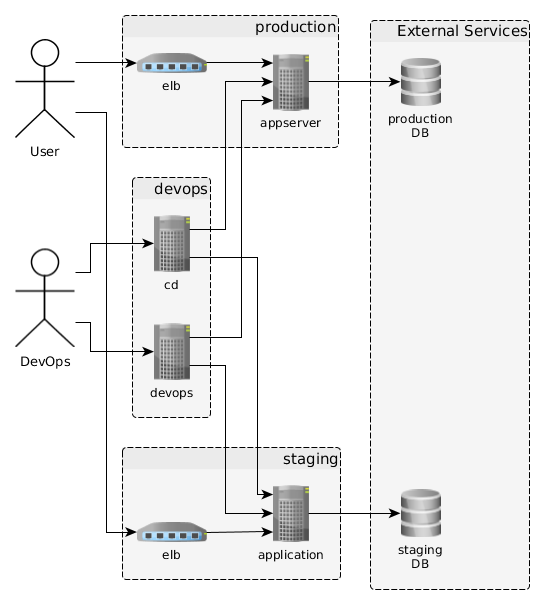
Despliegue en Heroku (día 2)

1. Configurar aplicación staging
2. Configurar aplicación producción
3. Configurar pipeline

Despliegue en Bluemix (día 3)

1. Configurar aplicación producción

Diagrama del Entorno de red de Amazon



Dominio externo: examplecorp.es

Máquina de acceso: devops.examplecorp.es

Máquina de CI/CD: cd.examplecorp.es

Usuario de acceso: studentXX

Contraseña de acceso: cangetin

Dirección de correo electrónico: [studentXX@examplecorp.es](mailto:studentXX@examplecorp.es) (asignados por el instructor)

Creación de un par de claves SSH

Para facilitar la interacción entre algunos servicios, vamos a generar un par de claves de usuario.

Para ello, desde la interfaz de administración de Amazon, el instructor nos proporcionará un fichero .pem con la clave privada correspondiente a Amazon. Con el siguiente comando, generaremos una clave pública para usar en algunos otros servicios:

$ ssh-keygen -y -f studentXX.pem

generará una salida similar a esta:

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCf5JaoUONmqawopEJ9zjq9GHMxM8i5fisj/aKM1AWT/CSuidZDXUtDU4sDU9qXRbyWKcHHytVC0Pl2exQU8YcF7Qg5+BgIshyvHTrMff7M5GLToddPqOiD6Yuguvec+6SbFD14n8JeLl+/6KWEQX437R41BpqKmqF+NW++hxDVr1luVdb4GFAN+F//if5lW1nwyHsb6p672ULhj1Vy01vG0EIZvELQZx+LUcn7b99pomksStvPqxpKNE5XDGHWG5CkjucaX6YbQezyBsP7cuX1hjgmB8dDGjGtNNnfdo5k97Hegzc4psu1myqzP4JRUJDHteEN1a47x60oHRyCxZTB

Este texto lo podremos copiar y pegar más adelante para vincular la clave ssh a un servicio

### Versionar el código de la aplicación

Dirección de código fuente: lab02/myapp.zip

Configuración de la identidad en git

Ejecutamos los siguientes comandos para establecer nuestra identidad en git. Esto sirve para identificar quién ha realizado un cambio en el código.

$ git config --global user.name "Student XX"  
$ git config --global user.email [studentXX@examplecorp.es](mailto:studentXX@examplecorp.es)

NOTA: puedes ampliar la configuración de tus datos de identidad estableciendo tu editor principal, y otros parámetros. Consulta la web de git[[1]](#footnote-0) para más información.

1.1. Descarga del código fuente

Descargamos el código fuente de la aplicación al directorio local del usuario student, y lo descomprimimos con la utilidad unzip:

$ unzip ~/Downloads/myapp.zip

Genera el directorio myapp. Entramos dentro del directorio e inicializamos el repositorio git:

$ cd myapp

$

$ git init .

Añadimos todos los ficheros al repositorio.

$ git add -A

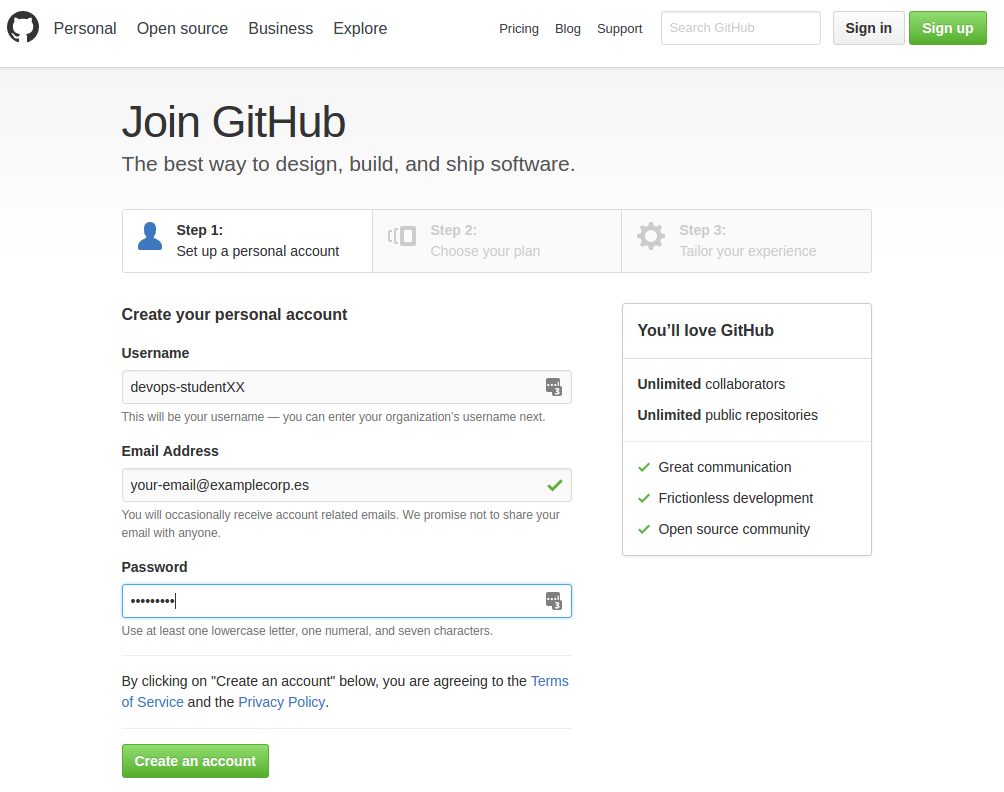
Generamos el primer commit.

$ git commit -m “initial commit”

1.2. Configuración del repositorio git

Creamos una cuenta de usuario en github.com, o si ya la tenemos, usamos la que ya tengamos y saltamos los siguientes pasos:

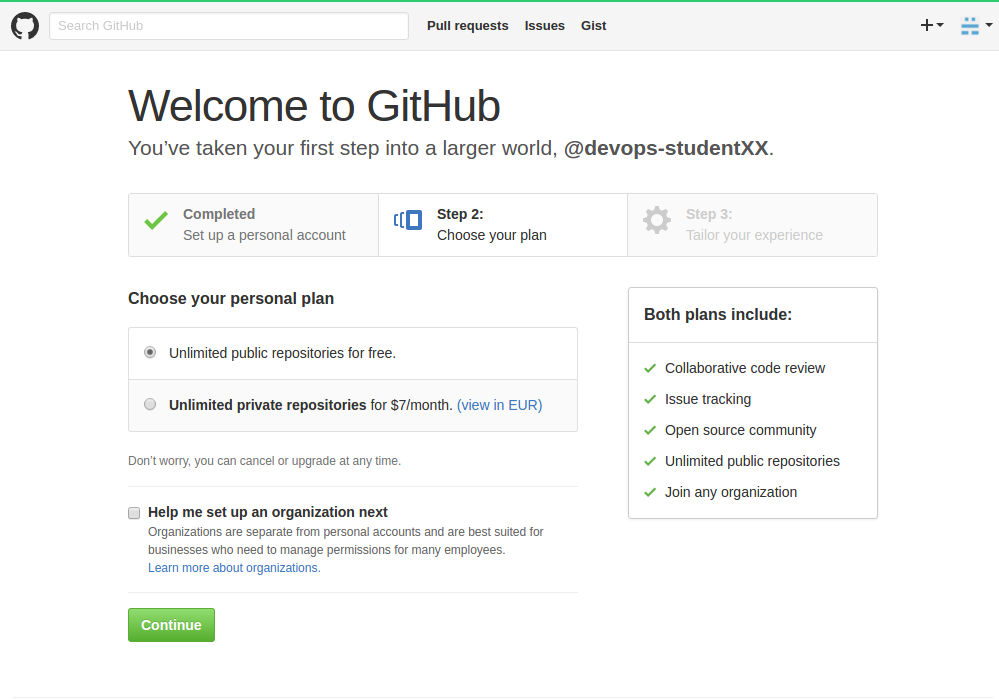
1.2.1. Creación del usuario



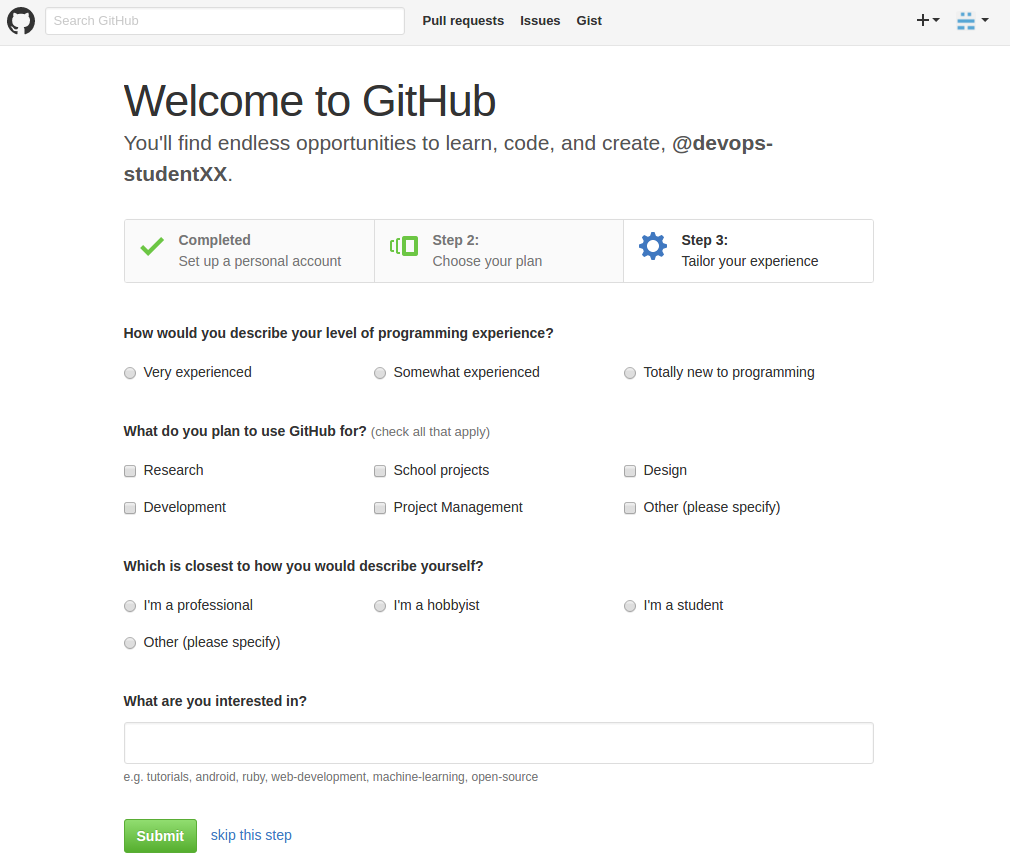
Nombre de usuario: devops-studentXX (desde 01 en adelante)

Dirección de correo: debe ser válida, ya que envían un correo de validación

Contraseña: cangetin123



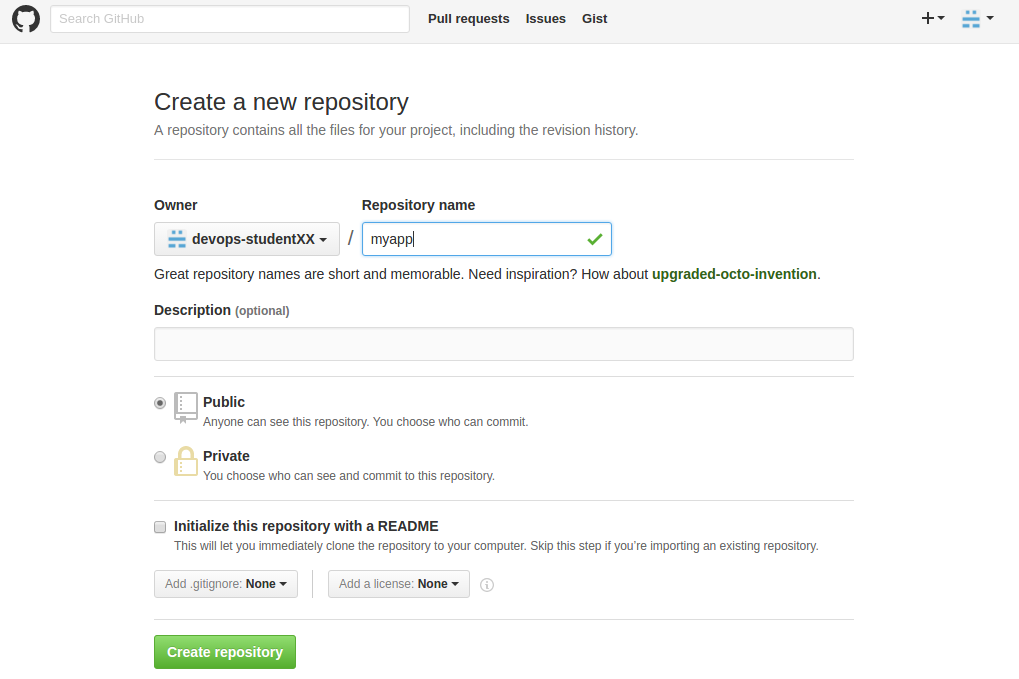
Señalamos la opción “Unlimited public repositories for free. Siguiente

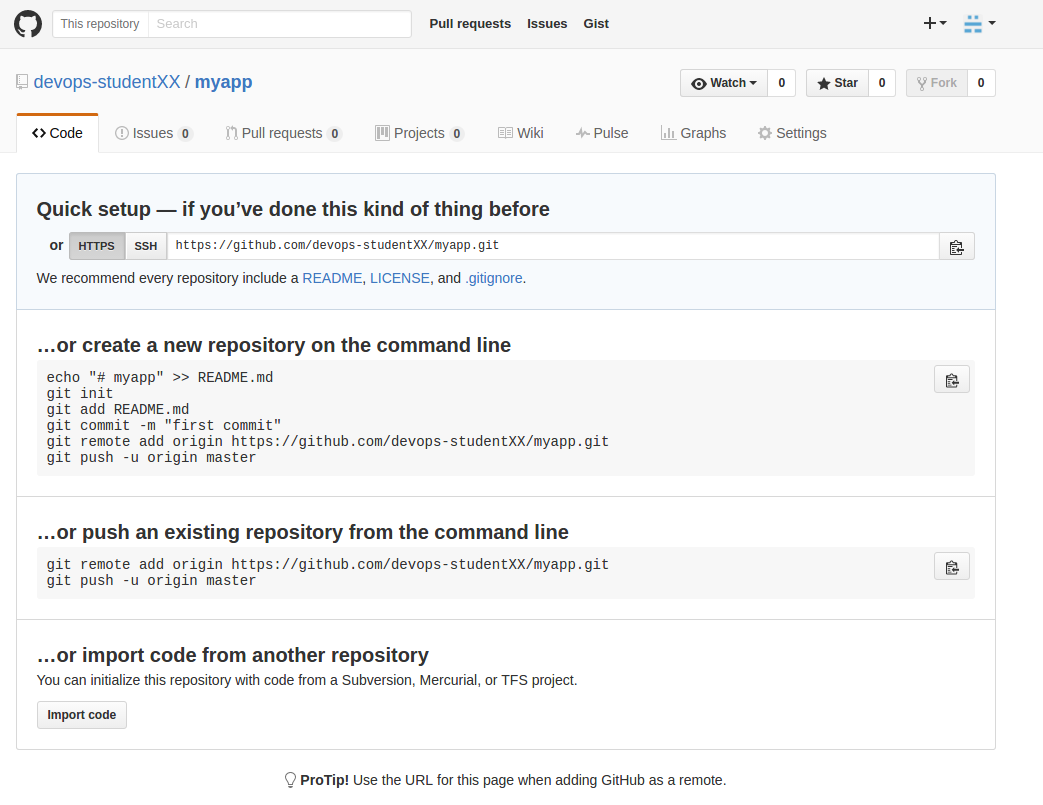


Pulsamos en “skip this step”.

Enhorabuena, ya tienes una cuenta en github.com. No te olvides de validad la dirección de tu correo electrónico para que puedas continuar creando repositorios.

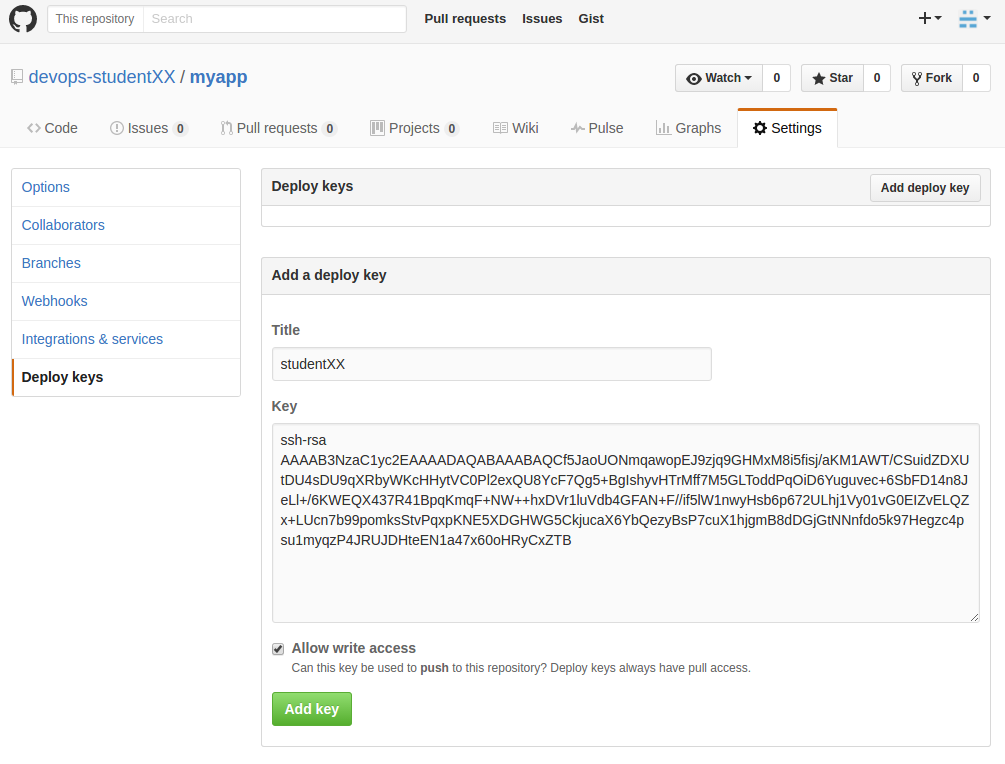
1.3. Creación del repositorio en github

Rellenamos el nombre del repositorio y lo establecemos como público.



Configuración del par de claves SSH para acceso al repositorio

En el repositorio recién creado, en el menú superior se puede acceder al submenú Settings. Seleccionamos la opción “Deploy keys”



Introducimos la clave pública generada para nuestra clave privada de amazon. **Marcar “allow write access”**

Usamos estas recomendaciones para subir nuestro código a github.

$ git remote add origin git@github.com/devops-studentXX/myapp.git

$ git push -u origin master

Con esto subiremos el código a nuestra nueva cuenta de github.com.

Con esto terminamos la primera parte del versionado de código. Cada vez que modifiquemos el código, realizaremos la siguiente secuencia de pasos para subir el commit a github:

$ git pull origin master # conveniente para evitar conflictos

$ git add fichero/s

$ git commit -m “comentario”

$ git push origin master

### Integrar maven en la aplicación

Para refrescar los conocimientos de maven, consulta la siguiente [guía rápida de maven](https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html).

Verificamos que maven funciona en nuestra cuenta ejecutando el comando mvn

$ mvn

NOTA: Si devuelve error de JAVA\_HOME, agregar la siguiente línea a nuestro fichero de configuración de la shell (~/.bashrc o ~/.bash\_profile):

export JAVA\_HOME=/usr

Y reabrir la sesión para aplicar los cambios (o ejecutar source ~/.bashrc)

Ejecutamos el comando mvn archetype:generate para crear nuestra aplicación:

$ mvn archetype:generate -DgroupId=com.examplecorp.studentXX -DartifactId=myapp -DarchetypeArtifactId=maven-archetype-webapp

Con este comando se genera una estructura de ficheros similar a la de la aplicación web original, y un fichero pom.xml.

Copiamos el fichero poml.xml en nuestro directorio de la aplicación web.

Ejecutamos:

$ mvn clean process-resources

Conectará a Internet y se bajará las dependencias.

Observa que descarga muchos ficheros desde lugares externos de la red. Maven siempre utiliza repositorios remotos para descargarse componentes.

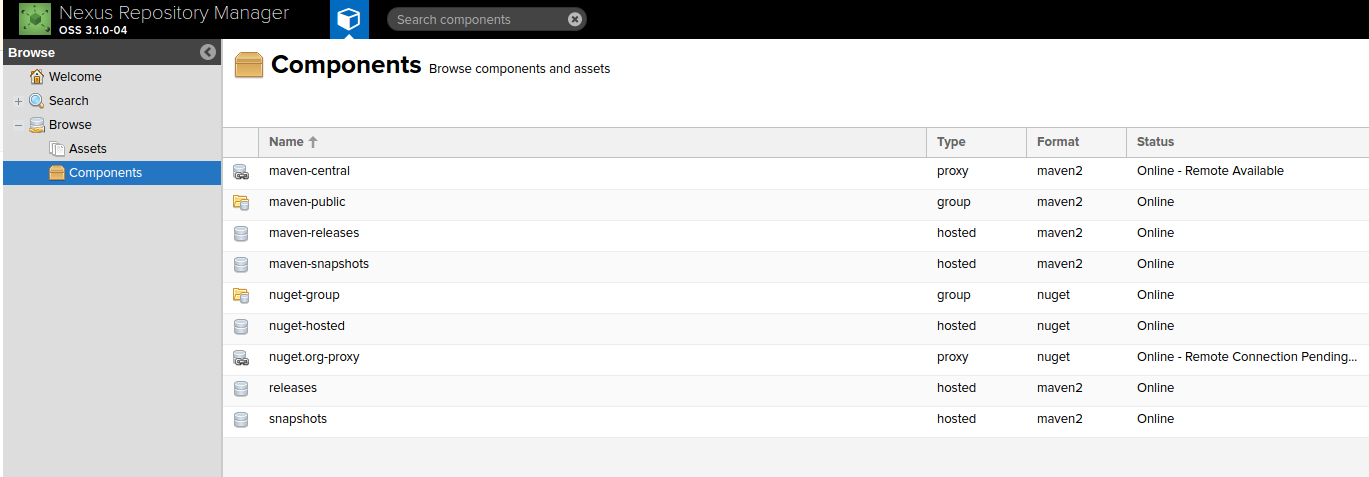
### Configurar un repositorio de artefactos

El instructor ha configurado un servidor nexus en la red para guardar y desplegar los artefactos que se generan durante la construcción de software. En este ejercicio vamos a configurar maven para que pueda guardar los artefactos en la rama correspondiente.

Nexus URL: <http://nexus.examplecorp.es:8081>

User/Password: devops / devops

En nexus, tendremos dos ramas: snapshots y releases.



* snapshots: son las construcciones incrementales de software. A cada commit, se construye una nueva revisión del software.
* releases: son las construcciones “liberadas” del software. Constituyen las release candidate o las versiones finales del software y se consideran como versiones independientes.

A medida que se generan modificaciones en el código, la CI genera y testea una revisión de la versión de código actual. Esa es la snapshot. Se puede desplegar y puede ser usada por otros para seguir desarrollando código. Pero no es la versión acabada del módulo. Todas las versiones snapshot comparten la misma versión, pero se diferencian en un timestamp, de forma que siempre se pueda seleccionar la última o más reciente.

El objetivo de esta práctica es configurar maven para que los artefactos se guarden en un repositorio después de cada construcción. Esto permite a otros usuarios compartir y acceder a módulos o dependencias necesarias para la realización de su propio trabajo.

Cada usuario de maven tiene un directorio .m2 en su directorio local, por ejemplo ~/.m2. En él se encuentra el directorio repository, con las dependencias descargadas por maven en construcciones anteriores.

El fichero settings.xml permite configurar parámetros de maven y ajustarlos a cada usuario. En nuestro caso, vamos a crear el fichero con el siguiente contenido:

|  |
| --- |
| <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0  https://maven.apache.org/xsd/settings-1.0.0.xsd">  <servers>  <server>  <id>nexus</id>  <username>devops</username>  <password>devops</password>  </server>  </servers>  <mirrors>  <mirror>  <id>nexus</id>  <name>nexus local</name>  <url>http://repo.examplecorp.es:8081/repository/maven-public/</url>  <mirrorOf>\*</mirrorOf>  </mirror>  </mirrors>  <proxies/>  <profiles/>  <activeProfiles/>  </settings> |

Con esta configuración estamos haciendo dos cosas:

1. definir un usuario y contraseña para un servidor llamado nexus
2. usar el servidor nexus como mirror local de todos los repositorios de maven remotos

Maven es popular por la cantidad de dependencias que llega a descargar para realizar su trabajo. Es común la observación “se está descargando medio internet”. La configuración mirror permite usar un servidor local de Nexus OSS como proxy y caché de los repositorios maven. De esta forma optimizamos la velocidad de descargas necesarias por maven en entornos de múltiples usuarios y/o construcciones concurrentes.

En el pom.xml también tendremos que realizar modificaciones para indicar dónde queremos guardar los artefactos que generamos.

Snapshots

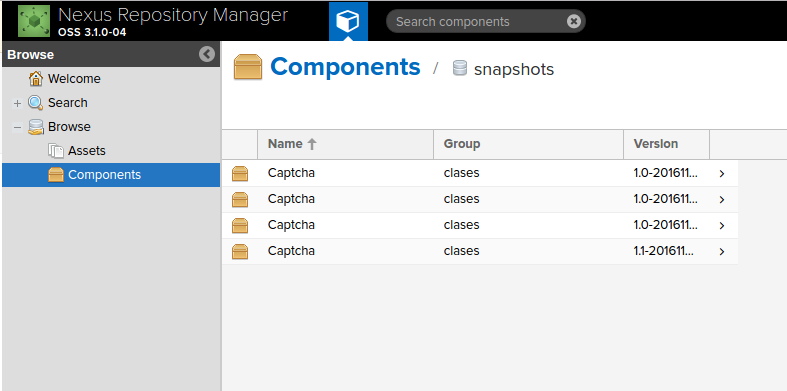
Como hemos dicho previamente, las snapshots son piezas incrementales del código que podemos compartir con otros para que puedan realizar su trabajo. Cada snapshot comparte la misma versión, pero son revisiones incrementales del código.

|  |
| --- |
| <distributionManagement>  <repository>  <id>nexus</id>  <name>Releases</name>  <url>http://repo.examplecorp.es:8081/repository/releases/</url>  </repository>  <snapshotRepository>  <id>nexus</id>  <name>Snapshots</name>  <url>http://repo.examplecorp.es:8081/repository/snapshots/</url>  </snapshotRepository>  </distributionManagement> |

Con esto, cuando ejecutemos la fase de despliegue en maven (mvn deploy), los artefactos generados en formato SNAPSHOT serán enviados al repositorio de snapshots.

$ mvn deploy

Generará una nueva revisión de nuestro software en el repo snapshots en nexus:



Releases

Por último, cuando se cierra el desarrollo de una versión de código, generamos una release (en terminología maven). Esta versión se supone estable y sin fallos y se puede “liberar” para su uso.

Para realizar la subida a un repositorio remoto, necesitamos modificar nuestro fichero pom.xml para activar un plugin llamado maven-release-plugin.

|  |
| --- |
| <build>  ...  <plugins>  ...  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-release-plugin</artifactId>  <version>2.5.3</version>  </plugin>  </plugins>  </build> |

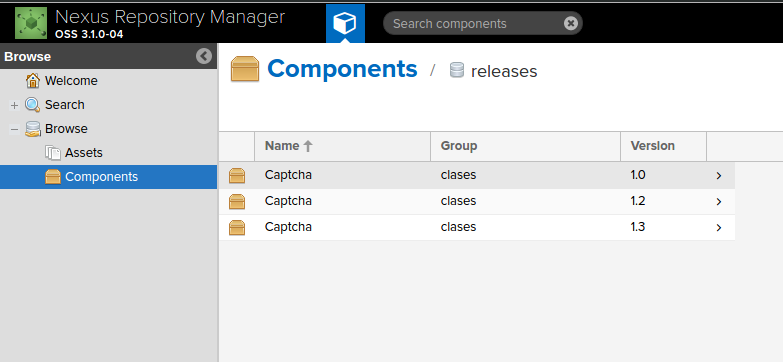
Esta configuración nos habilita otro goal: release, que consta de dos fases:

|  |
| --- |
| $ mvn release:prepare  …  What is the release version for "MyApp Maven Webapp"? (clases:MyApp) 1.3: :  What is SCM release tag or label for "MyApp Maven Webapp"? (clases:MyApp) MyApp.3: :  What is the new development version for "MyApp Maven Webapp"? (clases:MyApp) 1.4-SNAPSHOT: :  … |

Este comando nos pregunta cual es la versión que vamos a generar ahora, la etiqueta con la que marcaremos esta versión en el repositorio de código y cuál será la nueva versión en desarrollo (para los SNAPSHOTS).

$ mvn release:perform

Por lo tanto subimos este artefacto a una rama especial llamada releases, desde la cual, los servicios de despliegue pueden descargar el código e instalarlo en los sistemas.



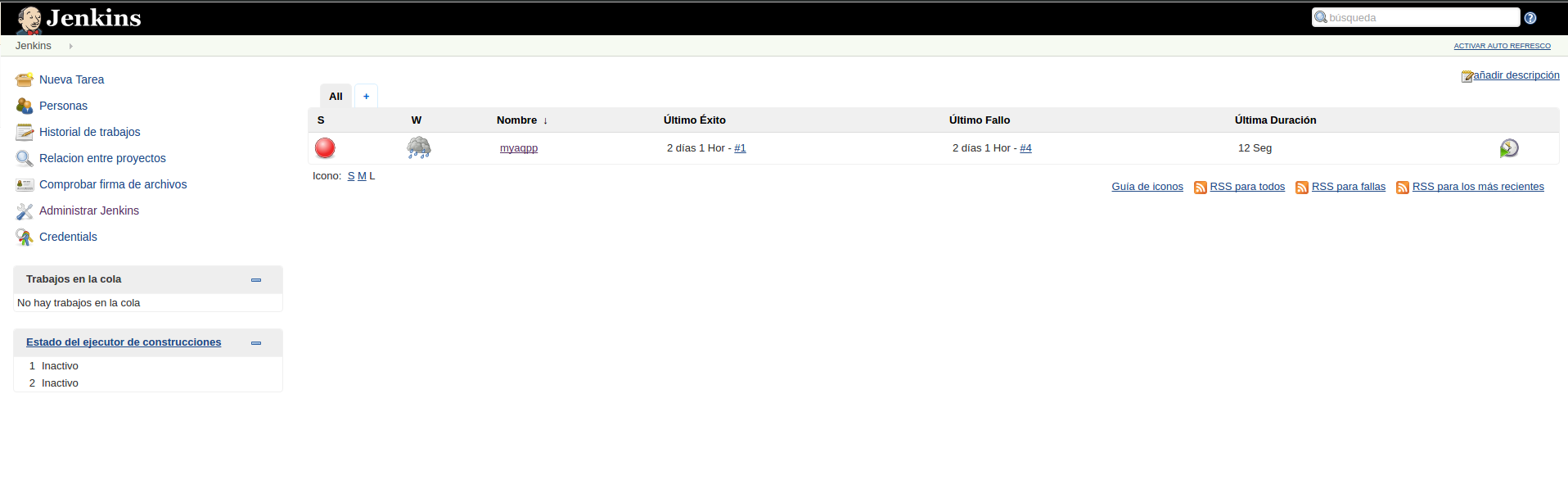
### Configurar un gestor de tareas automáticas

Empezamos a trabajar ahora con Jenkins. Este es un servicio muy popular de Automatización de tareas por ser de código abierto y muy modular, con múltiples plugins que ayudan a facilitar la integración con varios sistemas de desarrollo y despliegue.

El instructor nos dará acceso al servidor de integración continua

URL: <http://ci.examplecorp.es:8080/jenkins>

User / Password: devops / devops



El instructor mostrará las opciones de navegación de jenkins y cómo instalar plugins.

### Nomenclatura para Jenkins

Las tareas en jenkins se ordenan únicamente por el nombre de la tarea. Por lo tanto, si queremos mantener un orden, es preciso utilizar una nomenclatura precisa:

Ejemplo: **studentXX-myapp-00-build-development**

Donde:

Prefijo: studentXX-<app>

Orden: 00-99

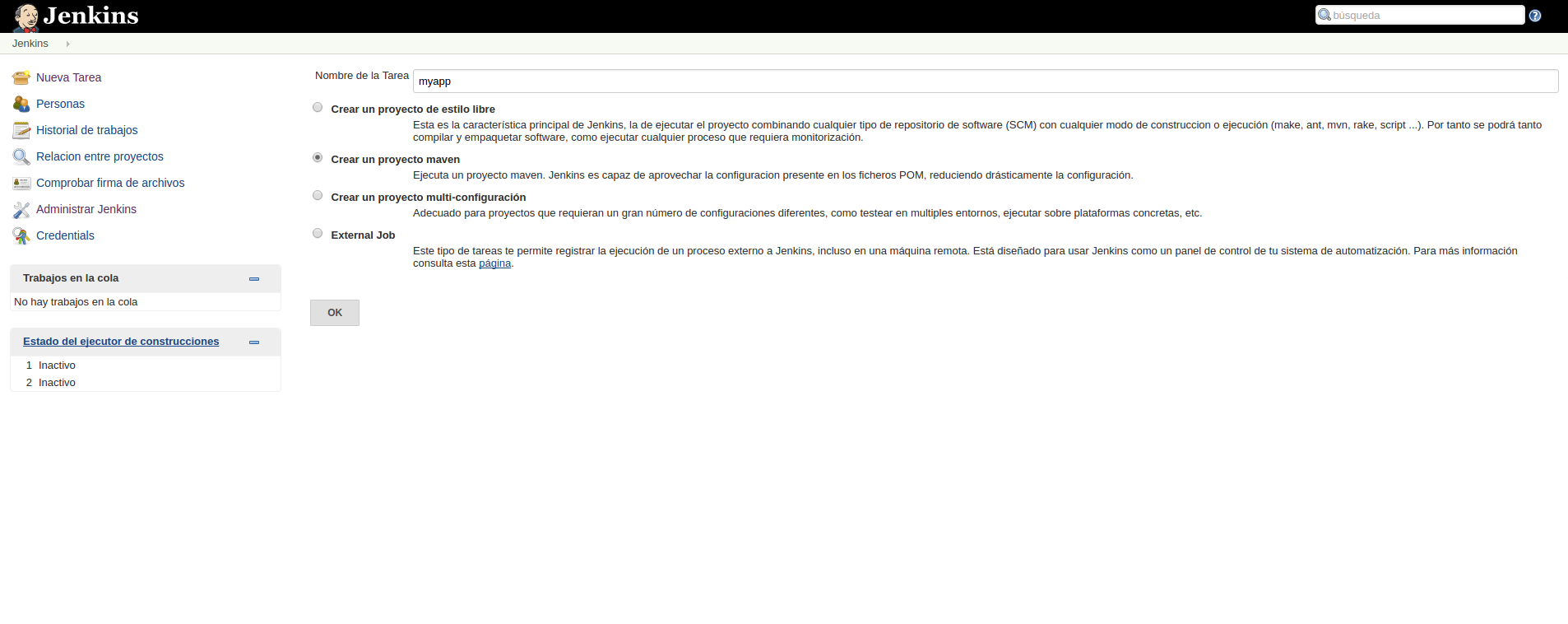
Nombre Tarea: nombre de la tarea

Sufijo: el entorno en que se realiza la tarea

### Configurar tarea de construcción maven en Jenkins

Vamos a crear una tarea de construcción de maven en jenkins.

Hacemos click en “crea una nueva tarea”.



Nombre de la tarea: studentXX-myapp-01-build-development.

Selecciona “crear un proyecto maven”.

Pulsa Ok.

La siguiente pantalla está llena de opciones que el instructor irá comentando para aclarar las definiciones.

Configurar el origen del código fuente

Selecciona GIT

Introduce la URL de github usando HTTPS: <https://github.com/devops-studentXX/myapp.git>

Pulsa en el botón add para añadir credenciales de acceso al repositorio.

Rellena los siguientes campos del formulario:

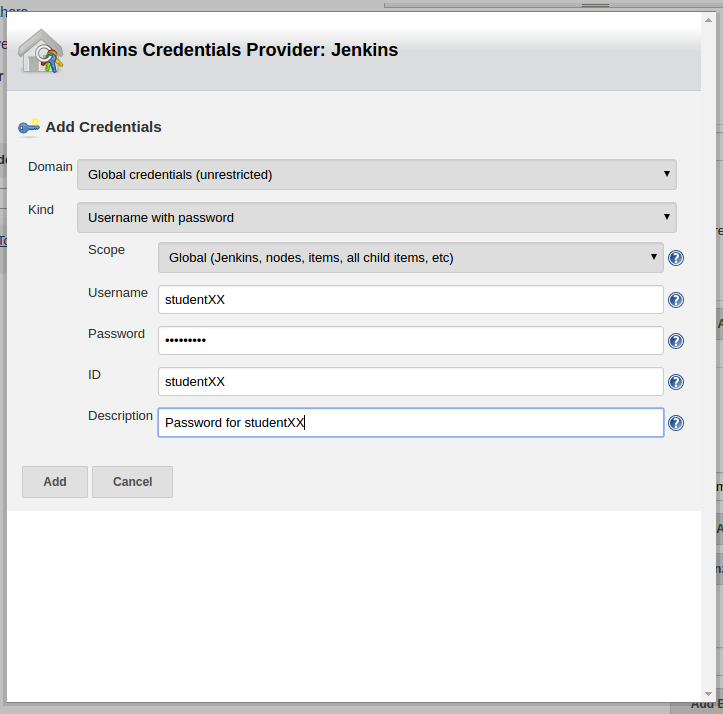
Kind: Username with password

Username: studentXX

Password: cangetin

ID: studentXX

Pulsa Add para añadir las nuevas credenciales.



En el formulario de configuración de la tarea, selecciona las credenciales que acabamos de crear: studentXX/\*\*\*\*\*\*

Disparadores de ejecuciones

Aquí hay muchas opciones de ejecución, pero vamos a dejar de momento la de por defecto y también “Build when a change is pushed to Github”. Esto permitirá activar la tarea cada vez que un commit sea subido a Github. Más adelante se realizará una configuración para esto.

Entorno de ejecución

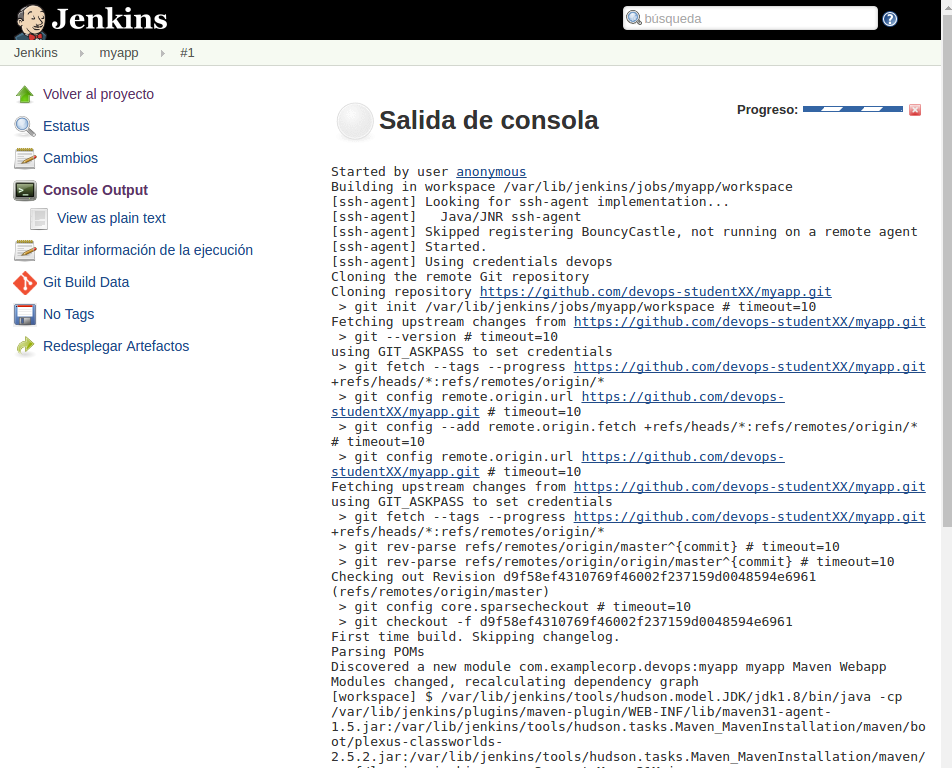
Selecciona la opción Maven release build

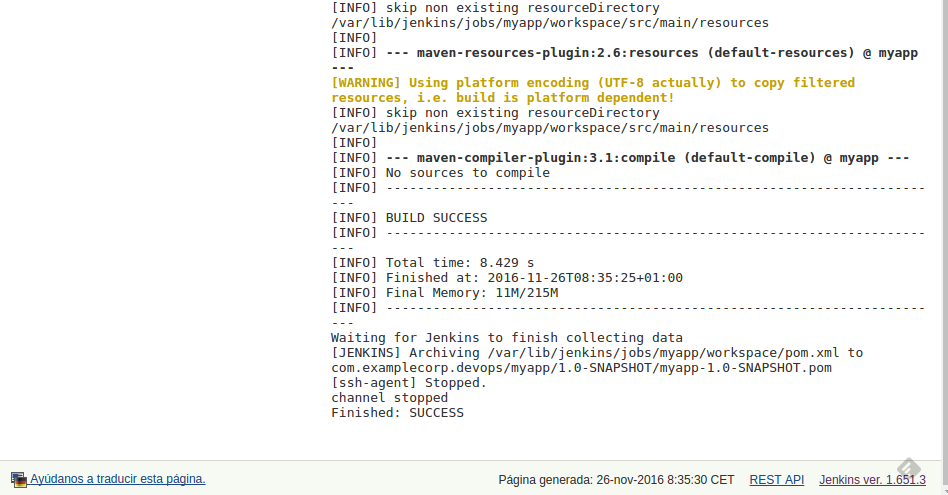
Proyecto

Goles y opciones: clean process-resources compile

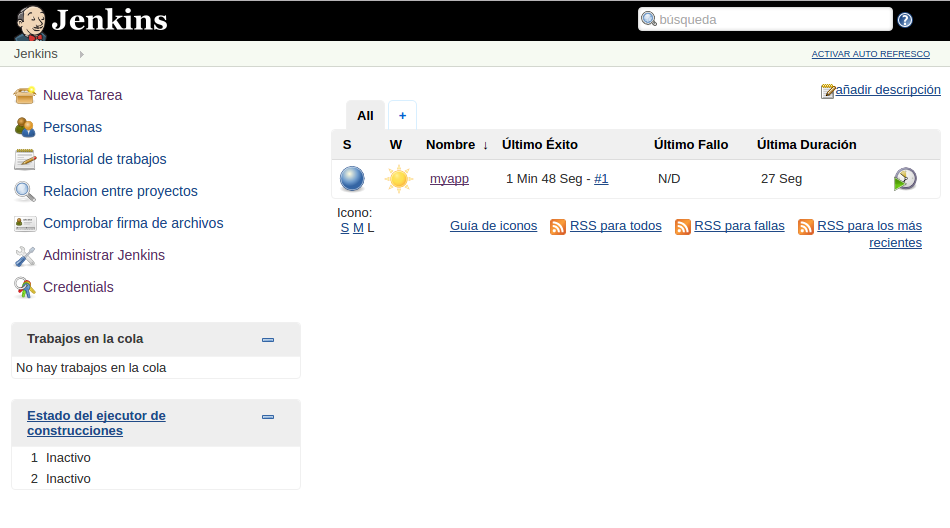
Guardar

Con esto, la tarea está configurada. Ahora podemos pulsar en “Construir ahora” para ejecutarla.





Si la ejecución termina con éxito, el panel principal de jenkins tendrá una tarea con una bolita azul y un sol.



### Configurar tarea de tests

Vamos a realizar los mismos pasos, pero ahora vamos a configurar un test para que se ejecute cada vez que se construye el software.

Creamos una nueva tarea, le ponemos

Nombre: studentXX-myapp-02-test-development

Seleccionamos la opción: Copiar una Tarea existente

Pulsamos OK.

En disparadores de ejecuciones, selecciona:

“Build after other projects are built:

Projects to watch: studentXX-myapp-01-build-development

Selecciona: Trigger only if build is stable.

En la opción de proyecto, establece el nuevo gol: test

Guarda los cambios.

Pulsa en Ejecutar la tarea studentXX-myapp-01-build-development.

Verás que ambas tareas se ejecutan en orden correctamente.

### Configurar tarea de creación y despliegue de SNAPSHOTS a Nexus

Seguimos los pasos que realizamos para crear una tarea de tests, pero esta vez para crear una tarea cuyo goal sea deploy, y dependa del proyecto myapp-test, con la misma condición de que se ejecute sólo si la tarea anterior se ha ejecutado correctamente.

Nombre studentXX-myapp-03-snapshot-development

“Build after other projects are built:

Projects to watch: studentXX-myapp-02-test-development

Selecciona: Trigger only if build is stable.

Proyecto:

Goal: deploy

Guardar.

Ejecuta la tarea de construcción de studentXX-myapp-01-build-development y observa cómo se van ejecutando el resto de tareas.

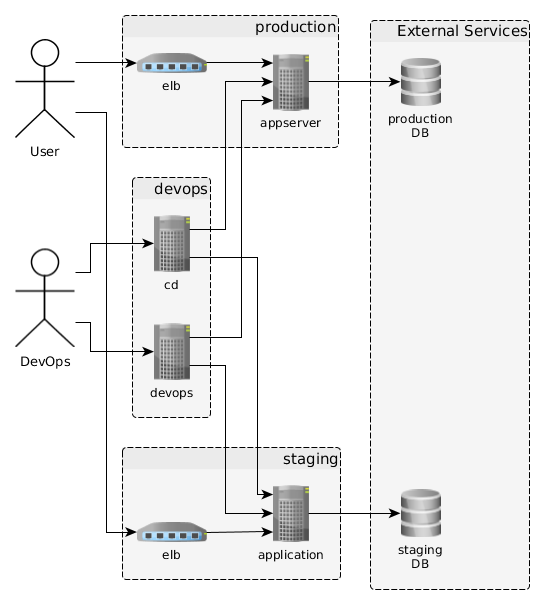
Verifica en nexus, en el repositorio snapshot, que se ha subido el artefacto generado.

URL: http://repo.examplecorp.es:8081/repository/snapshots/

## Prácticas con AWS y terraform

Es un conjunto de prácticas en las cuales usaremos la plataforma AWS para desplegar instancias de máquinas y crear infraestructuras complejas usando software definido por código, y manejando mediante versionado la evolución de la infraestructura.

Diagrama del Entorno de red de Amazon



### Configurar tarea de despliegue en entorno de desarrollo

Vamos a usar [terraform](https://www.terraform.io/) para la creación, mantenimiento y despliegue de la infraestructura y el código.

Para ello, accedemos al directorio lab02/terraform/dev y revisamos los ficheros que tenemos:

|  |
| --- |
| ├── aws.tf  ├── go.sh  ├── private.tf  ├── public.tf  ├── terraform.tfvars  ├── variables.tf  └── vpc.tf |

El fichero terraform.tfvars contiene las variables y personalización para el entorno de desarrollo. Tenemos que modificarlo para rellenar los datos que nos hacen falta:

|  |
| --- |
| aws\_access\_key = "AKIAJOAXI7UDMYFLXXXX"  aws\_secret\_key = "HiDh3wIyeWViIuTAZFO9xXXXX"  aws\_key\_path = "~/.aws/studentXX.pem"  aws\_key\_name = "studentXX"  aws\_environment = "development" |

Se ha añadido un fichero que permite instalar en la fase de ejecución la instalación de puppet y la configuración de la instancia:

|  |
| --- |
| #!/bin/bash  sudo apt-get -yy update  sudo apt-get -yy install git puppet  git clone <https://github.com/devops-studentXX/puppet.git>  cd puppet  sudo puppet apply init.pp --modulepath modules |

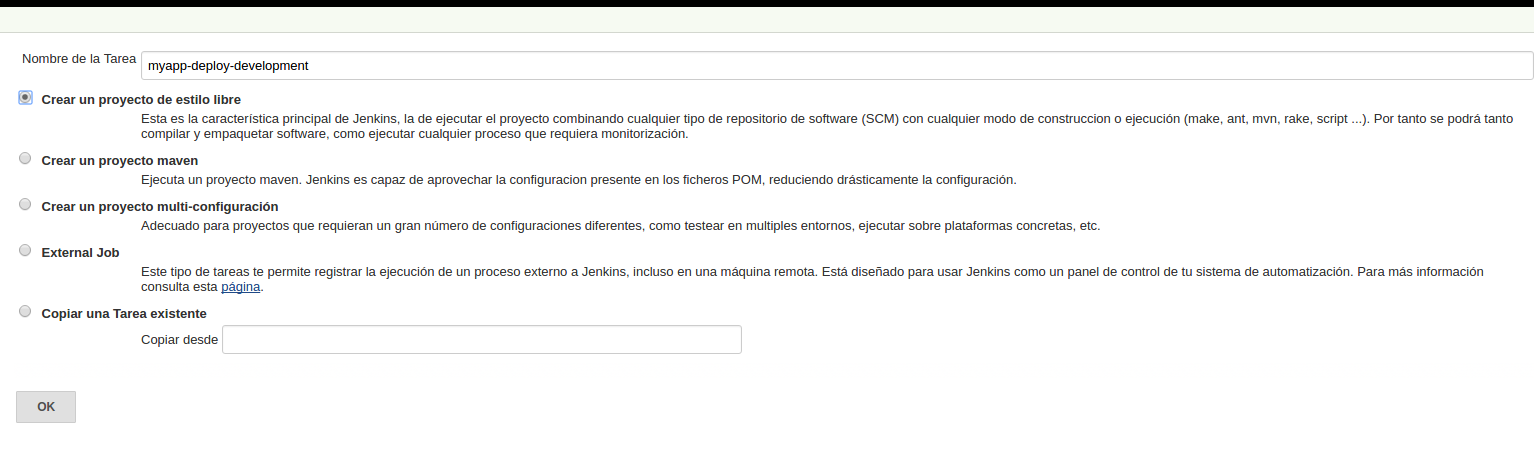
Este script es sólo un ejemplo para el desarrollo de nuestras prácticas. Existen múltiples maneras de inicializar una instancia con puppet.

##### Crear tarea en Jenkins

Creamos una tarea en jenkins con los siguientes pasos:

Nombre: studentXX-myapp-04-deploy-development

Tipo de proyecto: Crear un proyecto de estilo libre



Pulsamos Ok.

Configuramos el origen del código fuente:

Github: <https://github.com/devops-studentXX/lab02>

Establecemos como disparador, la tarea de despliegue del SNAPSHOT

Build after other projects are built: studentXX-myapp-03-snapshot-development

Ejecución

Creamos una tarea de tipo: ejecutar línea de comandos (shell)

Tenemos que inyectar las variables necesarias para que terraform pueda funcionar:

|  |
| --- |
| # Vamos a configurar las variables de entorno necesarias para que terraform pueda actuar  export TF\_VAR\_aws\_access\_key="XXXX"  export TF\_VAR\_aws\_secret\_key="XXXX"  export TF\_VAR\_aws\_key\_path="~/.aws/studentXX.pem"  export TF\_VAR\_aws\_key\_name="studentXX.pem"  export TF\_VAR\_aws\_environment="development"  # accedemos al directorio dev  cd terraform/aws/dev  # ejecutamos el comando  terraform apply |

Guardamos la tarea y comprobamos que podemos realizar todos los pasos desde jenkins.

Con esto tenemos terminado el proceso de despliegue en un entorno de desarrollo.

### Configurar tarea de creación y despliegue de RELEASE CANDIDATE a Nexus

Una vez se alcanza el nivel de desarrollo y test adecuado, se congela el desarrollo y se genera una release de maven.

NOTA: Maven no tiene concepto de RELEASE CANDIDATE, sólo de release.

Se llama **Release Candidate** a la **release generada por el equipo de desarrollo** mientras está en observación por el equipo de calidad. **La release candidate se convierte en release cuando el equipo de calidad le da el visto bueno**. Entonces está lista para ser desplegada en los siguientes entornos.

Creamos la tarea en Jenkins

Nombre studentXX-myapp-05-release-candidate-development

“Build after other projects are built:

Proyecto:

Goles y opciones: --batch-mode release:prepare release:perform

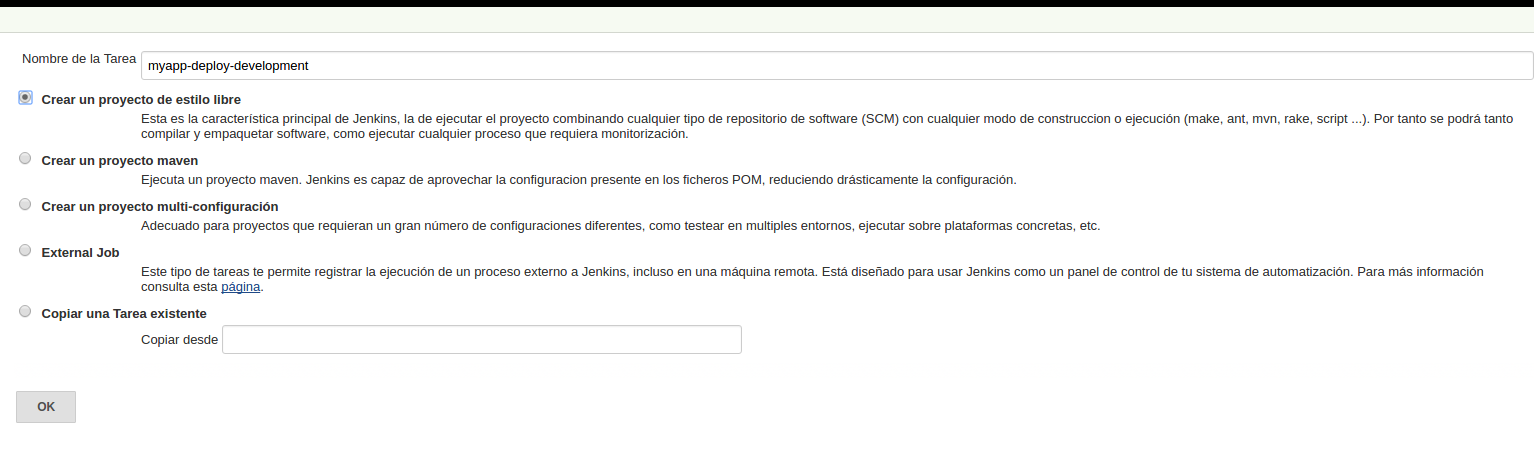
Guardar.

### Configurar tarea de despliegue en entorno de qa

Creamos una tarea en jenkins con los siguientes pasos:

Nombre: studentXX-myapp-06-deploy-qa

Tipo de proyecto: Crear un proyecto de estilo libre



Pulsamos Ok.

Configuramos el origen del código fuente:

Github: <https://github.com/devops-studentXX/lab02>

Establecemos como disparador, la tarea de despliegue del SNAPSHOT

Build after other projects are built: studentXX-myapp-05-release-candidate-development

Ejecución

Creamos una tarea de tipo: ejecutar línea de comandos (shell)

Tenemos que inyectar las variables necesarias para que terraform pueda funcionar:

|  |
| --- |
| # Vamos a configurar las variables de entorno necesarias para que terraform pueda actuar  export TF\_VAR\_aws\_access\_key="XXXX"  export TF\_VAR\_aws\_secret\_key="XXXX"  export TF\_VAR\_aws\_key\_path="~/.aws/studentXX.pem"  export TF\_VAR\_aws\_key\_name="studentXX.pem"  export TF\_VAR\_aws\_environment="qa"  # accedemos al directorio qa  cd terraform/aws/qa  # ejecutamos el comando  terraform apply |

Guardamos la tarea y comprobamos que podemos realizar todos los pasos desde jenkins.

Con esto tenemos terminado el proceso de despliegue en un entorno de qa.

### Configurar tarea de ejecución de tests

Creamos ahora una tarea de jenkins que permita lanzar automáticamente una batería de tests de QA usando el programa soapUI.

Crear tarea

Nombre: studentXX-myapp-07-test-qa

Tipo de tarea: maven

Origen

Github: <https://github.com/devops-studentXX/tests>

Build after other projects are built: studentXX-myapp-06-deploy-qa

Ejecución:

Goal: test

### Configurar Acción Manual Aceptación de RELEASE CANDIDATE

En ocasiones, es necesario establecer un punto de interrupción del procedimiento de despliegue continuo. Algunas organizaciones necesitan que alguien se responsabilice de los cambios, por lo que se crean tareas manuales que obliguen a alguien a ejecutarlas para así registrar las autorizaciones.

Crea una nueva tarea en jenkins que permita interrumpir el despliegue a producción si los resultados no son los adecuados.

Nombre: studentXX-myapp-07-acceptance-qa

Tipo de tarea: manual

Selecciona la opción “Esta ejecución debe parametrizarse”

Introduce un parámetro de tipo Elección (Choice)

Nombre: acceptance

Valores “No” y “Si” (Ojo, el orden es importante, ya que la primera opción será la de por defecto).

Descripción: Es esta release válida?

Marcar la opción: “Build after other projects are built” y establecer el nombre de la tarea de la que depende: studentXX-myaqpp-07-test-qa.

En los pasos de ejecución, agrega uno de tipo shell, con el siguiente código:

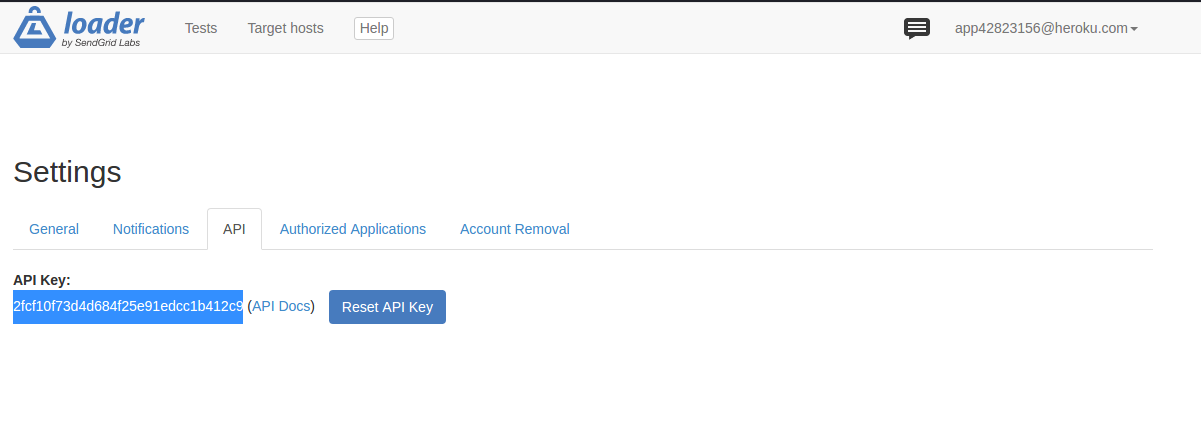
|  |
| --- |
| if [ "${acceptance}" != "Yes" ]  then  echo "Procedure interrupted by manual action"  exit 1  fi  exit 0 |

## Prácticas con Loader.io

Loader.io es una aplicación que permite capturar las acciones de un usuario en una aplicación web y repetirlas varias veces y con concurrencia. La idea establecer un tráfico constante sobre las aplicaciones y automatizar la realización de pruebas usando Jenkins. Para más información acerca de este plugin, ver <https://wiki.jenkins-ci.org/display/JENKINS/loaderio>

En Loader.io:

Accedemos a las settings del usuario y pulsamos en la pestaña de API:



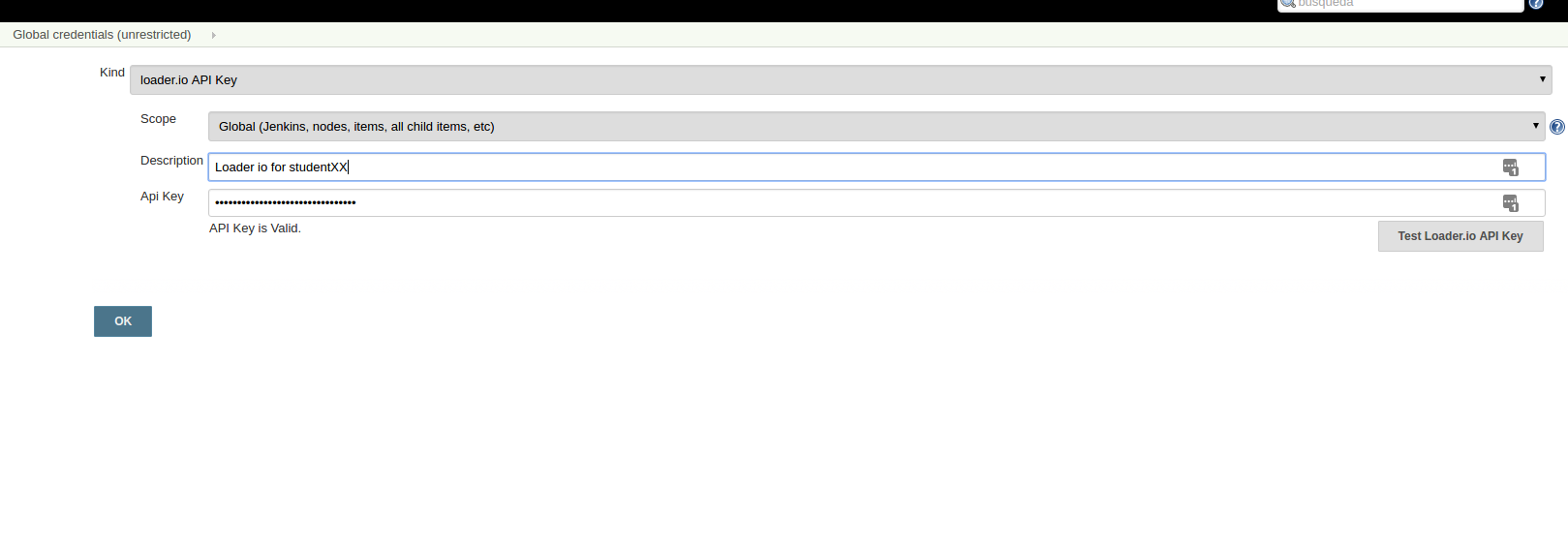
Copiamos la clave y la guardamos para la configuración del plugin de jenkins.

Para configurar loader.io, necesitamos configurar un nuevo plugin en jenkins llamado “loaderio-jenkins-plugin”.

Instalamos el plugin “loader-jenkins-plugin” y esperamos a que se reinicie el servidor jenkins.

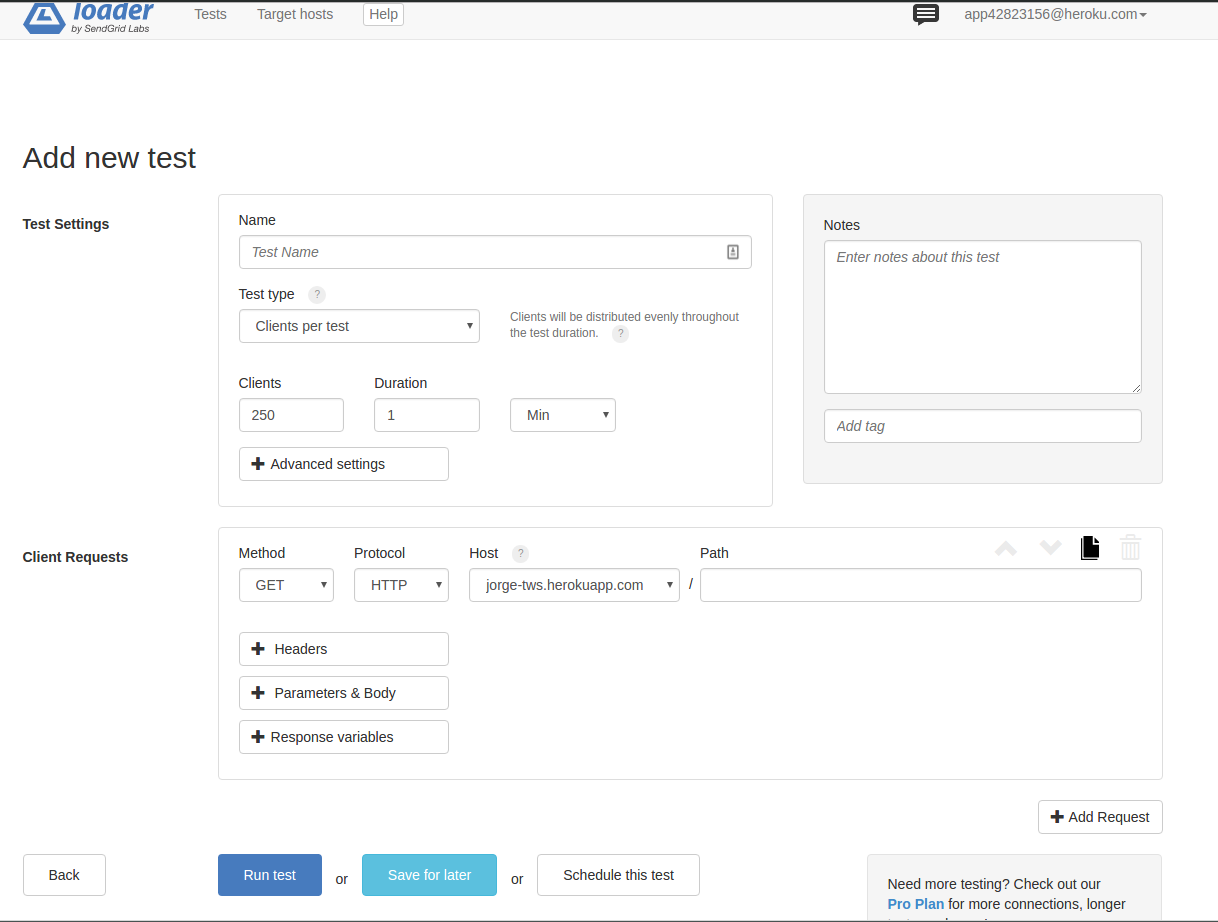
Después vamos a Jenkins > Credenciales > System > Add credentials

Añadimos una credencial de tipo Loader.io API Key.



En la descripción, indica que se trata de la API Key del usuario studentXX correspondiente.

Creación de un test:



Name: Home test

Test type: Clients per second

Clients: 10

Duration: 1 minute

Client Requests:

GET HTTP studentXX-myapp.examplecorp.es/

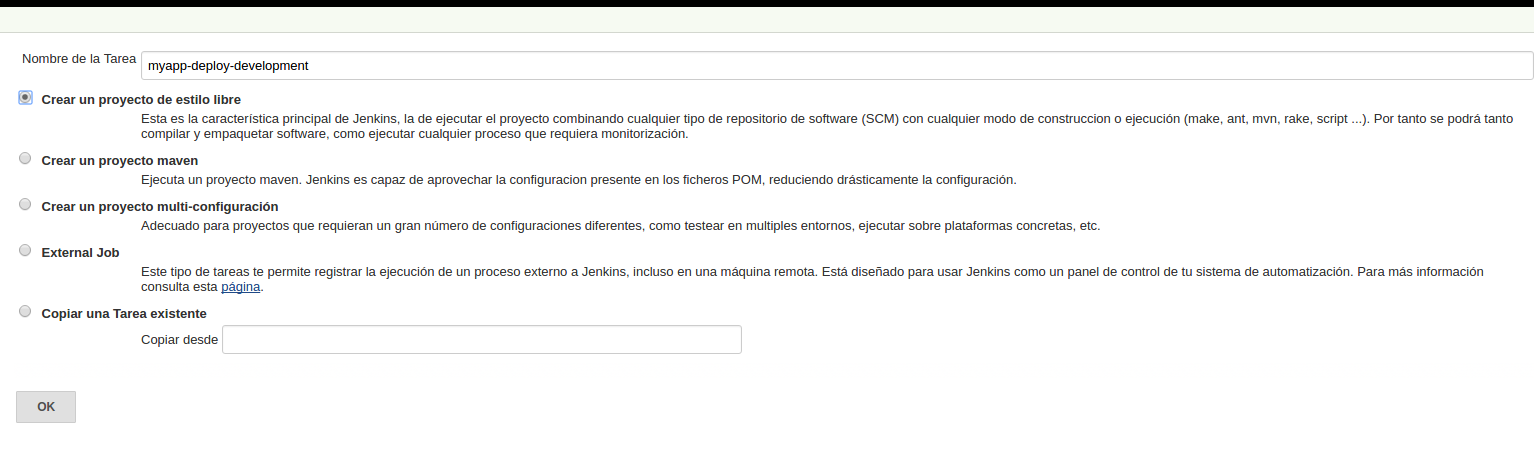
Guarda el test.

### Configurar tarea de despliegue en entorno de staging

Creamos una tarea en jenkins con los siguientes pasos:

Nombre: studentXX-myapp-08-deploy-staging

Tipo de proyecto: Crear un proyecto de estilo libre



Pulsamos Ok.

Configuramos el origen del código fuente:

Github: <https://github.com/devops-studentXX/lab02>

Establecemos como disparador, la tarea de despliegue del SNAPSHOT

Build after other projects are built: studentXX-myapp-07-acceptance-qa

Ejecución

Creamos una tarea de tipo: ejecutar línea de comandos (shell)

Tenemos que inyectar las variables necesarias para que terraform pueda funcionar:

|  |
| --- |
| # Vamos a configurar las variables de entorno necesarias para que terraform pueda actuar  export TF\_VAR\_aws\_access\_key="XXXX"  export TF\_VAR\_aws\_secret\_key="XXXX"  export TF\_VAR\_aws\_key\_path="~/.aws/studentXX.pem"  export TF\_VAR\_aws\_key\_name="studentXX.pem"  export TF\_VAR\_aws\_environment="staging"  # accedemos al directorio staging  cd terraform/aws/staging  # ejecutamos el comando  terraform apply |

Guardamos la tarea y comprobamos que podemos realizar todos los pasos desde jenkins.

Con esto tenemos terminado el proceso de despliegue en un entorno de staging.

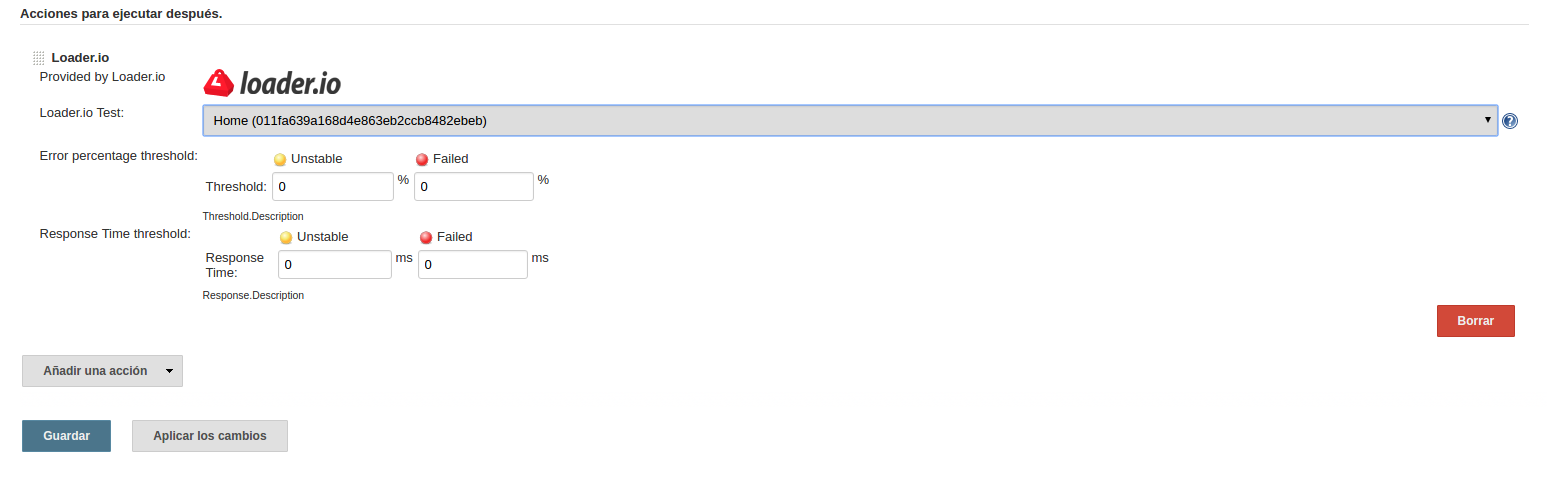
### Configurar tarea de testing continuo en staging

Crea una nueva tarea en jenkins con los siguientes parámetros:

Nombre: studentXX-myapp-09-test-staging

Tipo: proyecto libre

Ejecutar después de otros proyectos: studentXX-myapp-08-deploy-staging



En acciones para después, selecciona la API Key que hemos configurado anteriormente y podrás acceder al Test que se creó en los pasos anteriores.

Configura unos límites para determinar el resultado de la carga

Porcentaje de errores

Unstable si el resultado del test supera el 10 % de errores

Failure si el resultado supera el 20 % de errores

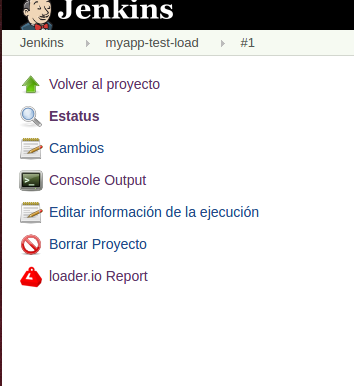
Tiempo de respuesta

Unstable si el resultado supera los 500 ms

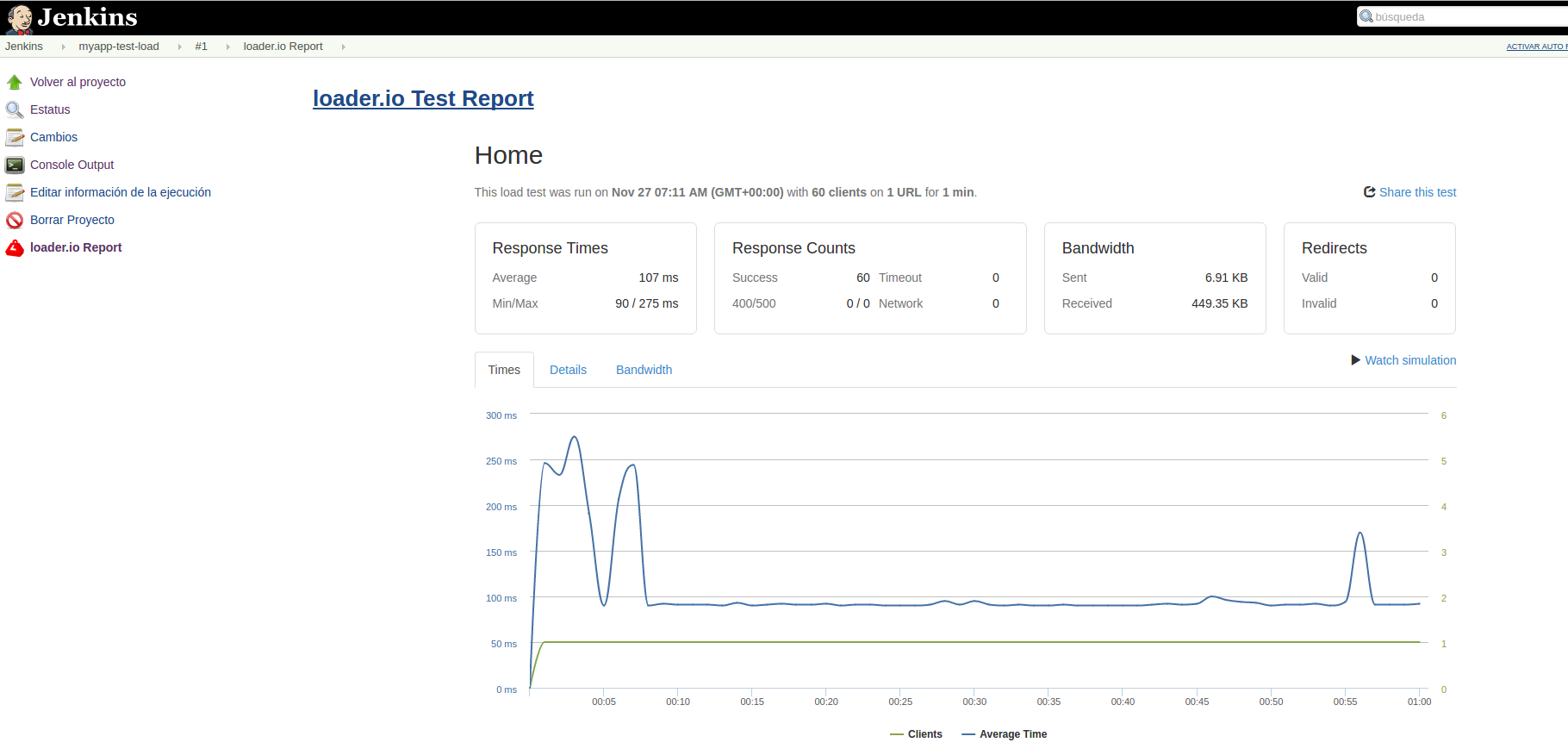
Failure si el resultado supera los 1000ms.

Guarda la tarea.

El test se ejecutará durante un minuto y terminará mostrando el color rojo o azul.



Para acceder al informe del test, accede a la tarea y en el menu de la derecha observarás que hay un enlace al loader.io report.



### Configurar monitorización de staging y producción

Ver más adelante las tareas de monitorización del laboratorio 3.

### Configurar tarea de Aceptación de RELEASE

En ocasiones, es necesario establecer un punto de interrupción del procedimiento de despliegue continuo. Algunas organizaciones necesitan que alguien se responsabilice de los cambios, por lo que se crean tareas manuales que obliguen a alguien a ejecutarlas para así registrar las autorizaciones.

Crea una nueva tarea en jenkins que permita interrumpir el despliegue a producción si los resultados no son los adecuados.

Nombre: studentXX-myapp-10-acceptance-staging

Tipo de tarea: manual

Selecciona la opción “Esta ejecución debe parametrizarse”

Introduce un parámetro de tipo Elección (Choice)

Nombre: acceptance

Valores “No” y “Si” (Ojo, el orden es importante, ya que la primera opción será la de por defecto).

Descripción: Es esta release válida?

Marcar la opción: “Build after other projects are built” y establecer el nombre de la tarea de la que depende: studentXX-myapp-09-test-staging.

En los pasos de ejecución, agrega uno de tipo shell, con el siguiente código:

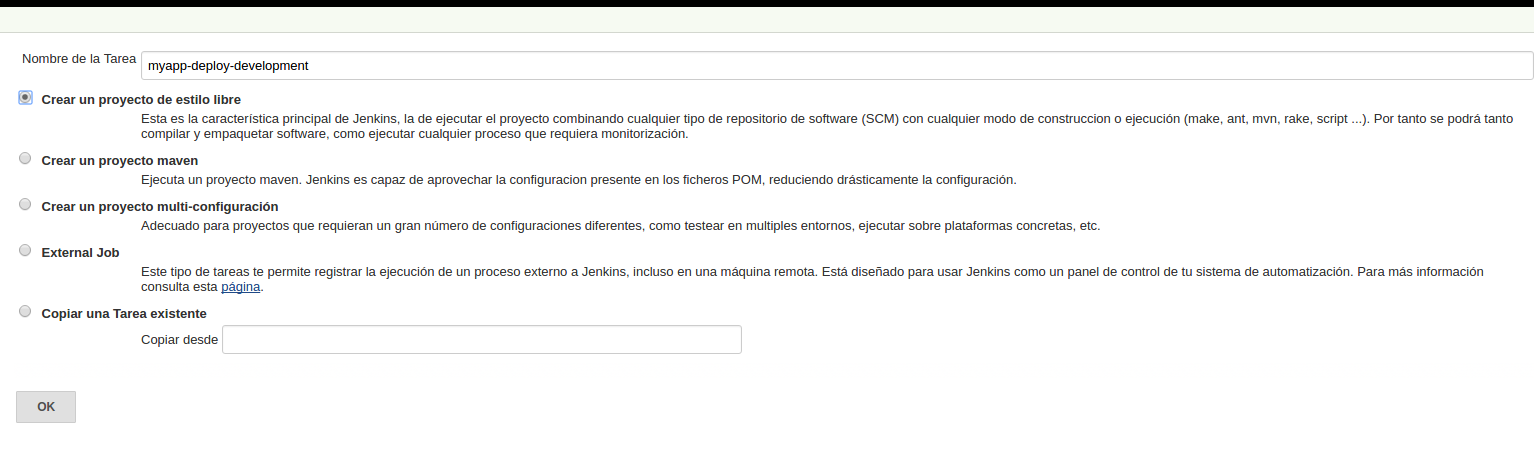
|  |
| --- |
| if [ "${acceptance}" != "Yes" ]  then  echo "Procedure interrupted by manual action"  exit 1  fi  exit 0 |

### Configurar tarea de despliegue en producción

Creamos una tarea en jenkins con los siguientes pasos:

Nombre: studentXX-myapp-11-deploy-production

Tipo de proyecto: Crear un proyecto de estilo libre



Pulsamos Ok.

Configuramos el origen del código fuente:

Github: <https://github.com/devops-studentXX/lab02>

Establecemos como disparador, la tarea de despliegue del SNAPSHOT

Build after other projects are built: studentXX-myapp-10-acceptance-staging

Ejecución

Creamos una tarea de tipo: ejecutar línea de comandos (shell)

Tenemos que inyectar las variables necesarias para que terraform pueda funcionar:

|  |
| --- |
| # Vamos a configurar las variables de entorno necesarias para que terraform pueda actuar  export TF\_VAR\_aws\_access\_key="XXXX"  export TF\_VAR\_aws\_secret\_key="XXXX"  export TF\_VAR\_aws\_key\_path="~/.aws/studentXX.pem"  export TF\_VAR\_aws\_key\_name="studentXX.pem"  export TF\_VAR\_aws\_environment="production"  # accedemos al directorio production  cd terraform/aws/production  # ejecutamos el comando  terraform apply |

Guardamos la tarea y comprobamos que podemos realizar todos los pasos desde jenkins.

Con esto tenemos terminado el proceso de despliegue en un entorno de production.

## Prácticas con Heroku

Heroku es una plataforma de despliegue muy automatizada, que permite a los desarrolladores desplegar sin necesidad de conocimientos en infraestructuras. Configuraremos dos aplicaciones y las desplegaremos usando la línea de comandos y la interfaz web.

El instructor proporcionará los datos de acceso a la plataforma de heroku

#### Creación de las aplicaciones de Heroku

##### Staging

Desde la línea de comandos y el directorio de código de la aplicación, ejecuta el siguiente comando:

|  |
| --- |
| $ cd lab02/terraform/heroku/staging  $ vi terraform.tfvars  [introduce los datos de la cuenta de terraform y de tu usuario]  $ terraform plan -var-file terraform.tfvars  $ terraform apply -var-file terraform.tfvars |

##### Production

Desde la línea de comandos y el directorio de código de la aplicación, ejecuta el siguiente comando:

|  |
| --- |
| $ cd lab02/terraform/heroku/production  $ vi terraform.tfvars  [introduce los datos de la cuenta de terraform y de tu usuario]  $ terraform plan -var-file terraform.tfvars  $ terraform apply -var-file terraform.tfvars |

##### Configuración pipeline

Heroku tiene un sistema propio para la gestión del continuous deployment denominado [*pipelines*](https://devcenter.heroku.com/articles/pipelines).

En este ejercicio vamos a usar parte de ese sistema para facilitar el despliegue entre la fase de staging y la fase de producción.

Instalación del plugin de pipelines

Necesitaremos instalar el plugin de pipelines tanto en nuestra cuenta, como en la cuenta de jenkins (ya lo habrá hecho el instructor).

|  |
| --- |
| $ heroku plugins:install heroku-pipelines |

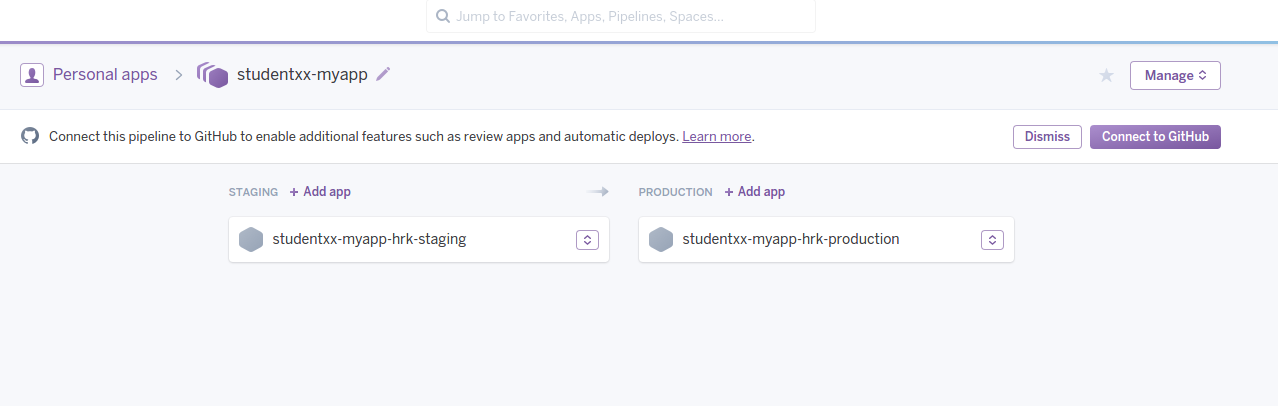
Creamos el pipeline para la aplicación en produccion

|  |
| --- |
| $ heroku pipelines:create -a studentxx-myapp-hrk-production  ? Pipeline name: studentxx-myapp-pipeline ? Stage of studentxx-myapp-hrk-production: production |

Asociamos la aplicación de staging al pipeline

|  |
| --- |
| $ heroku pipelines:add -a studentxx-myapp-hrk-staging studentxx-myapp-pipeline  ? Stage of example-staging: staging |

Así se verá en la interfaz gráfica:



Con estos pasos tendremos creada la infraestructura necesaria para operar en heroku.

#### Preparar la máquina jenkins con las tareas necesarias para desplegar en heroku.

Necesario tener el heroku toolkit instalado en la cuenta de tomcat7 (usuario que ejecuta jenkins)

Para aplicaciones java, necesitamos maven y el heroku maven plugin instalado.

Plugin: heroku-maven-plugin

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>com.heroku.sdk</groupId>  <artifactId>heroku-maven-plugin</artifactId>  <version>1.1.2</version>  <configuration>  <includes>  <include>target/dependency/newrelic-agent.jar</include>  </includes>  <appName>**YOUR APP NAME HERE**</appName>  </configuration>  </plugin>  …  </plugins>  …  </build> |

|  |
| --- |
| **NOTA: Introduce el nombre de la aplicación para poder desplegar correctamente.** |

### Configurar aplicación staging

Vamos a crear ahora la tarea de despliegue de la aplicación en un entorno de staging de Heroku.

Heroku tiene configurados por defecto dos entornos: staging y producción. La forma en la que vamos a desplegar, permite reutilizar la misma imagen de disco (llamada slug) generada durante la construcción del entorno, para el siguiente entorno (producción).

Este mecanismo puede no ser recomendable en todas las situaciones, ya que si elementos compilados (assets) incluídos en el slug están relacionados con datos del entorno (como urls, dominios, etc), puede dar errores.

Dicho esto, comenzamos con la tarea de jenkins:

Crear una nueva tarea en jenkins

Nombre studentXX-myapp-20-heroku-deploy-staging

Tipo de tarea: estilo libre

Origen del código fuente:

Git: <https://github.com/devops-studentXX/myapp.git>

Disparadores de ejecución:

Seleccionar “Build after other projects are build”: establecer studentXX-myapp-07-acceptance-qa como antecesor

Ejecución:

Seleccionar “Ejecutar línea de comandos shell” y escribir el siguiente script:

|  |
| --- |
| #!/bin/bash  #  # from https://gist.github.com/caarlos0/6366804  # login in the jenkins server with:  #  # heroku login  # heroku keys:add  #  # Doing so, jenkins will have permission to deploy to  # the heroku remote.  #  # exit 1 on errors  set -e  # deal with remote  echo "Checking if remote exists..."  if ! git ls-remote heroku; then  echo "Adding heroku remote..."  # HERE IT GOES THE NAME OF YOUR HEROKU APP  git remote add heroku git@heroku.com:devops-studentxx-myapp-staging.git  fi  # push only origin/master to heroku/master - will do nothing if  # master doesn't change.  echo "Updating heroku master branch..."  git push heroku origin/master:master |

Guardar los cambios

### Despliegue a producción

Para desplegar a producción desde jenkins, sólo tenemos que crear una tarea en jenkins que se encargue de ejecutar el comando:

|  |
| --- |
| $ heroku pipelines:promote -a studentxx-myapp-hrk-staging |

Nombre studentXX-myapp-21-studentXX-heroku-deploy-production

Tipo de tarea: estilo libre

Origen del código fuente:

Git: <https://github.com/devops-studentXX/myapp.git>

Disparadores de ejecución:

Ejecución:

Seleccionar “Ejecutar línea de comandos shell” y escribir el siguiente comando:

|  |
| --- |
| $ heroku pipelines:promote -a studentxx-myapp-hrk-staging |

## Prácticas con IBM BlueMix

BlueMix es una plataforma de despliegue automatizada con mucho foco en DevOps, de forma que las aplicaciones puedan ser desplegadas con velocidad y precisión.

Usando las herramientas de línea de comandos que nos proporciona BlueMix, desplegaremos dos aplicaciones y realizaremos cambios con ellas.

Configuración inicial

En la máquina de devops ya se tienen instalados los toolkits que permiten la comunicación a través de línea de comandos con la plataforma.

Si quiere instalarlos en otro equipo, siga los siguientes pasos:

Instalación de Bluemix CLI

<http://clis.ng.bluemix.net/ui/home.html>

Instalación de CloudFoundry

<https://github.com/cloudfoundry/cli>

Adaptación del proyecto a Bluemix

Se necesita crear un fichero *manifest.yml* con la configuración de la aplicación según bluemix. Más sobre el fichero manifest en la [documentación](https://docs.cloudfoundry.org/devguide/deploy-apps/manifest.html).

|  |
| --- |
| applications:  - path: target/myapp.war  memory: 256M  instances: 1  domain: eu-gb.mybluemix.net  name: studentxx-myapp-bluemix-production  host: studentxx-myapp-bluemix-production  disk\_quota: 1024M  buildpack: java\_buildpack |

Conectar con Bluemix

|  |
| --- |
| $ bluemix api <https://api.eu-gb.bluemix.net>  Invocando 'cf api https://api.eu-gb.bluemix.net'...  Estableciendo un punto final de API en https://api.eu-gb.bluemix.net...  Aceptar  Punto final de la API: https://api.eu-gb.bluemix.net (Versión de la API: 2.54.0)  Usuario: studentXX@examplecorp.es  Organización: examplecorp  Espacio: production |

Login con Bluemix CLI

|  |
| --- |
| $ bluemix login -u studentXX@examplecorp.es -o "examplecorp" -s "production"  Invocando 'cf login -u studentXX@examplecorp.es -o examplecorp -s production...  Punto final de la API: https://api.eu-gb.bluemix.net  Password>  Autenticando...  Aceptar  Organización de destino examplecorp  Espacio de destino production  Punto final de la API: https://api.eu-gb.bluemix.net (Versión de la API: 2.54.0)  Usuario: studentXX@examplecorp.es  Organización: examplecorp  Espacio: production |

Despliegue de la aplicación

|  |
| --- |
| $ cf push "studentxx-myapp-bluemix-production"  Utilización del archivo de manifiesto /Users/instructor/Documents/Workspaces/personal/myapp/manifest.yml  Actualizando la app studentxx-myapp-bluemix-production en la organización examplecorp / espacio production como studentXX@examplecorp.es...  Aceptar  Utilización de la ruta studentxx-myapp-bluemix-production.eu-gb.mybluemix.net  Subiendo studentxx-myapp-bluemix-production...  Subiendo archivos de app desde: /var/folders/wj/sgn4pf1j6\_s4bj93jgd5r1nm0000gn/T/unzipped-app303501021  Subida de archivos 3.8K, 11  Done uploading  Deteniendo app studentxx-myapp-bluemix-production en la organización examplecorp / espacio production como studentXX@examplecorp.es...  Aceptar  Iniciando app studentxx-myapp-bluemix-production en la organización examplecorp / espacio production como studentXX@examplecorp.es... |

### Configurar aplicación producción

Nombre studentXX-myapp-30-bluemix-deploy-production

Tipo de tarea: estilo libre

Origen del código fuente:

Git: <https://github.com/devops-studentXX/myapp.git>

Disparadores de ejecución:

Ejecución:

Seleccionar “Ejecutar línea de comandos shell” y escribir el siguiente script:

|  |
| --- |
| #!/bin/bash  # finish the deploy if anything goes wrong  sed -e  bluemix api <https://api.eu-gb.bluemix.net>  bluemix login -u studentXX@examplecorp.es -o "examplecorp" -s "production"  cf push "studentxx-myapp-bluemix-production" |

### 

# Laboratorio 3

## Prácticas de Monitorización

En este caso, vamos a agregar a las aplicaciones un elemento de monitorización, un servicio llamado UptimeRobot, y un servicio de gestión del rendimiento de aplicación (APM),llamado NewRelic, para obtener información del estado de la aplicación y sacar métricas del servicio.

También usaremos el servicio Geckoboard para crear dashboards con las métricas más representativas del servicio.

## Con UptimeRobot

### Configurar los endpoints para que se detecte la caída/recuperación del servicio.

Accede a la cuenta de uptimerobot creada para el curso y registra un endpoint con los datos siguientes:

<http://uptimerobot.com>

Usuario: [devops@examplecorp.es](mailto:devops@examplecorp.es)

Contraseña: cangetin

Agrega un nuevo monitor. Introduce los siguientes datos:

Monitor Type: HTTP(s)

Friendly Name: myapp-studentXX

URL: [http://myapp-studentXX.examplecorp.es](http://myapp-studentxx.examplecorp.es)

Monitor interval: every 5 minutes

Create Monitor

### 

### Configurar una alerta para avisar cuando cambia la versión de un servicio.

Agrega un nuevo monitor. Introduce los siguientes datos:

Monitor Type: Keyword

Friendly Name: myaqpp-studentXX version

URL: [http://myaqpp-studentXX.examplecorp.es/version](http://myaqpp-studentxx.examplecorp.es/version)

Keyworkd: 1.1.0 (o la versión actual en las pruebas que se hayan realizado)

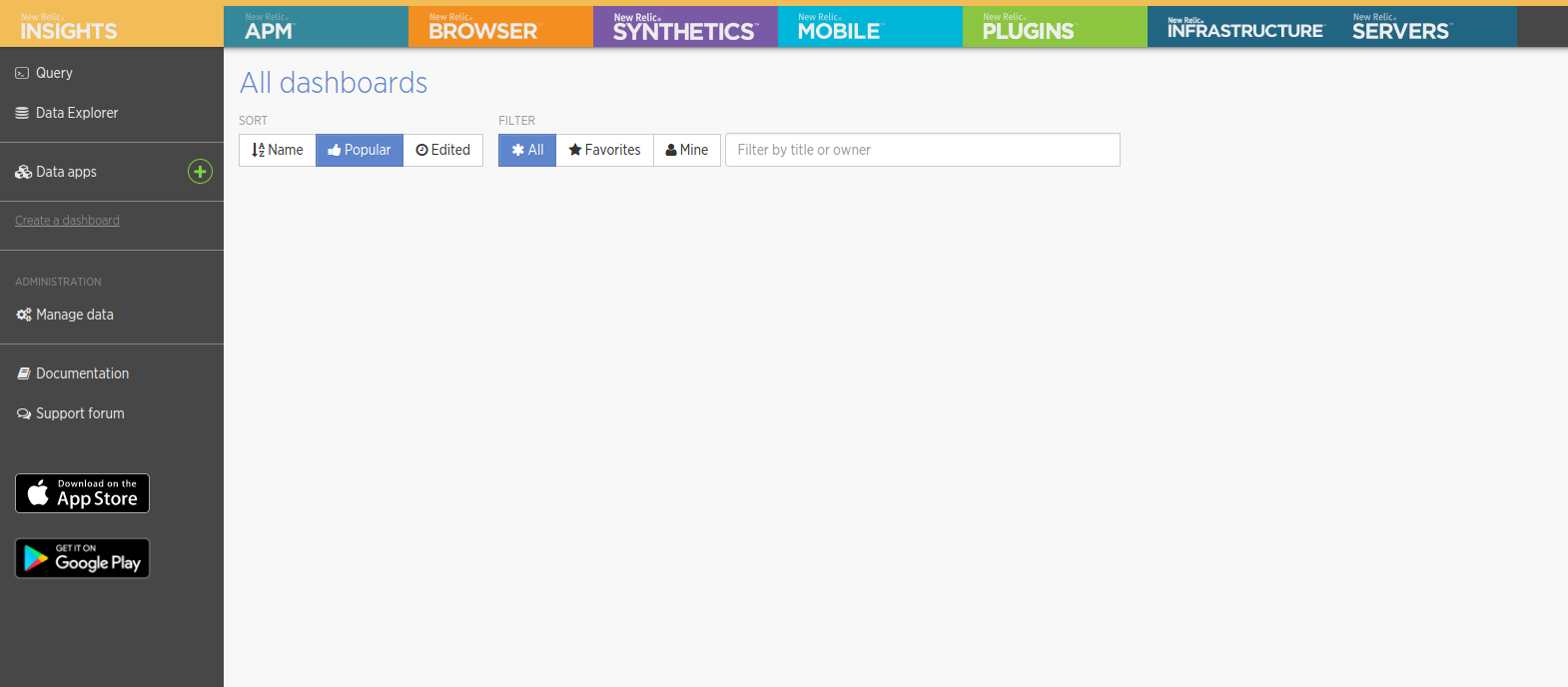
Alert When: Keyword Not Exists

Monitoring Interval: 5 minutes

Con esto tenemos configurados unos monitores que nos pueden avisar cuándo es servicio está caído o no coincide con nuestras expectativas.

Pregunta: ¿Cómo plantearías la detección de anomalías en los servicios de tus sites web? No hablamos de caídas, sino degradación de servicio o pérdida parcial de servicio en la web.

## Con New Relic

Cuando se abra la interfaz de New Relic, pulsamos en el menú superior la opción “INSIGHTS”. Desde aquí vamos a crear un dashboard con varias métricas y KPIs.

Creación de un dashboard

Lo primero de todo es crear un dashboard que aloje todas las métricas que vamos a crear.

Pulsa en el enlace “Create a dashboard” en el menu de opciones de la izquierda.

En el campo de texto introduce el nombre del dashboar. En nuestro caso lo llamaremos “Metricas”. Con esto se dispone de un sitio donde guardar las métricas y eventos que vamos a configurar a continuación.

No hace falta guardar nada, ya que vamos a seguir editando el dashboard.

Para agregar una métrica, disponemos de dos herramientas:

**Data Explorer**: es una vista sobre todos los eventos y métricas que guarda New Relic.

Principalmente tenemos dos tablas: Page Views y Transactions.

Page Views: contiene información sobre actividad realizada por navegadores y dispositivos.

Transactions: contiene información más próxima al backend y al funcionamiento interno de la aplicación.

**NRQL Query Interface**: Es la interfaz de consultas de New Relic. Un lenguaje similar a SQL que permite realizar consultas y operaciones básicas sobre datos de la aplicación guardados en las tablas de New Relic.

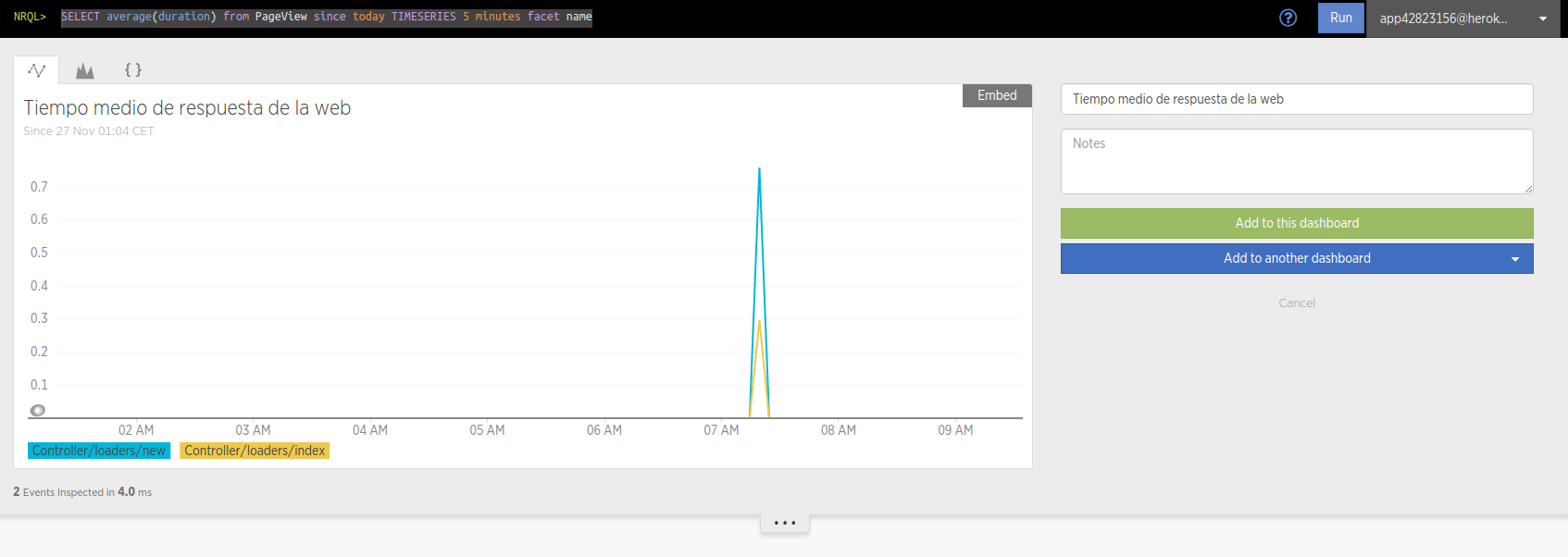
|  |
| --- |
| **NOTA: es recomendable dejar realizando tests de carga en la aplicación para mostrar datos durante la configuración de las gráficas.** |

### Métrica: tiempo medio de respuesta de la web

En el dashboad “Metricas” vamos a crear una primera consulta usando la interfaz NRQL.

Introduce la siguiente query:

SELECT average(duration) from PageView since today TIMESERIES 5 minutes facet name

Se generará un gráfico como el de la imagen. Introduce el nombre de la métrica en el cajón de la derecha. Por último pulsa en “Add to dashboard”.

Con esto tenemos la primera métrica en nuestro dashboard.

### Métrica: Nº de transacciones por minuto

Crea un gráfico con la siguiente query:

SELECT count(name) from Transaction since 1 hour ago timeseries 1 minute facet name

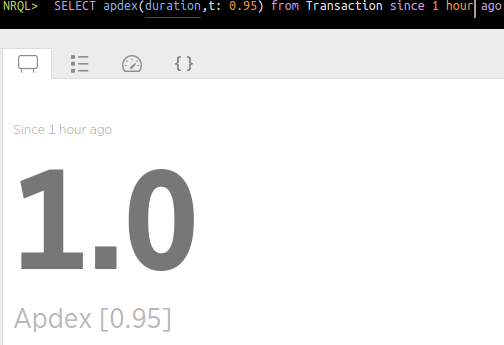
### KPI: Apdex de la aplicación

Ahora vamos a agregar un gráfico que nos indique el nivel de calidad de experiencia de navegación (Apdex) de nuestra web.

El apdex es una función que devuelve un valor entre 0 y 1 indicando el nivel de satisfacción de un usuario en base al tiempo de respuesta de la aplicación. Más información aquí:

<https://docs.newrelic.com/docs/apm/new-relic-apm/apdex/apdex-measuring-user-satisfaction>

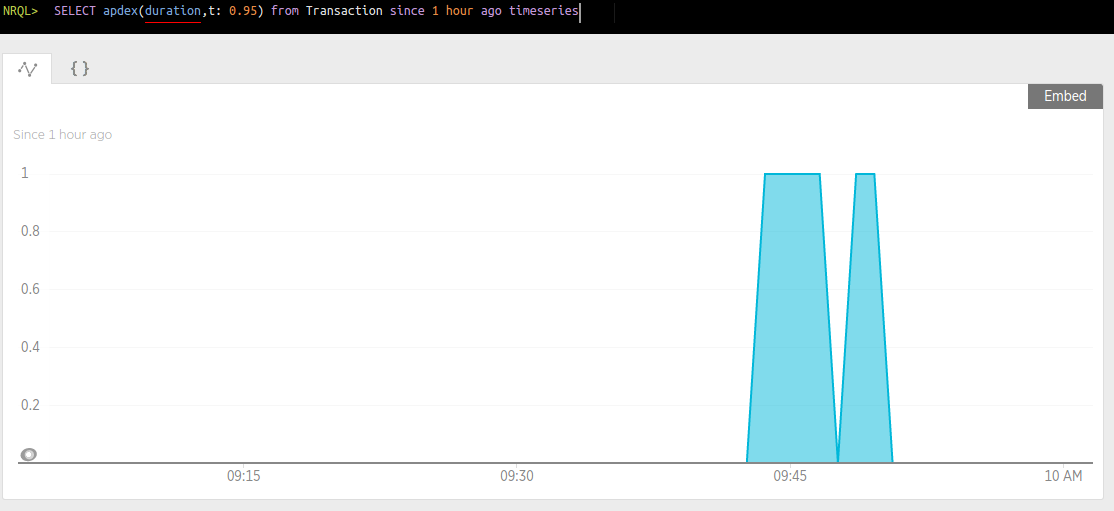
Para mostrar un valor único por unidad de tiempo, la query resultante es



SELECT apdex(duration,t: 0.5) from Transaction since 1 hour ago

Para mostrar una línea temporal, usa la keyword TIMESERIES

SELECT apdex(duration,t: 0.5) from Transaction since 1 hour ago



Si quieres ver los apdex separados por transacción, añade la siguiente keyword: FACET name

SELECT apdex(duration,t: 0.95) from Transaction since 1 hour ago facet name



### Métrica: Nº de objetos creados por minuto

Usando la funcionalidad de Custom metrics de New Relic, podemos enviar datos del funcionamiento de la aplicación a New Relic para generar gráficas.

Más informacion en <https://docs.newrelic.com/docs/agents/manage-apm-agents/agent-data/custom-metrics>

### KPI: Tiempo medio entre despliegues

Todavía no está implementada esta funcionalidad en NewRelic. Como ejercicio extra, sería interesante ver cómo crear a través de [Custom events via Insights API](https://docs.newrelic.com/docs/insights/new-relic-insights/custom-events/insert-custom-events-insights-api) los eventos y obtener la diferencia entre ellos.

### Goal: SLA 99% diario

New Relic proporcionaba una interfaz de dashboards donde se podía mostrar el nivel de servicio de una aplicación. Esta interfaz está deprecada en favor de Insights, pero esta funcionalidad de APM todavía no está integrada dentro de Insights. Más información en <https://docs.newrelic.com/docs/data-analysis/legacy-custom-dashboards/custom-dashboards-v2/custom-dashboards-legacy>

# Laboratorio 4

Realizaremos una Retrospectiva del Workshop para obtener información por parte de los asistentes y se responderán a las tres preguntas:

* Qué fue mal, o no tan bien como se esperaba?
* Qué podía haber hecho para resolverlo?
* Por último, qué fue bien, de qué se siente orgulloso?

Esta práctica ayuda a conocer la experiencia de cada alumno, así como al instructor a definir próximos elementos del workshop.

También definiremos unos objetivos usando la técnica OKR y la herramienta weekdone. Mediante la técnica OKRs se establecerán unos objetivos personales que permitan a los alumnos aplicar lo aprendido en el seminario en su día a día.

Por último, revisitaremos la arquitectura para ver qué puntos se pueden mejorar, empleando la técnica descrita en Teoría de las Limitaciones (Theory of Constrains) aplicada al diseño de sistemas.

1. <https://git-scm.com/book/es/v1/Empezando-Configurando-Git-por-primera-vez> [↑](#footnote-ref-0)