**Integración continua**

Jenkinks

## Administrar Jenkins

Plugins a tener en cuenta:

* Subversion Plug-in
* Mailer Plugin
* JUnit Plugin / CVS Plug-in / Git full stack
* SonarQube Plugin
* NodeJS Plugin
* Pipeline full stack

Plugins ?????

* Ansicolor (log)
* Timestamper (log)
* Parallel test executor
* Yet another docker (\*)
* Monitoring
* Datadog (muchos jobs)
* Blue ocean

## Configuración

SonarQube servers

* Instalaciones de SonarQube
  + Name: Sonnar Server
  + URL del servidor: <http://sonar:9000>
  + Server version: 5.3 or higher

Maven Configuration

* Default settings provider: Use default maven settings
* Default global settings provider : Use default maven global settings

Copyartifact: Upstream build that triggered this job

* Which for multiple upstream: Use the oldest build

SonarQube Scanner

* instalaciones de SonarQube Scanner
  + SonarQube Scanner
    - Name: Sonnar Runner
    - Instalar automáticamente: check
    - Install from Maven Central
      * Versión: SonarQube Scanner 2.5.1

Ant

* instalaciones de Ant
  + Ant
    - Nombre: Ant v1.9.4
    - ANT\_HOME: /usr/share/ant

Maven

* instalaciones de Maven
  + Maven
    - Nombre: Maven v3.0.5
    - MAVEN\_HOME: /usr/share/maven
  + Maven
    - Nombre: Maven v3.3.9
    - Instalar automáticamente: check
      * Instalar desde Apache
        + Versión: 3.39

NodeJS

* instalaciones de NodeJS
  + NodeJS
    - Nombre: Node JS v4.2.6
    - Instalar automáticamente: check
      * Install from nodejs.org
        + Versión: NodeJS 4.2.6
        + Global npm packages to install: bower@~1.7.9 grunt-cli@~0.1.13 [selenium-webdriver@~2.52.0](mailto:selenium-webdriver@~2.52.0)
        + Global npm packages refresh hours: 72

Rebuild

* Rebuild Configuration
  + Remember Password Enabled: check

Jenkins Location

* Dirección web de Jenkins: <http://xxx.xxx.xx.xx/>

Subversion

* Subversion Workspace Version: 1.8

Notificación por correo electrónico

* Servidor de correo saliente (SMTP): hellboy.satec.es

Publish over SSH

* SSH Servers
  + SSH Server
    - Name: server
    - Hostname: <url-server>
    - Username: root
    - Remote Directory: /
    - Use password authentication, or use a different key: check
    - Passphrase / Password: <password>
    - Port: 22
    - Timeout (ms): 300000
  + SSH Server
    - Name: server front
    - Hostname: <url-server-front>
    - Username: root
    - Remote Directory: /
    - Use password authentication, or use a different key: check
    - Passphrase / Password: <password>
    - Port: 22
    - Timeout (ms): 300000

# Pipelines

## Pipeline Backend

### SCM

Tarea donde configuraremos la obtención de nuestro proyecto para ser procesado.

#### Configurar el origen del código fuente

Lugar donde tenemos que indicar el donde está nuestro repositorio y configurar el acceso a el.

Para ello seleccionamos el tipo, configuramos un módulo, seleccionamos el tipo de estrategia a seguir y el navegador del repositorio.

Ejemplo:

* Subversion
* Modulo:
  + Repository URL: svn://svn.aplicaciones.satec.es:33690/ANGOLA\_TELECOM\_OSS\_INVyPROV/trunk/2Inventario/trunk/1Código
  + Credentials: none
  + Repository depth: infinty
  + Ignore externals: check
* Check-out Strategy: Use ‘svn update’ as much as possible
* Navegador del repositorio: (Auto)

#### Disparadores de ejecuciones

Donde configuraremos cuando se va a lanzar la ejecución de la tarea.

Ejemplo:

* Ejecutar periódicamente: H H(0-8) 1,10,20 \* \*

#### Acciones para ejecutar después

Apartado donde indicaremos subtareas a realizar una vez se ejecute la tarea.

En este apartado configuraremos la copia de los ficheros procesados en la tarea (para no tenerlos que descargar otra vez), copiarlos a un almacen compartido con otras tareas y lanzar la siguiente tarea a ejecutar en nuestro pipeline.

Ejemplo:

* Guardar los archivos generados
  + Ficheros para guardar: \*/\*\*
* Almacenar firma de ficheros para poder hacer seguimiento
  + Almacenar la firma de los ficheros: \*/\*\*
* Ejecutar otros proyectos
  + Proyectos a ejecutar: quality (Trigger only if build is stable)

### QUALITY

Tarea o tareas donde configuraremos la ejecución de las pruebas de nuestro código, ya sean pruebas unitarias, end-to-end, etc.

Si se van a ejecutar distintas tipos de pruebas, lo suyo sería crear una tarea por cada una de ellas, enlazadas entre si. Esto se hace para dividir el procesamiento de cada una y saber donde está el posible fallo o error.

#### Configurar el origen del código fuente

Lugar donde tenemos que indicar el donde está nuestro repositorio y configurar el acceso a el.

Como este paso ya lo hemos ejecutado en la tarea anterior, tendremos que indicarle que no lo vamos a usar.

Ejemplo:

* Ninguno

#### Disparadores de ejecuciones

Ejemplo:

* Ejecutar siempre que cualquier 'SNAPSHOT' de los que dependa sea creado: check

#### Pasos previos

Donde realizaremos las tareas necesarias antes de la ejecución de nuestra tarea.

En nuestro caso será el acceso a los ficheros descargados del repositorio que se ha realizado en el paso anterior.

Ejemplo:

* Ejecutar linea de comandos (shell)
  + Comando: rm –rf $WORKSPACE/\*
* Copy artifacts from another project
  + Project name: scm
  + Which build: Latest successful build
    - Stable build only: check
  + Artifacts to copy: \*\*/\*
  + Parameter filters: - (Fingerprint Artifacts)

#### Proyecto

Donde lanzaremos las pruebas de nuestro codigo.

Ejemplo:

* Versión de maven: Maven v3.3.9
* Fichero POM raíz: 00\_parent/pom.xml
* Goles y opciones: clean compile test org.jacoco:jacoco-maven-plugin:prepare-agent

#### Pasos posteriores

Lugar donde configuraremos acciones que queramos hacer después de la ejecución de nuestras pruebas.

En nuestro caso querremos enviar los resultados a Sonnar para tener una visión más amplia de nuestras pruebas.

Ejemplo:

* Ejecutar siempre (sea cual sea el resultado): check
* Execute SonarQube Scanner:
  + JDK: (Inherit From Job)
  + Analysis properties:

# required metadata

sonar.projectKey=app

sonar.projectName=App

sonar.projectVersion=1.0

# path to source directories (required)

sonar.sources=src/main/java

sonar.language=java

sonar.binaries=target/classes

# List of the module identifiers

sonar.modules=app,appPlugin

# Properties can obviously be overriden for

# each module - just prefix them with the module ID

app.sonar.projectName=App - Application

appPlugin.sonar.projectName=AppPlugin – Application Plugin

# path to test source directories (optional)

sonar.tests=src/test/java

sonar.dynamicAnalysis=reuseReports

sonar.junit.reportsPath=target

sonar.java.coveragePlugin=jacoco

sonar.jacoco.reportPath=target/jacoco.exec

#### Propiedades del proyecto

Donde podemos configurar la notificación vía email del resultado de la tarea.

Ejemplo:

* Notificación por E-mail Recipients: angel.yague@satec.es
  + Send e-mail for every unstable build: check

#### Acciones para ejecutar después

Apartado donde indicaremos subtareas a realizar una vez se ejecute la tarea.

Igual que hicimos en la tarea ‘scm’, en este apartado configuraremos la copia de los ficheros procesados en la tarea (para no tenerlos que descargar otra vez), copiarlos a un almacen compartido con otras tareas y lanzar la siguiente tarea a ejecutar en nuestro pipeline.

Ejemplo:

* Guardar los archivos generados
  + Ficheros para guardar: \*/\*\*
* Almacenar firma de ficheros para poder hacer seguimiento
  + Almacenar la firma de los ficheros: \*/\*\*
* Ejecutar otros proyectos
  + Proyectos a ejecutar: build (Trigger only if build is stable)

### BUILD

Tarea donde configuraremos el proceso de construcción del proyecto despues de haber pasado todas las pruebas.

#### Configurar el origen del código fuente

Lugar donde tenemos que indicar el donde está nuestro repositorio y configurar el acceso a el.

Como este paso ya lo hemos ejecutado en la primera anterior, tendremos que indicarle que no lo vamos a usar.

Ejemplo:

* Ninguno

#### Disparadores de ejecuciones

Ejemplo:

* Ejecutar siempre que cualquier 'SNAPSHOT' de los que dependa sea creado: check

#### Pasos previos

Donde realizaremos las tareas necesarias antes de la ejecución de nuestra tarea.

En nuestro caso será el acceso a los ficheros descargados del repositorio que se ha realizado en el paso anterior.

Ejemplo:

* Ejecutar linea de comandos (shell)
  + Comando: rm –rf $WORKSPACE/\*
* Copy artifacts from another project
  + Project name: quality
  + Which build: Copy from WORKSPACE of latest completed build
  + Artifacts to copy: \*\*/\*
  + Parameter filters: - (Fingerprint Artifacts)

#### Proyecto

Donde lanzaremos la construcción de nuestra aplicación.

Ejemplo:

* Versión de maven: Maven v3.3.9
* Fichero POM raíz: 00\_parent/pom.xml
* Goles y opciones: install package -Dmaven.test.skip=true -P <profile>

#### Pasos posteriores

Lugar donde configuraremos acciones que queramos hacer después de la ejecución de nuestras pruebas.

En nuestro caso no querremos realizar ninguna accción adiccional.

Ejemplo:

* Ejecutar siempre (sea cual sea el resultado): check

#### Propiedades del proyecto

Donde podemos configurar la notificación vía email del resultado de la tarea.

Ejemplo:

* Notificación por E-mail Recipients: angel.yague@satec.es
  + Send e-mail for every unstable build: check

#### Acciones para ejecutar después

Apartado donde indicaremos subtareas a realizar una vez se ejecute la tarea.

Igual que hicimos en las otras tareas, guardaremos los ficheros para que la siguiente tarea pueda usarlo. En nuestro caso nos quedaremos con fichero que contiene a la aplicación.

Ejemplo:

* Guardar los archivos generados
  + Ficheros para guardar: inventory/target/app\*.war
* Ejecutar otros proyectos
  + Proyectos a ejecutar: deploy (Lanzar incluso si el resultado de la ejecución fue inestable)

### DEPLOY

Tarea donde configuraremos el despliegue de la aplicación generada en los servidores.

#### Configurar el origen del código fuente

Lugar donde tenemos que indicar el donde está nuestra aplicación y configurar el acceso a ella.

Como este paso ya lo hemos ejecutado en anterior paso, tendremos que indicarle que no lo vamos a usar.

Ejemplo:

* Ninguno

#### Pasos previos

Donde realizaremos las tareas necesarias antes de la ejecución de nuestra tarea.

En nuestro caso será el acceso a los ficheros descargados del repositorio que se ha realizado en el paso anterior.

Ejemplo:

* Ejecutar linea de comandos (shell)
  + Comando: rm –rf $WORKSPACE/\*
* Copy artifacts from another project
  + Project name: build
  + Which build: Latest successful build
  + Parameter filters: - (Flattern directories / Fingerprint Artifacts)
* Send files or execute commands over SSH
  + SSH Publishers:
    - SSH Server:
      * Name: server
      * Transfer Set:
        + Exec command : rm -rf /srv/datos/wildfly/standalone/deployments/app\*
      * Transfer Set:
        + Source files: app\*.war
        + Remote directory: /srv/datos/wildfly/standalone/deployments
        + Exec command: service wildfly restart

#### Acciones para ejecutar después

Apartado donde indicaremos subtareas a realizar una vez se ejecute la tarea.

Aquí solo notificaremos del resultado de la tarea.

Ejemplo:

* Notificación por correo
  + Destinatarios: angel.yague@satec.es
  + Enviar correo para todos las ejecuciones con resultado inestable: check

# Pipeline Frontend

### SCM

#### Configurar el origen del código fuente

Ejemplo:

* Subversion
* Modulo:
  + Repository URL: svn://svn.aplicaciones.satec.es:33690/ANGOLA\_TELECOM\_OSS\_INVyPROV/trunk/3InventarioFrontEnd/trunk/1Código
  + Credentials: none
  + Repository depth: infinty
  + Ignore externals: check
* Check-out Strategy: Use ‘svn update’ as much as possible
* Navegador del repositorio: (Auto)

#### Disparadores de ejecuciones

Ejemplo:

* Ejecutar periódicamente: H H(20-23) 1,10,20 \* \*

#### Acciones para ejecutar después

Ejemplo:

* Guardar los archivos generados
  + Ficheros para guardar: \*/\*\*
* Almacenar firma de ficheros para poder hacer seguimiento
  + Almacenar la firma de los ficheros: \*/\*\*
* Ejecutar otros proyectos
  + Proyectos a ejecutar: quality (Trigger only if build is stable)

### QUALITY

#### Configurar el origen del código fuente

Ejemplo:

* Ninguno

#### Disparadores de ejecuciones

Ejemplo:

* Ejecutar siempre que cualquier 'SNAPSHOT' de los que dependa sea creado: check

#### Entorno de ejecución

Apartado de configuración del entorno en el cual se ejecutará la tarea.

En nuestro caso son tareas que se tienen que ejecutar en un entorno Node JS usando bower, npm y grunt.

Ejemplo:

* Provide Node & npm bin/ folder to PATH
  + NodeJS Installation: Node JS v4.2.6
  + npmrc file: - use system defult –

#### Ejecutar

Donde lanzaremos las diversas tareas para ejecutar las pruebas de nuestra aplicación.

Ejemplo:

* Copy artifacts from another project
  + Project name: scm
  + Which build: Latest successful build
    - Stable build only: check
  + Artifacts to copy: \*\*/\*
  + Parameter filters: - (Fingerprint Artifacts)
* Ejecutar linea de comandos (shell)
  + Comando:

bower cache clean

npm cache clean

bower install --force

npm install

grunt karma:local

grunt sonarRunner:docker

#### Acciones para ejecutar después

Ejemplo:

* Guardar los archivos generados
  + Ficheros para guardar: \*/\*\*
* Ejecutar otros proyectos
  + Proyectos a ejecutar: build (Trigger only if build is stable)
* Notificación por correo
  + Destinatarios: [angel.yague@satec.es](mailto:angel.yague@satec.es)
    - Enviar correo para todos las ejecuciones con resultado inestable.

### BUILD

#### Configurar el origen del código fuente

Ejemplo:

* Ninguno

#### Entorno de ejecución

Ejemplo:

* Provide Node & npm bin/ folder to PATH
  + NodeJS Installation: Node JS v4.2.6
  + npmrc file: - use system defult –

#### Ejecutar

Donde lanzaremos la construcción del paquete de nuestra aplicación.

Ejemplo:

* Copy artifacts from another project
  + Project name: quality
  + Which build: Latest successful build
    - Stable build only: check
  + Artifacts to copy: \*\*/\*
  + Artifacts not to copy: node\_modules/\*\*
  + Parameter filters: - (Fingerprint Artifacts)
* Ejecutar linea de comandos (shell)
  + Comando:

npm install

grunt build:<profile>

#### Acciones para ejecutar después

Ejemplo:

* Guardar los archivos generados
  + Ficheros para guardar: dist/build/\*.zip
* Ejecutar otros proyectos
  + Proyectos a ejecutar: deploy (Trigger only if build is stable)
* Notificación por correo
  + Destinatarios: [angel.yague@satec.es](mailto:angel.yague@satec.es)
    - Enviar correo para todos las ejecuciones con resultado inestable.

### DEPLOY

#### Configurar el origen del código fuente

Ejemplo:

* Ninguno

#### Ejecutar

Ejemplo:

* Copy artifacts from another project
  + Project name: build
  + Which build: Latest successful build
    - Stable build only: check
  + Artifacts to copy: dist/build/\*\*
  + Parameter filters: - (Flattern directories / Fingerprint Artifacts)
* Send files or execute commands over SSH
  + SSH Publishers:
    - SSH Server:
      * Name: server front
      * Transfer Set:
        + Exec command : rm -rf /srv/datos/apache/\*
      * Transfer Set:
        + Source files: \*.zip
        + Remote directory: /srv/datos/apache
        + Exec command: unzip /srv/datos/apache/\*.zip -d /srv/datos/apache/ && rm -rf /srv/datos/apache/\*.zip

#### Acciones para ejecutar después

Ejemplo:

* Notificación por correo
  + Destinatarios: [angel.yague@satec.es](mailto:angel.yague@satec.es)
    - Enviar correo para todos las ejecuciones con resultado inestable.

SonarQube

Aplicación que nos evaluará el resultado de nuestras pruebas.

Informa sobre código duplicado, estándares de codificación, pruebas unitarias, cobertura de código, complejidad ciclomática, potenciales errores, comentarios, etc.

En un principio se había pensado para Java, acepta otros lenguajes mediante extensiones.