

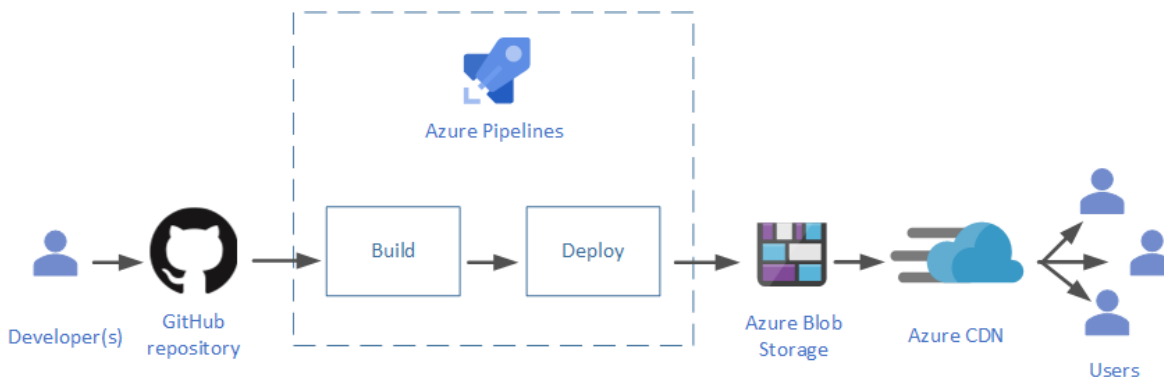
Docente: EDWIN MARAVI PÉREZ

POR: CARRION HUACANI GEAN CARLO

ACTIVIDAD ASINCRÓNICA N° 13

JENKINS

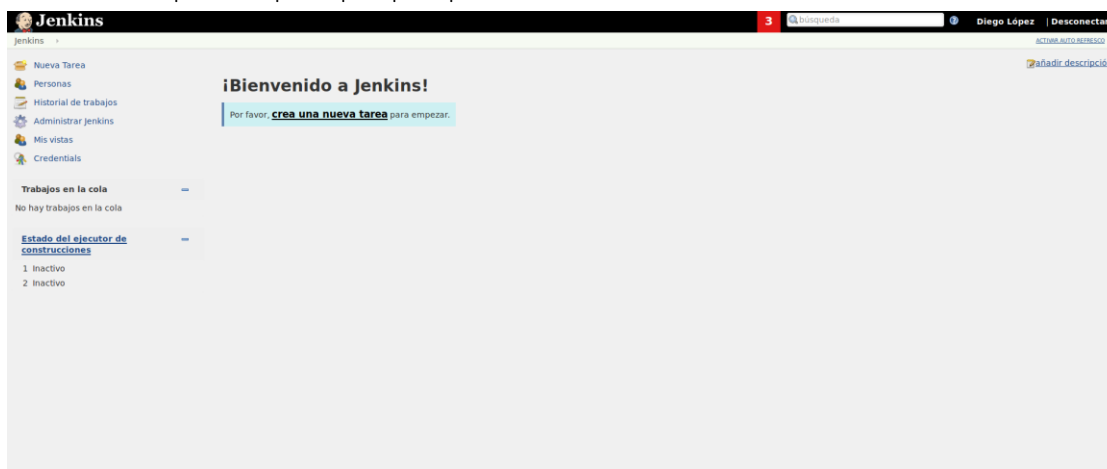
En el siguiente diagrama se describe la canalización de CI/CD que se usa en este front-end de ejemplo



Primeros pasos con Jenkins

Esta es una guía básica de funcionamiento para usuarios con conocimientos básicos o una persona que se dedique en específico al devops.

Tras instalarlo la pantalla principal que aparecerá es como esta:



Jenkins se basa en tareas. Como su nombre indica una tarea es un trabajo o un conjunto de instrucciones que podemos programar para que ocurran con una determinada acción. Si pinchas en crear una nueva tarea:

Enter an item name

» This field cannot be empty, please enter a valid name

Crear un proyecto de estilo libre

Esta es la característica principal de Jenkins, la de ejecutar el proyecto combinando cualquier tipo de repositorio de software (SCM) con cualquier modo de construcción o ejecución (make, ant, mvn, rake, script ...). Por tanto se podrá tanto compilar y empaquetar software, como ejecutar cualquier proceso que requiera monitorización.

Crear un proyecto multi-configuración

Adecuado para proyectos que requieran un gran número de configuraciones diferentes, como testear en multiples entornos, ejecutar sobre plataformas concretas, etc.

Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

OK

Voy a crear una tarea de ejemplo para ver como funciona este sistema. Para ello pongo un nombre a la tarea y selecciono crear proyecto estilo libre para poder configurarlo a mi gusto:

General
Configurar el origen del código fuente
Disparadores de ejecuciones
Entorno de ejecución
Ejecutar
Acciones para ejecutar después.

Proyecto nombre:

Tarea de ejemplo

Descripción

[Plain text]
Visualizar

☐ Desechar ejecuciones antiguas
☐ Esta ejecución debe parametrizarse
☐ Github project

Rebuild options:

☐ Rebuild Without Asking For Parameters
☐ Disable Rebuilding for this job

☐ Throttle builds
☐ Desactivar la ejecución
☐ Lanzar ejecuciones concurrentes en caso de ser necesario

Avanzado...

Configurar el origen del código fuente

☒ Ninguno
☐ Git

Disparadores de ejecuciones

☐ is lejem: desde 'scripts'
☐ re built

Guardar

Apply

Si has elegido plugins distintos al mío seguramente te salgan otras opciones. La primera sección es para la configuración básica como el nombre del proyecto la descripción, cómo vamos a querer las ejecuciones si queremos desechar las antiguas automáticamente, etc.

Más abajo podrás encontrar el origen del código, en mi caso he puesto git y he pegado la dirección del repositorio de Github, como ves, puedes seleccionar el origen de la rama, credenciales y qué disparador vamos a querer. Las ejecuciones pueden ser desde scripts, cuando se haga otro build, periódicamente (en cuyo caso hay que indicar cada cuanto tiempo), disparadores para gitscm y periódicamente consultando el repositorio. Yo voy a escoger consultar repositorio y voy a introducir H/15** para que se ejecute si hay un cambio en el repositorio cada 15 minutos. Si no sabes lo que significa H/15** esta expresión no te preocupes porque te voy a dejar un generador de estas expresiones, para que sepas cómo programar tareas en Jenkins: <https://crontab.guru/>

Configurar el origen del código fuente

☐ Ninguno
☒ Git

Repositories

Repository URL

https://github.com/Prostqui/tabstart.git

Credentials

Prostqui*****

Add

Name

Tabstart repo

Refspec

Add Repository

Branches to build

Branch Specifier (blank for 'any')

*/master

Add Branch

Navegador del repositorio

(Auto)

Additional Behaviours

Adadir

Disparadores de ejecuciones

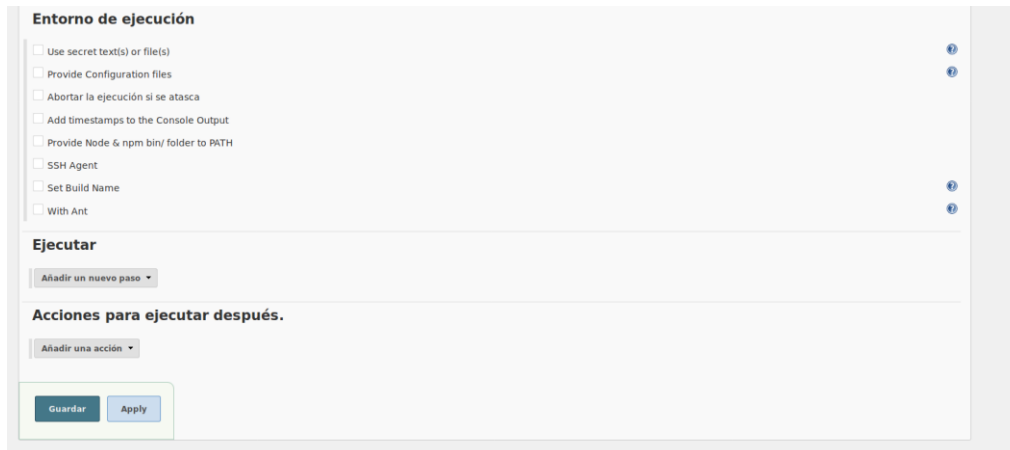
☐ Lanzar ejecuciones remotas (ejem: desde 'scripts')
☐ build after other projects are built
☐ Ejecutar periódicamente

☒ Periodic build trigger for Git (Scm polling)

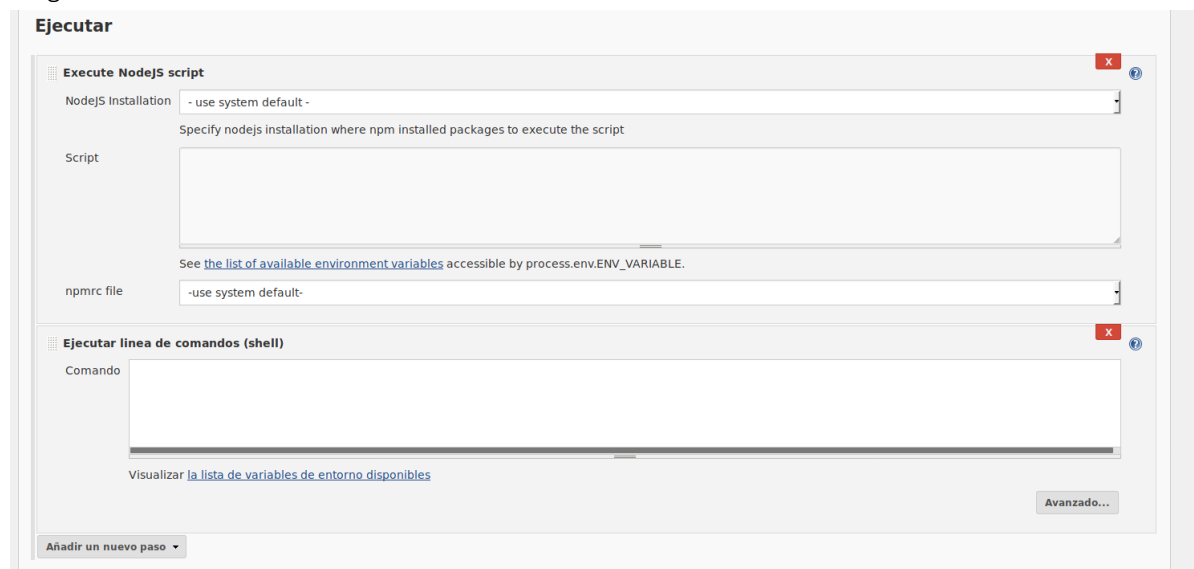
Guardar

Apply

Al final encontramos dónde queremos que se ejecuten las tareas y qué queremos ejecutar antes y después. Desde las opciones de entorno de ejecución podemos configurar cómo se van a ejecutar los build, si vamos a poner ficheros de configuración, por ejemplo, o si vamos a indicar rutas para los PATH de node y npm, o de si vamos a darle un nombre específico a la ejecución (si tenemos instalado el plugin), etc.



Para decidir qué queremos ejecutar hay muchas opciones dependiendo del tipo de proyecto que vayamos a ejecutar, pueden ser comandos de línea de comandos, comandos nodejs, comandos de windows, etc. Como ves se puede adaptar a todo tipo de proyectos y todavía se puede ampliar más con plugins. También se pueden encadenar distintos pasos por ejemplo ejecutar un script de shell y luego uno de windows.



Por último decidir que hacemos después, si notificar usuarios, guardar logs, imprimir resultados, hacer commit, publicar cobertura etc.

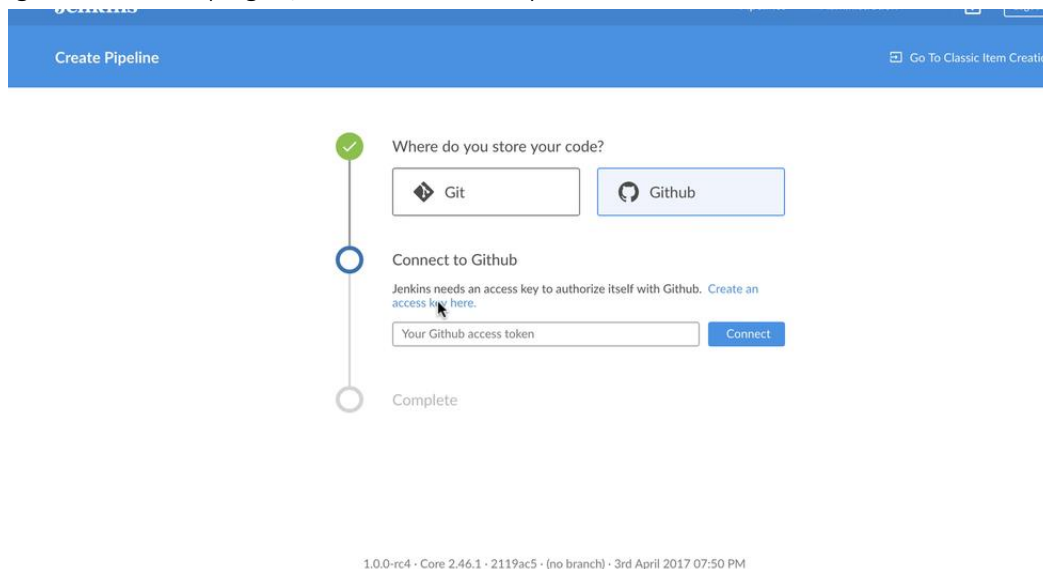
Tras crear la tarea tendrás acceso al panel de control de la tarea, desde aquí podrás seleccionar si lanzarla manualmente, ver los logs, las builds anteriores, volver a configurar la tarea, etc. En mi caso sale el icono del último build en rojo indicando que algo ha fallado:



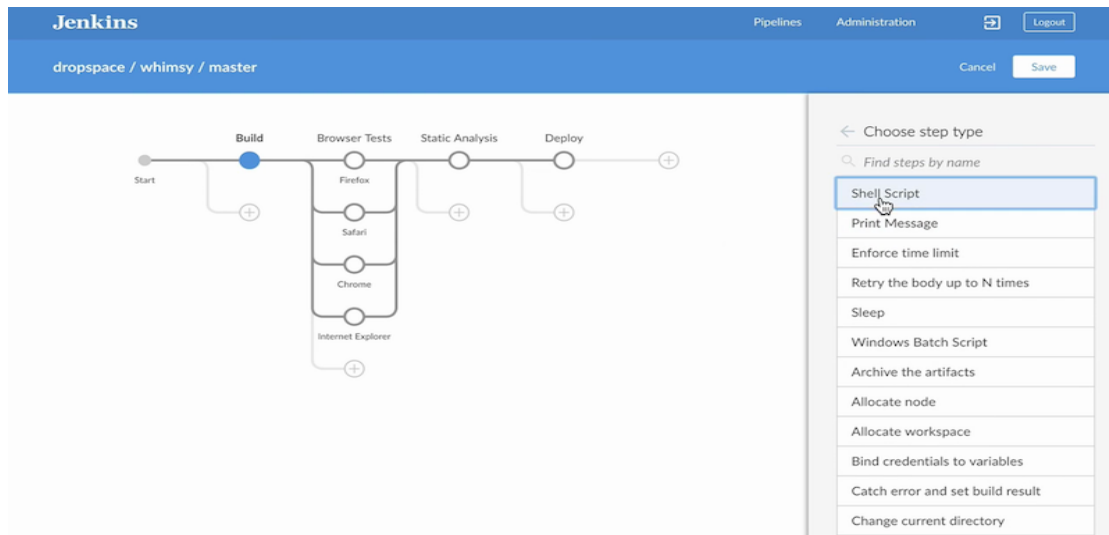
Si hacemos clic sobre el build, podemos ver su estado y la salida de consola para ver exactamente que ha fallado.

Blue ocean el mejor plugin para Jenkins para crear pipelines

Con esto ya seríamos capaces de tener tareas automatizadas funcionando pero como la creación de tareas es algo tediosa, hay un plugin muy famoso el cual recomiendo encarecidamente que se llama Blue Ocean. Este plugin sirve para crear pipelines al completo gestionando mucho mejor la conexión con el repositorio. Para instalarlo te tienes que dirigir a la configuración de Jenkins > Administración de plugins > Todos los plugins, buscas Blue ocean y lo instalas.



Como ves la interfaz de usuario es mucho más amigable y más pensada para todo el mundo. Desde aquí vas a poder configurar los pipelines mucho más fáciles. Además, te permite seleccionar mejor cuando saltan los builds, por ejemplo, al crear un pull request, al aceptarlo, al crear una rama, etc, que hasta ahora no lo habíamos hecho.



Al crear todo el sistema de integración continua es un complejo, si se tiene un proyecto muy pequeño no valdría mucho la pena el esfuerzo de montar todo, pero a la larga viene muy bien para no perder tanto tiempo al haciendo los builds, despliegues, etc, de forma manual. Para los proyectos open source también viene bien para restringir cada merge. Si el merge no cumple ciertos requisitos de estilo o no pasa los tests, el merge no se puede hacer. La integración continua es perfecta para tener un control de lo que pasa en el proyecto, para saber cómo de estable es un proyecto y para avisar a la gente cuando un build a producción ha salido mal.