

Descripción general del proyecto

El proyecto va a consistir en un ciclo completo de integración continua y despliegue continuo de una aplicación (CI/CD).

Para eso vamos a tener que preparar primero nuestra maquina con las herramientas necesarias y luego automatizar las tareas oportunas.

Para poder entender un poco mejor lo que se quiere hacer vamos a explicar en qué consiste la metodología de DevOps.

Que es DevOps y porque surge:

«DevOps es un acrónimo inglés de development (desarrollo) y operations (operaciones), que se refiere a una metodología de desarrollo de software que se centra en la comunicación, colaboración e integración entre desarrolladores de software y los profesionales de sistemas en las tecnologías de la información (IT)».

La principal característica del movimiento DevOps es defender enérgicamente la automatización y el monitoreo en todos los pasos de la construcción del software, desde la integración, las pruebas, la liberación hasta la implementación y la administración de la infraestructura. Es una respuesta a la interdependencia del desarrollo de software y las operaciones IT. Su objetivo es ayudar a una organización a producir productos y servicios software más rápidamente, de mejor calidad y a un coste menor.

Se divide en las siguientes fases:

Fase 1: el plan

La primera fase del ciclo se basa en habilitar los mecanismos necesarios para dar la posibilidad a cualquier usuario de aportar de forma continua sus ideas y que estas puedan ser transformadas en requerimientos u objetivos, siendo estos priorizados e incluidos en próximas iteraciones en forma de historias de usuario.

Fase 2: build

¿Cuántas veces habremos oído a un desarrollador la afirmación “Funcionaba en mi máquina”? Normalmente esta afirmación siempre se escucha después de ver

cómo falla el software cuando otro desarrollador coge esta funcionalidad para evolucionarla o peor aún, cuando se pasa a un entorno de integración el código que se suponía funcionando y este no cumple su función. DevOps ayuda a la construcción de software propiciando mecanismos que sean capaces de favorecer la creación de entornos de desarrollo repetibles, utilizando por ejemplo tecnologías como Docker.

Fase 3: integración continua

La integración continua (CI) es una técnica que tiene como objetivo detectar los posibles problemas que pueda tener el software de forma temprana, permitiendo así solucionarlos antes de que sea demasiado tarde o la solución implique un gran cambio en la aplicación o componente. Para esta labor es conveniente realizar de forma periódica compilación del código, ejecución de tests unitarios, revisión de la calidad del código y detección de vulnerabilidades. Los resultados de las acciones anteriores se pueden exponer mediante herramientas apropiadas que permitan al desarrollador visualizarlas de una forma ágil, como por ejemplo Jenkins.

Fase 4: despliegue

Uno de los grandes objetivos de DevOps es facilitar el paso de un software que se está desarrollando a una versión funcional. En una metodología tradicional, se requiere de una intervención manual desde la construcción de los artefactos hasta el paso de los mismos a producción. DevOps proporciona técnicas que permiten automatizar esta transición, pero también para la subsanación de posibles “catástrofes” que puedan ocurrir.

- Programación de despliegues. Existen herramientas que permiten programar los despliegues, organizando las diferentes historias de usuario en versiones, lo que permite además disponer de una trazabilidad completa de qué y cuándo se despliega en cada entrega.
- Compatibilidad de componentes. Aseguramiento de que todos los componentes son compatibles unos con los otros. Cuando hablamos de componentes no solo nos referimos a software, sino que también hay que poner en la ecuación a sistemas externos, hardware o requerimientos de arquitectura.
- Transición entre entornos. Es muy importante mantener la integridad mientras avanza el software entre los diferentes entornos: desarrollo, testing, staging y producción. No es nada raro que el software sufra pequeños ajustes durante esta transición, por lo que es fundamental garantizar que se siga manteniendo la integridad de la versión.

- Solución de desastres. Aunque las técnicas DevOps ayudan a evitar que llegue una versión inestable al entorno productivo, siempre existe una pequeña posibilidad de que se encuentre un fallo, el cual haga que sea necesario realizar un rollback al último estado estable. Dada la automatización de la que disponen las herramientas de despliegue, es posible recuperar de una forma muy rápida el correcto funcionamiento del aplicativo.

Fase 5: operación

Una vez una versión de software sea funcional, es fundamental realizar un seguimiento de la misma:

- Monitorización del rendimiento de la aplicación y servidor. Para ello hay que realizar de forma automatizada la lectura de parámetros que permitan dar la voz de alarma en caso de fallo.
- Seguimiento de problemas, incidentes y cambios. Una comunicación fluida entre las diferentes personas es fundamental para poder identificar y resolver los posibles problemas que se vayan encontrando.

Conclusión

Adoptar herramientas de Integración Continua y automatización no es suficiente para asegurar la calidad del desarrollo del software, se necesita también un cambio de cultura o filosofía que permita adoptar y utilizar de forma correcta todas las herramientas y lo que es más importante, una comunicación constante entre todas las implicados en el desarrollo y puesta en producción de un aplicativo.

Para que quede claro lo que se va a realizar voy a definir las siglas del ciclo

CI/CD:

CI equivale a integración continua o continuous integration.

CD por su parte, puede equivaler a entrega continua (continuous delivery) o despliegue continuo (continuous deployment).

Sin embargo, conviene apuntar que aunque continuous delivery y continuous deployment tengan cosas en común, son diferentes tal y como veremos a continuación.

CI (continuous integration) o integración continua tiene como objetivo principal preparar y servir cuanto antes una release del Software de forma rápida, sencilla y ágil.

Los desarrollares mezclan los cambios de código en la rama o branch principal tantas veces como puedan.

Los cambios son validados generando una build para permitir ejecutar después los tests contra esa build.

De hecho, uno de los pilares fundamentales de la integración continua tiene una estrecha vinculación con la automatización de tests, asegurando que la aplicación no está rota.

Es lo que seguramente hayas oído comentar muchas veces como “se ha roto la build”, un argot bastante frecuente en desarrollo de Software e integración continua.

Por todo esto, los tests requieren una importancia relevante para que el código que escribimos sea confiable y que la integración continua sea un elemento importante dentro de la calidad de la funcionalidad del código escrito.

Obviamente, necesitaremos un servidor que realice todo este proceso de forma automatizada cada vez que un desarrollador haga un commit de sus cambios de código.

Esto nos ahorrará mucho tiempo, aunque como es lógico también, un servicio de estas características tendrá un coste asociado.

CD (continuous delivery) o entrega continua es el siguiente paso de la integración continua.

De hecho, en muchos círculos se la conoce como una extensión de la integración continua.

Su objetivo no es otro que el de proporcionar una manera ágil y fiable de poder entregar o desplegar los nuevos cambios a los clientes.

Esto se consigue automatizando el proceso de despliegue, aunque esto ocurrirá al hacer clic sobre la acción de despliegue.

Podremos indicar también cuando llevar a cabo el proceso de despliegue en producción.

En algunas empresas, el despliegue se realiza con dos ó tres despliegues por día.

En otras es semanalmente o quincenalmente, y en otras incluso mensualmente. Todo depende de los propósitos de negocio de la empresa, complejidades, criticidades, etc.

No obstante, el mantra de entrega continua invita a que el despliegue en producción sea lo más rápido posible en el tiempo.

CD (continuous deployment) o despliegue continuo tiene mucha relación con la entrega continua, sin embargo va más allá.

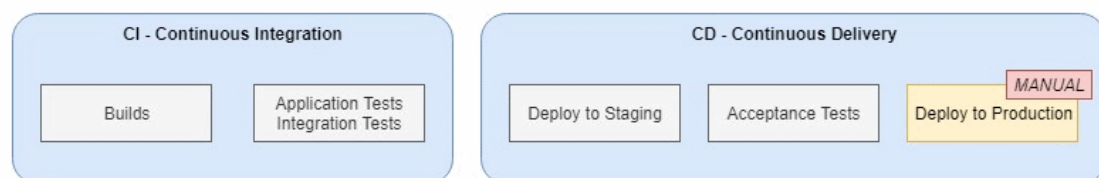
El objetivo a diferencia de la entrega continua es que no exista intervención humana a la hora de realizar el despliegue de nuestro Software en producción. Esta es quizás la clave y diferencia más importante con respecto a la entrega continua.

Para lograr este propósito, el pipeline de producción tiene una serie de pasos que deben ejecutarse en orden y forma satisfactoria.

Si alguno de esos pasos no finalizan de forma esperada, el proceso de despliegue no se llevará a cabo.

Hay empresas de primer orden en el mercado que sigue esta filosofía.

El resumen de CI + Continuous Delivery se puede ver en la siguiente imagen:



El resumen de CI + Continuos Deployment se puede ver en la siguiente imagen:



Como conclusión de lo que se va a hacer, es coger una aplicación de ejemplo java, mediante Spring boot que es una solución para el framework Spring de Java. Para ello, Spring Boot proporciona la estructura básica configurada del proyecto, que incluye las pautas para usar el marco y todas las bibliotecas de terceros relevantes para la aplicación, lo que nos allana el camino para comenzar a desarrollarla lo más rápidamente posible.

Una vez con nuestra app añadiremos los ficheros para realizar el ciclo de integración y despliegue continuo.

Aspectos funcionales y de diseño de la aplicación.

Primero vamos a preparar el ordenador con las herramientas necesarias.

Para eso voy a definir las herramientas utilizadas, para conocer un poco mejor su utilidad:

Kubernetes: es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Orquesta la infraestructura de cómputo, redes y almacenamiento para que las cargas de trabajo de los usuarios no tengan que hacerlo.

Los conceptos clave de Kubernetes son:

- **Pods:** Conjunto de contenedores y volúmenes.
- **Replication Controllers:** Gestor de Pods que asegura que están levantadas las réplicas y permite escalar de forma fácil. Una **réplica** es una copia exacta de un Pod. Levanta Pods en caso de fallos o reinicios.
- **Service:** Define como acceder a un grupo de Pods.

Jenkins: básicamente es una aplicación que supervisa ejecuciones de trabajos repetitivos, como la construcción de un proyecto de software y la ejecución de pruebas automáticas del mismo mediante integración continua y facilita ciertos aspectos de la entrega continua. Es decir, es un sistema de automatización.

Que se realiza mediante una pipeline, que es un conjunto de acciones conectadas entre sí, de modo que al ejecutarse una de ellas, esta da paso a la siguiente y así sucesivamente hasta completar todo el proceso.

Docker: es una herramienta open source que nos permite crear contenedores para poder empaquetar entornos y aplicaciones que posteriormente podremos desplegar en cualquier sistema que disponga de esta tecnología.

Para utilizar Docker debemos dominar dos conceptos básicos:

- **Imágenes:** se podrían ver como un componente estático, pues no son más que un sistema operativo base, con un conjunto de aplicaciones empaquetadas.
- **Contenedor:** es la instanciación o ejecución de una imagen. Se pueden ejecutar varios contenedores a través de una misma imagen.

Git: son todas las herramientas que nos permiten hacer las modificaciones en nuestro código y hacen que sea más fácil la administración de las distintas versiones de cada producto desarrollado.

Ya con esto explicado vamos a comenzar nuestro ciclo CI/CD.

Para ello necesitamos primero instalarnos todas las herramientas necesarias.

Instalamos minikube, que es una herramienta que administra máquinas virtuales en donde corre un cluster (conjuntos de servidores unidos entre sí y que se comportan como si fuesen un único servidor), o mejor dicho una instancia de Kubernetes en un solo nodo en nuestra máquina.

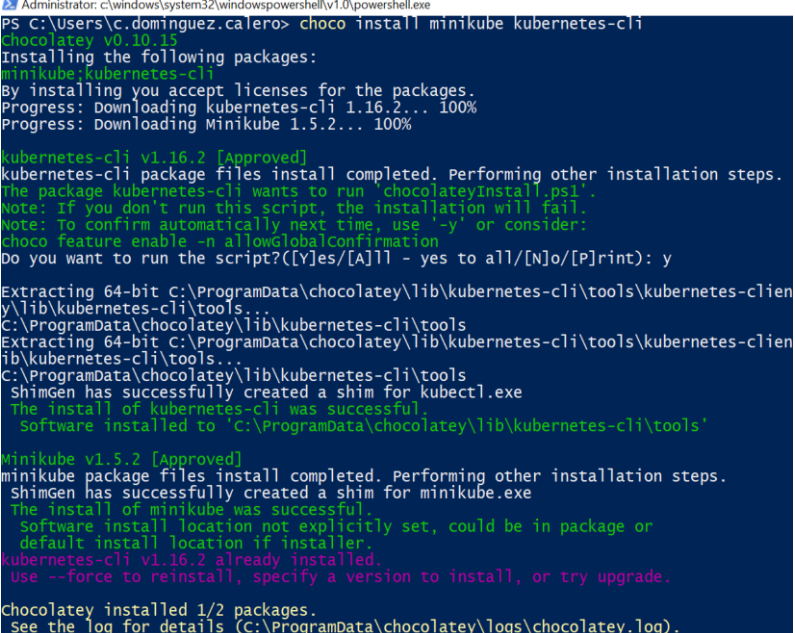
Para la instalación, primero vamos a instalar Chocolatey, que es un Gestor de Paquetes para el Sistema Operativo Windows (como apt-get o yum, pero para Windows), con esta herramienta podrás instalar múltiples aplicaciones y herramientas usando solamente PowerShell, de esta forma la instalación será más rápida.

Para ello abrimos powershell como administrador y ponemos el siguiente comand:

```
Set-ExecutionPolicy Bypass -Scope Process -Force; ` iex ((New-Obj System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
```

Ya podemos instalar minikube:

```
choco install minikube kubernetes-cli
```



```
Administrator: c:\windows\system32\windowspowershell\v1.0\powershell.exe
PS C:\Users\c.dominguez.calero> choco install minikube kubernetes-cli
chocolatey v0.10.15
Installing the following packages:
minikube,kubernetes-cli
By installing you accept licenses for the packages.
Progress: Downloading kubernetes-cli 1.16.2... 100%
Progress: Downloading Minikube 1.5.2... 100%
kubernetes-cli v1.16.2 [Approved]
kubernetes-cli package files install completed. Performing other installation steps.
The package kubernetes-cli wants to run 'chocolateyinstall.ps1'.
Note: If you don't run this script, the installation will fail.
Note: To confirm automatically next time, use '-y' or consider:
choco feature enable -n allowGlobalConfirmation
Do you want to run the script?([Y]es/[A]ll - yes to all/[N]o/[P]rint): y
Extracting 64-bit C:\ProgramData\chocolatey\lib\kubernetes-cli\tools\kubernetes-clien
y\lib\kubernetes-cli\tools...
C:\ProgramData\chocolatey\lib\kubernetes-cli\tools
Extracting 64-bit C:\ProgramData\chocolatey\lib\kubernetes-cli\tools\kubernetes-clien
ib\kubernetes-cli\tools...
C:\ProgramData\chocolatey\lib\kubernetes-cli\tools
ShimGen has successfully created a shim for kubect1.exe
The install of kubernetes-cli was successful
Software installed to 'C:\ProgramData\chocolatey\lib\kubernetes-cli\tools'
minikube v1.5.2 [Approved]
minikube package files install completed. Performing other installation steps.
ShimGen has successfully created a shim for minikube.exe
The install of minikube was successful.
Software install location not explicitly set, could be in package or
default install location if installer.
kubernetes-cli v1.16.2 already installed.
Use --force to reinstall, specify a version to install, or try upgrade.
chocolatey installed 1/2 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
```

Una vez Minikube ha terminado de instalarse, cierro la sesión cliente actual y reinicio.

Minikube debería haberse añadido a mi \$PATH automáticamente.

Arrancamos la minikube con el comando:

Minikube start

Ahora vamos a crear un pod de Jenkins.

Como ya tengo kubernetes, en mi Docker local, para confirmar que me encuentro en el cluster de minikube, pongo este comando:

kubectl config -h

Que me da las
diferentes opciones



```
Administrator: c:\windows\system32\windowspowershell\v1.0\powershell.exe
1. If the --kubeconfig flag is set, then only that file is loaded. The flag may only be set once and no merging takes place.
2. If $KUBECONFIG environment variable is set, then it is used a list of paths (normal path delimiting rules for your system). These paths are merged. When a value is modified, it is modified in the file that defines the stanza. When a value is created, it is created in the first file that exists. If no files in the chain exist, then it creates the last file in the list.
3. Otherwise, ${HOME}/.kube/config is used and no merging takes place.

Available Commands:
  current-context  Displays the current-context
  delete-cluster   Delete the specified cluster from the kubeconfig
  delete-context   Delete the specified context from the kubeconfig
  get-clusters     Display clusters defined in the kubeconfig
  get-contexts     Describe one or many contexts
  rename-context   Renames a context from the kubeconfig file.
  set              Sets an individual value in a kubeconfig file
  set-cluster      Sets a cluster entry in kubeconfig
  set-context      Sets a context entry in kubeconfig
  set-credentials  Sets a user entry in kubeconfig
  unset            Unsets an individual value in a kubeconfig file
  use-context      Sets the current-context in a kubeconfig file
  view            Display merged kubeconfig settings or a specified kubeconfig file

Usage:
  kubectl config SUBCOMMAND [options]

Use "kubectl <command> --help" for more information about a given command.
Use "kubectl options" for a list of global command-line options (applies to all commands).
```

En este caso para verificar que me encuentro en el cluster de minikube utilizo:

kubectl config current-context

Una vez confirmado empezamos con Jenkins:

Instalamos primero Helm (es una herramienta que se utiliza para la gestión de paquetes Kubernetes) :

choco install kubernetes-helm

He tenido problemas con el comando Helm debido al tiller, (Tiller es la pieza que actúa como servidor de Helm y es desplegada en Kubernetes) lo he solucionado con:

helm init --wait

Ya con todo en marcha volvemos a la instalación de Jenkins, para eso usamos:

helm install --name jenkins --set rbac.create=false --set master.adminUser=admin --set master.adminPassword=admin stable/jenkins

Asigno automáticamente usuario y contraseña, porque me daba problemas en el login default de Jenkins, para ello me he apoyado en

<https://github.com/helm/charts/tree/master/stable/jenkins>

En este caso nos
saltamos el paso uno,
ya que hemos
generado
automáticamente el
usuario y contraseña.



```
Administrator: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
No resources found.
PS C:\Users\c.dominguez.calero> helm install --name jenkins --set rbac.create=false --set master.adminUser=admin --set
word=admin stable/jenkins
Error: could not find tiller
PS C:\Users\c.dominguez.calero> helm init --wait
$HELM_HOME has been configured at C:\Users\c.dominguez.calero\.helm.

Tiller (the Helm server-side component) has been installed into your Kubernetes cluster.

Please note: by default, Tiller is deployed with an insecure 'allow unauthenticated users' policy.
To prevent this, run 'helm init' with the --tiller-tls-verify flag.
For more information on securing your installation see: https://docs.helm.sh/using_helm/#securing-your-helm-installati
PS C:\Users\c.dominguez.calero> helm install --name jenkins --set rbac.create=false --set master.adminUser=admin --set
word=admin stable/jenkins
NAME: jenkins
LAST DEPLOYED: Wed Nov 13 09:54:32 2019
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/ConfigMap
NAME AGE
jenkins 1s
jenkins-tests 1s

==> v1/Deployment
NAME AGE
jenkins 1s

==> v1/PersistentVolumeClaim
NAME AGE
jenkins 1s

==> v1/Pod(related)
NAME AGE
jenkins-588884c78-vh77z 1s

==> v1/Service
NAME AGE
jenkins 1s
jenkins-agent 1s

==> v1/ServiceAccount
NAME AGE
jenkins 1s

NOTES:
1. Get your 'admin2' user password by running:
printf $(kubectl get secret --namespace default jenkins -o jsonpath='{.data.jenkins-admin-password}' | base64 --decode);echo
2. Get the Jenkins URL to visit by running these commands in the same shell:
export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/component=jenkins-master" -l "app.kubernetes.io/i
nstance=jenkins" -o jsonpath='{.items[0].metadata.name}')
echo http://127.0.0.1:8080
kubectl --namespace default port-forward $POD_NAME 8080:8080
3. Login with the password from step 1 and the username: admin2

For more information on running Jenkins on Kubernetes, visit:
https://cloud.google.com/solutions/jenkins-on-container-engine
```

```
kubectl port-forward deploy/jenkins
8080
```

Genera una tubería que nos daría acceso con localhost:8080.

Es un poco engorroso porque mientras necesitamos que esté en marcha, necesitamos un powershell abierto solo para eso.

Con LoadBalancer: expone el servicio externamente usando el equilibrador de carga de un proveedor de la nube. Los servicios NodePort y ClusterIP, a los que se enrutará, se crean automáticamente.

```
kubectrl expose deployment jenkins --port=8080 --type=LoadBalancer
```

Yo en este caso me voy a crear un servicio nodePort que expone el servicio en la IP de cada nodo en un puerto estatico (NodePort). Se crea automáticamente un servicio ClusterIP, al que se enrutará el servicio NodePort, desde fuera del clúster, solicitando <NodeIP>:<NodePort>.

```
kubectl expose deployment jenkins --name=jenkins --port=8080 --target-port=8080 --type=NodePort
```

Con este comando
Podemos acceder a
Jenkins en nuestro
equipo mediante la ip
del cluster

```
minikube ip
```

[illegible]

```
Administrator: C:\Windows\system32\WindowsPowerShell\v1.0\powershell.exe
PS C:\Users\c.dominguez.calero> kubectl expose deployment jenkins --port=8080 --type=LoadBalancer
service "jenkins" exposed
PS C:\Users\c.dominguez.calero> kubectl get svc

```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
jenkins	LoadBalancer	10.96.145.250	<pending>	8080:31081/TCP	8s
jenkins-agent	ClusterIP	10.101.17.229	<none>	50000/TCP	1h
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	14d

```

PS C:\Users\c.dominguez.calero> minikube ip
192.168.240.158
PS C:\Users\c.dominguez.calero> kubectl delete svc jenkins
service "jenkins" deleted
PS C:\Users\c.dominguez.calero> kubectl expose deployment jenkins --name=jenkins --port=8080 --target-port=8080 --type=NodePort
service "jenkins" exposed
PS C:\Users\c.dominguez.calero> kubectl get svc

```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
jenkins	NodePort	10.108.196.184	<none>	8080:32740/TCP	12s
jenkins-agent	ClusterIP	10.101.17.229	<none>	50000/TCP	1h
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	14d

```

PS C:\Users\c.dominguez.calero>

```

Como nos genera un puerto aleatorio entre 30000 y 32767, para ver el puerto que se nos ha asignado usamos

kubectl set svc.

Vamos a nuestro navegador
y añadimos la ip de
minikube y nodePort →



Para ver los detalles de nuestro servicio podemos usar:

kubectl describe svc Jenkins

y ver el yaml del servicio creado:

kubectl get svc jenkins -o yaml

Para entrar en el contenedor de Jenkins:

kubectl exec -ti jenkins-58f5b8cf45-wg7zb bash

Ejemplo comprobar que tiene acceso a internet:

ping 8.8.8.8

Acceder a docker mediante minikube

Vamos a probar que podemos acceder a Dockerhub desde la máquina, ya que nuestro pipeline de Jenkins lo que va a hacer es crear una imagen que tenemos que guardar, en nuestro repositorio online.

Primero ponemos el comando:

minikube docker-env

Nos da la ayuda para poder usar el comando docker en nuestra maquina:

minikube docker-env | Invoke-Expression

Y comprobamos que funciona con un simple *docker ps*

```
PS C:\Users\c.dominguez.calero> minikube docker-env
$Env:DOCKER_TLS_VERIFY = "1"
$Env:DOCKER_HOST = "tcp://192.168.240.158:2376"
$Env:DOCKER_CERT_PATH = "C:\Users\c.dominguez.calero\.minikube\certs"
# Run this command to configure your shell:
# & minikube docker-env | Invoke-Expression
PS C:\Users\c.dominguez.calero> minikube docker-env | Invoke-Expression
PS C:\Users\c.dominguez.calero> docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
NAMES
111559a7a378        jenkins/jenkins    "/sbin/tini -- /usr/..." 2 hours ago         Up 2 hours
k8s_jenkins-jenkins-5c564d7df7-nbn8f_default_58b98374-bd84-4cbd-8e9d-f796118c7310_1
64754bbaa173        c2c9a0406787       "kube-apiserver --ad..." 2 hours ago         Up 2 hours
```

También se podría con ssh, como se ve en la imagen de abajo.

```
kubecfg: configured
PS C:\Users\c.dominguez.calero> minikube ssh --native-ssh=true

minikube

$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
NAMES
1cf7a4a6d2b7        6802d83967b9       "/dashboard --insecu..." 3 minutes ago       Up 3 minutes
k8s_kubernetes-dashboard_kubernetes-dashboard-57f4cb4545-t77j7_kubernetes-dashboard_677347a4-f0d7-4520-2c7d57ac4360
2c7d57ac4360        jenkins/jenkins    "/sbin/tini -- /usr/..." 3 minutes ago       Up 3 minutes
k8s_jenkins-jenkins-5c564d7df7-nbn8f_default_58b98374-bd84-4cbd-8e9d-f796118c7310_2
```

Para salir seria con (Ctrl+D).

Vamos a crear una imagen de prueba, en este caso con nginx, que es un servidor web.

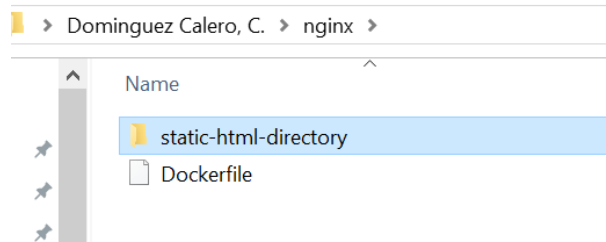
Creamos Dockerfile, archivo de texto plano que contiene las instrucciones necesarias para automatizar la creación de una imagen:

```
FROM nginx # Descarga la imagen de nginx

COPY static-html-directory /usr/share/nginx/html # Copia el html de la carpeta static-html-directory

EXPOSE 80 # Indica los puertos TCP/IP los cuales se pueden acceder a los servicios del contenedor
```


Y dentro de la carpeta, donde se encuentra Dockerfile, añadimos otra carpeta, que dentro tenga un html con un simple Hello!!!!



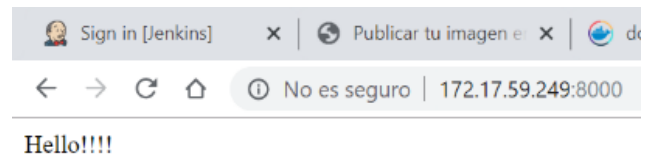
Construimos la imagen

```
docker build -t cecilia/nginx:nginximage
```

Ejecutamos para ver que funciona correctamente

```
docker run --name cecilia/nginx:nginximage -p 8000:80 -d cecilia/nginx:nginximage
```

Ip de minikube y el puerto expuesto



Ahora vamos a subir la imagen a dockerhub

Utilizo docker tag para generar una variante de mi imagen con ese nombre.

```
docker tag cecilia/nginx:nginximage ceciliadominguez/nginx:nginximage
```

Si ejecuto docker images, no ha modificado la imagen original, sino que

simplemente se ha

creado una especie de

alias sobre la misma

imagen, ya que el id de

la imagen, tanto de la

```
Administrator: c:\windows\system32\windowspowershell\v1.0\powershell.exe
gcr.io/k8s-minikube/storage-provisioner v1.8.1 4689081edb10 2 years ago 80.8MB
PS C:\Users\c.dominguez.calero\.docker\nginx> docker build -t cecilia/nginx:nginximage .
Sending build context to Docker daemon 3.584kB
Step 1/3 : FROM nginx
--> 540a289bab6e
Step 2/3 : COPY static-html-directory /usr/share/nginx/html
--> Using cache
--> 68808dc4ceea
Step 3/3 : EXPOSE 80
--> Using cache
--> 7ee5679191fb
Successfully built 7ee5679191fb
Successfully tagged cecilia/nginx:nginximage
SECURITY WARNING: You are building a Docker image from windows against a non-windows Docker host. All files and di
ild context will have '-rwxr-xr-x' permissions. It is recommended to double check and reset permissions for sensit
ries.
PS C:\Users\c.dominguez.calero\.docker\nginx> docker rm some-content-nginx
some-content-nginx
PS C:\Users\c.dominguez.calero\.docker\nginx> docker tag cecilia/nginx:nginximage ceciliadominguez/nginx:nginximag
PS C:\Users\c.dominguez.calero\.docker\nginx> docker push ceciliadominguez/nginx:nginximage
The push refers to repository [docker.io/ceciliadominguez/nginx]
4ab2dfe96387: Pushed
a89b8f05da3a: Layer already exists
6eaad811af02: Layer already exists
b67d19e65ef6: Layer already exists
nginximage: digest: sha256:762db9211838f7527c5344b1da559591f0f448735b35ca950a4a4f52b48f372 size: 1155
```


original como de esta, es el mismo.

Ahora que ya cumplimos los requerimientos *docker push* para subir la imagen:

docker push ceciliadominguez/nginx:nginximage

Borramos la imagen

docker rmi -f 7ee5679191fb

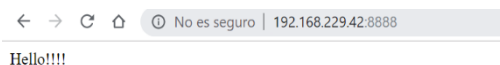
Arrancamos nuestra imagen

docker run -p 8888:80 ceciliadominguez/nginx:nginximage

Ahora es mi imagen la que está la nube, por lo que inicialmente no la encontrará en local y entonces decidirá ir a buscarla a Docker Hub, que es el registro por defecto.

```
Administrator: c:\windows\system32\windowspowershell\v1.0\powershell.exe
PS C:\Users\c.dominguez.calero> docker stop f748474fb6f8
f748474fb6f8
PS C:\Users\c.dominguez.calero> docker rmi -f 7ee5679191fb
Untagged: ceciliadominguez/nginx:nginximage
Deleted: sha256:7ee5679191fbacbf43515ca9da4cf7aa6cbd68e8ef2f4fcc9525e82f3ab655cd
PS C:\Users\c.dominguez.calero> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
gcr.io/kubernetes-helm/tiller    v2.16.0            aedca6b7890b       8 days ago         91.2MB
jenkins/jenkins                lts                 fac78e370c0b       2 weeks ago        568MB
k8s.gcr.io/kube-proxy           v1.16.2            8454cbe08dc9       4 weeks ago        86.1MB
k8s.gcr.io/kube-apiserver        v1.16.2            c2c9a0406787       4 weeks ago        217MB
k8s.gcr.io/kube-scheduler        v1.16.2            ebac1ae204a2       4 weeks ago        87.3MB
k8s.gcr.io/kube-controller-manager v1.16.2            6e4bffa46d70       4 weeks ago        163MB
k8s.gcr.io/etcd                 3.3.15-0           b2756210eeab       2 months ago       247MB
k8s.gcr.io/coredns              1.6.2              bf261d157914       3 months ago       44.1MB
k8s.gcr.io/kube-addon-manager    v9.0.2             bd12a212f9dc       3 months ago       83.1MB
k8s.gcr.io/kube-addon-manager    v9.0               119701e77cbc       10 months ago      83.1MB
k8s.gcr.io/kubernetes-dashboard-amd64 v1.10.1            f9aed6605b81       11 months ago      122MB
k8s.gcr.io/k8s-dns-sidecar-amd64  1.14.13            4b2e93f0133d       14 months ago      42.9MB
k8s.gcr.io/k8s-dns-kube-dns-amd64  1.14.13            55a3c5209c5e       14 months ago      51.2MB
k8s.gcr.io/k8s-dns-dnsmasq-nanny-amd64 1.14.13            6dc8ef8287d3       14 months ago      41.4MB
k8s.gcr.io/pause                3.1                da86e6ba6ca1       23 months ago      742kB
gcr.io/k8s-minikube/storage-provisioner v1.8.1             4689081edb10       2 years ago         80.8MB
PS C:\Users\c.dominguez.calero> docker run -p 8888:80 ceciliadominguez/nginx:nginximage
Unable to find image 'ceciliadominguez/nginx:nginximage' locally
nginximage: Pulling from ceciliadominguez/nginx
8d691f585fa8: Already exists
5b07f4e08ad0: Already exists
abc291867bca: Already exists
e64d46b7436a: Already exists
Digest: sha256:762db9211838f7527c5344b1da559591f0f448735b35caf950a4a4f52b48f372
Status: Downloaded newer image for ceciliadominguez/nginx:nginximage
PS C:\Users\c.dominguez.calero> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ceciliadominguez/nginx    nginximage          7ee5679191fb       21 hours ago       126MB
gcr.io/kubernetes-helm/tiller    v2.16.0            aedca6b7890b       8 days ago         91.2MB
```

Comprobamos que es nuestra imagen:



Ya tenemos nuestra maquina funcionando con Jenkins y Docker, correctamente.

Empezamos nuestra pipeline:

Primero vamos a clonar el código de nuestra app, que esta subido a nuestro repositorio de GitHub, en una nuestra carpeta creada en local proyectoCecilia.

git clone https://github.com/ceciliacalero/proyectoFCT.git proyectoCecilia/

```
PS C:\Users\c.dominguez.calero> git clone https://github.com/ceciliacalero/proyectoFCT.git proyectoCecilia/
Cloning into 'proyectoCecilia'...
remote: Enumerating objects: 214, done.
remote: Counting objects: 100% (214/214), done.
remote: Compressing objects: 100% (145/145), done.
Receiving objects: 100% (214/214), 72.86 KiB | 6.62 MiB/s, done.
Resolving deltas: 100% (88/88), done.
PS C:\Users\c.dominguez.calero> cd proyectoCecilia
PS C:\Users\c.dominguez.calero\proyectoCecilia> ls

Directory: C:\Users\c.dominguez.calero\proyectoCecilia

Mode                LastWriteTime         Length Name
----                -
d-----          02/12/2019     9:27         .deploy
d-----          02/12/2019     9:27         .mvn
d-----          02/12/2019     9:27         src
-a-----          02/12/2019     9:27         5508 Jenkinsfile
-a-----          02/12/2019     9:27         9399 mvnw
-a-----          02/12/2019     9:27         5971 mvnw.cmd
-a-----          02/12/2019     9:27         1694 pom.xml
```

Una vez comprobamos que se ha clonado en nuestro local, usamos:

Git init

Cada vez que generemos un cambio de nuestro código, para que este actualizado en todo momento en GitHub, tenemos que realizar:

Git add .

git commit -m "add jenkinsfile"

git push

O más rápido:

git commit -am "add jenkinsfile" (add y commit, en el mismo comando)

git push

```
Administrator: c:\windows\system32\windowspowershell\v1.0\powershell.exe

On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   .deploy/Dockerfile
        deleted:    secret.yml

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\c.dominguez.calero\New folder\app> git add .
PS C:\Users\c.dominguez.calero\New folder\app> git commit -m "change port"
[master 91aada3] change port
Committer: Dominguez Calero <c.dominguez.calero@accenture.com>
Your name and email address were configured automatically based
on the commit message. You can suppress this message by setting them explicitly: Run the
following command and follow the instructions in your editor to edit
your configuration file:
    git config --global --edit

After doing this, you may fix the identity used for this commit with:
    git commit --amend --reset-author

2 files changed, 1 insertion(+), 9 deletions(-)
 delete mode 100644 secret.yml
PS C:\Users\c.dominguez.calero\New folder\app> git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 448 bytes | 448.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/ceciliacalero/pruebaGit.git
  51d8cc3..91aada3 master -> master
PS C:\Users\c.dominguez.calero\New folder\app> git add .
```

Comandos utiles de git:

git ls-tree --full-tree -r --name-only HEAD (para ver los archivos)

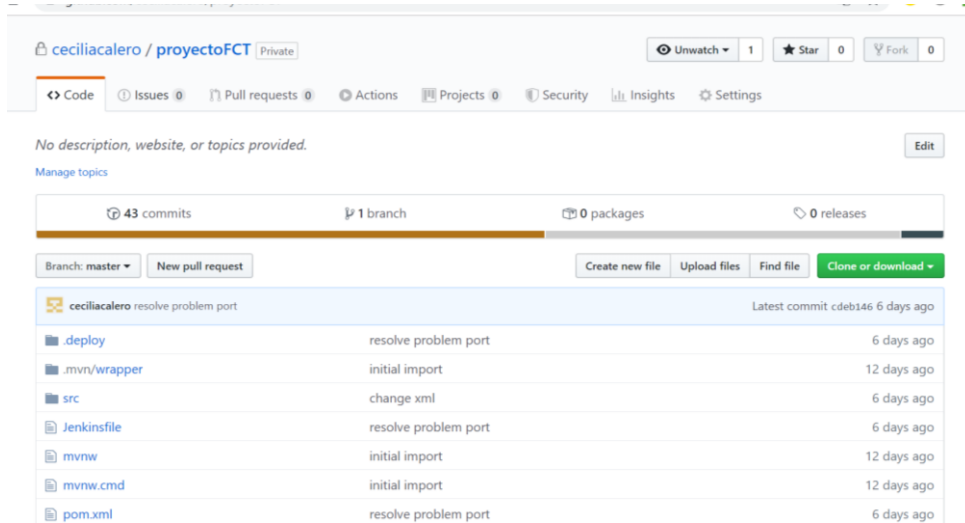
git remote add origin <https://github.com/ceciliacalero/pruebaGit> (Si no has clonado un repositorio ya existente y quieres conectar tu repositorio local a un repositorio remoto)

git pull --rebase origin master (rebase solo utilizar cuando se trabaja con una única rama, ya que borras todas las ramas y dejas la última como master)

git checkout -b cecilia (Crea una nueva rama llamada Cecilia y cámbiate a ella)

git push origin cecilia (Subir la rama a tu repositorio remoto)

Desde GitHub se puede ver todos los cambios y las ramas, como se muestra en la imagen de abajo.

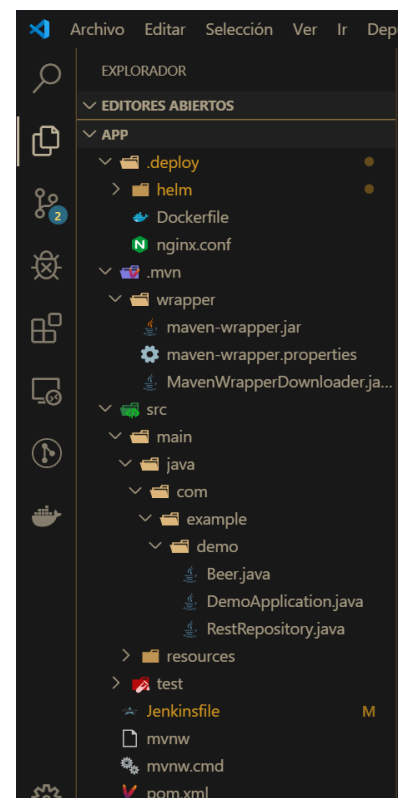


Ya con el código de nuestra app ejemplo, creamos el archivo Jenkinsfile, que es un script basado en Groovy que permite definir un pipeline de Jenkins y DockerFile, que es un archivo de texto plano que contiene las instrucciones necesarias para automatizar la creación de una imagen.

En la carpeta src , se encuentra el código java y los test unitarios .

Pom.xml, es la “unidad” principal de un proyecto Maven. Contiene información acerca del proyecto, fuentes, test, dependencias, versión...

Tenemos una carpeta deploy, en la cual vamos a tener una carpeta con los valores, chart, service y deployment, que se utilizara para que la app se pueda desplegar mediante Kubernetes.



Así queda nuestra pipeline, realizando todas las pruebas hasta que se ha obtenido el resultado deseado.

```
// <c.dominguez.calero@accenture.com>

pipeline {
    agent { /*Crea un pod de kubernetes con contenedor maven, docker y
helm, se usa docker in docker(De manera predeterminada,
el complemento Docker Pipeline se comunicará con un demoni
o Docker local,
al que normalmente se accede a través de /var/run/docker.s
ock.)*/*
    kubernetes {
        label 'pipeline-as-code'
        yaml """
            apiVersion: v1
            kind: Pod
            metadata:
                labels:
                    app: jenkins
            spec:
                containers:
                    - name: maven
                      image: maven:3.6.2-jdk-14
                      command:
                        - cat
                      tty: true
                    - name: docker
                      image: docker:stable-dind
                      command:
                        - cat
                      tty: true
                      volumeMounts:
                        - name: dockersock
                          mountPath: "/var/run/docker.sock"
                    - name: helm
                      image: alpine/helm:3.0.0
                      command:
                        - cat
                      tty: true
                volumes:
                    - name: dockersock
                      hostPath:
                        path: "/var/run/docker.sock"
            """
    }
}

environment {
```

```

        gitBranch      = "${BRANCH_NAME}" //nombre de la rama
        gitCommit      = "${GIT_COMMIT}" //devuelve los
commit realizados
        shortGitCommit = "${gitCommit[0..10]}"
        // this variable get the last tag on the branch
        //release      = sh(returnStdout: true, script: 'git tag | head
-1').trim()
        release        = "0.0.1"
        /*
        * these variables must be configured
        */

        // docker
        dockerfile      = ".deploy/Dockerfile" //ruta dockerfile
        imageName        = "ceciliadominguez/app:${gitBranch}" //nombre de
la imagen
        registry        = "ceciliadominguez/app:${gitBranch}" //registro
de dockerHub
        credentialsId    = 'dockerhub' //credenciales de dockerhub en jenk
ins

        // k8s deploy
        appName          = "app-api-${gitBranch}" //nombre de la app
        appChart          = ".deploy/helm" //ruta de helm
        helmAppVersion    = "1"
    }
    stages {
        stage('Maven Build') {
            steps {
                container('maven') { //descargar las dependencias y constru
r la app
                    sh """
                        mvn compile
                        mvn -B -DskipTests clean package
                    """
                }
            }
        }
        stage('Code Analysis') {
            parallel {
                stage ('Maven Test') { //test unitarios
                    steps {
                        container('maven') {
                            sh '''
                                mvn test
                                ls `pwd`/target
                            '''

```

```

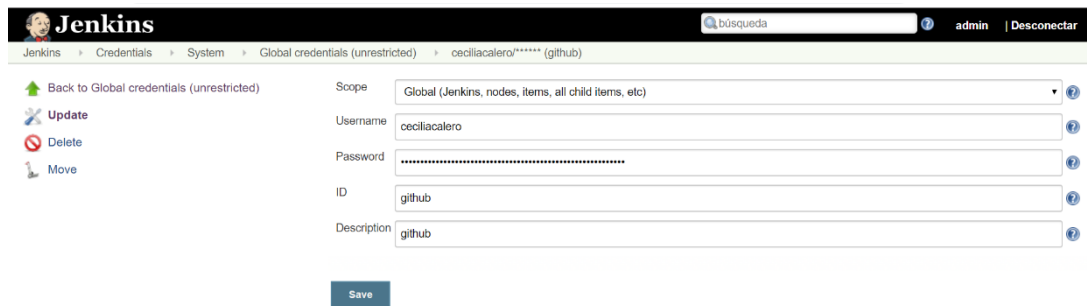
    }
  }
}
}
stage('Create Docker Image') {
  steps {
    container('docker') {
      script{
        def app = docker.build("${imageName}", "-
f ${dockerfile} $WORKSPACE") //construye la imagen
        sh """
            sed -i.bak -
e "s/mydemo/${gitBranch}/" "${dockerfile}"
            pwd
            """
        docker.withRegistry("", "${credentialsId}") { //login
dockerHub
            app.push "${gitBranch}-
${shortGitCommit}" //subimos la imagen
            app.push "${gitBranch}"
        }
      }
    }
  }
}
stage('Deploy to Kubernetes') { // deliegue de la imagen
//1.cambio automatico de los valores de la app (expresiones reg
ulares)
//2.cat leemos el archivo y descargamos
//3. creamos el pod
  steps {
    container('helm') {
      sh """
        sed -i -
e "s/\\(tag:\\).*/\\1 ${gitBranch}/" ${appChart}/values.yaml
        sed -i -
e "s/\\(^appVersion:\\).*/\\1 ${helmAppVersion}/" ${appChart}/Chart.yaml
        sed -i -
e "s/\\(^name:\\).*/\\1 ${appName}/" ${appChart}/Chart.yaml
        sed -i -
e "s/\\(^version:\\).*/\\1 ${release}/" ${appChart}/Chart.yaml
        cat ${appChart}/values.yaml
        cat ${appChart}/Chart.yaml
        helm upgrade -i --recreate-
pods ${appName} ${appChart}
        helm list
        ls
        pwd
        """
      }
    }
  }
}

```

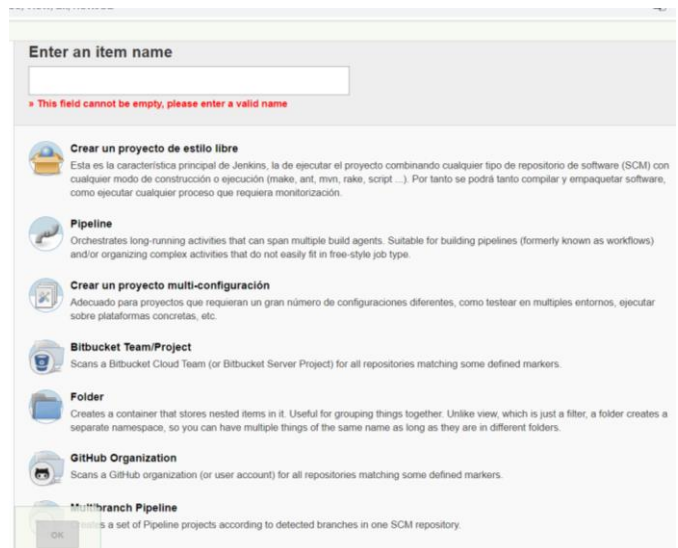
Ahora vamos a ver como funciona nuestra pipeline, y se ejecuta mediante Jenkins.

Primero añadimos en Jenkins las credenciales de GitHub y DockerHub.

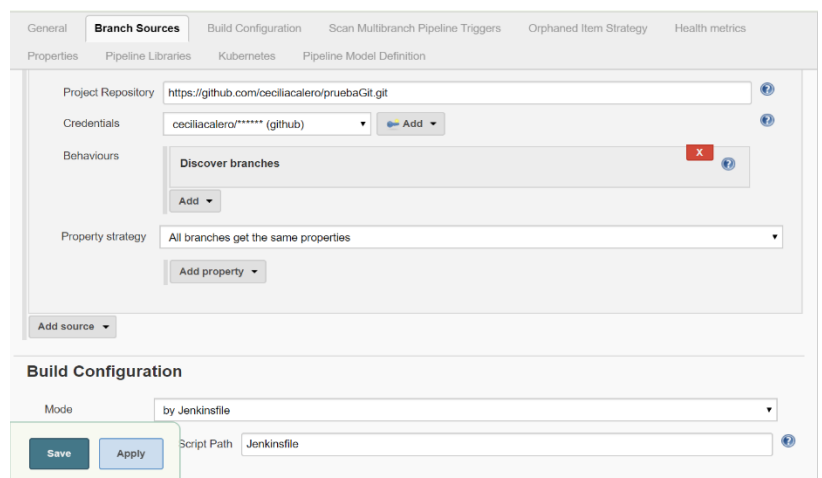
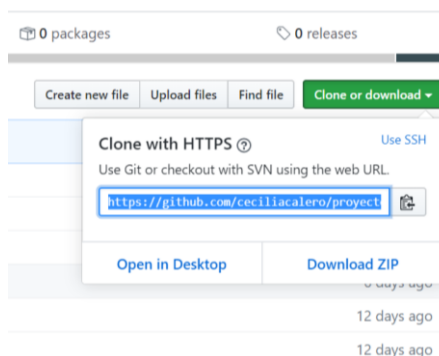
La primera es para que pueda acceder a nuestro repositorio y la segunda, para una vez creada nuestra imagen la suba a DockerHub.



Creamos una multibranch pipeline, ya que el código de nuestro Jenkinsfile, se ha creado de tal manera, que sea un código reutilizable y se pueda utilizar por diferentes ramas, aunque en este caso solo vamos a usar la master, ya que estoy yo sola en el proyecto.

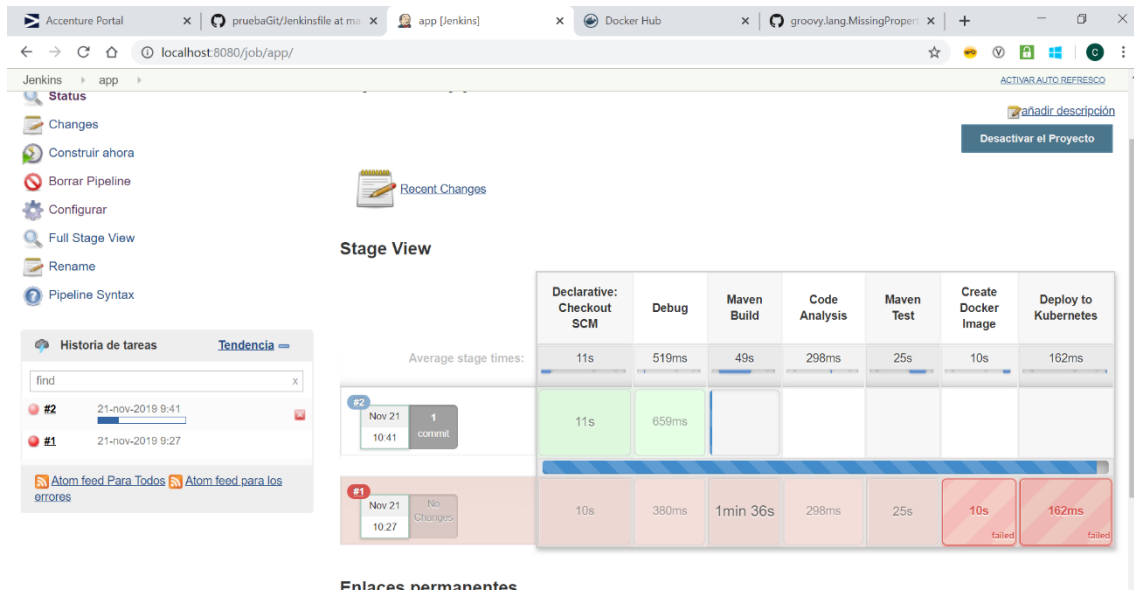


Y añadimos las credenciales de GitHub, creadas anteriormente y la URL de nuestro repositorio.

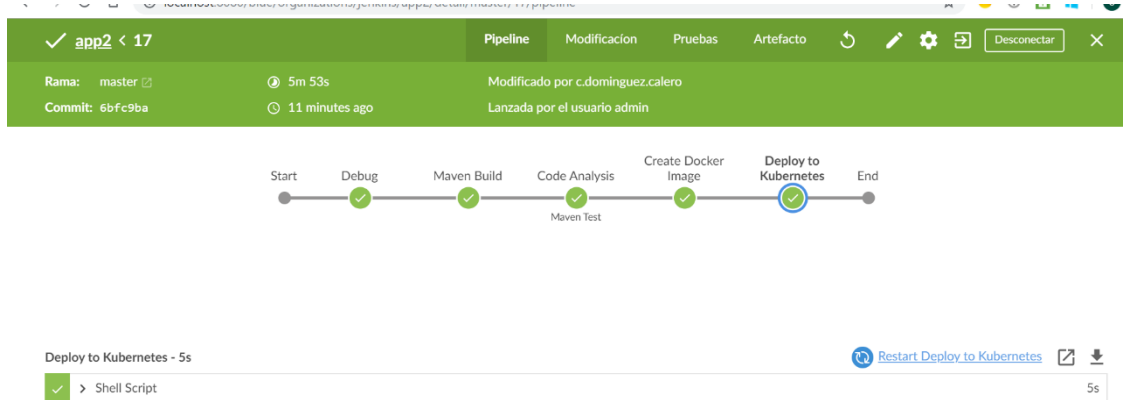


Le damos a construir ahora y empezara a ejecutarse nuestra pipeline.

Como se puede ver en la imagen, va marcando las acciones, que va realizando y en el caso de error, donde se detiene.



Otra manera ver como se ejecuta la pipeline de una interfaz mas visual es descargandose los plugins blue ocean.



Podemos ver en la consola, como se ha creado el pod en Kubernetes, que se declarado en el .yaml de la pipeline.



```
set-credentials Sets a user entry in kubeconfig
unset           Unsets an individual value in a kubeconfig file
use-context     Sets the current-context in a kubeconfig file
view           Display merged kubeconfig settings or a specified kubeconfig file

Usage:
  kubectl config SUBCOMMAND [options]

Use "kubectl <command> --help" for more information about a given command.
Use "kubectl options" for a list of global command-line options (applies to all commands).
PS C:\Users\c.dominguez.calero> kubectl config current-context
docker-for-desktop
PS C:\Users\c.dominguez.calero> kubectl get pods
NAME                                READY    STATUS    RESTARTS   AGE
jenkins-55d6fc89fb-9zxmz            1/1      Running   22         31d
PS C:\Users\c.dominguez.calero> kubectl get pods
NAME                                READY    STATUS    RESTARTS   AGE
jenkins-55d6fc89fb-9zxmz            0/1      Running   24         31d
pipeline-as-code-9159z-46jtz        4/4      Running   0          5m
PS C:\Users\c.dominguez.calero> kubectl get pods
```


Para comprobar que el despliegue de la aplicación se ha generado correctamente, en consola

Kubect! `get svc` (para saber el puerto que ha generado en el servicio creado nodePort)

Añadimos en el navegador localhost: (nodePort) y se nos carga nuestra app.

The HAL Browser (for Spring Data REST) Go To Entry Point About The HAL Browser (for Spring Data REST)

Explorer

http://localhost:32538 Go!

Custom Request Headers

Properties

```
{
}
```

Links

rel	title	name / index	docs	GET	NON-GET
beers				→	i
profile				→	i

Inspector

Response Headers

```
200 success

content-type: application/hal+json;charset=UTF-8
date: Mon, 25 Nov 2019 13:13:45 GMT
transfer-encoding: chunked
```

Response Body

```
{
  "_links": {
    "beers": {
      "href": "http://localhost:32538/beers"
    },
    "profile": {
      "href": "http://localhost:32538/profile"
    }
  }
}
```