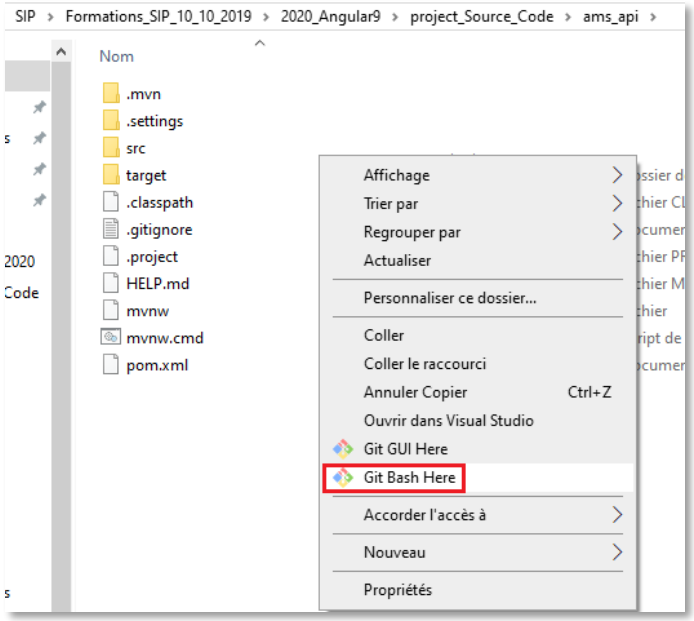
Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins

WORKSHOP 2: GIT

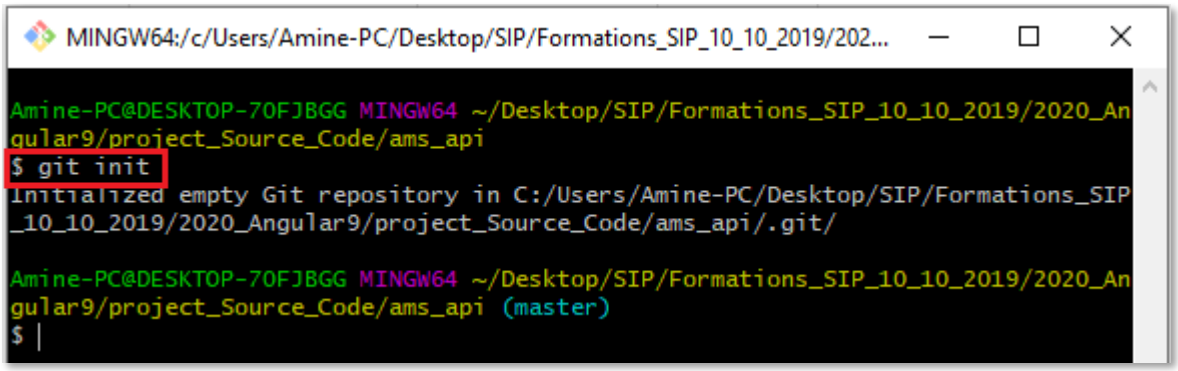
1-Git Basics

Start by installing git in your computer.

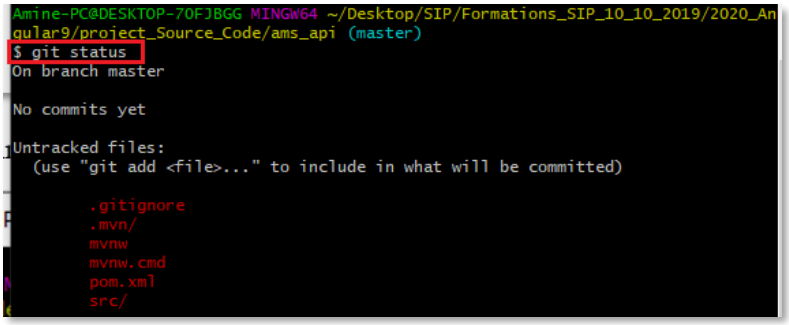


- Then type : git init command

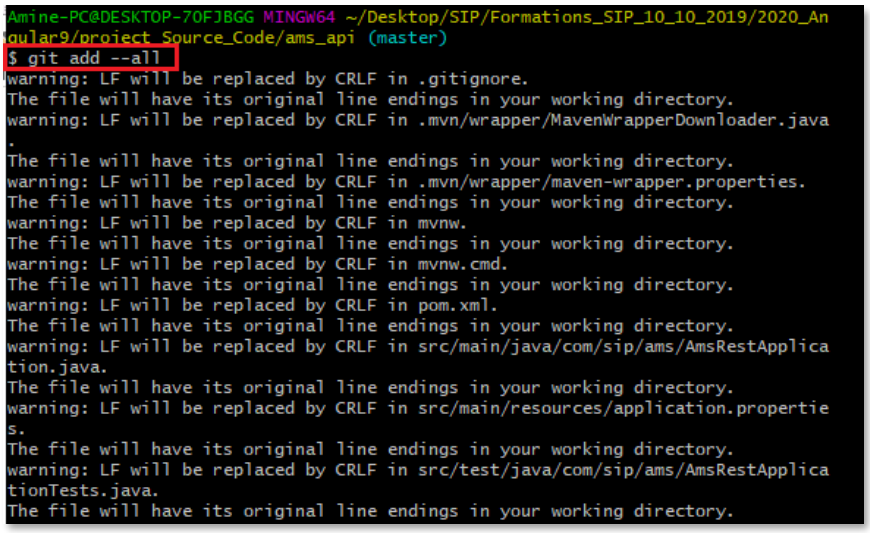
Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins



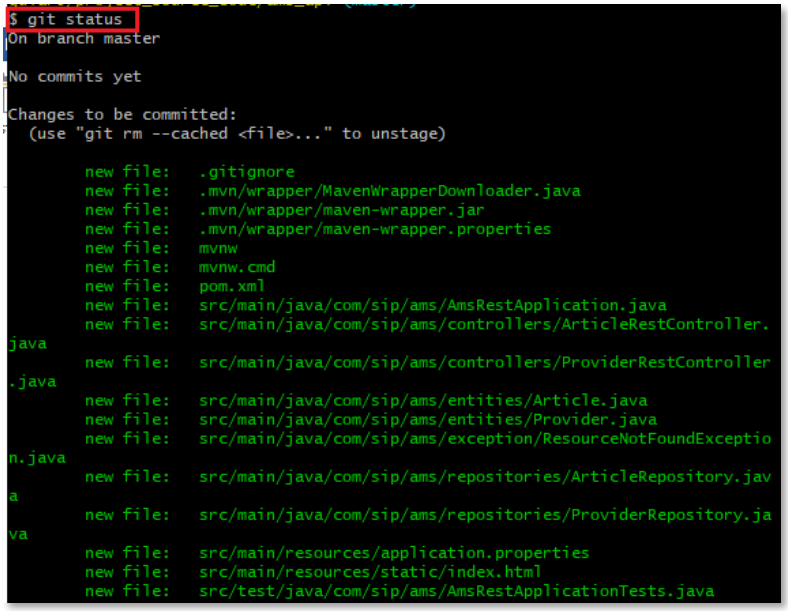
- The second command is git status



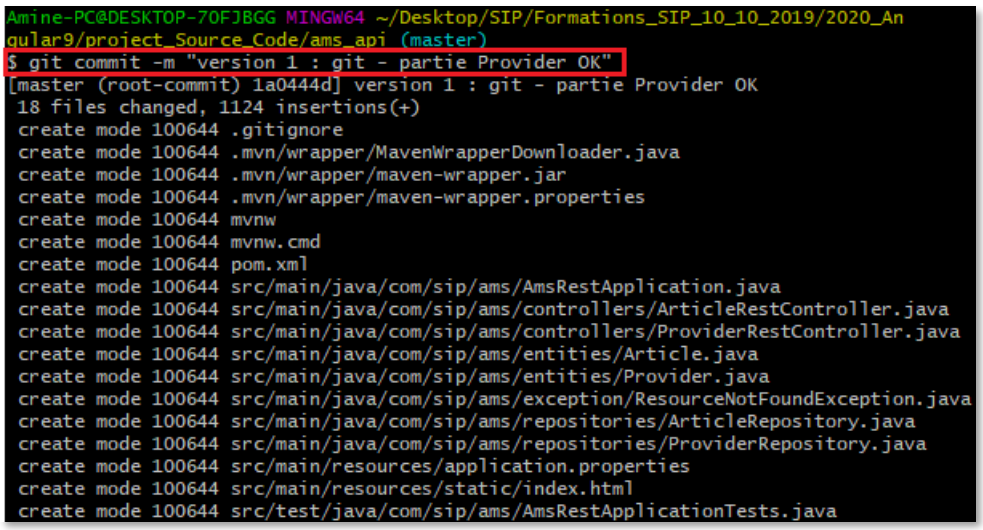
- Then type git add –all



- So , now if we type git status, it will be OK, git knows the content of our project



- Let's perform our first commit : git commit –m "your message"

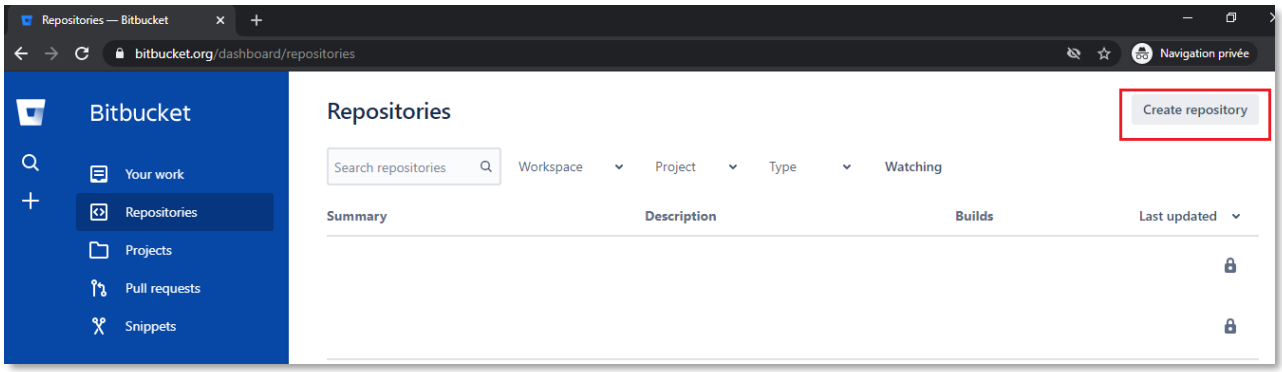


Now, what if you want to collaborate on the project with other person? We can use github, but for a private solution we will use bitbucket.

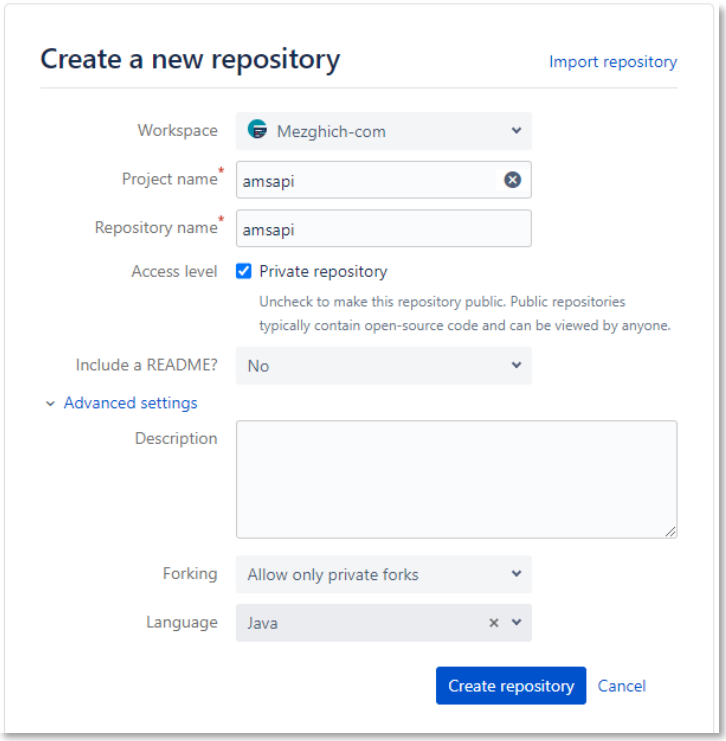
Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins

BitBucket

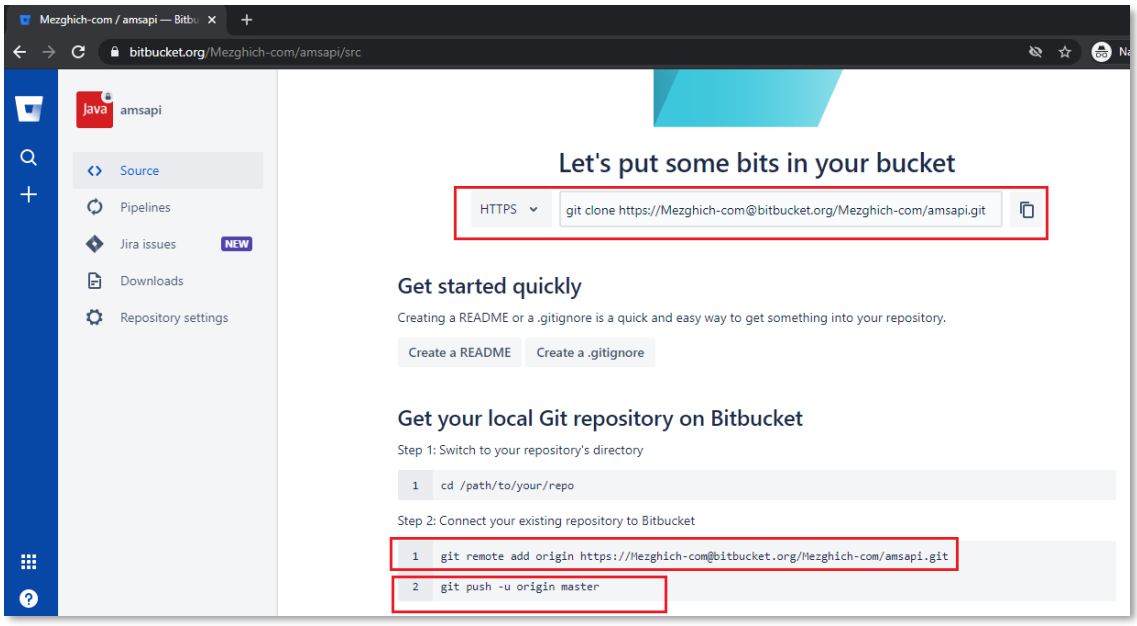
- Let's start by creating a new account on bitbucket.



- Then set the following configurations



Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins



git clone https://Mezghich-com@bitbucket.org/Mezghich-com/amsapi.git

```
git remote add origin https://Mezghich-com@bitbucket.org/Mezghich-com/amsapi.git
```

```
git push -u origin master
```

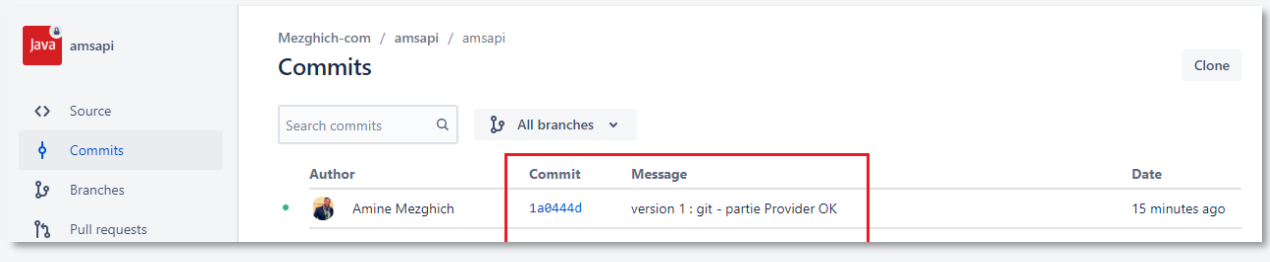
git clone https://Mezghich-com@bitbucket.org/Mezghich-com/amsapi2020.git

```
git remote add origin https://Mezghich-com@bitbucket.org/Mezghich-com/amsapi2020.git

git push -u origin master
```

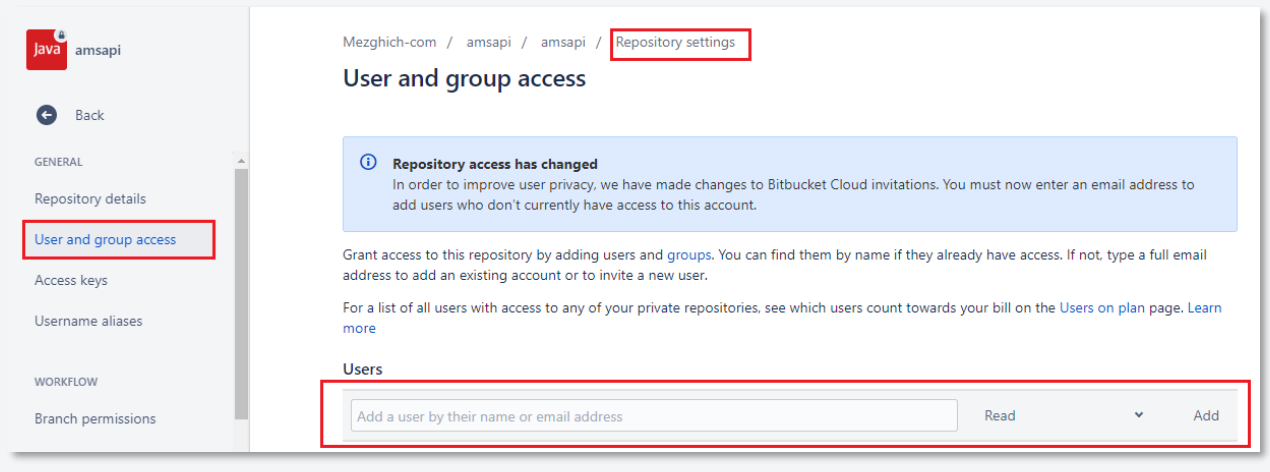
Now, we have to link our local git project to the remote repository

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins

```
Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/SIP/Formations_SIP_10_10_2019/2020_Angular9/project_Source_Code/ams_api (master)
$ git remote add origin https://Mezghich-com@bitbucket.org/Mezghich-com/amsapi.git

Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/SIP/Formations_SIP_10_10_2019/2020_Angular9/project_Source_Code/ams_api (master)
$ git push -u origin master
Enumerating objects: 39, done.
Counting objects: 100% (39/39), done.
Delta compression using up to 4 threads.
Compressing objects: 100% (30/30), done.
Writing objects: 100% (39/39), 56.54 KiB | 2.09 MiB/s, done.
Total 39 (delta 1), reused 0 (delta 0)
To https://bitbucket.org/Mezghich-com/amsapi.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```



You can invite others person to collaborate on the same project using it push/poll.



2-Git Branchs

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins

Une branche dans Git est simplement un pointeur léger et déplaçable vers un de ces **commits**. La branche par défaut dans Git s'appelle `master`. Au fur et à mesure des validations, la branche `master` pointe vers le dernier des **commits** réalisés. À chaque validation, le pointeur de la branche `master` avance automatiquement.

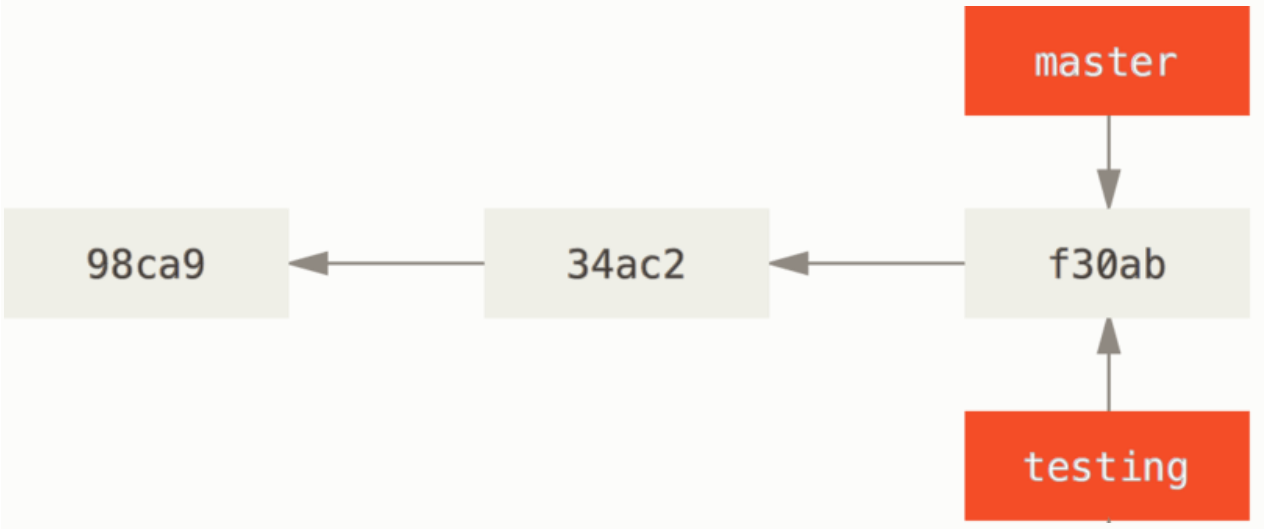
Note	La branche ``master`` n'est pas une branche spéciale. Elle est identique à toutes les autres branches. La seule raison pour laquelle chaque dépôt en a une est que la commande `git init` la crée par défaut et que la plupart des gens ne s'embêtent pas à la changer.

Créer une nouvelle branche

Que se passe-t-il si vous créez une nouvelle branche ? Eh bien, cela crée un nouveau pointeur pour vous. Supposons que vous créez une nouvelle branche nommée `testing`. Vous utilisez pour cela la commande `git branch` :

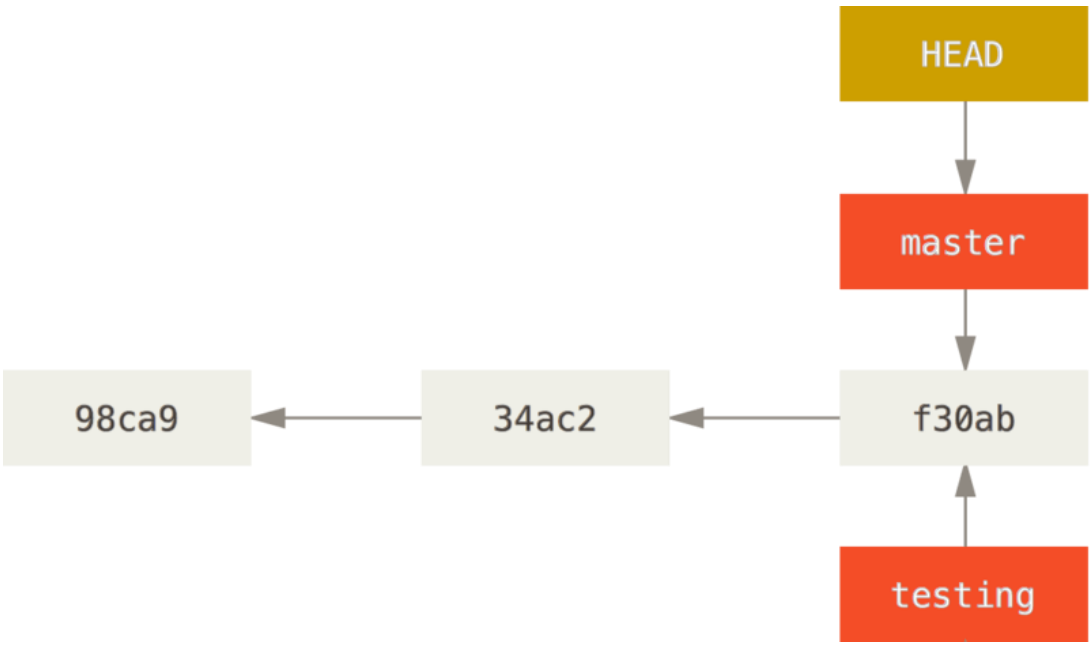
```
$ git branch testing
```

Cela crée un nouveau pointeur vers le **commit** courant.



Comment Git connaît-il alors la branche sur laquelle vous vous trouvez ? Il conserve à cet effet un pointeur spécial appelé `HEAD`.

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins



La commande **git branch** n'a fait que créer une nouvelle branche — elle n'a pas fait basculer la copie de travail vers cette branche.

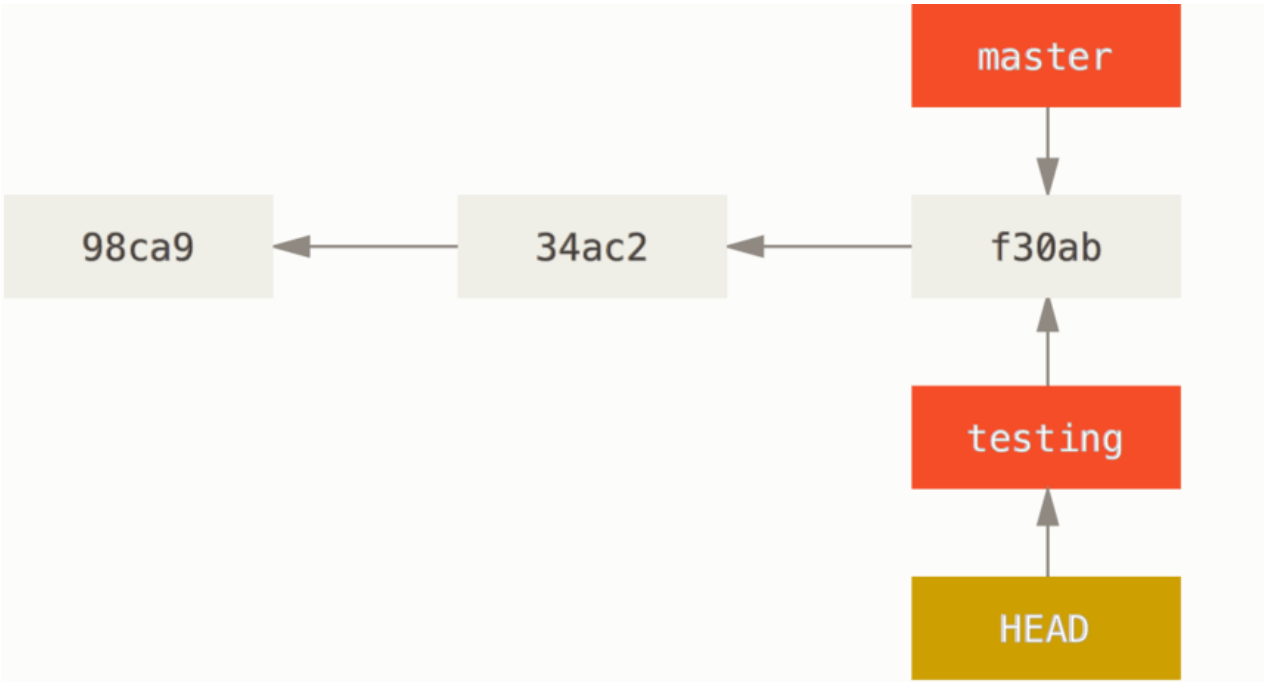
Basculer entre les branches

Pour basculer sur une branche existante, il suffit de lancer la commande `git checkout`. Basculons sur la nouvelle branche `testing` :

```
$ git checkout testing
```

Cela déplace `HEAD` pour le faire pointer vers la branche `testing`.

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins



Apportons quelques modification et ensuite faisons un commit.

```
1  package com.sip.ams;
2
3  import org.springframework.boot.SpringApplication;
5
6  @SpringBootApplication
7  public class AmsApiApplication {
8
9      public static void main(String[] args) {
10         System.out.println("GIT-Les Branchs");
11         SpringApplication.run(AmsApiApplication.class, args);
12     }
13
14 }
```

Next

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins

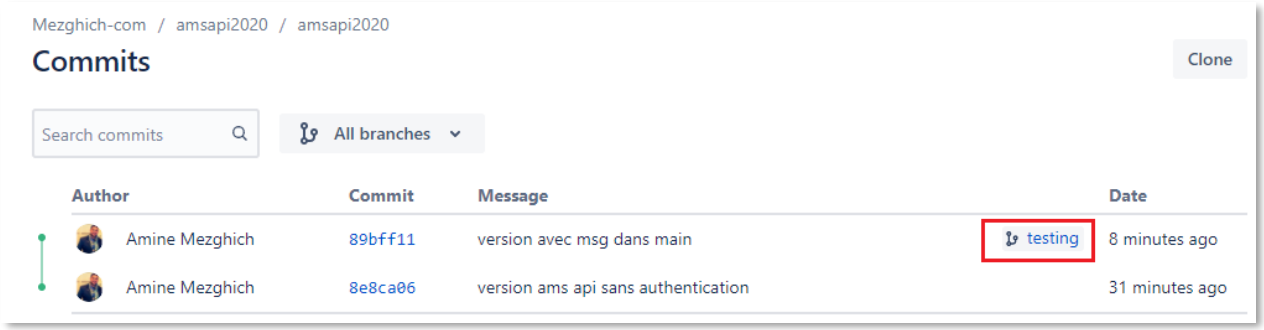
```
Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/AMS_GIT_Deploy/amsApi (master)
$ git branch testing

Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/AMS_GIT_Deploy/amsApi (master)
$ git checkout testing
Switched to branch 'testing'

Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/AMS_GIT_Deploy/amsApi (testing
)
$ git status
On branch testing
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   src/main/java/com/sip/ams/AmsApiApplication.java

no changes added to commit (use "git add" and/or "git commit -a")
```

Now, let's commit

```
Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/AMS_GIT_Deploy/amsApi (testing
)
$ git commit -m "version avec msg dans main"
[testing 89bff11] version avec msg dans main
 1 file changed, 1 insertion(+)
```

We can also push on the bitbucket

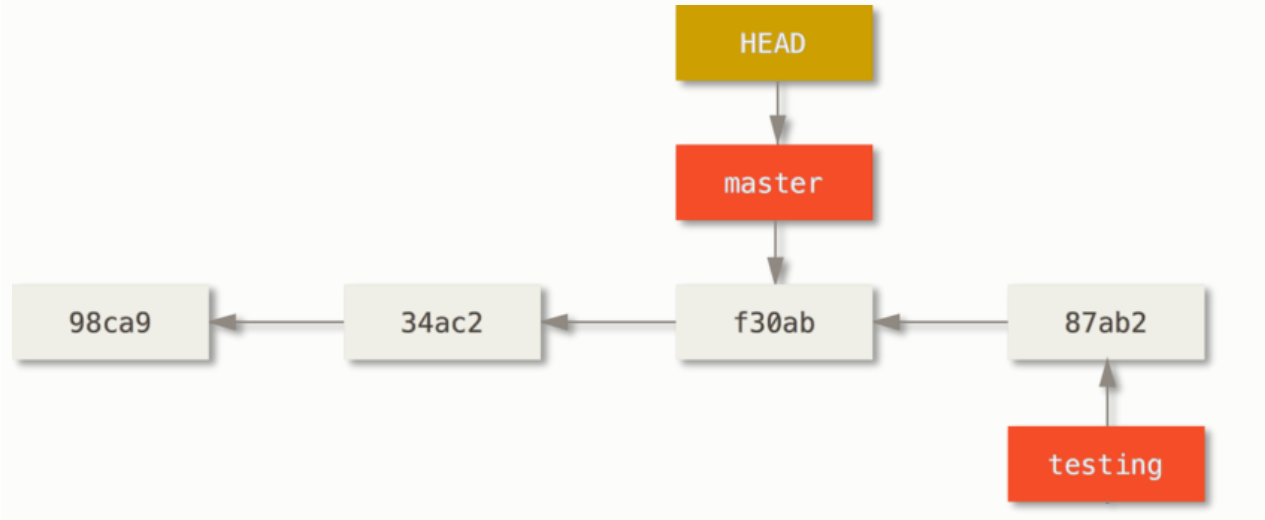
git push --set-upstream origin testing



Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins

C'est intéressant parce qu'à présent, votre branche testing a avancé tandis que la branche master pointe toujours sur le **commit** sur lequel vous étiez lorsque vous avez lancé la commande git checkout pour changer de branche. Retournons sur la branche master :

```
$ git checkout master
```

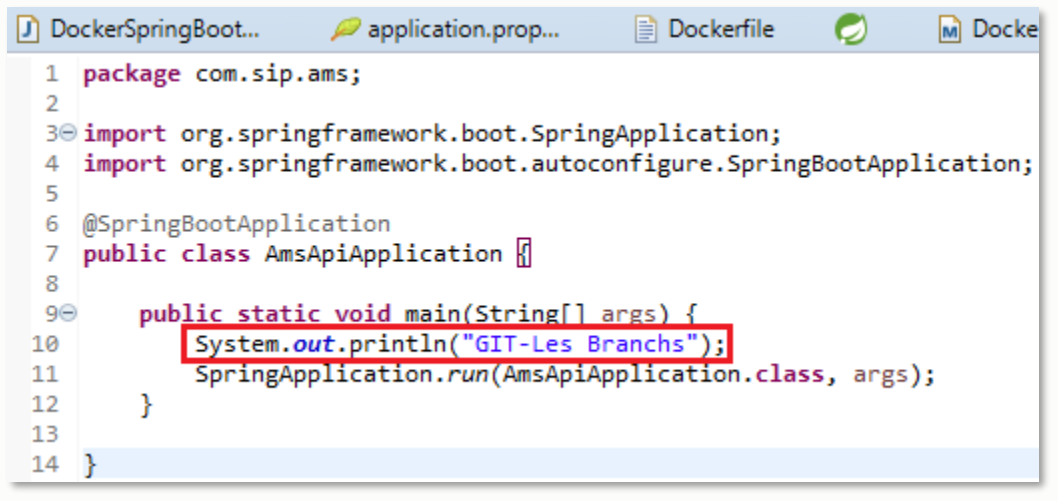


Cette commande a réalisé deux actions. Elle a remis le pointeur HEAD sur la branche master et elle a replacé les fichiers de votre répertoire de travail dans l'état du **snapshot** pointé par master. Cela signifie aussi que les modifications que vous réalisez à partir de ce point divergeront de l'ancienne version du projet. Cette commande annule les modifications réalisées dans la branche test pour vous permettre de repartir dans une autre direction.

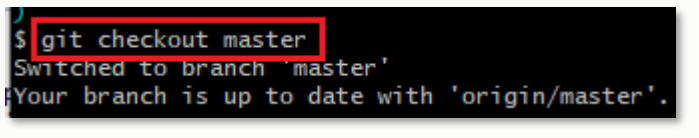
Voyons ce qui se passe au niveau STS

Lorsqu'on est sur la branch testing :

```
Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/AMS_GIT_Deploy/amsApi (master)
$ git checkout testing
Switched to branch 'testing'
Your branch is up to date with 'origin/testing'.
```



➔Switch vers la branch master,



Le code revient à l'état de ce commit.

Note	**Changer de branche modifie les fichiers dans votre répertoire de travail** Il est important de noter que lorsque vous changez de branche avec Git, les fichiers de votre répertoire de travail sont modifiés. Si vous basculez vers une branche plus ancienne, votre répertoire de travail sera remis dans l'état dans lequel il était lors du dernier **commit** sur cette branche. Si git n'est pas en mesure d'effectuer cette action proprement, il ne vous laissera pas changer de branche.

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins

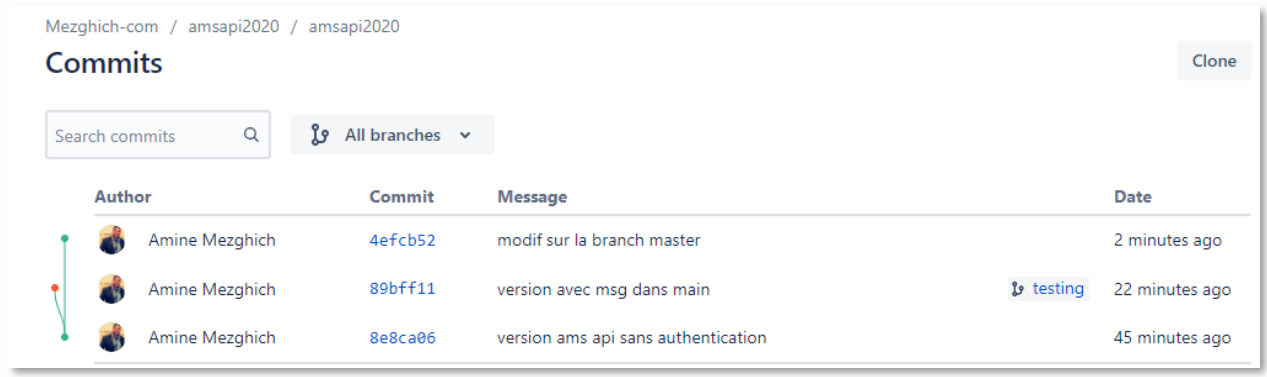
```java
@SpringBootApplication
public class AmsApiApplication {

    public static void main(String[] args) {
        SpringApplication.run(AmsApiApplication.class, args);
    }

}
```

Faisons une modification puis un commit puis un push

```java
@SpringBootApplication
public class AmsApiApplication {

    public static void main(String[] args) {
        System.out.println("Modif sur la brach master");
        SpringApplication.run(AmsApiApplication.class, args);
    }
```

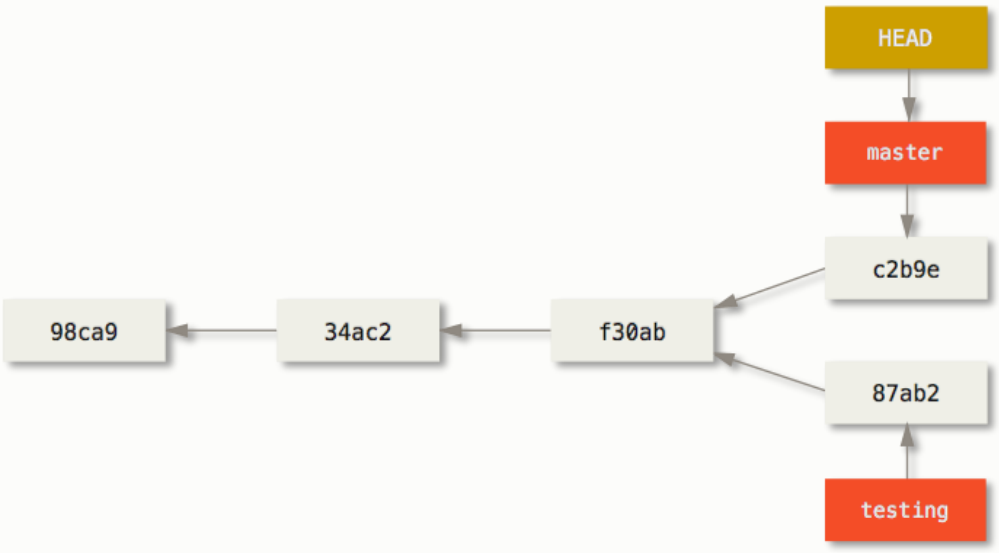
Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   src/main/java/com/sip/ams/AmsApiApplication.java

no changes added to commit (use "git add" and/or "git commit -a")

Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/AMS_GIT_Deploy/amsApi (master)
$ git add --all

Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/AMS_GIT_Deploy/amsApi (master)
$ git commit -m "modif sur la branch master"
[master 4efcb52] modif sur la branch master
 1 file changed, 1 insertion(+)

Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/AMS_GIT_Deploy/amsApi (master)
$ git push
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 662 bytes | 220.00 KiB/s, done.
Total 9 (delta 3), reused 0 (delta 0)
To https://bitbucket.org/Mezghich-com/amsapi2020.git
   8e8ca06..4efcb52  master -> master
```



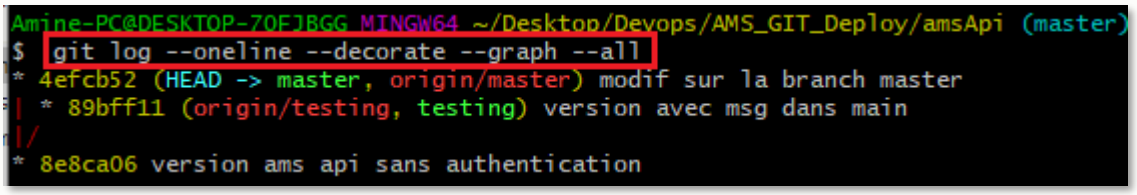
Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins

Maintenant, l'historique du projet a divergé. Vous avez créé une branche et basculé dessus, y avez réalisé des modifications, puis vous avez rebasculé sur la branche principale et réalisé d'autres modifications. Ces deux modifications sont isolées dans des branches séparées : vous pouvez basculer d'une branche à l'autre et les fusionner quand vous êtes prêt. Et vous avez fait tout ceci avec de simples commandes : `branch`, `checkout` et `commit`.



Vous pouvez également voir ceci grâce à la commande `git log`. La commande

`git log --oneline --decorate --graph --all` : va afficher l'historique de vos **commits**, affichant les endroits où sont positionnés vos pointeurs de branche ainsi que la manière dont votre historique a divergé.



Remarque :

```
$ git checkout -b testing

Switched to a new branch "testing"
```

Cette commande est un raccourci pour :

```
$ git branch testing

$ git checkout testing
```

La fusion (Merge)

Cas 1 : Modification sur le même fichier

Actuellement, nous travaillons sur le même fichier .java ; un merge va automatiquement créer un conflit.

Voyons ce que ça donne :

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins

```
Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/AMS_GIT_Deploy/amsApi (master)
$ git checkout master  (1)
Already on 'master'
Your branch is up to date with 'origin/master'.

Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/AMS_GIT_Deploy/amsApi (master)
$ git merge testing  (2)
Auto-merging src/main/java/com/sip/ams/AmsApiApplication.java
CONFLICT (content): Merge conflict in src/main/java/com/sip/ams/AmsApiApplicatio
n.java
Automatic merge failed; fix conflicts and then commit the result.

Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/AMS_GIT_Deploy/amsApi (master|
MERGING)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   src/main/java/com/sip/ams/AmsApiApplication.java

no changes added to commit (use "git add" and/or "git commit -a")
```

(3)

Git status nous affiche sur STS le problème, et c'est à nous de le résoudre manuellement

```
 8
 9      public static void main(String[] args) {
10  <<<<<<< HEAD
11          System.out.println("Modif sur la brach master");
12  =======
13          System.out.println("GIT-Les Branchs");
14  >>>>>>> testing
15          SpringApplication.run(AmsApiApplication.class, args);
16      }
17
18  }
```

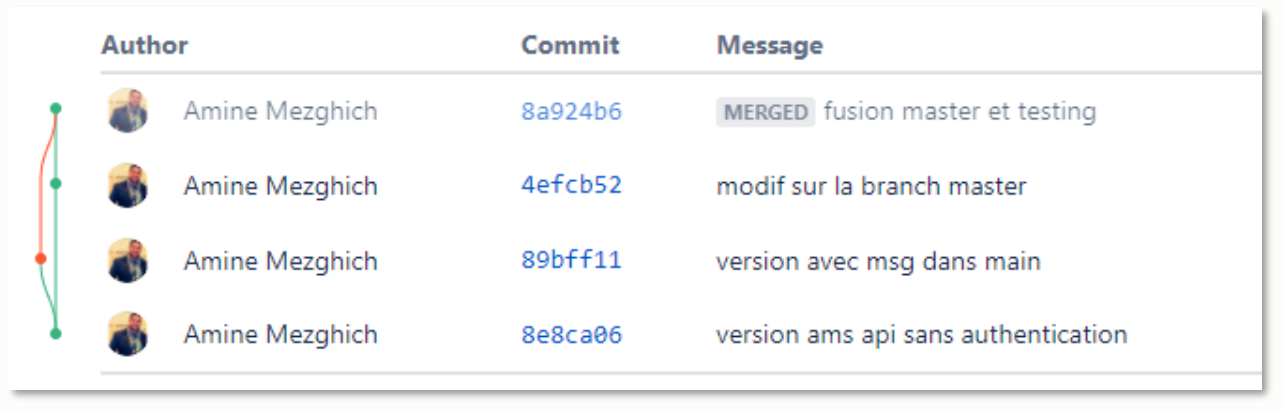
Disons que nous avons résolus le problème,

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins

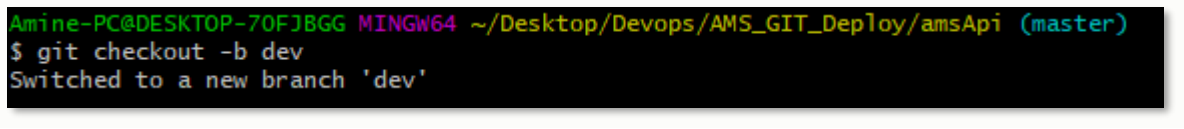
```java
public static void main(String[] args) {

    System.out.println("Modif sur la brach master");

    System.out.println("GIT-Les Branchs");

    SpringApplication.run(AmsApiApplication.class, args);
}
```

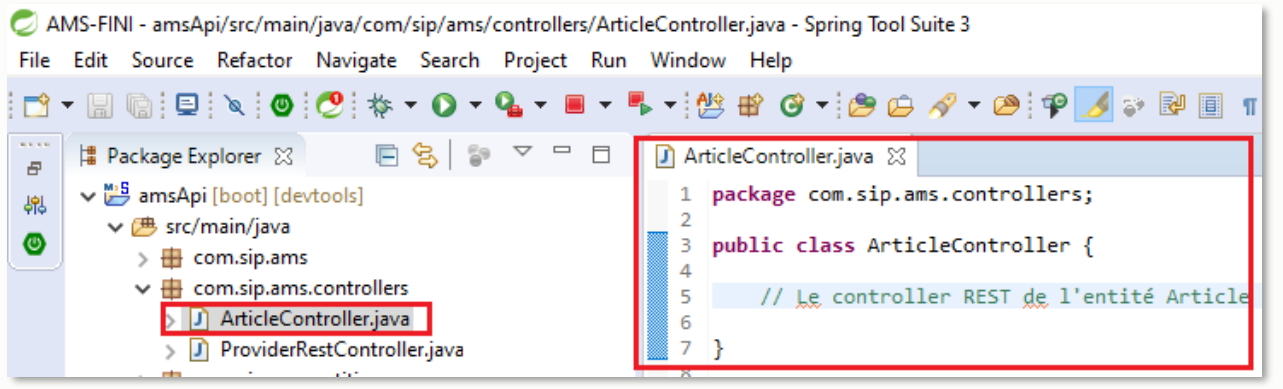
```
Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/AMS_GIT_Deploy/amsApi (master|
MERGING)
$ git add --all  (1)

Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/AMS_GIT_Deploy/amsApi (master|MERGING)
$ git status  (2)
On branch master
Your branch is up to date with 'origin/master'.

All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:

        modified:   src/main/java/com/sip/ams/AmsApiApplication.java


Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/AMS_GIT_Deploy/amsApi (master|MERGING)
$ git commit -m "fusion master et testing"  (3)
[master 8a924b6] fusion master et testing

Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/AMS_GIT_Deploy/amsApi (master)
$ git push  (4)
Enumerating objects: 25, done.
Counting objects: 100% (25/25), done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 680 bytes | 170.00 KiB/s, done.
Total 9 (delta 3), reused 0 (delta 0)
To https://bitbucket.org/Mezghich-com/amsapi2020.git
   4efcb52..8a924b6  master -> master
```

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins

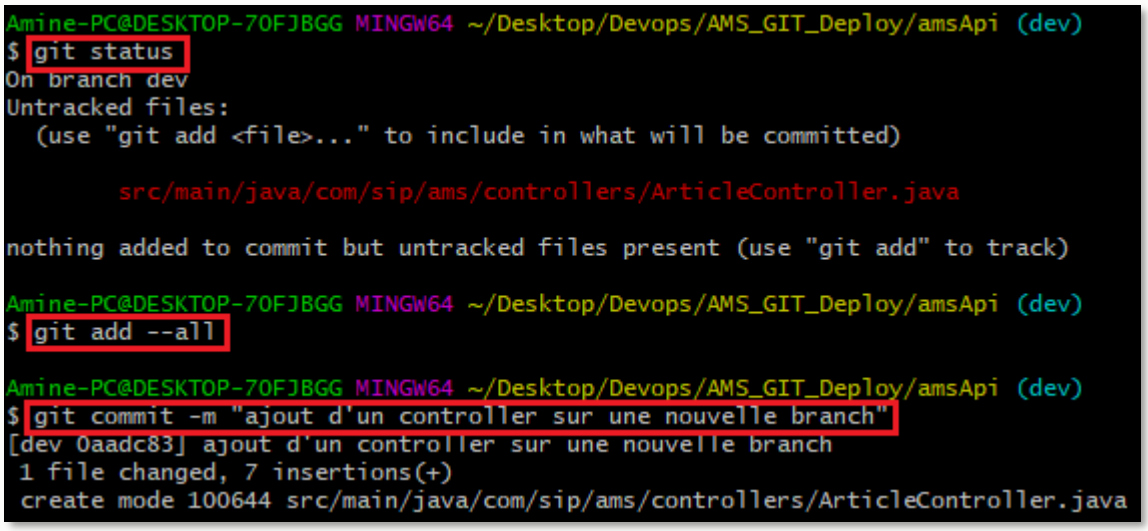


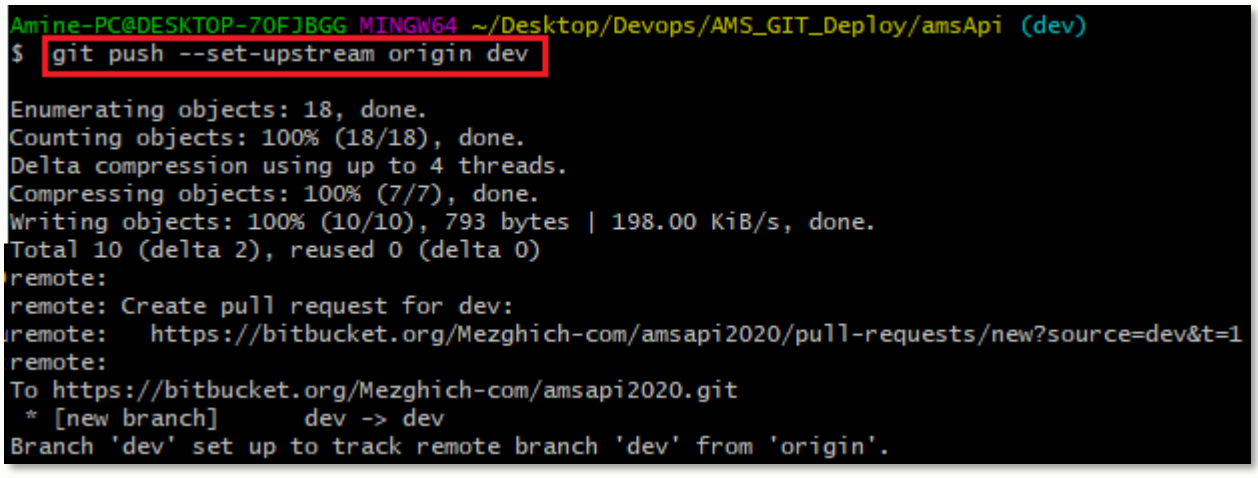
Cas 2 : Modification sur un fichier différent (sans conflit)



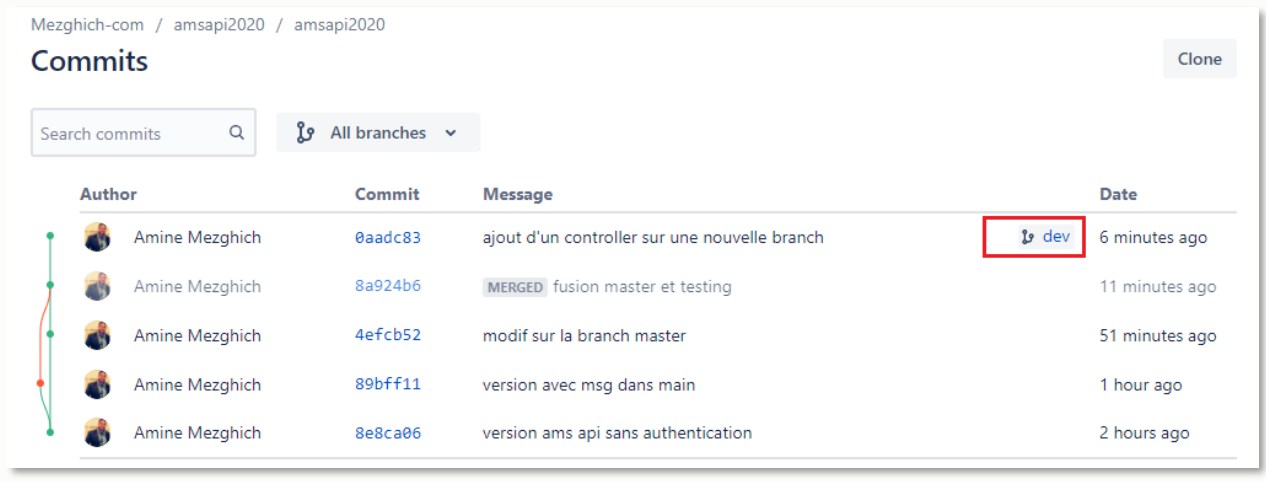


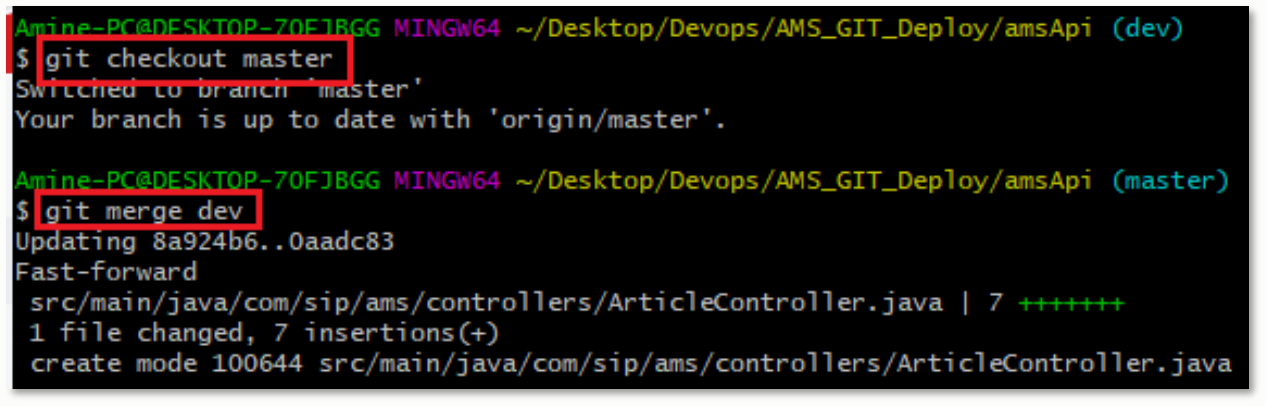
Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins





Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins





Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins

WORKSHOP 2: DOCKER

1- What is Docker?

Docker is a platform to **build**, **ship** and **run** applications by wrapping them in **containers**.

In docker, the applications are **composed as images** and **run them in containers**. So docker is all about creating **images** and run them inside **containers**.

By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

Another definition:

Docker

Docker is a tool designed to simplify creating, deploying, and running applications by using containers. A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

A Docker image is a file, comprised of multiple layers, used to execute code in a Docker container. Dockerfile is a text document that contains all the commands to assemble an image. The image is created with the docker build command.

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins

1.1- What is Image and Container ?

A **image** is an lightweight, standalone and executable package of the software application. The image contains everything (including compiled source code,runtime dependencies, executable jars and libraries etc) that it needs to run the application.

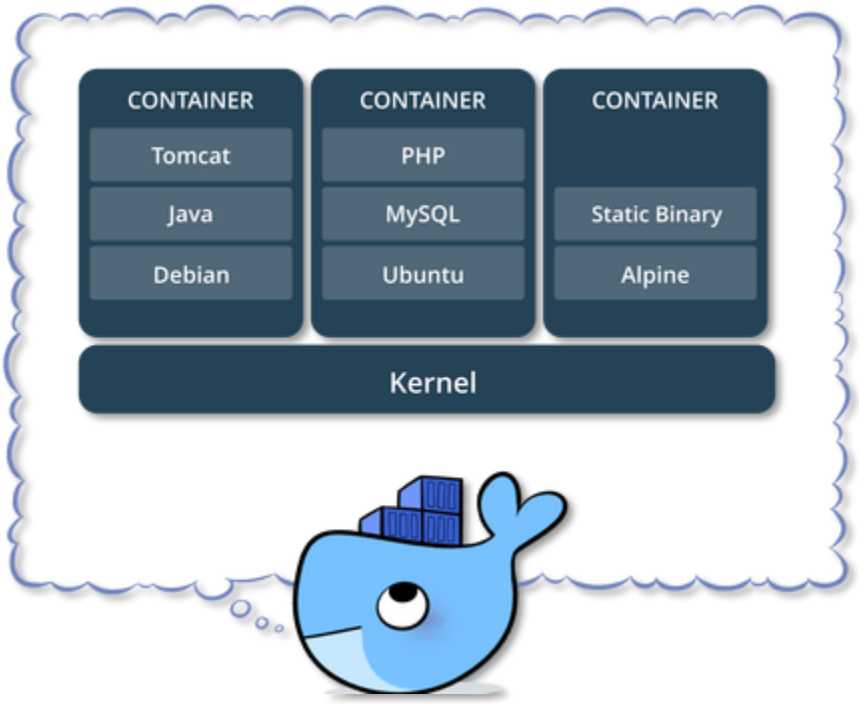
A **container** is a runtime instance of an image—what the image becomes in memory when actually executed. It runs completely isolated from the host environment by default, only accessing host files and ports if configured to do so.

The relationship between **image** and **container** can be described as follows. The image will be composed from the application source code and it is the runtime executable version of the application. The **container** is the runtime representation of the **image** and the whole image will be run/executed in the container.

For instance, assumed that a php application that requires **php** and **mysql** in the runtime environment. Therefore the image should be packaged by providing those two dependencies. when the application is run in the container, those two dependencies will also run inside the container.

Multiple containers can be executed on the **docker** platform and they all will run as independent containers.

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins



As you can see that the docker platform can have multiple containers running on it. each container runs their own set of libraries and servers that are required to run the underlying docker image.

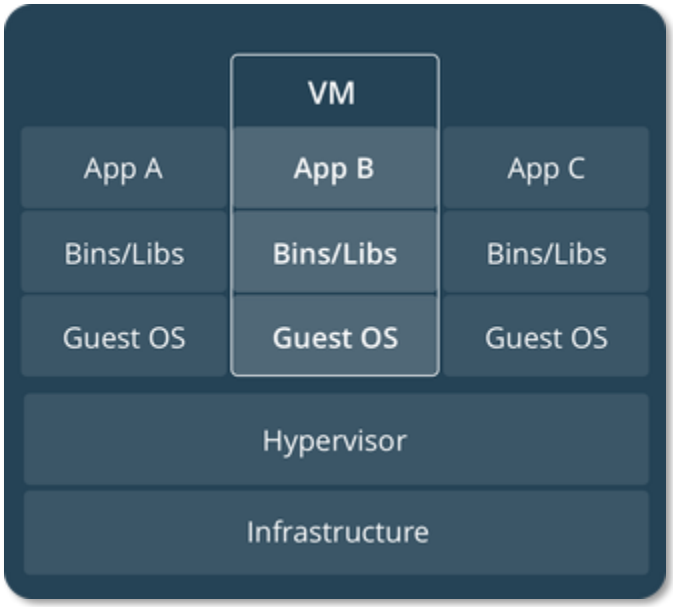
1.2- Are Docker Containers similar to Virtual Machines (VM) ?

No… They looks like same. But the are completely different. Lets look at why they are different and what are their differences.

Virtual Machine (VM)

As the name implies Virtual Machine is a machine that virtually running on a physical machine. Each VM has their own operating system(full version of the OS) , runtime libraries and installed apps. Therefore the size of a VM may

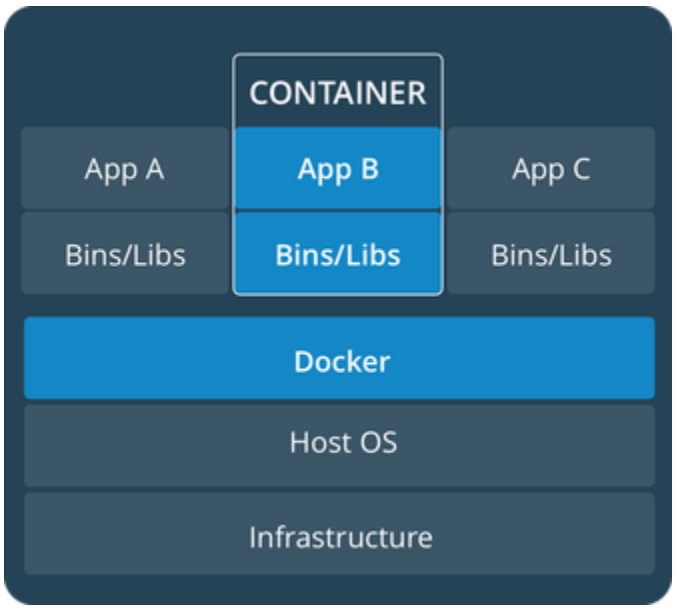
take **GB** of space from the physical machine (It consumes high resources of the physical machine).



Docker Containers

Docker containers run on top of the docker platform. They contains only the executable package of the application, libraries and dependent software/servers that are required to run the application. They do not have separate OS installations running on their own and they utilize the host machine's OS. Therefore the size of the container may take MB of spaces (this may varies based on the size of the dependent softwares and libraries in the container) and thus containers are considered as light weight with compared to VM. The below diagram will demonstrate multiple dockers containers running on the docker platform installed in a single machine.

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins



2- Installation

https://docs.docker.com/engine/install/

Once you have completed the installation process, you can verify the installation by checking the docker version.

Checking the docker version

There are two commands available to check the version of the docker. you can run one of the commands in the terminal. Then it will print the installed version.

```
docker -v
```

OR

```
docker --version
```

3- Create a Docker Hub

This is one of the most important place that you should be aware of. This is a sort of repository that contains the published docker images. You can create your own docker image of your application and publish it here for later use or someone else to use. In addition, you can find any official docker image through this repository hub.

https://hub.docker.com/

4- Exercise 1- (Hello-world) :

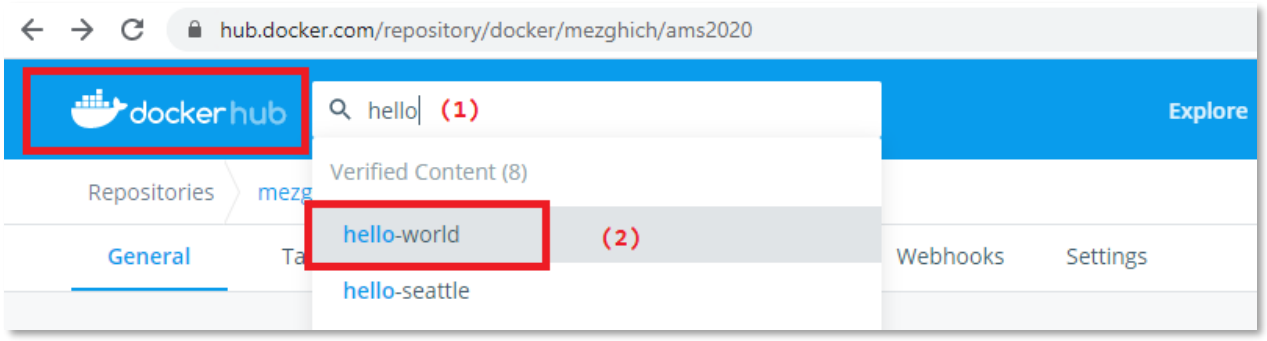
Consider the following docker image:

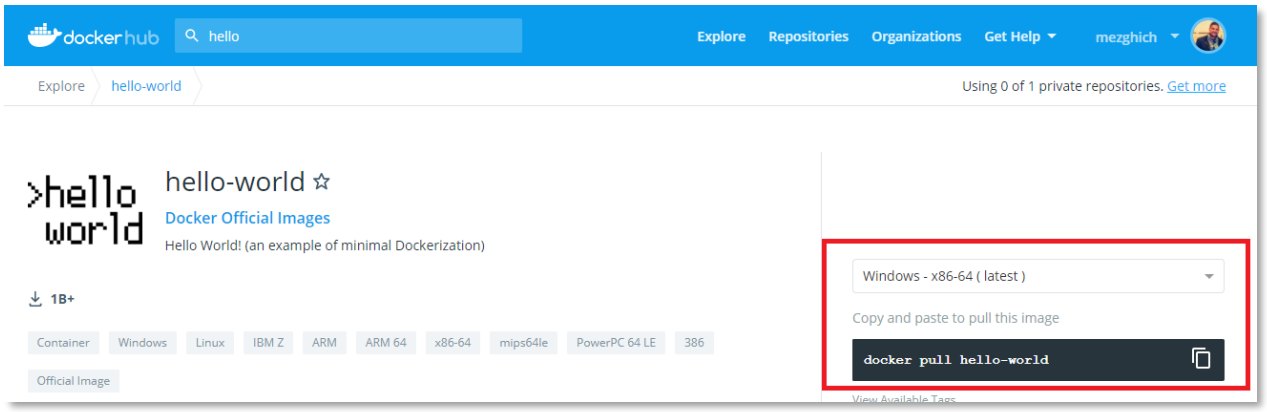
https://github.com/docker-library/hello-world

https://github.com/docker-library/hello-world/blob/master/hello.c

Lets search for the "**hello-world**" docker image. You can see the list of found container images. Is is always advised to go with **official image** if available.

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins






```
Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/Docker/doodle/cheers2019 (master)
$ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
0e03bdcc26d7: Pulling fs layer
0e03bdcc26d7: Verifying Checksum
0e03bdcc26d7: Download complete
0e03bdcc26d7: Pull complete
Digest: sha256:4cf9c47f86df71d48364001ede3a4fcd85ae80ce02ebad74156906caff5378bc
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest
```

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins

```
Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/Docker/doodle/cheers2019 (master)
$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
mezghich/cheers2019     latest              e1912f9e3e00        5 hours ago         4.01MB
<none>                  <none>              9fa4446857ef        5 hours ago         357MB
docker101tutorial       latest              420072d96127        6 hours ago         27.3MB
<none>                  <none>              4d3e2ad59f69        6 hours ago         85.5MB
<none>                  <none>              8a1edd2e67cd        6 hours ago         72MB
<none>                  <none>              5acf61ed844b        6 hours ago         224MB
python                  alpine              0f03316d4a27        2 days ago          42.7MB
nginx                   alpine              6f715d38cfe0        4 weeks ago         22.1MB
node                    12-alpine           18f4bc975732        6 weeks ago         89.3MB
hello-world             latest              bf756fb1ae65        8 months ago        13.3kB
golang                  1.11-alpine         e116d2efa2ab        12 months ago       312MB
```

Finally run the container

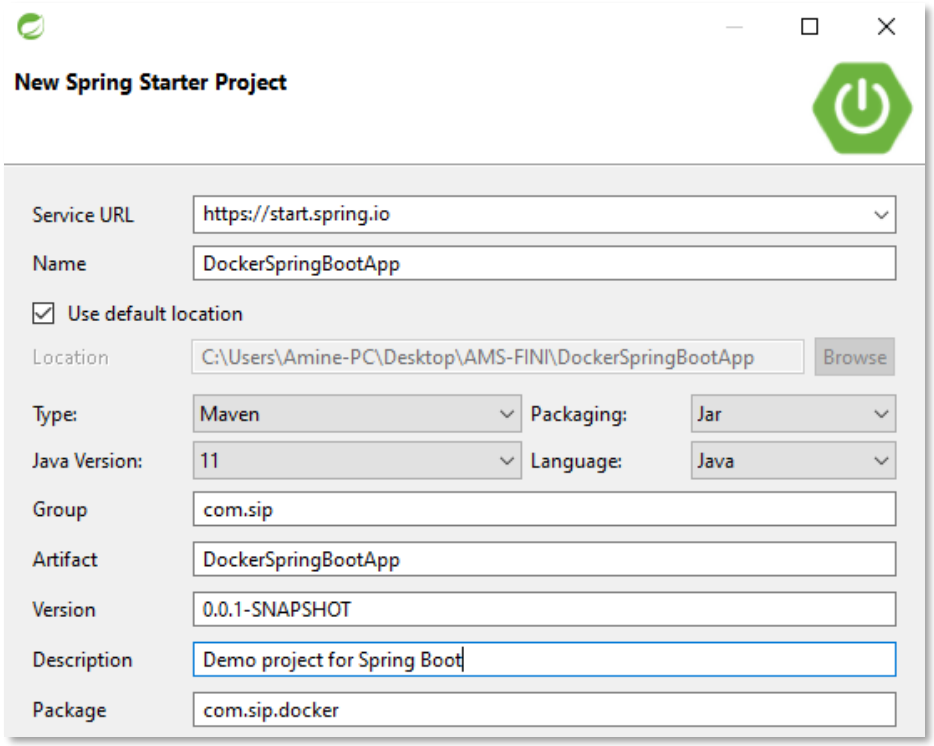
```
Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/Devops/Docker/doodle/cheers2019 (master)
$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

5- Exercise 2 (Springboot App on docker):

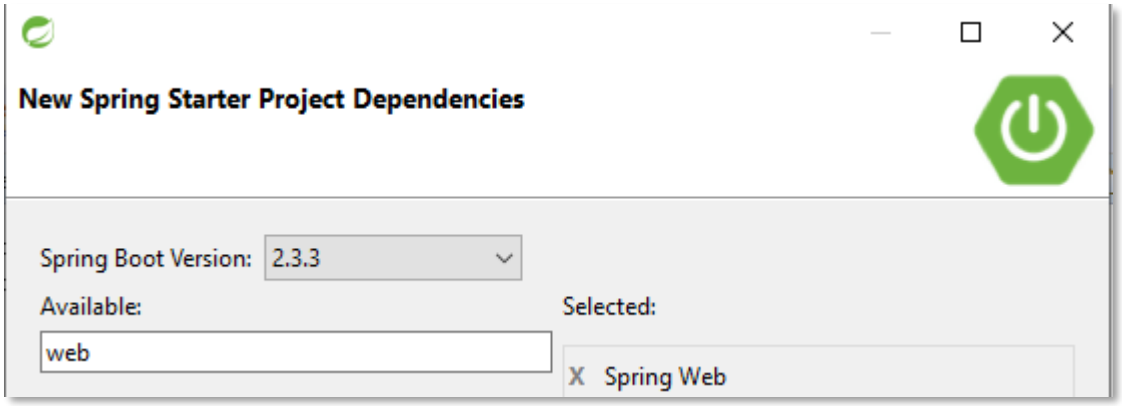
5.1-Creating the spring boot App

In the following example, we are creating a Spring Boot application and placing it into the Docker image.

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins



We will use only a single dependency (Change to Java 8)

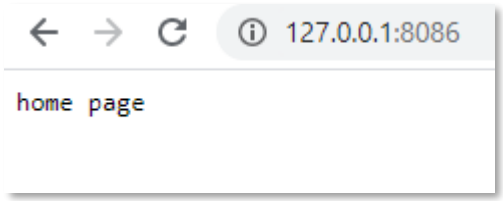


Let's add the following code to the principal class.

```
package com.sip.docker;

import org.springframework.boot.SpringApplication;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController



public class DockerSpringBootAppApplication {
        @GetMapping(value="/", produces=MediaType.TEXT_PLAIN_VALUE)
          public String home() {

             return "home page";
          }

       public static void main(String[] args) {
              SpringApplication.run(DockerSpringBootAppApplication.class, args);
       }

}
```

The application consists of this simple file. The application returns a simple text.

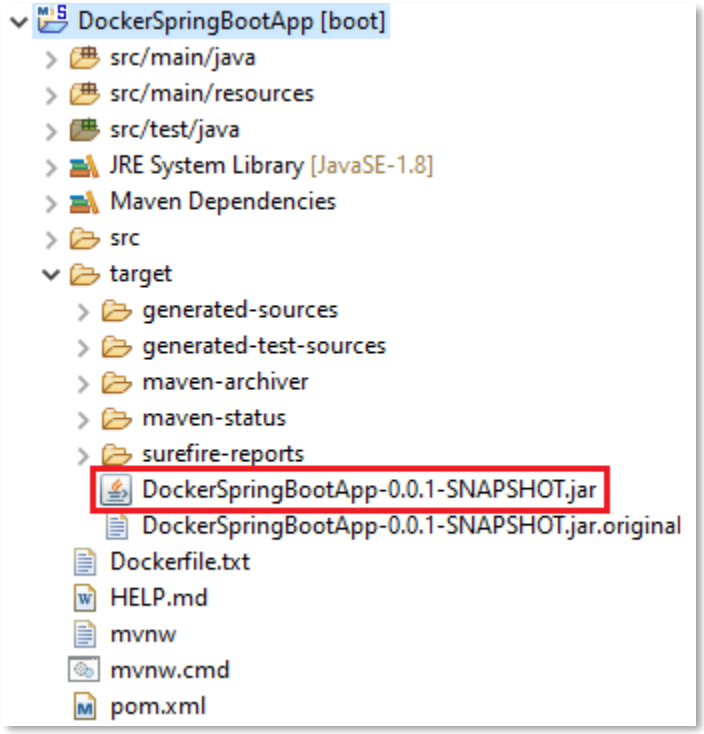


Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins

5.2- Dockerizing using Dockerfile

A Dockerfile is just a regular `text` file that includes native Docker commands that are used to specify the layers of an image. To do so, let's create a text file named "Dockerfile":

Before moving further, we need a Spring Boot .jar file. This file will be used to create the Docker image as mentioned above.



Run the mvn clean install command to make sure that it's generated.

The content of the file itself can look something like this:

```
FROM java:8-jdk-alpine
COPY ./target/DockerSpringBootApp-0.0.1-SNAPSHOT.jar /usr/app/
WORKDIR /usr/app
EXPOSE 8086

ENTRYPOINT ["java","-jar","DockerSpringBootApp-0.0.1-SNAPSHOT.jar"]
```

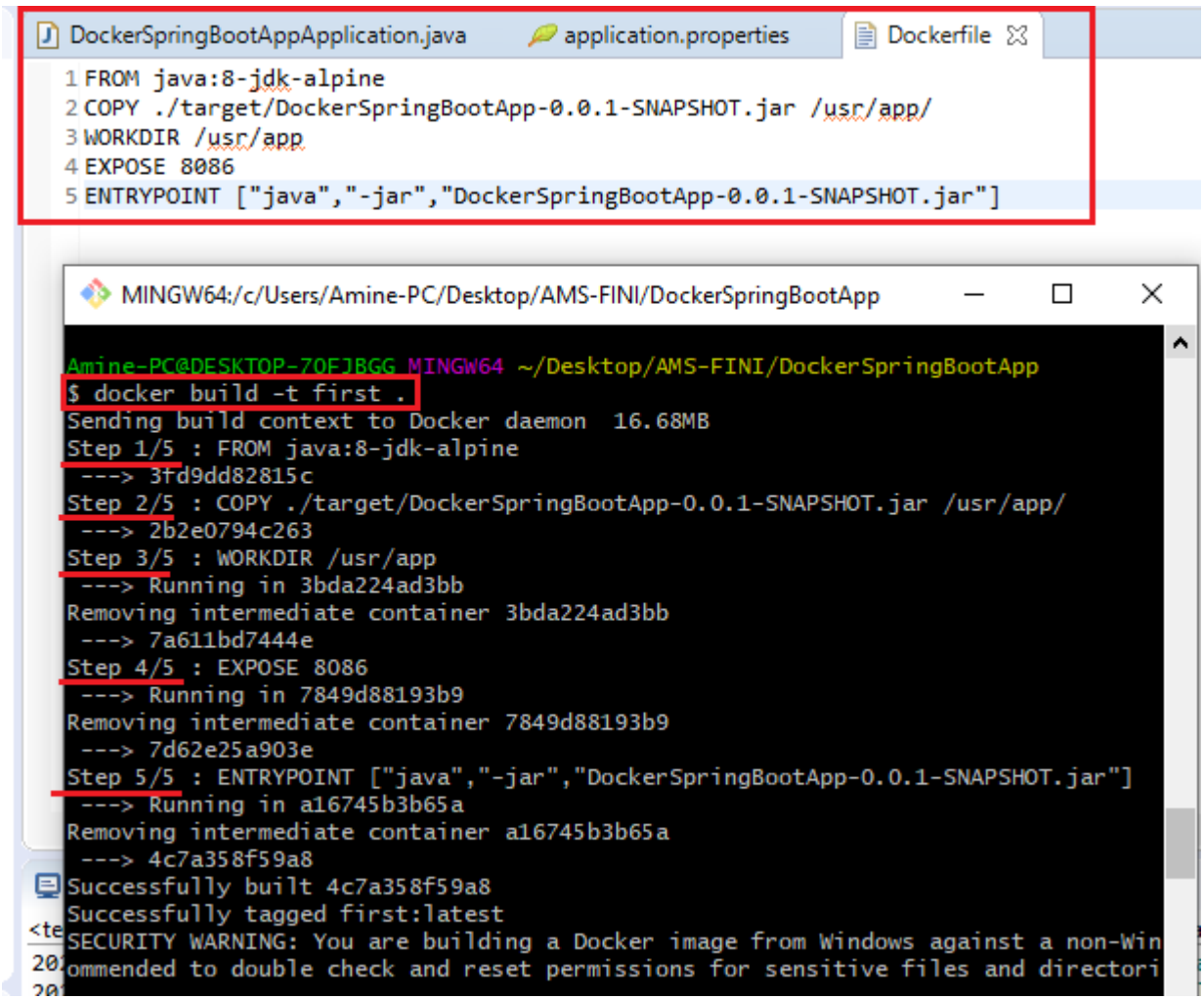
Let's take a look at the commands and fully understand them before proceeding:

- **FROM –** The keyword FROM tells Docker to use a given base image as a build base. We have used 'java' with tag '8-jdk-alpine'. Think of a tag as a version. The base image changes from project to project. You can search for images on docker-hub.
- **COPY** - This tells Docker to copy files from the local file-system to a specific folder inside the build image. Here, we copy our .jar file to the build image (Linux image) inside /usr/app.
- **WORKDIR** - The WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow in the Dockerfile. Here we switched the workdir to /usr/app so as we don't have to write the long path again and again.
- **ENTRYPOINT** - This allows you to configure a container that will run as an executable. It's where you tell Docker how to run your application. We know we run our spring-boot app as java -jar <app-name>.jar, so we put it in an array.

Let's build the image using this Dockerfile. To do so, move to the root directory of the application and run this command:

```
$ docker build -t first .
```

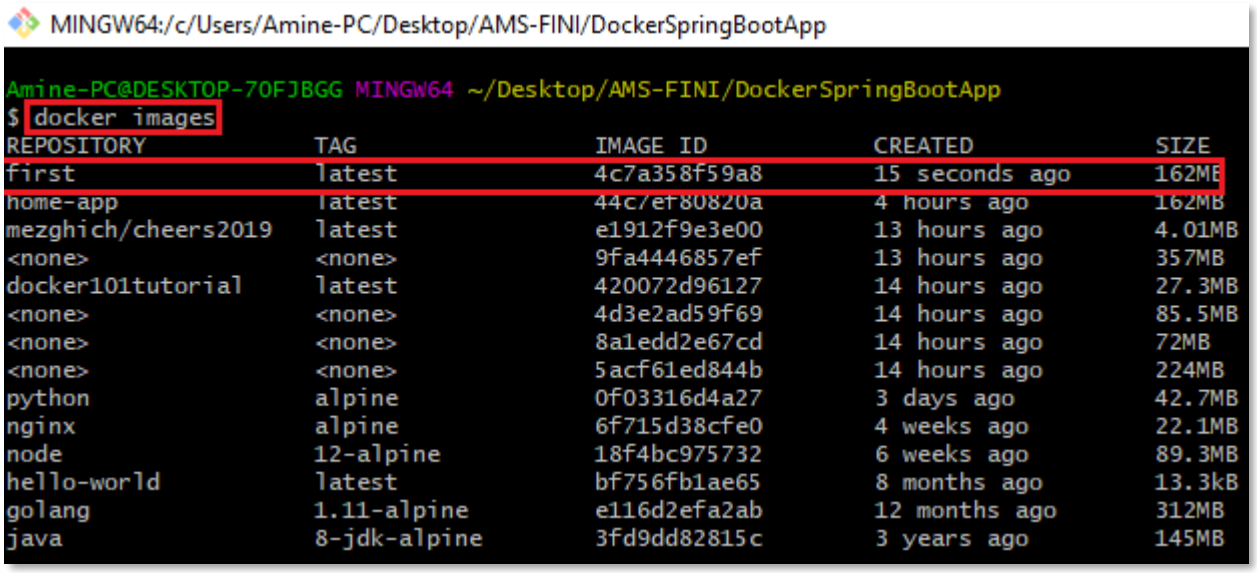
Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins



We built the image using `docker build`. We gave it a name with the `-t` flag and specified the current directory where the Dockerfile is. The image is built and stored in our local docker registry. Let's check our image:

```
$ docker images
```

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins



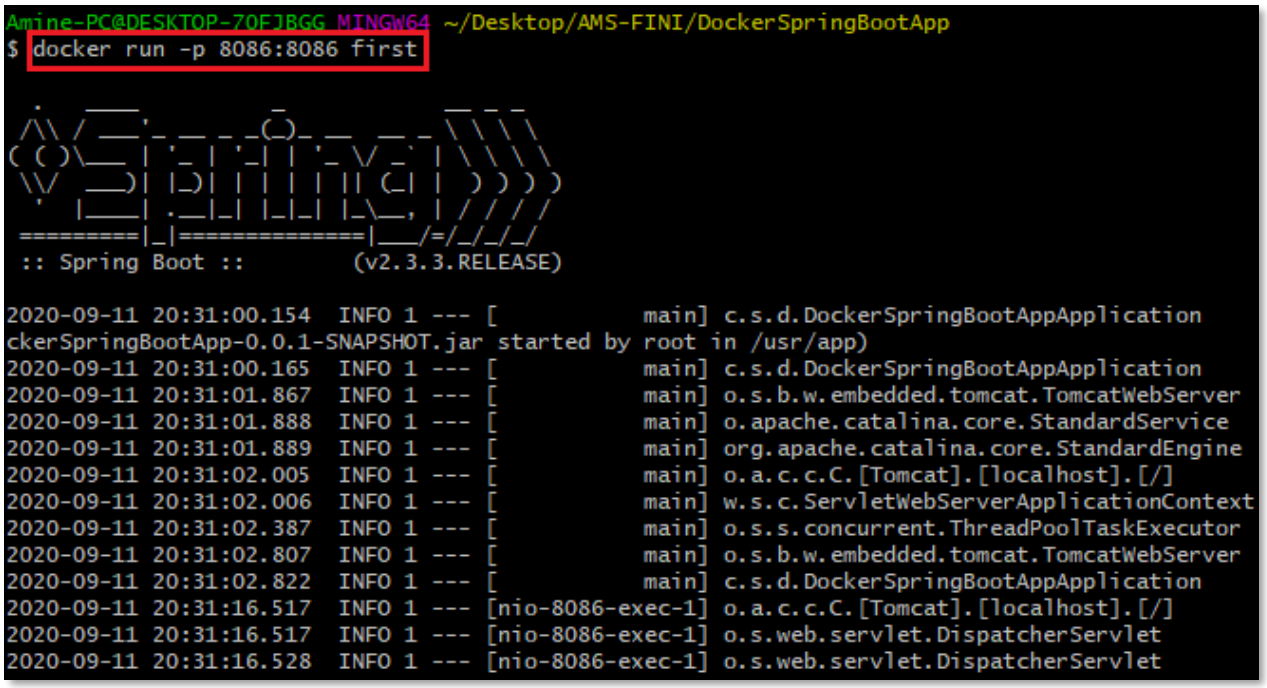
And finally, let's run our image:

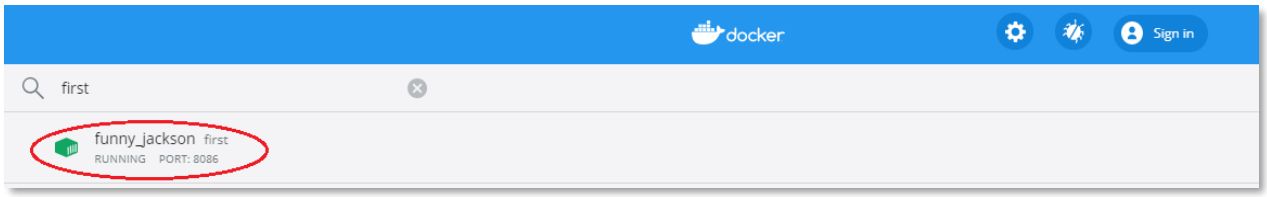
```
$ docker run -p 8086:8086 home-app
```

We can run Docker images using the `docker run` command.

We know that each container is an isolated environment in itself and we have to map the port of the host operating system - `8086` and the port inside the container - `8086`, which is specified as the `-p 8086:8086` argument.

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins





With the `docker ps`, we can list the running containers.

We stop the container with the `docker stop` command.

Example docker stop 4c7a358f59a8

Running the docker image in the background, in detached mode.

You can use the `-d` option in `docker run` command to run the container in the background –

The above command starts the container in the background and gives you the container ID. You can see the list of all containers running in your system using the following command -

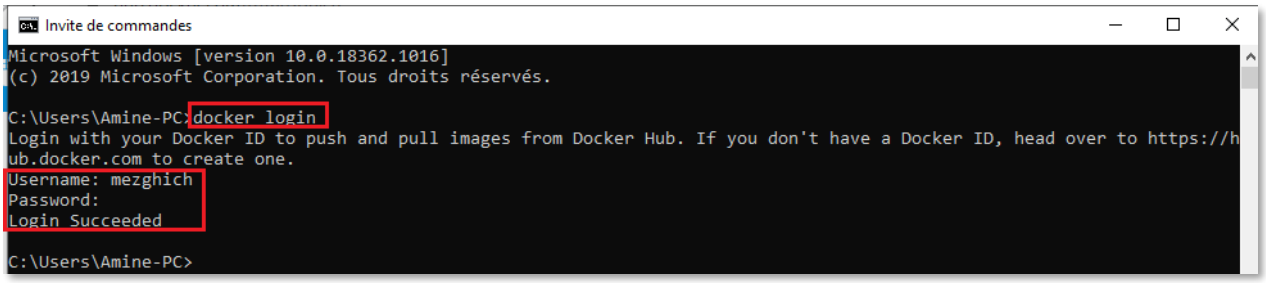
```
$ docker container ls
```

5.3- Pushing the docker image to docker hub

Now let's push the docker image to docker hub so that other people can download and consume our image.

5.3.1-Login with your Docker Id

- docker login



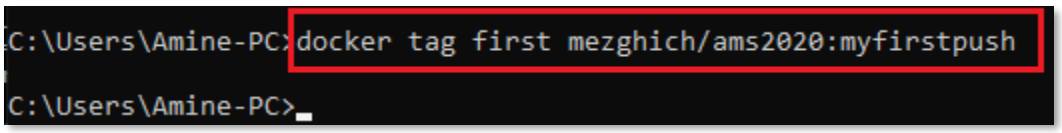
5.3.2-Tag the image

To push a local image to docker registry, you need to associate the local image with a repository on the docker registry. The notation for the repository on docker registry is `username/repository:tag`.

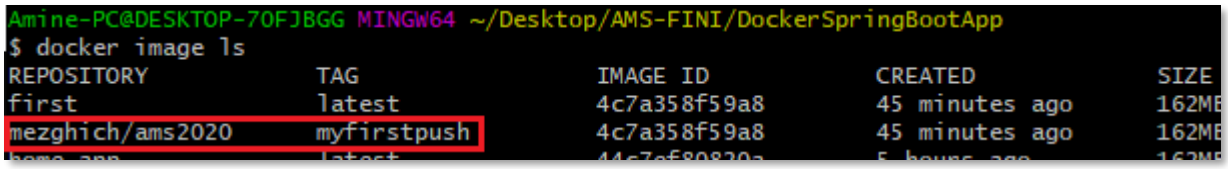
To tag an image, we use the `docker tag` command -

```
$ docker tag image username/repository:tag
```

For example, Here is how we can tag the local image of our spring boot application -

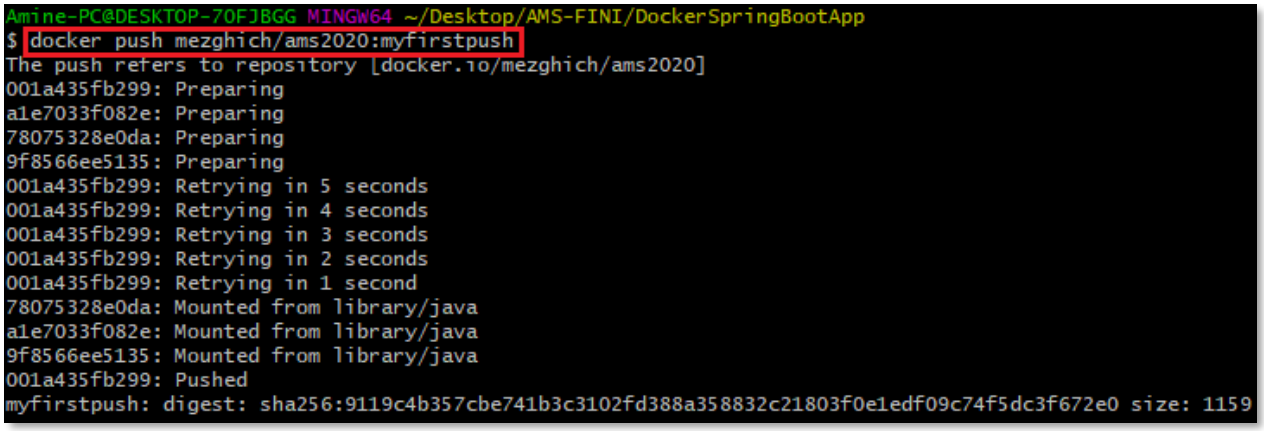


Now tape docker image ls in the terminal



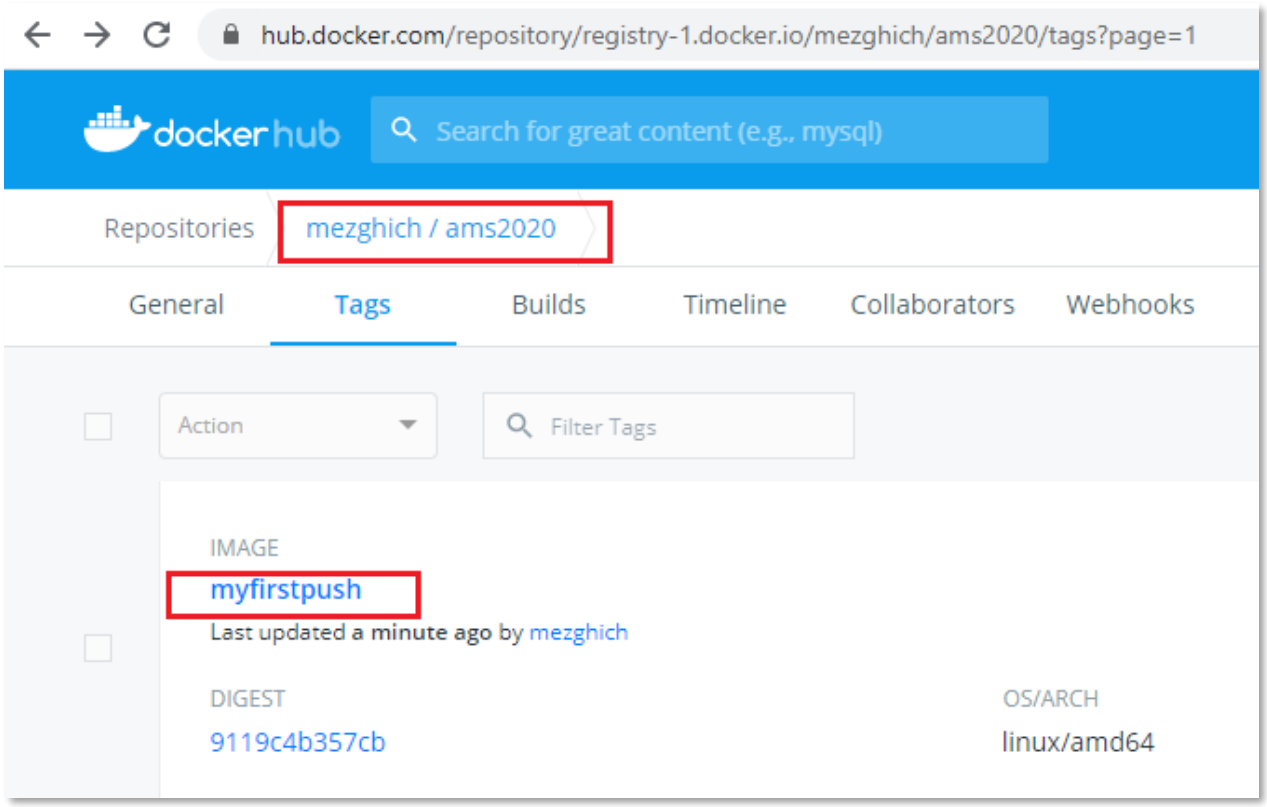
5.3.3-Push the image to docker hub

Finally, use the `docker push` command to push the tagged image to docker hub like so -



On my docker hub

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins



We remove the local image with the `docker rmi` command.

```
Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/AMS-FINI/DockerSpringBootApp
$ docker image ls
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
first                   latest              4c7a358f59a8        45 minutes ago      162MB
mezghich/ams2020        myfirstpush         4c7a358f59a8        45 minutes ago      162MB
home-app                latest              44c7ef80820a        5 hours ago         162MB
mezghich/cheers2019     latest              e1912f9e3e00        14 hours ago        4.01MB
```

```
Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/AMS-FINI/DockerSpringBootApp
$ docker rmi -f 4c7a358f59a8
Untagged: first:latest
Untagged: mezghich/ams2020:myfirstpush
Untagged: mezghich/ams2020@sha256:9119c4b357cbe741b3c3102fd388a358832c21803f0e1edf09c74f5dc3f672e0
Deleted: sha256:4c7a358f59a8ac4411e835bab525eaea5d7370f801a16637beb3c3f710e6b055
Deleted: sha256:7d62e25a903e9adb65c579d723c32cdfc57e54dc8378fac014dff96efbbae168
Deleted: sha256:7a611bd7444e5d958a18a4d2b117db7ebd4247b576cc42ed3226b6154a939734
Deleted: sha256:2b2e0794c263587ed868881ef6c47329cf29b3406bc77a63366973575d54266c
```

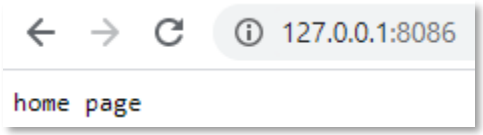
We can see that the image is being removed.

```
Amine-PC@DESKTOP-7OFJBGG MINGW64 ~/Desktop/AMS-FINI/DockerSpringBootApp
$ docker image ls
REPOSITORY            TAG                 IMAGE ID            CREATED             SIZE
home-app              latest              44c7ef80820a        5 hours ago         162MB
mezghich/cheers2019   latest              e1912f9e3e00        14 hours ago        4.01MB
<none>                <none>              9fa4446857ef        14 hours ago        357MB
docker101tutorial     latest              420072d96127        15 hours ago        27.3MB
<none>                <none>              4d3e2ad59f69        15 hours ago        85.5MB
<none>                <none>              8a1edd2e67cd        15 hours ago        72MB
<none>                <none>              5acf61ed844b        15 hours ago        224MB
python                alpine              0f03316d4a27        3 days ago          42.7MB
nginx                 alpine              6f715d38cfe0        4 weeks ago         22.1MB
node                  12-alpine           18f4bc975732        6 weeks ago         89.3MB
hello-world           latest              bf756fb1ae65        8 months ago        13.3kB
golang                1.11-alpine         e116d2efa2ab        12 months ago       312MB
java                  8-jdk-alpine        3fd9dd82815c        3 years ago         145MB
```

So let's install it

```
C:\Users\Amine-PC>docker pull mezghich/ams2020:myfirstpush
myfirstpush: Pulling from mezghich/ams2020
709515475419: Already exists
38a1c0aaa6fd: Already exists
5b58c996e33e: Already exists
c6b3c05ba05b: Already exists
Digest: sha256:9119c4b357cbe741b3c3102fd388a358832c21803f0e1edf09c74f5dc3f672e0
Status: Downloaded newer image for mezghich/ams2020:myfirstpush
docker.io/mezghich/ams2020:myfirstpush

C:\Users\Amine-PC>docker images
REPOSITORY           TAG                 IMAGE ID            CREATED             SIZE
mezghich/ams2020     myfirstpush         4c7a358f59a8        About an hour ago   162MB
home-app             latest              44c7ef80820a        5 hours ago         162MB
```

```
C:\Users\Amine-PC>docker run -d -p 8086:8086 mezghich/ams2020:myfirstpush
fe21908aae7b15f4f4e567b3b2cdcea1497c0672d659de05b2fbcd3827f9b186
```



You see how easy it is to share your image with others. People don't need to install anything whatsoever to run your application. They just need to pull the image and run it with docker.

5.3.4-Automating the Docker image creation and publishing using dockerfile-maven-plugin

You can automate everything from building the docker image to publishing it on docker hub using dockerfile-maven-plugin.

Add the plugin to the pom.xml file with the following configurations -

```
        <plugin>
                <groupId>com.spotify</groupId>
                <artifactId>dockerfile-maven-plugin</artifactId>
                <version>1.4.0</version>
                <configuration>
                        <repository>mezghich/ams2020</repository>
                        <tag>${project.version}</tag>
                        <buildArgs>

<JAR_FILE>target/${project.build.finalName}.jar</JAR_FILE>
                        </buildArgs>
                </configuration>
                <executions>
                        <execution>
                                <id>default</id>
                                <phase>install</phase>
                                <goals>
                                        <goal>build</goal>
                                        <goal>push</goal>
                                </goals>
                        </execution>
                </executions>
        </plugin>
```

6- Exercise 3 (Springboot App & Mysql on docker):

Formation: Devops **Formateur :** Dr. Mohamed Amine MEZGHICH **Durée de la formation :8h** **Email :** ma.mezghich@smart-it-partner.com	**Workshop DEVOPS:** **Objectifs:** 1-GIT 2-Docker 3-Maven 4-Jenkins

https://springbootdev.com/2017/11/30/docker-spring-boot-and-spring-data-jpa-mysql-rest-api-example-with-docker-without-docker-compose/

7- Exercice 4(Angular App on docker)

https://dzone.com/articles/how-to-dockerize-angular-app

8- Commands

https://springbootdev.com/2017/11/10/docker-most-important-and-frequently-used-commands/