A simple solution to design a good path to Obtain an Automated Deploy

Aka DevOps, Continuous Integration and Continuous Delivery how to

Summary

Abstract

This guide want to explain how design and build a simple path to automated deploy, you can learn how to install and configure, some product that can be support this solution, we use:

Git version 2.7.4
GitHab
Jenkins ver. 2.89.4
Apache/2.4.18 (Ubuntu)
Eclipse IDE Oxygen2
Ubuntu 16.04.4 LTS
WebSphere 8.5x

Windows/Unix Differences

This guide was written using Linux as the base operating system, however the steps/concepts listed in this guide are independent of operating system.
The only significant difference is that for Windows, you must use the batch file commands instead of the UNIX shell commands listed in this guide.
For example:
UNIX: ./startServer.sh WebSphere_Portal
Windows: startServer.bat WebSphere_Portal
Or
UNIX: ./ConfigEngine.sh cluster-node-config-cluster-setup
Windows: ConfigEngine.bat cluster-node-config-cluster-setup

Hostnames Used in this Guide

To avoid confusion with my own hostnames, I've replaced each instance of the hostnames of my servers with a sample value that corresponds to the server it belongs to so that it may be easier to understand which server I'm referring to in my examples.
I use the following values:
Jenkins Server: build.net2action.com

DevOps Concepts

Today many Organizations need to create innovative applications or services to solve business problems, they may need to address internal business problems or to help their customers to develop new business or new application. Many organizations to lost too much time and resource during the path to perform an IT Project and finally aren't successful. However, and their failures are often related to challenges in software development and delivery.

Although most enterprises feel that software development and delivery are critical, a recent IBM survey of the industry found that only 25% believe that their teams are efficient. DevOps applies agile and lean principles across the entire software supply chain.

It enables a business to maximize the speed of its delivery of a product or service, from initial idea to production release. This philosophy want to engage a customer to have feedback and develop features and enhancements based on that feedback.
DevOps improves the way that a business delivers value to its customers, suppliers, and partners. It's an essential business process, not just an IT capability.

DevOps provides significant return on investment in three areas:

- ✓ Enhanced customer experience
- ✓ Increased capacity to innovate
- ✓ Faster time to deliver

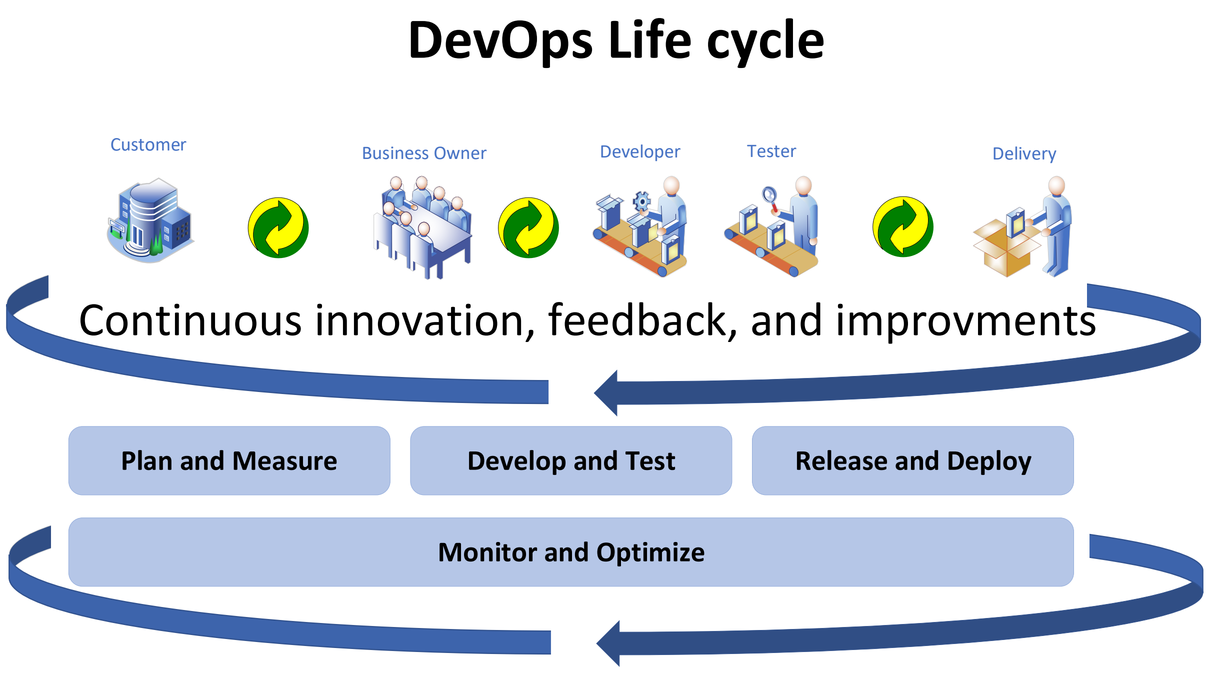
The DevOps philosophy has produced several principles that have evolved over time and are still evolving. All major solution providers, including IBM, have developed their own philosophy.

All these principles, however, take a holistic approach to DevOps, and organizations of all sizes can adopt them.
These principles are

- ✓ Develop and test against production-like systems
- ✓ Deploy with repeatable, reliable processes
- ✓ Monitor and validate operational quality
- ✓ Amplify feedback loops

DevOps Adoption

The capabilities that make up DevOps are a broad set that span the software delivery life cycle. Where an organization starts with DevOps depends on its business objectives and goals. what challenges it's trying to address and what gaps in its software delivery capabilities need to be filled.
This logical architecture provides a template of a proven solution by using a set of predefined methods and capabilities. These capabilities in turn may be provided by a single component or a group of components working together. This idea can be evolves to concrete form, these capabilities are provided by a set of effectively enabled people, defined practices (methodology) , and automation tools.



The DevOps Life cycle architecture shown in figure proposes the following four sets of adoption paths:

- ✓ Plan and measure
- ✓ Develop and test
- ✓ Release and deploy
- ✓ Monitor and optimize

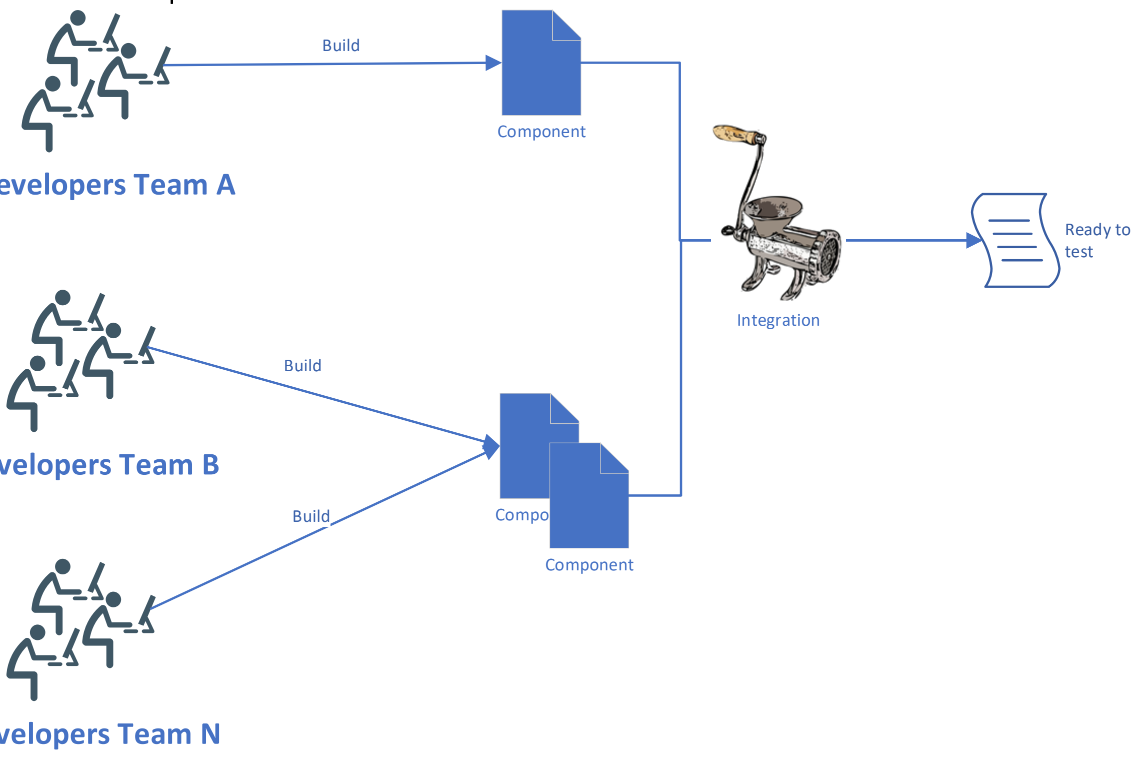
Plan and measure

This module consists of one practice that focuses on the lines of business and their analysis, that will become planning processes: **Continuous Planning**.
Businesses need to be agile and able to react quickly to customer feedback, consequently, many Company today must employ light thinking techniques. These techniques begin by identifying the outcomes and resources needed to test the business goal, and then continuously adapting and adjusting based on customer feedback. To achieve these goals, organizations can measure progress, find out what customers really want, and then adjust direction by updating their business plans accordingly, allowing them to make continuous trade-off decisions, without thinking to have an infinite resources environment.

Develop and test

This module will be develop in two practices: **Collaborative Development** and **Continuous Testing.** Collaborative development enables these developer to work together by providing a common set of artefacts, that they can use to create and release software. The software delivery efforts today involve more of cross-functional teams, including lines of business owners, business analysts, enterprise and software architects, developers, QA tester, security specialists, suppliers, and partners. The members of these teams can be work on multiple platforms and may be spread across multiple locations. The natural answer is "Collaboration".

The developers will be involved in Continuous Integration path, because we'll have more concurrent team to release a specific solution.



The idea is that the developers to regularly integrate their work with that of the rest of the developers on their team and then test the integrated work.

In the complex systems made up of multiple systems or services, the developers also regularly integrate their work with the others.

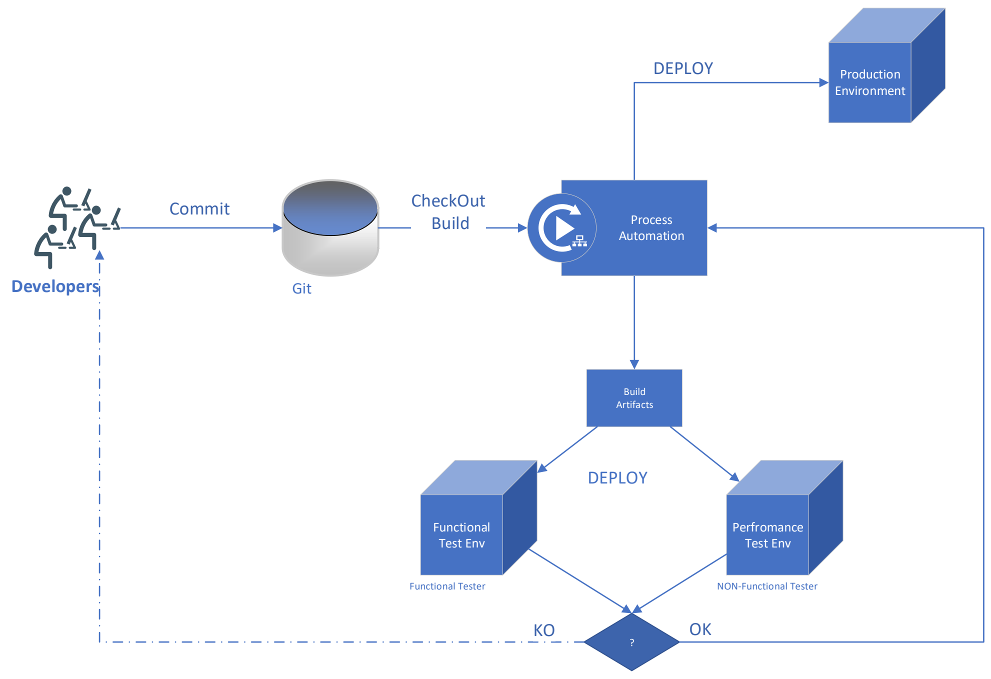
Regular integration of results leads to early discovery the possibility exposure of integration risks.

Continuous testing can permit to obtains the testing earlier and continuously, which results in continuous feedback about quality. This occurs by adopting capabilities that grant to execute some task automatically, and can perform testing in an environment like production-like environments and that permit continuous testing feasible.

Release and Deploy

Continuous release and deployment derived the idea from continuous integration. The practice that enables release and deploy also enables the creation of a delivery pipeline. This pipeline facilitates continuous deployment of software to QA and then to production in an efficient, automated manner.

The goal of continuous release and deployment is to release new features to customers and users as soon as possible.



Monitoring and Optimize

The monitor and optimize adoption path includes two sub path, Continuous monitoring and Customers' Continuous Feedback .
That allow businesses to monitor how released applications are performing in production and use feedback from customers, to tune the application with these feedbacks.
Continuous monitoring provides data and metrics to QA, Development, lines-of-business, and other stakeholders about applications at different stages of the delivery cycle. These metrics aren't limited to production environment, they are collected in each environment, between Developers and Production. Such metrics allow stakeholders to react by enhancing or changing the features being delivered and/or the business plans required to deliver them.
Customer's continuous feedback, allows you to collect the two most important types of information that a software delivery team can get is data on how customers use the application. New technologies enable companies to capture customer behaviour and critical customer points while using the application. This feedback allows different stakeholders to take appropriate action to improve applications and improve the customer experience. Business lines can adjust their business plans, development can adjust the capabilities it provides and the systems engineer can improve the environment in which the application is deployed.
This continuous feedback loop is an essential component of DevOps, allowing companies to be more agile and responsive to customer needs.

Infrastructure Technologies

Technology enables people to focus on high-value creative work while delegating routine tasks to automation. Technology also allows teams of practitioners to leverage and scale their time and abilities.
Standardizing automation also makes people more effective, in some Organizations may experience turnover in employees, contractors, or resource providers; the people may move from project to project.

A common set of tools allows practitioners to work anywhere, and new team members need to learn only one set of tools — a process that's efficient, cost-effective, repeatable, and scalable.
In this articles, I discuss about a specific set of tools that can working together to simplify DevOps adoption, following:

- Git & GitHub, as repository and version control
- Jenkins, as orchestrator
- Jira suite as management and planning

So, start our implementation.

We must start from package repository. A package repository (also referred to as an asset repository or artefact repository) is a common storage mechanism for the binaries created during the build stage. These repositories also need to store the assets associated with the binaries to facilitate their deployment, such as configuration files, infrastructure-as-code files, and deployment scripts.

Install and configure repository

Install git your local repository

Connect to your Ubuntu machine, and bring the root privileges, with command

```
user01@ubuntu:~$ sudo -i
[sudo] password for user01: <type root password>
root@ubuntu:~#
```

update software repository and install git

```
sudo apt-get update
sudo apt-get install git -y
```

Configure the username, replace sample user

```
git config --global user.name "your git user admin”
```

Configure the email, replace sample

```
git config --global user.email "gitAdmin@example.com"
```

Now that Git has been installed, first step will be initialize it:

If you must attach your local path artifacts to git, you must move to this directory and init git from it.

Make directory /opt/JavaProject and inside of it make Project1 and Project2

```
mkdir -p /opt/JavaProject/Project1
mkdir -p /opt/JavaProject/Project2
```

build two sample classi like inside of them:

```
public class first {

  public static void main(String[] args){

    for(int i=1;i<=10;i++){
        System.out.println(" my first java class...");
    }
  }
}

public class second {

  public static void main(String[] args){

    for(int i=1;i<=10;i++){
        System.out.println(" my second java class...");
    }
  }
}
```

Called first.class and second.class

And from /opt/JavaProject run

```
Git status
--------
root@ubuntu:/opt/JavaProject# git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

              Project1/
              Project2/

nothing added to commit but untracked files present (use "git add" to track)
```

to commit your change on git run

```
git add .
```

from /opt/JavaProject

and again check the status

```
root@ubuntu:/opt/JavaProject# git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   Project1/first.class
        new file:   Project2/second.class
```

now you are ready to commit, with command:

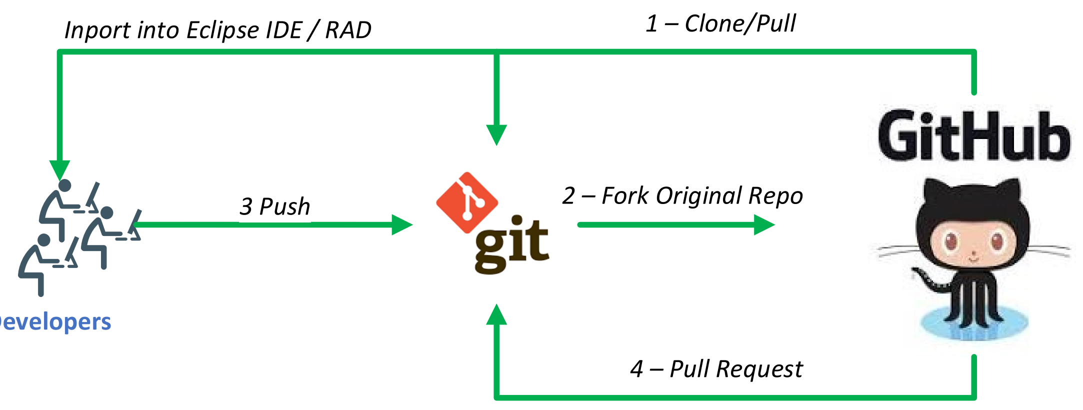
```
git commit -m "commit my first & second class" (-m commentabout commit is mandatory)

root@ubuntu:/opt/JavaProject# git commit -m "commit my first & second class"
[master (root-commit) c2c6355] commit my first & second class
 2 files changed, 18 insertions(+)
 create mode 100644 Project1/first.class
 create mode 100644 Project2/second.class
```

and the status will be:

```
root@ubuntu:/opt/JavaProject# git status
On branch master
nothing to commit, working directory clean
root@ubuntu:/opt/JavaProject#
```

Now we can configure git and GitHub to working together refer to the image below for help.

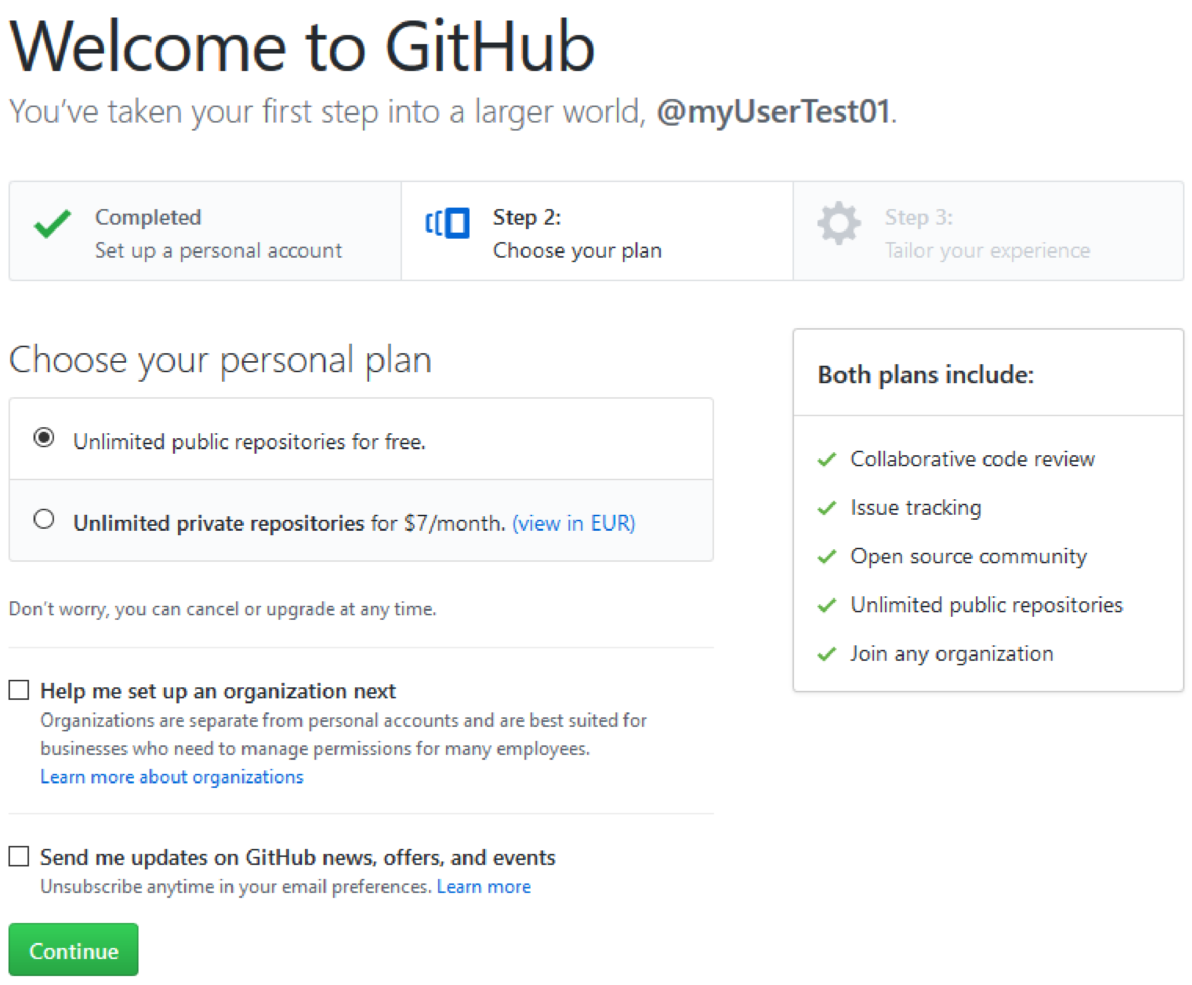


Configure Git and GitHub to working together

Go to github home page and register your account:

https://github.com/

register as “myUserTest01” (use different this is already used by me)



Welcome to GitHub

You’ve taken your first step into a larger world, **@myUserTest01**.

✓ Completed Set up a personal account	Step 2: Choose your plan	Step 3: Tailor your experience

Choose your personal plan

- [✓] Unlimited public repositories for free.
- [ ] **Unlimited private repositories** for $7/month. (view in EUR)

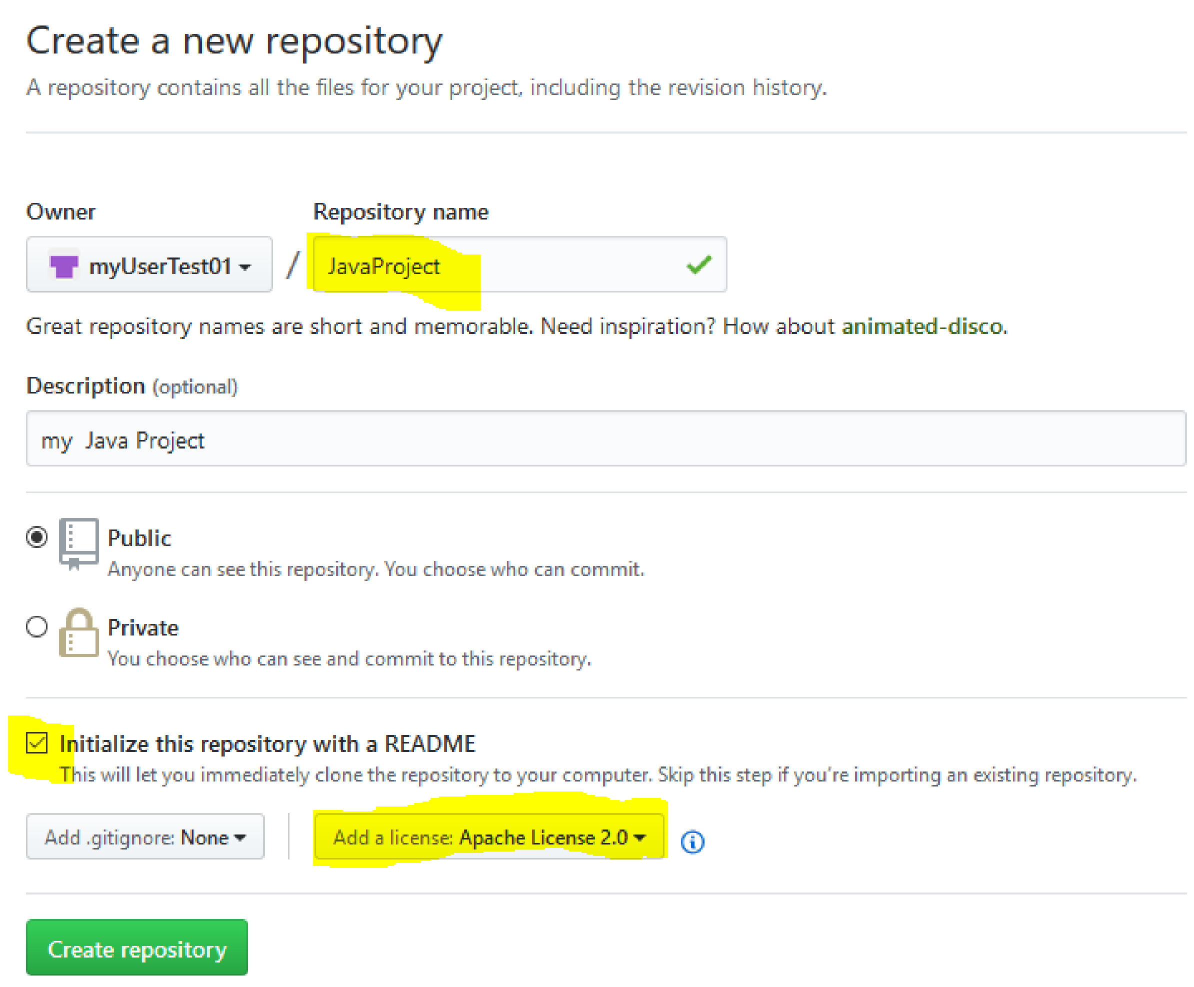
Don’t worry, you can cancel or upgrade at any time.

- [ ] **Help me set up an organization next**
  Organizations are separate from personal accounts and are best suited for businesses who need to manage permissions for many employees.
  Learn more about organizations
- [ ] **Send me updates on GitHub news, offers, and events**
  Unsubscribe anytime in your email preferences. Learn more

Both plans include:

- ✓ Collaborative code review
- ✓ Issue tracking
- ✓ Open source community
- ✓ Unlimited public repositories
- ✓ Join any organization

Choose your profile, public or private and continue
Confirm your email and create first project, choose repository name, and if you like reference license type.



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: myUserTest01 / Repository name: JavaProject ✓

Great repository names are short and memorable. Need inspiration? How about animated-disco.

Description (optional): my Java Project

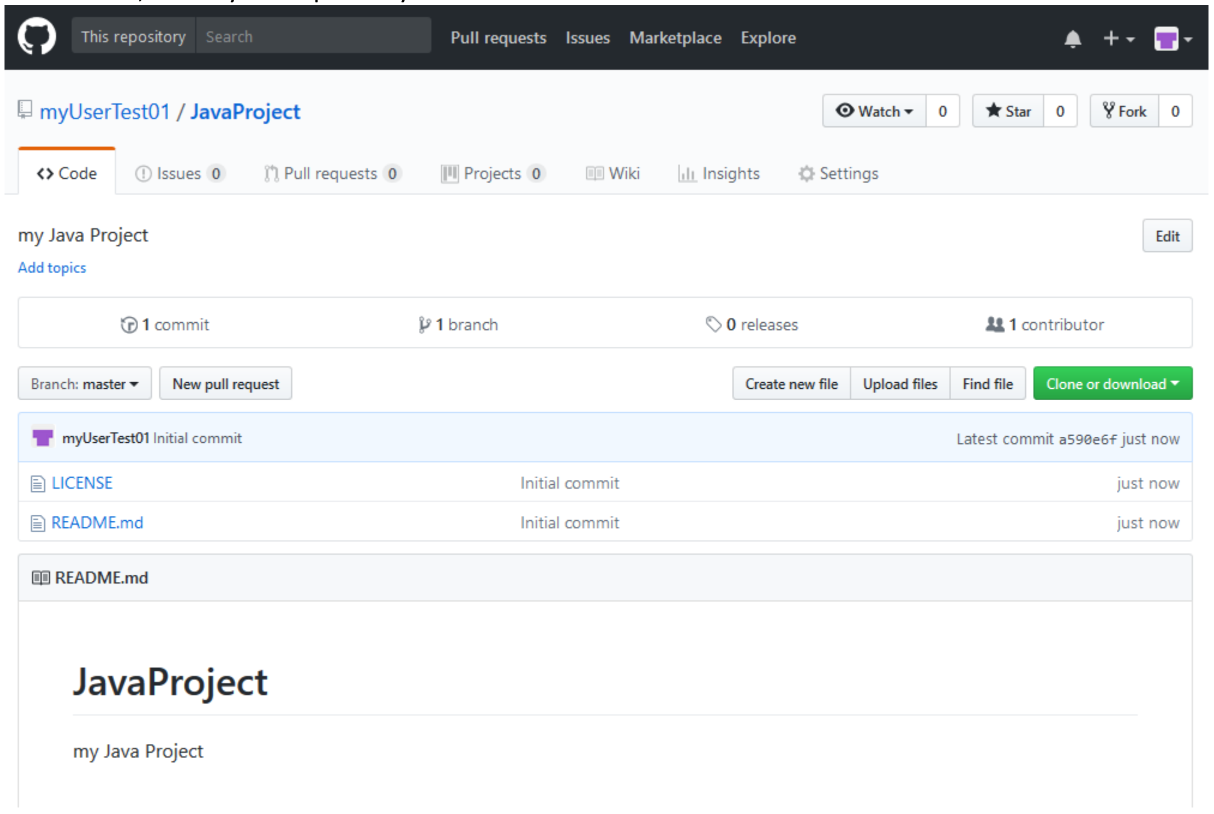
● Public
Anyone can see this repository. You choose who can commit.

○ Private
You choose who can see and commit to this repository.

☑ Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: Apache License 2.0

And Create, this is your repository with master branch.



myUserTest01 / JavaProject

my Java Project

1 commit | 1 branch | 0 releases | 1 contributor

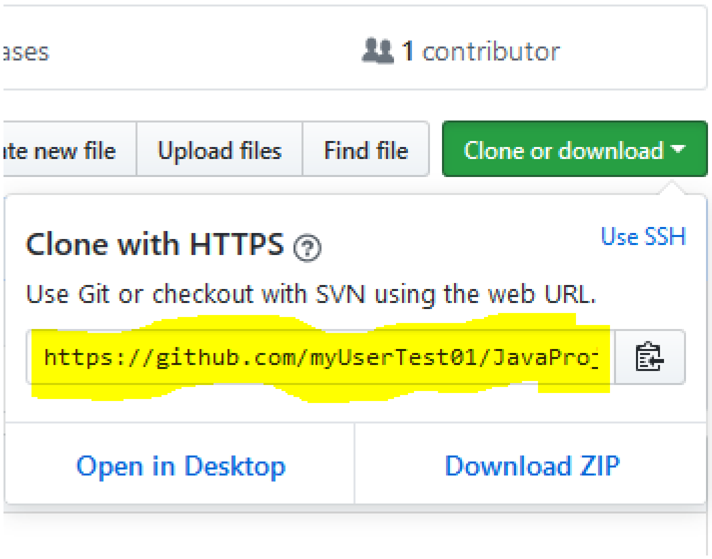
myUserTest01 Initial commit		Latest commit a590e6f just now
LICENSE	Initial commit	just now
README.md	Initial commit	just now

README.md

JavaProject

my Java Project

Now we can connect your GitHub cloud repository with your local Git repository, clik on “Clone or Download” and copy url from popup.



Now come back to your git environment and attach your GitHub repository, to doing execute following command:

```
git remote add origin https://github.com/myUserTest01/JavaProject.git
```

and now you can push your project to GitHub

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

Username for 'https://github.com': myUserTest01
Password for 'https://myUserTest01@github.com':
To https://github.com/myUserTest01/JavaProject.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/myUserTest01/JavaProject.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

opps.. you received an error….. why?

Because when you create repository you have choose Licence and readme sample, and you do not have in your local repository, you must get it before push for first time your project ….

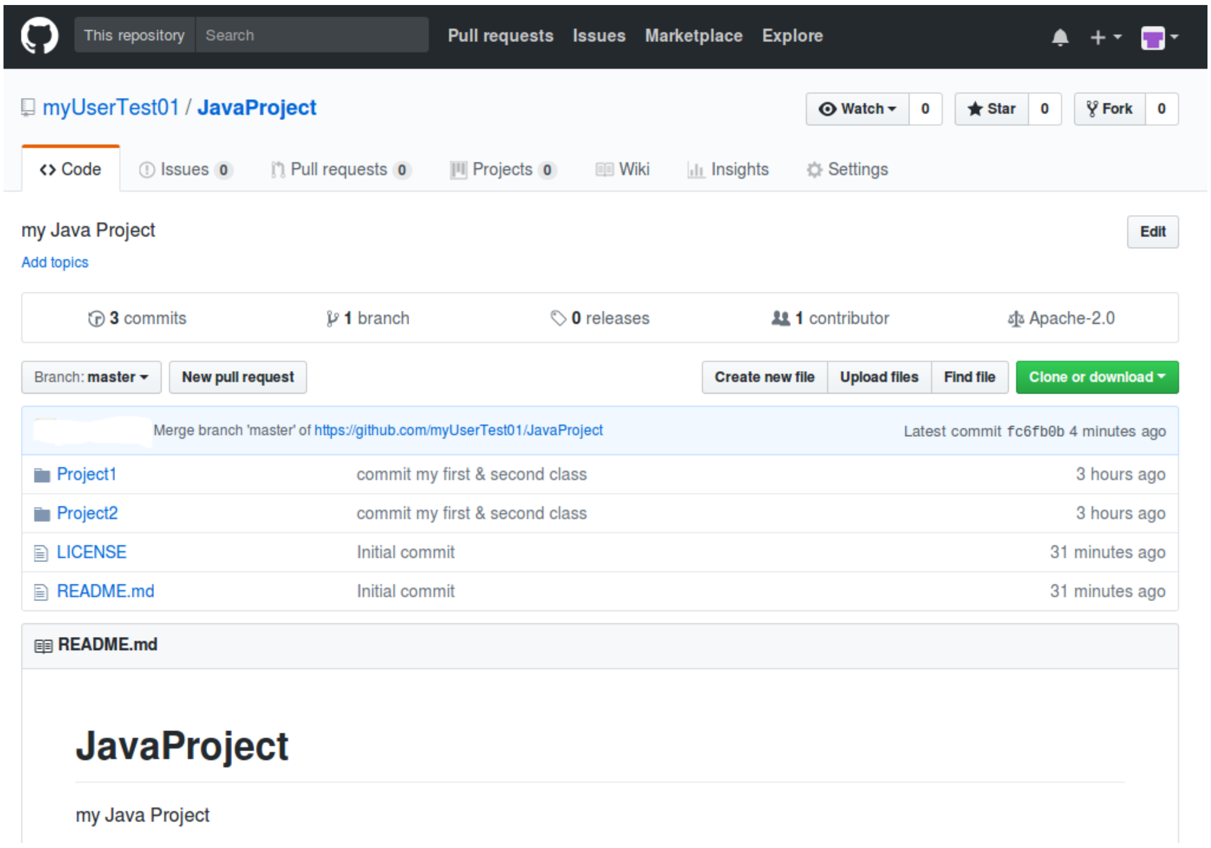
To get update from your remote repository, use following command:

```
git pull https://github.com/myUserTest01/JavaProject.git master
---- ------------------------
warning: no common commits
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), done.
From https://github.com/myUserTest01/JavaProject
 * branch            master     -> FETCH_HEAD
Merge made by the 'recursive' strategy.
 LICENSE   | 201
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++
 README.md |   2 ++
 2 files changed, 203 insertions(+)
 create mode 100644 LICENSE
 create mode 100644 README.md
```

Now you can push your project

```
git push -u origin master
----- -------------------------------------------------------------
Username for 'https://github.com': myUserTest01
Password for 'https://myUserTest01@github.com':
Counting objects: 8, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (8/8), 879 bytes | 0 bytes/s, done.
Total 8 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/myUserTest01/JavaProject.git
   a590e6f..fc6fb0b  master -> master
Branch master set up to track remote branch master from origin.
```

Now you have attach your git with your GitHub, and if you go to github and refresh page you can see your last push activities.



myUserTest01 / JavaProject

my Java Project

3 commits | 1 branch | 0 releases | 1 contributor | Apache-2.0

Merge branch 'master' of https://github.com/myUserTest01/JavaProject Latest commit fc6fb0b 4 minutes ago

Project1	commit my first & second class	3 hours ago
Project2	commit my first & second class	3 hours ago
LICENSE	Initial commit	31 minutes ago
README.md	Initial commit	31 minutes ago

README.md

JavaProject

my Java Project

In next articles, we can see how attach your git/GitHub with Jenkins, our Orchestrator…..

Andrea Fontana
IT Architect & WebSphere Collab. Solution Specialist
a.fontana@net2action.com
mobile: +39.334.8314267
IBM CHAMPION
IBM developerWorks Contributing Author