CI/CD Set up

1 About

Continuous Integration/Continuous Deployment (CI/CD) is an important software development process. The idea is that when a developer pushes to the repository, the entire code is built, tested, and deployed automatically. If any of these three stages fail, then the developer would need to fix the problem and try again. This way, any successful push would imply a properly tested version that is also deployed. Note that pushes to all branches need not go through this process. Usually, pushes to production or master branches would need this kind of rigor.

Here, we will discuss gitlab specific CI/CD mechanisms.

The remaining document is organized in the following manner:

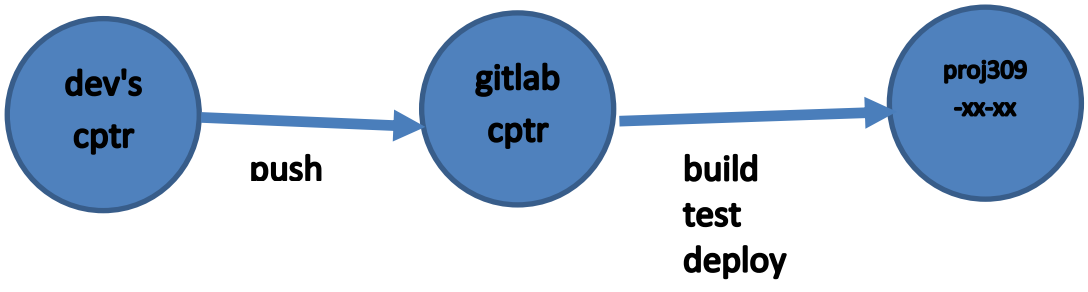
1. First, a brief overview of some concepts and terms.
2. Next, a brief description of the YAML file used for the example project.
3. Finally, concrete steps that each team needs to carry out to get CI/CD working for their project.

2 Concepts/Terms

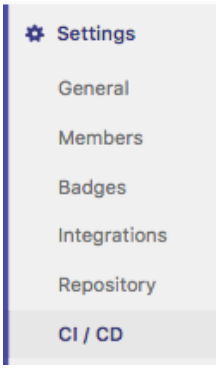
2.1 Computers involved

There are three main computers involved in this process:

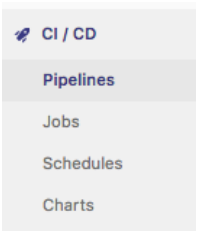
1. the developer's computer from which files are being pushed to gitlab,
2. the computer where gitlab is being hosted, and
3. the computer on which the "building, testing, and deployment" is taking place. In our case, this third computer is the one that has been given to each team (with the proj309-xx-xx.misc.iastate.edu name). Of course, we could deploy on a totally different fourth computer.



2.2 CI/CD on gitlab menu



On gitlab, there are two areas that deal with CI/CD. One is inside Settings menu on the left on gitlab. The last item on that list is CI/CD. This is used to set up the CI/CD operations.



The other area is the CI/CD menu on the left on gitlab. This is used to view the status of CI/CD operations.

2.3 runner

Runners are processes that will run on the project computer to accomplish the build, test, and deploy jobs. First, these need to be installed on the project computer. Next, the gitlab host needs to be linked to this runner (using the settings menu). Several runners with different capabilities may be used. For example, one runner may be able to use Docker container to provide an android environment and to help build and test android code. Another runner could be used to build and test Spring code.

When runners are created, they can be assigned "tags". These tags are used in the following configuration file to map build/test/deploy tasks to specific runners.

2.4 configuration file (.gitlab-ci.yml)

This file is very important and has detailed instructions on what to do to build, test, and deploy. These instructions are carried out by the runner(s).

Instructions are organized by stages. So, a pipeline is composed of stages. A stage consists of jobs. These jobs are carried out by specific runners.

3 Example CI/CD for 309 Android/Spring projects

The pet-clinic example has CI/CD enabled. You are encouraged to take a close look at it. Note that there are multiple ways to accomplish the goals. For example, one could use a Docker container in our runner to build and test Spring code. We have opted to use the shell instead.

The pet-clinic example uses two runners. One for android and the other for Spring.

The .gitlab-ci.yml file in the petclinic repository has instructions for all the stages.

The example file shown below has six parts. The first part enumerates the five stages in the pipeline. The other parts has instructions for each stage. Do not be intimidated by the YML file. It is pretty easy to read and follow. In addition, all you need to do is just copy/paste and make slight modifications.

You can learn about YAML (https://learnxinyminutes.com/docs/yaml/) here. You will find a few advanced tricks you can use to make your YAML file more helpful.

In the .gitlab-ci.yml file excerpts below, you will need to know of six keys:

a) stages (used to enumerate stages in the pipeline)
b) stage (indicates which of many stages in the pipeline these instrns refers to)
c) tags (this is used to indicate which of the many runners to use)
d) script (this is where instructions for the runner are listed)
e) artifacts (indicates location of key files)

3.1 First part (enumeration of stages in the pipeline)

```
# Stages of pipeline, should match with stage tag inside each job.
# Each stages executes in sequence, if previous job fails, then all the preceeding jobs
# are skipped.
stages:
  - mavenbuild
  - maventest
  - autodeploy
  - androidbuild
  - androidtest
```

3.2 Second Part (maven build instructions for the spring code)

```
# tags: "shell" should match with the tag name provided to runner, for spring runner should execute in shell.
# Notice that in petclinic project, the spring project is in root of master, if yours is in
# folder then cd to that folder.
# maven -B means batch mode (i.e. not interactive)
maven-build:
  stage: mavenbuild
  tags:
    - shell
  script:
    - cd Spring
    - mvn package -B
```

3.3 Third part (instructions to test the spring part)

```
# artifacts are created when job executes successfully, and can be manually downloaded from GitLab GUI.
# artifacts are not mandatory, but it is good practice, in case auto deploy fails, you
# can manually download the jar.
maven-test:
  stage: maventest
  tags:
    - shell
  script:
    - cd Spring
    - mvn test
  artifacts:
    paths:
      - Spring/target/*.jar
```

3.4 Fourth part (instructions to deploy the spring part)

```
# Below stage builds, deploys and executes the jar as service.
# Make sure there is /target directory in your server, else script will fail, or use any
# other folder you like.
# If you are changing the folder, reflect the change in systemd-web-demo service
autoDeploy:
  stage: autodeploy
  tags:
   - shell
  script:
    - cd Spring
    - mvn package
    - sudo mv target/*.jar /target/web-demo.jar
    - sudo systemctl stop systemd-web-demo
    - sudo systemctl start systemd-web-demo
```

Note that your TA created a service to run the spring code to make it easier to start and stop it. Nice!

3.5 Fifth part (instructions to build the android project)

```
# To build android projects
# Notice that we cd into Android_test folder which contains all Android files before
# building apk.
# Ensure the tag matches with the tag created for Android runner
# Android runner should have docker as executor.
android-build:
  image: javiersantos/android-ci:latest
  stage: androidbuild
  before_script:
    - export GRADLE_USER_HOME=`pwd`/.gradle
    - chmod +x ./Android_test/gradlew
  tags:
   - android_tag1
  script:
    - cd Android_test
    - ./gradlew assemble
  artifacts:
    paths:
    - Android_test/app/build/outputs/
```

3.6 Sixth part (instructions to test the android part)

```
# To run Android unit tests.
unitTests:
  image: javiersantos/android-ci:latest
  stage: androidtest
  before_script:
    - export GRADLE_USER_HOME=`pwd`/.gradle
    - chmod +x ./Android_test/gradlew
  tags:
   - android_tag1
  script:
    - cd Android_test
    - ./gradlew test
```

3.7 Advanced tricks

```
# You may not want to always autodeploy every commit you send to the repository.
# Especially if you are on an experimental branch and the feature you are
# implementing is incomplete or if the frontend is not ready for all of the changes the
#backend has implemented. What you can do in this situation is add some conditions
#to your yml file which will only run certain stages on a certain branch. You can also
#add conditions to detect a certain commit message to skip/run specific stages in
#your pipeline.
autoDeploy:
  stage: autodeploy
  tags:
   - shell
  only: # Will only run if these conditions are met, check documentation for more!
    refs:
      - master
    variables:
      - $CI_COMMIT_MESSAGE =~ /autodeploy/
  script:
    - cd Spring
    - mvn package
    - sudo mv target/*.jar /target/web-demo.jar
    - sudo systemctl stop systemd-web-demo
    - sudo systemctl start systemd-web-demo
```

4 CI/CD for 309 Android/Spring projects

What you need to do:

1. Install gitlab-runner on your server
2. Install docker on your server (for Android projects)
3. Install jdk on your server
4. Install maven on your server
5. Add the .gitlab-ci.yml to your project repository. Modify it as necessary.
6. Register the runners on the gitlab host
7. Test the working of your CI/CD

Some of the steps are described below.

4.1 Install GitLab-Runner on your server.

Source - https://docs.gitlab.com/runner/install/linux-manually.html

Step 1: Simply download one of the binaries for your system:

```
sudo wget -O /usr/local/bin/gitlab-runner https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-amd64
```

Step 2: Give it permissions to execute:

```
sudo chmod +x /usr/local/bin/gitlab-runner
```

Step 3: Create a GitLab CI user:

```
sudo useradd --comment 'GitLab Runner' --create-home gitlab-runner --shell /bin/bash
```

Step 4: Provide Sudo access to “GitLab Runner”

```
sudo usermod -aG wheel gitlab-runner
```

Step 5: We want to auto deploy jar to server using user name – “GitLab Runner”, we have given sudo permissions to “GitLab Runner” but we want to bypass giving password, so that our scripts are not interrupted. For that -

```
sudo visudo
```

Uncomment the line

#%Wheel ALL= (ALL) NOPASSWD: ALL

by removing the # in front by pressing `x (lower case)`

`:wq` to save and exit. (that is colon w q)

Important Note: DO NOT CORRUPT THIS FILE, IF YOU MAKE ANY UNWANTED CHANGES, SIMPLY EXIT BY ’:q!’ (exit without saving)

Step 6: Install and run as service:

```
sudo gitlab-runner install --user=gitlab-runner --working-directory=/home/gitlab-runner
```

```
sudo gitlab-runner start
```

Sometimes the gitlab runners are not started. You can try doing

```
sudo gitlab-runner run
```

4.2 Install Docker on your server.

Source: https://www.tecmint.com/install-docker-and-learn-containers-in-centos-rhel-7-6/

Step 1: Install docker

yum install docker

yum install epel-release

yum install docker-io

Step 2: After, Docker package has been installed, start the daemon, check its status and enable it system wide using the below commands

systemctl start docker

systemctl status docker

systemctl enable docker

Step 3: Enable Docker service

service docker start

service docker status

chkconfig docker on

Step 4: Verify if Docker is on - “You see a hello from docker message.”

docker run hello-world

4.3 Install JDK

Note that server has only JRE installed and not JDK, you try that by running javac option.

Source: https://www.digitalocean.com/community/tutorials/how-to-install-java-on-centos-and-fedora

To install JDK:

sudo yum install java-1.8.0-openjdk-devel

Verify JDK is installed by running javac option again.

4.4 Install maven

Source: https://www.tecmint.com/install-apache-maven-on-centos-7/

Step1: cd /usr/local/src

Step2: sudo wget http://www-us.apache.org/dist/maven/maven-3/3.5.4/binaries/apache-maven-3.5.4-bin.tar.gz

Step3: sudo tar -xf apache-maven-3.5.4-bin.tar.gz

Step4: sudo mv apache-maven-3.5.4/ apache-maven/

Step5: cd /etc/profile.d/

Step6: sudo vim maven.sh

Type i to enter insert mode and type in:

Apache Maven Environment Variables

MAVEN_HOME for Maven 1 - M2_HOME for Maven 2

export M2_HOME=/usr/local/src/apache-maven

export PATH=${M2_HOME}/bin:${PATH}

Step7: sudo chmod +x maven.sh

Step8: source /etc/profile.d/maven.sh

Step9: mvn --version

Maven should be running now.

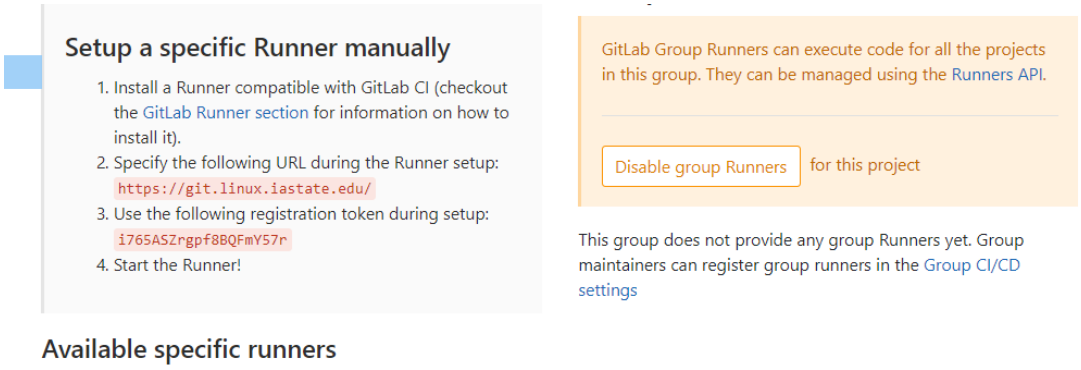
4.5 Register GitLab Runner

We would need at least 2 GitLab runners – Frontend + Backend.

Step1: Verify if you have any Runners registered already.

Navigate to settings-> CI/CD in your GITLAB repo. Expand runner option.

You would notice either you have no specific runner, or they are not enabled.



Available specific runners

Step2: To Create a specific runner for Frontend – Android.

- Type the command

`sudo gitlab-runner register` by logging into your server.

- You will be prompted to add a gitlab-ci coordinator url , enter-

https://git.linux.iastate.edu/

- Next you will be prompted to enter a token, you can find the token value in your git repo by navigating to *settings-> CI/CD, expand runner, under specific runner, you will find the registration token,* copy and paste the value in your console.
- Next enter any description
- Next enter tags, *tags are important,* as they will be used later, name them appropriately, so that it's easier for you to identify, just in case you end up creating a lot of runners – example – android_tag.
- Finally, you will be asked to select executor, *enter docker.*
- Default Docker image enter - *alpine:latest*
- Your runner is up and running and you can verify this by again navigating to *settings->CI/CD,* expanding runner, you should see an active runner created (green for active), with the provided tags.

Step3: Create a specific runner for Backend – Spring

- Repeat the same steps as above, name your tags appropriately and for *executor enter shell* and **NOT Docker.**

5 Acknowledgements

Preethi Pandian did the vast majority of work on this.

If some parts of this document are not clear, or you feel that there could be more examples – please let us know on Piazza.