> Data Science Technology and Systems PG
>
> Week 7
>
> Ibrahim Radwan
>
> 14/9/2022

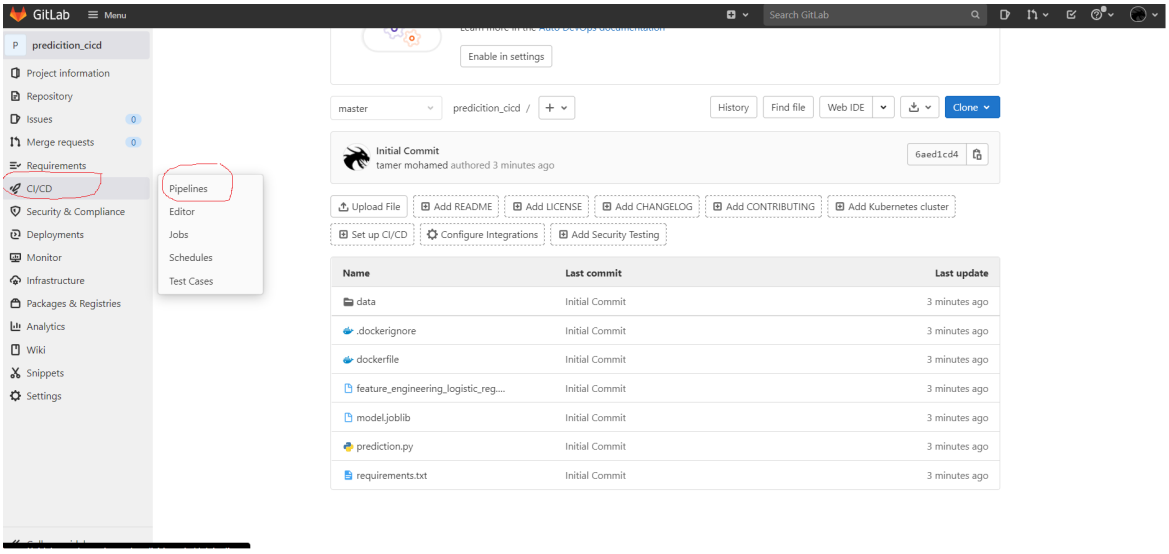
Continuous Integration, Delivery (CI/CD)

What is the CI/CD?

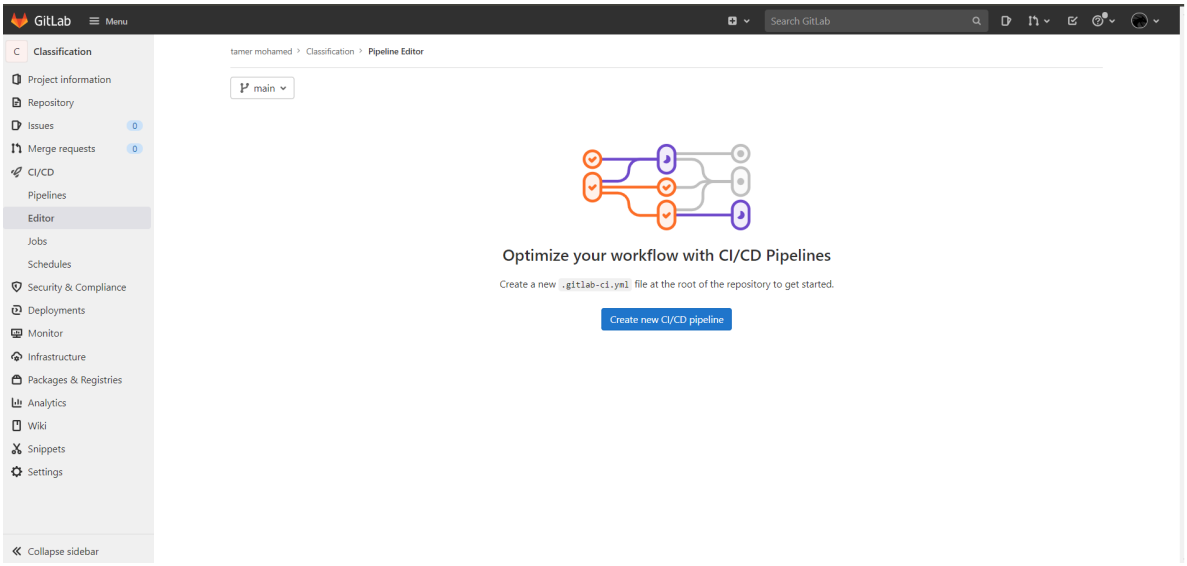
Let's discuss the problem first to understand wgy we need to use CI/CD. Imagine you are working with a team, and everyone does his own tasks, commits and pushes the repository, there may be an issue in the code pushed by one of the team, which needs to be checked to make sure that the combined code is running is not broken somewhere. Moreover, the application should be tested and checked to ensure that they cover all the required features and with the expected accuracy. Next, the code should be deployed on the servers in order to be used by the customers. Performing this process manually is time consuming especially if we have a large team working on many features and will be deployed on multiple servers and there is a high demand to deploy releases of the code more frequently.
CI/CD solves this problem by automating the process of building and delivering the applications. Instead of waiting to finish all the features at once, we can make small patches of features and allow automatic deployment to be occured immediately. This will minimize the complexity of building the applications and will reduce the effort to deploy them
In general, CI stands for **Continuous Integration** which is the process of automating the code merging and building. While, CD stands for **Continuous Delivery** , whcih means building and packaging the code to be deployed to the production server. This is followed by a manual approval to be released to the production. The manual step is just a click of buttons to approve and may be needed in situations when we need privileges or approval from other parties to access these servers. However, CD sometimes refers to **Continuous Deployment**, whcih refers to making the whole process automatically

CI/CD in Practice

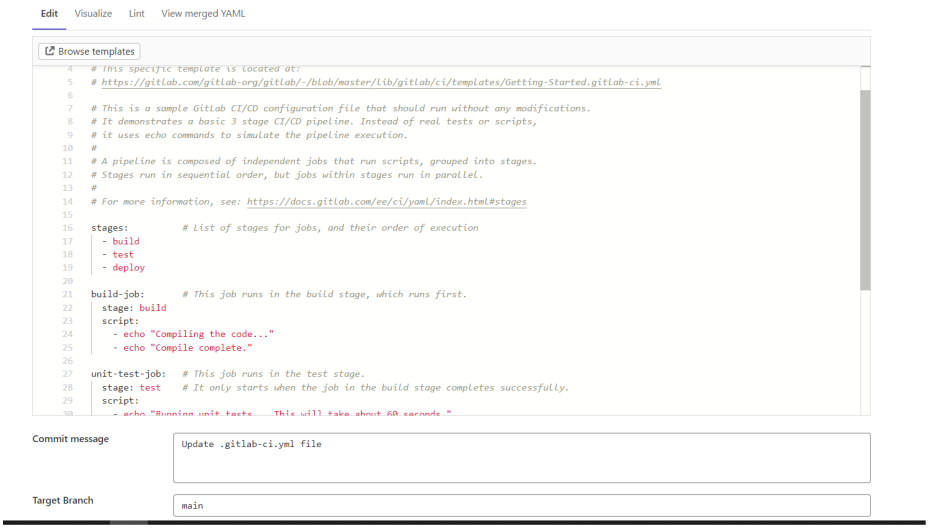
in this section we will take a quick view of how to make this operation using GitLab and Docker. First we push our code to our accounts on GitLab, then we open the project in GitLab, and from the left-hand side menu we click CI/CD.



This will open the pipline, if it exists. To create a new pipeline, we select editor option from the menue under CI/CD. Then, we select **Create New CI/CD Pipelines**



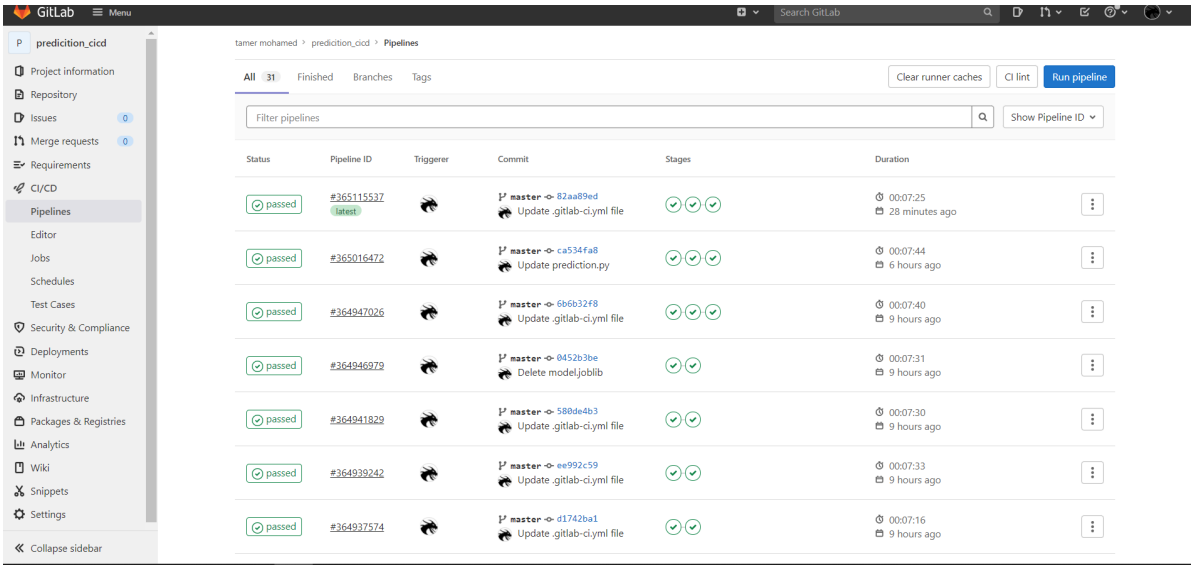
This will open a YAML file. YAML is a human readable data serialization language. It's similar to JSON format and it has a similar structure and specefication for writing the configurations. We use this type of files in many applications to write the configuration of the creating and deployment of the code in a standard way.



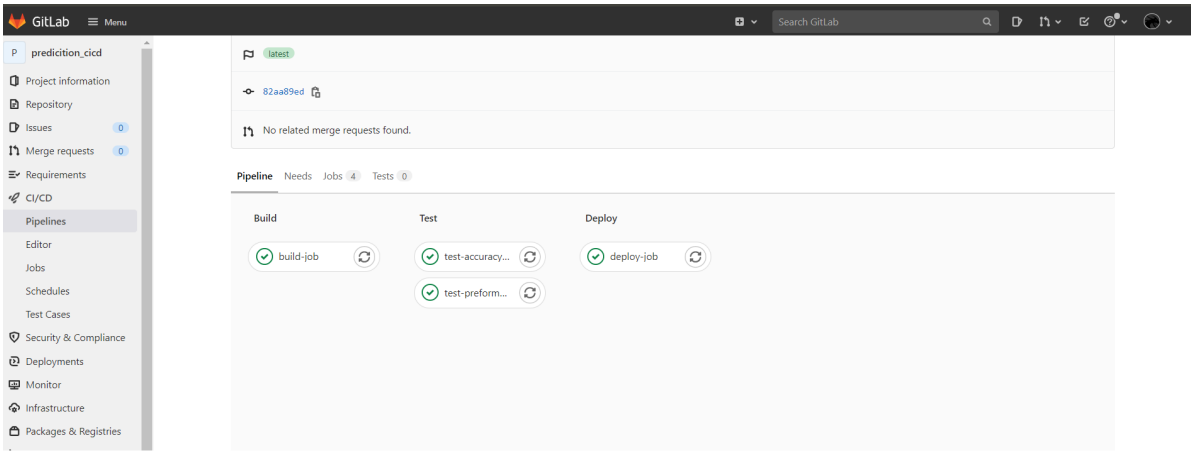
We will change the default YAML instructions in this file and add the following configurations:

```
stages:          # List of stages for jobs, and their order of execution
  - build
  - test
  - deploy
build-job:       # This job runs in the build stage, which runs first.
  stage: build
  image: python:3.8
  script:
    - python -m pip install -r requirements.txt
    - pylint --disable=R,C,W  *.py
test-preformance-job:
  stage: test
  script:
    - echo "Run Test Performance"
deploy-job:
  stage: deploy
  image: gitlab/dind
  before_script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
  script:
    - docker build -t
$CI_REGISTRY/$GITLAB_USER_LOGIN/$CI_PROJECT_NAME/prediction:phasethree .
    - docker push
$CI_REGISTRY/$GITLAB_USER_LOGIN/$CI_PROJECT_NAME/prediction:phasethree
  only:
    - master
test-accuracy-job:
  stage: test
  script:
    - echo "Run Test accuracy"
```

Then, we commit the changes, wait for the process to complete, then click Pipeline under CI/CD in the left menu, we will find the running pipelines and will notice the passed ones, which means the pipeline has run successfully for these instances.



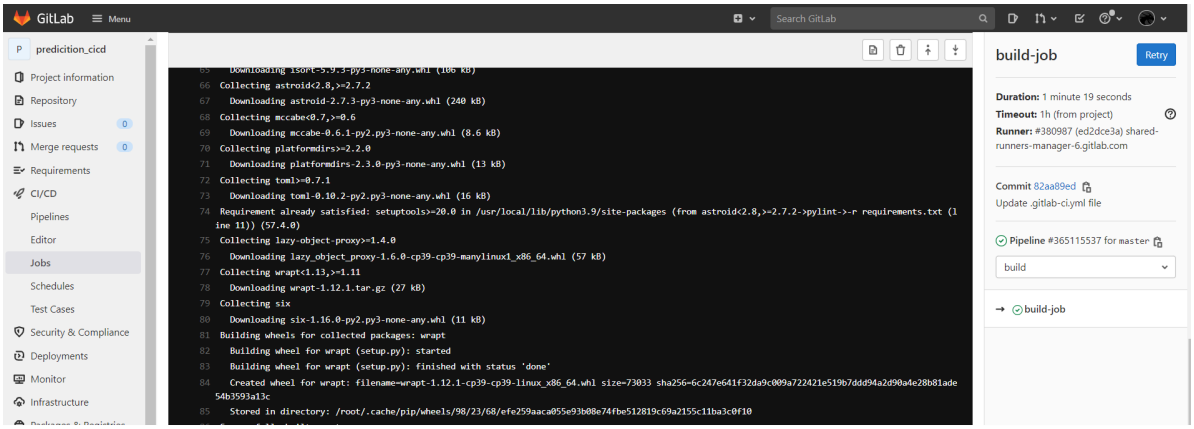
To ensure that the code has been created and can be deployed automatically, click the pipeline and you will get the following screen.



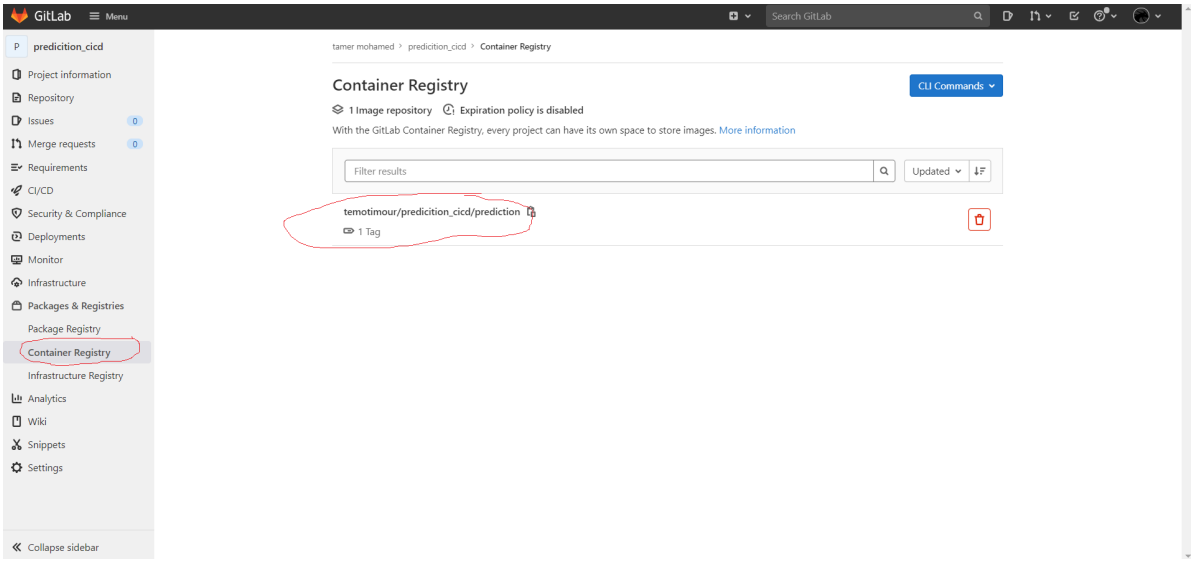
CI/CD uses pipelines which include stages and jobs, where stages run in the sequence that has been defined in the configuration file, and the jobs in the same stage run independently. In the above configurations, we created build, test and deploy stages, so they run in this sequence. Stages can be devided into jobs which can run in parallel so the instructions in these jobs should be completed independently from other jobs in the same satge. For instance, you can not execute a command that install python packages and in another job in the same stage execute a python command such as 'python app.py'.

Now, let's have a look at build-job that belongs to the build stage, you will notice an image keyword, which simply defines a docker image that is needed to run this job. In our case, we need a python image to execute the scripts in this job so GitLab will pull this python image from the docker hub first and then use it to run the python code. Also, in the same scripts' section, we lint the python files to make sure they build correctly without issues. In the deploy-job, we used another Docker image to enable the docker commands on the GitLab server. Another thing to look at is the `$CI_REGISTRY_USER` variable. This is an example of the predefined variables, which are predefined by default in the GitLab CI/CD pipeline. It contains some information you may need in your pipeline such as your username and password that we use to login to GitLab and to publish the docker image we created on GitLab registry (GitLab registry is a repository for the docker images similar to the Docker Hub). The last thing inside the deploy-job we used `only` and passed the branch name, this basically means that this deployment job will only run for the master branch and if you run this pipeline in any other branch, it will execute all jobs and except this one.

In cases where the pipeline failed to run, you can click on any job and you will get a full log, which shows what exactly occurs during this job. You can investigate to know the source of the issue and resolve it.



You can check the saved docker images in gitlab, from the left menue - Packages&Registries>Container Registry.



Then., you can try to pull this image into yor working space and run it using the normal docker commands as discusse in the previous lecture.

```
docker pull registry.gitlab.com/temotimour/predicition_cicd/prediction:v2
```

```
docker images
REPOSITORY                                                      TAG           IMAGE
ID        CREATED          SIZE
registry.gitlab.com/temotimour/predicition_cicd/prediction   v2
06c021a38d2c    8 minutes ago    1.46GB
```

```
```docker run image_id

      Date Location  MinTemp  ...  RainToday  RISK_MM  RainTomorrow
0  2008-12-01   Albury     13.4  ...         No      0.0            No
1  2008-12-02   Albury      7.4  ...         No      0.0            No
```

What is Next?

So far, we have not deployed our application to any production server. In order to do that, there is an open source system called **Kubernetes**. It can be used to handle containerized application where we can manage and automate the deployment operations on the production server, kubernetes can run across multiple clusters, and it can scale-up or scale-down the resources as needed such as memory and CPUs. Kubernetes can also auto scale if there is a load on your application, where it can run another container and route some of the traffic to the new container. This will reduce the loads on other containers and increase the performance. It also can be used to track the requests and keep logs of these requests. Kubernetes can be used on our machines as it is already installed by default when installing the docker desktop but not enabled. To enable it, you may open the Docker GUI and from Settings - Kubernetes - check the enable button. For more information, you may check this article Docker VS Kubernetes

There is a concept which is called `DevOps`. This means Development and Operations and the main purpose of it is to deliver the software applications quicker than the classical way by using some of the practices and tools such as version control (GitLab, GitHub, ... etc.), Container Management tools (docker,Kubernetes, ... etc), CI/CD tools (Gitlab CI/CD, CirecleCI, Jenkins,... ect),
```

and etc..

Machine learning also has its own DevOps which is called `MLOPs`, and in big data there is `DataOps`. if we look at any these concepts, we will find that it is all about automating the process of delivery and deployment by following some practices and using some tools. The main difference between any of these SomethingOps is neither the concept nor the main idea, but the life cycle. In MLOps, we have some differences such as ensuring data quality, continues training if the data your are using for training the model has been changed in structure, type or relation between features or between features and output overtime. In these cases, we need to retrain the model to keep the accuracy of the model. There is also situations in ML where you may need to automate the machine learning pipeline which may consist of of data extraction, data processing, feature engineering, model training, model registry and model deployment. In order to do that, there are many open source platforms that can be used in MLOps such as MLFlow and KubeFlow.
