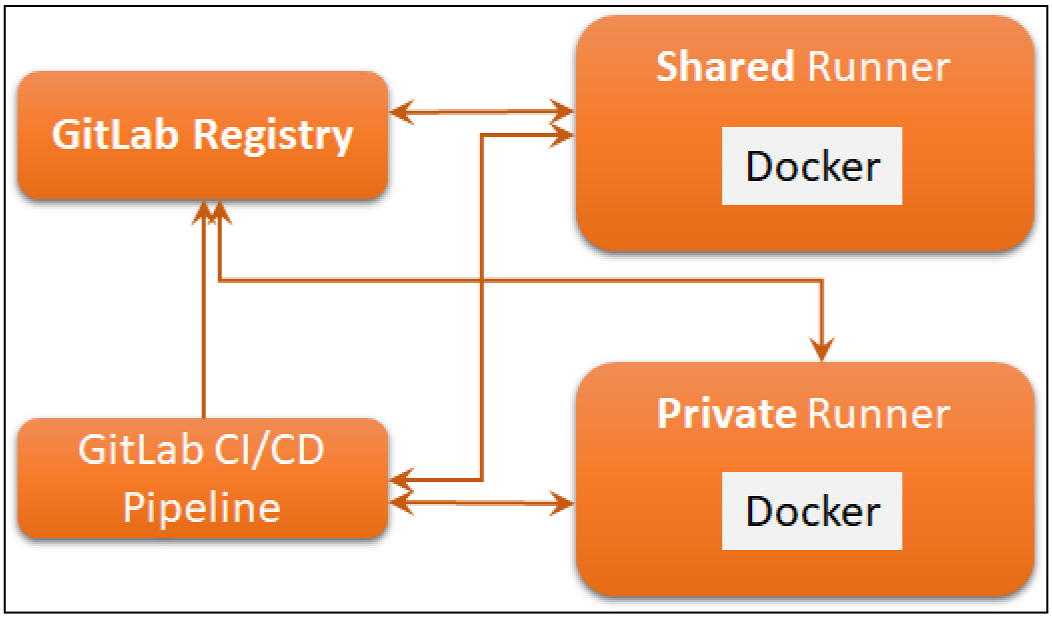
Practice Session-07:- CI/CD with GitLab

Make sure that you have already gone through Lab-06.

In this session you're going to learn the basic building blocks of the GitLab CI/CD pipeline and create the CI/CD pipelines to automatically deploy your custom code on the k8s.

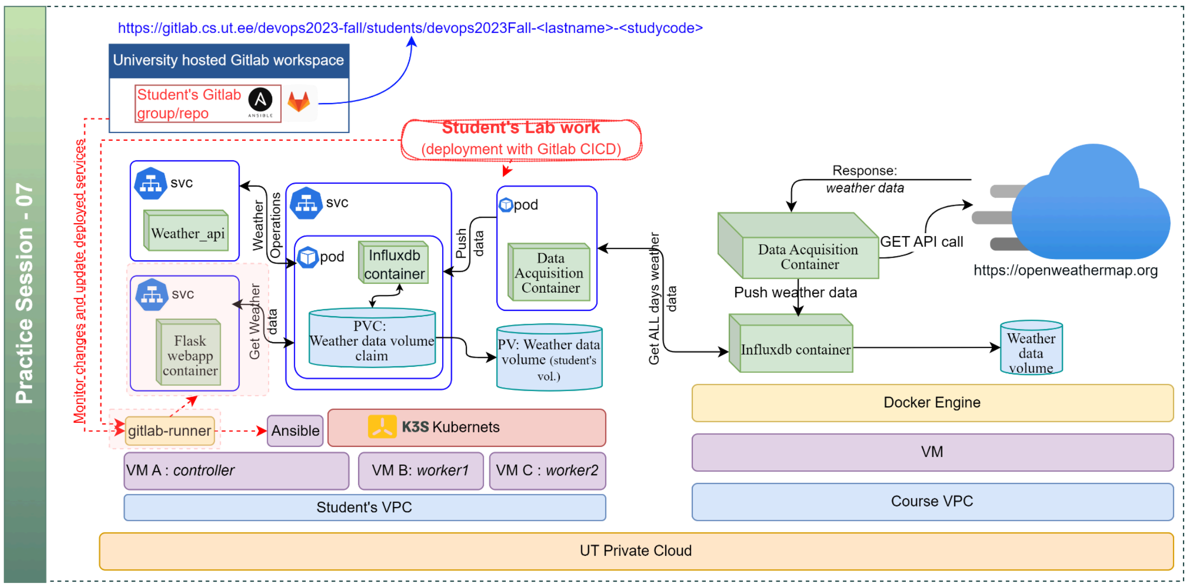


Understanding the communication among GitLab CICD, runneres and the registry.

Prerequisite

- Basic knowledge on Python, pandas library, csv file format
- Basic knowledge on YAML
- Familiar with Dockerfile
- Familiar with Git

What are you going to do?



Practice Session - 07
https://gitlab.cs.ut.ee/devops2023-fall/students/devops2023Fall-<lastname>-<studycode>
University hosted Gitlab workspace
Student's Gitlab group/repo
Student's Lab work
(deployment with Gitlab CICD)
Monitor changes and update deployed services
svc
Weather_api
Weather Operations
svc
pod
Influxdb container
PVC: Weather data volume claim
Push data
pod
Data Acquisition Container
PV: Weather data volume (student's vol.)
svc
Flask webapp container
Get Weather data
Get ALL days weather data
Response: weather data
GET API call
Data Acquisition Container
Push weather data
Influxdb container
Weather data volume
https://openweathermap.org
gitlab-runner
Ansible
K3S Kubernetes
VM A : controller
VM B: worker1
VM C : worker2
Student's VPC
Docker Engine
VM
Course VPC
UT Private Cloud

In the practice session, you will work with gitlab CI/CD. Using gitlab CI/CD, you're going to create pipelines to deploy the flask web application using ansible and gitlab runner, when code is modified. Further, you will work with image versioning and sync the images in the deployment.

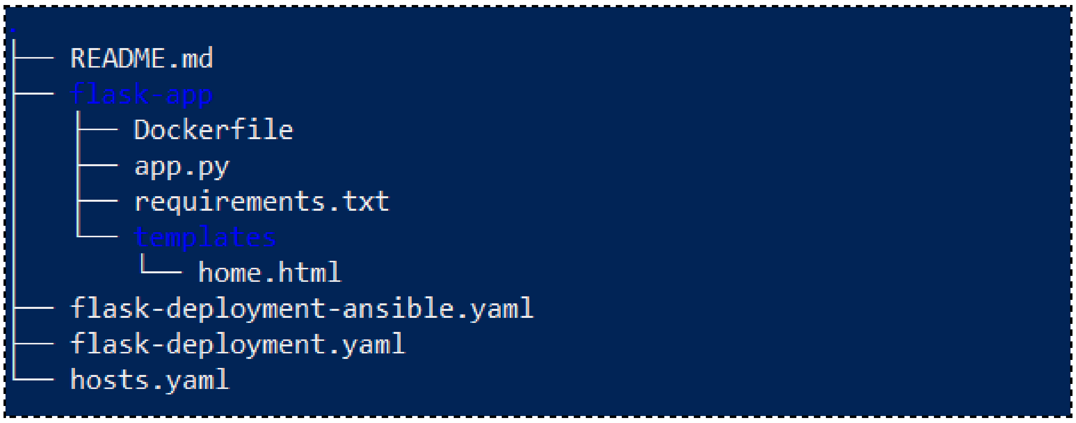
Exercise 1: Setting up of gitlab project and runners

In this exercise you will create a project and set up GitLab Runner and configure the runner, configure pipeline, etc. In this lab we are going to use the flask application code from Lab 02 and k8s deployment files and ansible playbooks from Lab 06.

1.1 Create a gitlab project and add necessary files

- Create a project in the gitlab with name `prac07-gitlab-cicd` under group `devops2023-fall/students/devops2023Fall-<lastname>-<studyCode>`
- Clone the project to `microk8scontroller` VM and go to the project `cd prac07-gitlab-cicd`
- Copy `flask-app` directory from `prac02-docker` (flask web application code used in Lab 02). It should include the following files (!! Please don't add **venv-** you can delete it if needed)
  - Dockerfile
  - templates/home.html
  - app.py
  - requirements.txt

- Copy the `flask_deployment.yaml, hosts.yaml` and `flask-deployment-ansible.yaml` from `prac06-ansible` (Lab 06 files) `and` files should be copied outside the `flask-app` directory.
- The final directory of `prac07-gitlab-cicd` should look like



```
├── README.md
├── flask-app
│   ├── Dockerfile
│   ├── app.py
│   ├── requirements.txt
│   └── templates
│       └── home.html
├── flask-deployment-ansible.yaml
├── flask-deployment.yaml
└── hosts.yaml
```

- Commit with the message "`Added the required code and deployment files`" and push the code.

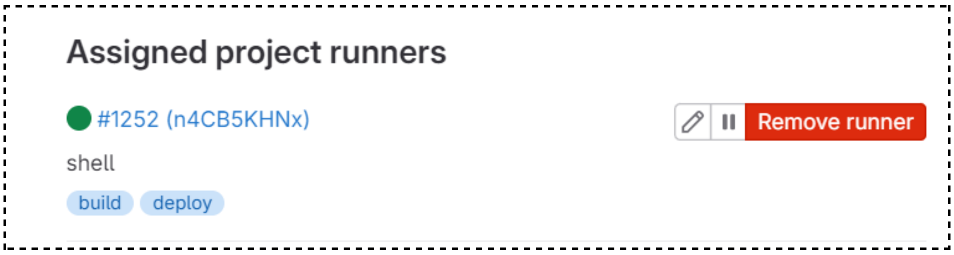
1.2 Installation and registration of Gitlab runners

In this task, we are going to set up a runner with `shell` as executor. The `shell` executor based runner used for building the docker images and to deploy the application on the kubernetes cluster using ansible.

- Make sure that you are logged in to your `microk8scontroller` VM.
- Install the gitlab-runner on k8s-controller virtual machine
  - Add the gitlab runner package - `curl -L "https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh" | sudo bash`
  - Install the runner `sudo apt-get install gitlab-runner`
  - Add the gitlab-runner to docker, microk8s group
    - `sudo usermod -aG docker gitlab-runner`
    - `sudo usermod -aG microk8s gitlab-runner`
- Get the token for Gitlab runner registration for `shell` runner
    - Go to your `prac07-gitlab-cicd` Gitlab project
    - Go to `Setting --> CI/CD` → Expand `Runner`
    - Click on `New Project Runner`
      - Platform(Operating systems)--> `Linux`
      - Tags--> `build,deploy`
      - Details--> `shell`
      - Click on `Create runner`
      - Now copy the command mentioned in Step 1

For example: `sudo gitlab-runner register  --url https://gitlab.cs.ut.ee  --token glrt-...sp`

- Paste the command (make sure to use `sudo`) in `microk8scontroller` VM and it asks for input as described below :
  - Enter the GitLab instance URL: `https://gitlab.cs.ut.ee/`
  - Enter a name for the runner: `shell`
  - Enter an executor: `shell`
  - Press enter and gitlab runner is registered
- Now you can see the registered runner in your gitlab account in `Settings → CI/CD → Runners`
- You should see your registered runner: (If you see your runner without green symbol, then run the command `sudo gitlab-runner run shell` in the *microk8scontroller* VM)



Assigned project runners

● #1252 (n4CB5KHNx)

shell

build deploy

1.3 Creating an access token and creating k8s secret for gitlab registry

- Next, we need to create an access token for authentication when reading and writing to the GitLab container registry. This access token is required in later exercise, and please note it down carefully (Note!! Please copy the access token and save it in a text file at some place)
  - Go to Settings→Repository→Depoy tokens and Expand it.
    - Name: k8s
    - Expiration date: Choose some date
    - Username (optional): Your Gitlab Username
    - Scopes (select at least one): read_registry,write_registry
    - Click on Create deploy token
    - Copy the access token and save it in a text file someplace.

- Create k8s secret for docker registry to access the gitlab registry in the `microk8scontroller` VM using the command (Change the values of red coloured marked text, `YOUR_ACCESS_TOKEN` is noted in the previous step). It would be nice to copy the text file and modify it, before running the command.

```
microk8s kubectl create secret docker-registry
registry-credentials
```

```
--docker-server=https://gitlab.cs.ut.ee:5050
--docker-username=YOUR_GITLAB_USERNAME
--docker-password=YOUR_ACCESS_TOKEN
--docker-email=YOUR_GITLAB_EMAIL_ID -n ex3
```

1.4 Check for the prerequisites in the `microk8scontroller` VM

- Make sure ansible is running `ansible --version`
- Make sure the **influxdb** database is running and have the data in weather_data bucket
- Make sure **influxdbdata** pod is running
- Check your configmap (**studhost**) and secret (**influx**) are present and correct.

Exercise 2: Building your first pipeline

In this exercise, you're going to create the `.gitlab-ci.yml` CI file, a YAML file containing a specific set of jobs, stages, tags, etc., for the GitLab CI/CD pipeline. For information on the keywords used in this CI file can be found at https://docs.gitlab.com/ee/ci/yaml/
In this file, you define

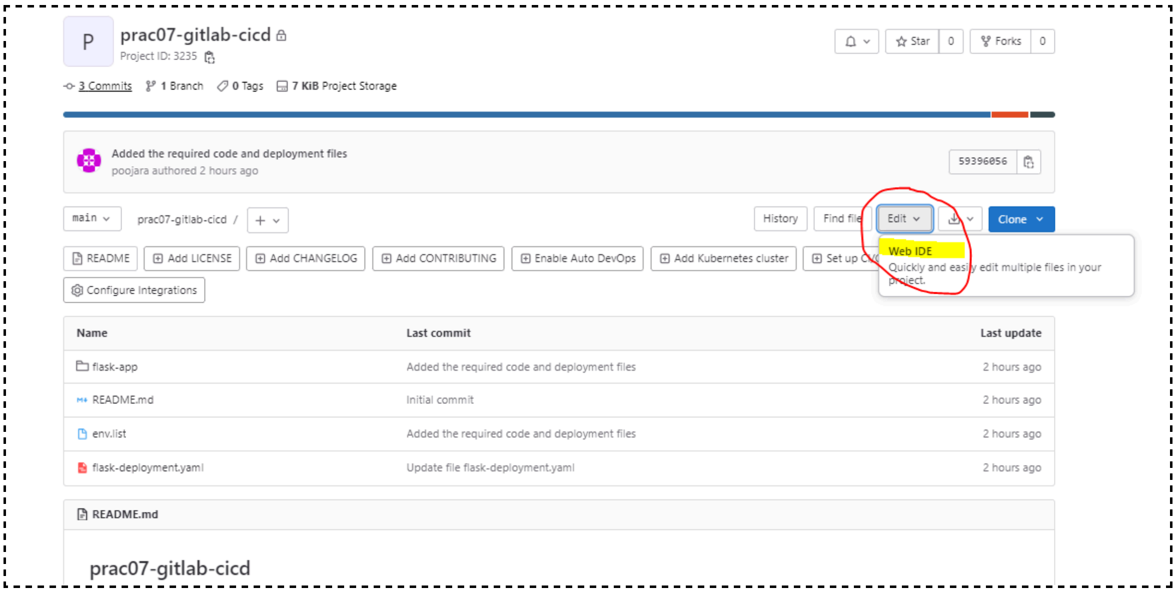
- The structure and order of jobs that the runner should execute.
- The decisions the runner should make when specific conditions are encountered.

From now onwards, you will work most of the time in gitlab UI under the project `prac07-gitlab-cicd`

At this point, you already have registered one GitLab runner.

Points to Remember,

1. In this exercise, you will very frequently commit the changes and push the code to your repo (you may use Gitlab Web IDE to modify the files). How to find Gitlab Web IDE is here:


P prac07-gitlab-cicd
Project ID: 3235

3 Commits | 1 Branch | 0 Tags | 7 KiB Project Storage

Added the required code and deployment files
poojara authored 2 hours ago
59396056

main / prac07-gitlab-cicd / + | History | Find file | Edit | Clone

Web IDE
Quickly and easily edit multiple files in your project.

README | Add LICENSE | Add CHANGELOG | Add CONTRIBUTING | Enable Auto DevOps | Add Kubernetes cluster | Set up CI/CD
Configure Integrations

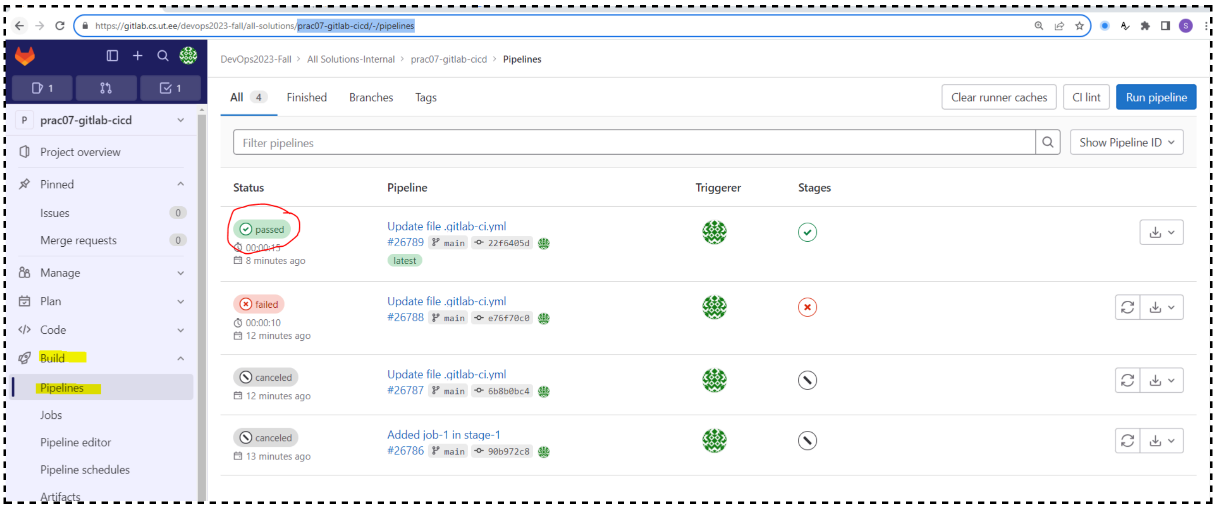
Name	Last commit	Last update
flask-app	Added the required code and deployment files	2 hours ago
README.md	Initial commit	2 hours ago
env.list	Added the required code and deployment files	2 hours ago
flask-deployment.yaml	Update file flask-deployment.yaml	2 hours ago

README.md

prac07-gitlab-cicd

2. At the end of this exercise, we may go through the commits history. So make sure that you have followed and implemented each step one after another.

- After each commit, you can see pipeline status in `CI/CD` → `Pipelines`, as below


https://gitlab.cs.ut.ee/devops2023-fall/all-solutions/prac07-gitlab-cicd/-/pipelines

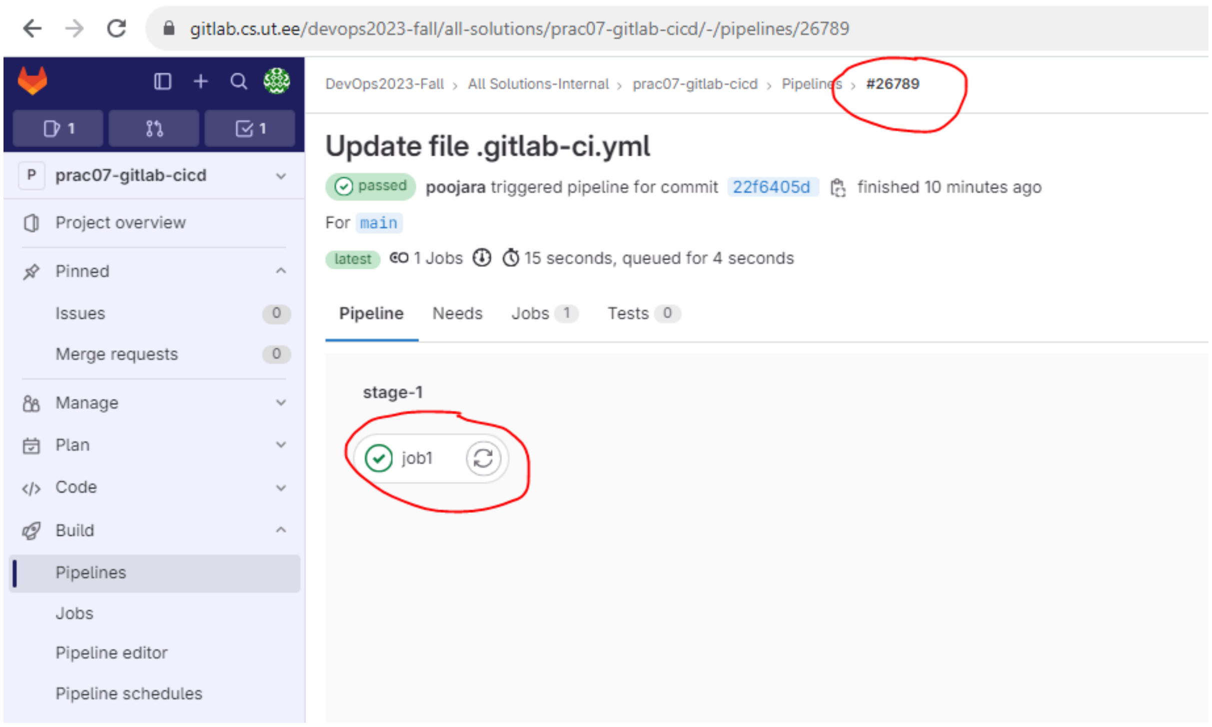
DevOps2023-Fall > All Solutions-Internal > prac07-gitlab-cicd > Pipelines

All 4 | Finished | Branches | Tags | Clear runner caches | CI lint | Run pipeline

Filter pipelines | Show Pipeline ID

Status	Pipeline	Triggerer	Stages
passed 00:00:15 8 minutes ago	Update file .gitlab-ci.yml #26789 main 22f6405d latest		✓
failed 00:00:10 12 minutes ago	Update file .gitlab-ci.yml #26788 main e76f70c0		✗
canceled 12 minutes ago	Update file .gitlab-ci.yml #26787 main 6b8b0bc4		⊘
canceled 13 minutes ago	Added job-1 in stage-1 #26786 main 90b972c8		⊘

- Click on the Pipeline ID to see the list of Jobs. for example, when I click on Pipeline ID `#26789`, I see the following list and status of jobs:



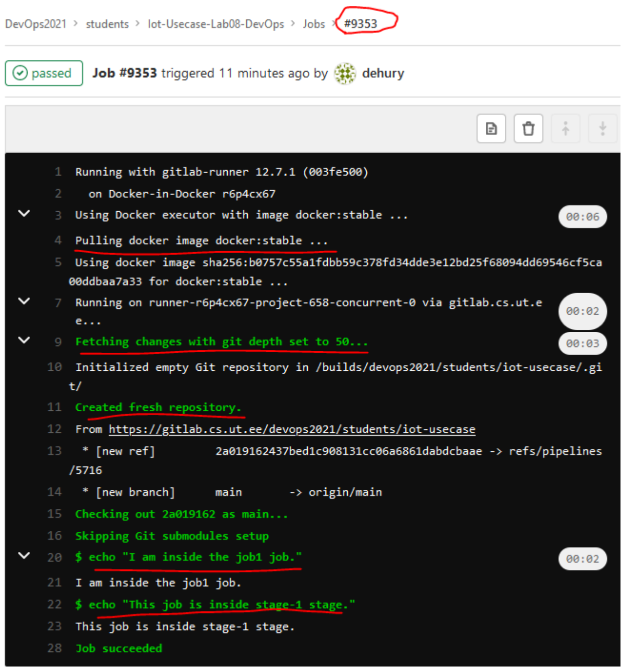
Update file .gitlab-ci.yml

passed poojara triggered pipeline for commit 22f6405d finished 10 minutes ago

For main

latest 1 Jobs 15 seconds, queued for 4 seconds

- Now, if you click on the job (e.g. `job1` in the above figure), you can see the logs of gitlab runner. For example, when I click on the above job1, I see the following output:


DevOps2021 › students › Iot-Usecase-Lab08-DevOps › Jobs › #9353

passed Job #9353 triggered 11 minutes ago by dehury

```
Running with gitlab-runner 12.7.1 (003fe500)
  on Docker-in-Docker r6p4cx67
Using Docker executor with image docker:stable ...                    00:06
Pulling docker image docker:stable ...
Using docker image sha256:b0757c55a1fdbb59c378fd34dde3e12bd25f68094dd69546cf5ca
00ddbaa7a33 for docker:stable ...
Running on runner-r6p4cx67-project-658-concurrent-0 via gitlab.cs.ut.e  00:02
e...
Fetching changes with git depth set to 50...                          00:03
Initialized empty Git repository in /builds/devops2021/students/iot-usecase/.gi
t/
Created fresh repository.
From https://gitlab.cs.ut.ee/devops2021/students/iot-usecase
 * [new ref]         2a019162437bed1c908131cc06a6861dabdcbaae -> refs/pipelines
/5716
 * [new branch]      main       -> origin/main
Checking out 2a019162 as main...
Skipping Git submodules setup
$ echo "I am inside the job1 job."                                    00:02
I am inside the job1 job.
$ echo "This job is inside stage-1 stage."
This job is inside stage-1 stage.
Job succeeded
```

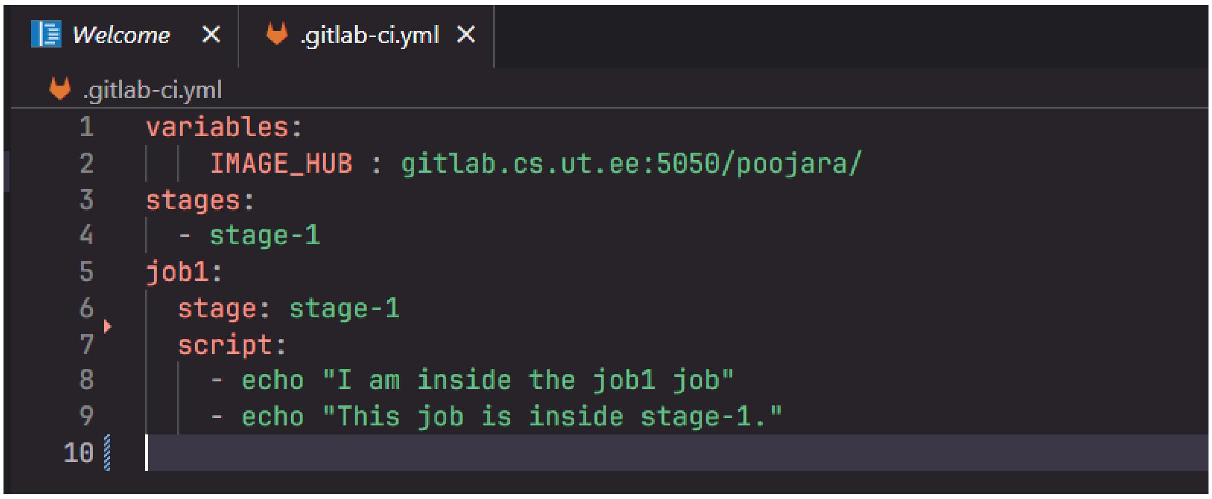
Note!! You will commit and push the code several times in further exercises.

Remember that, we may see the commit history while grading your submission. So, please specify the commit messages as prescribed.

Let's move ahead and prepare our first pipeline.

2.1: Single stage pipeline

- Create `.gitlab-ci.yml` CI file (You may use GitLab UI/Web IDE to create this file) and add only one stage `stage-1` as below:

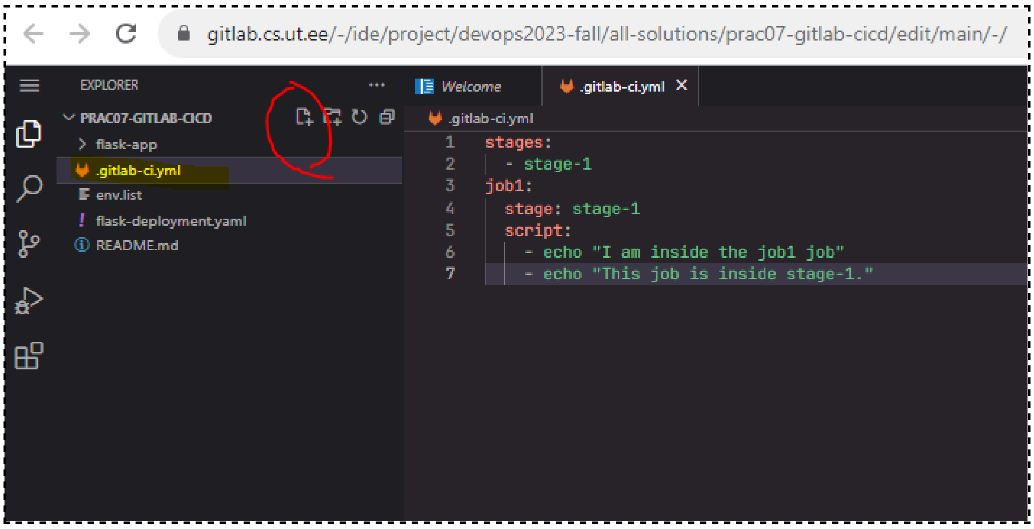


Welcome ✕ | .gitlab-ci.yml ✕

.gitlab-ci.yml

```
variables:
    IMAGE_HUB : gitlab.cs.ut.ee:5050/poojara/
stages:
  - stage-1
job1:
  stage: stage-1
  script:
    - echo "I am inside the job1 job"
    - echo "This job is inside stage-1."
```

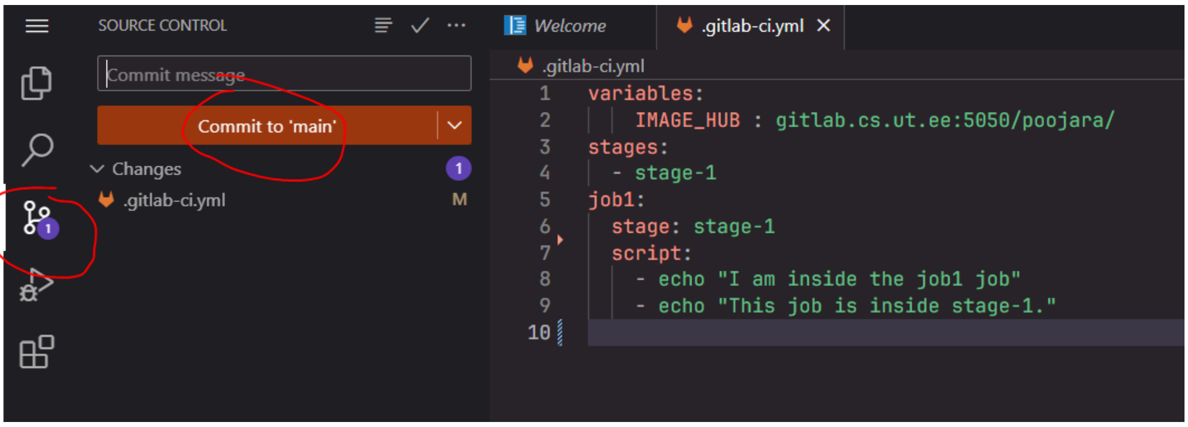
  - Creating a file using Web IDE



gitlab.cs.ut.ee/-/ide/project/devops2023-fall/all-solutions/prac07-gitlab-cicd/edit/main/-/

```
stages:
  - stage-1
job1:
  stage: stage-1
  script:
    - echo "I am inside the job1 job"
    - echo "This job is inside stage-1."
```

- Here the first and only job is `job1`, which will run in `stage-1` stage and tag is `shell`.
- Commit with the message “Added job-1 in stage-1” and push the above changes. Commit using Web IDE

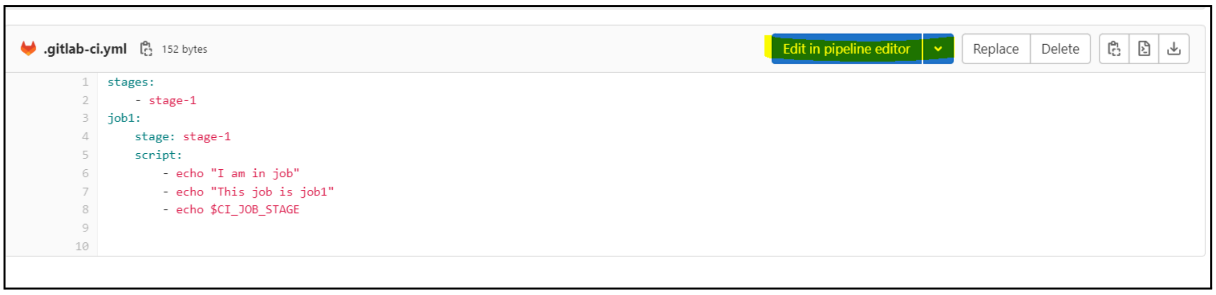


```
variables:
    IMAGE_HUB : gitlab.cs.ut.ee:5050/poojara/
stages:
  - stage-1
job1:
  stage: stage-1
  script:
    - echo "I am inside the job1 job"
    - echo "This job is inside stage-1."
```

- See the newly created pipeline and job.(You can check how to see the pipelines here)

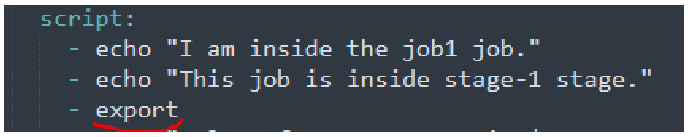
2.2 : Lets print some predefined and custom variables

- To modify the `.gitlab-ci.yml` in further steps, you may use Pipeline Editor in GitLab UI.



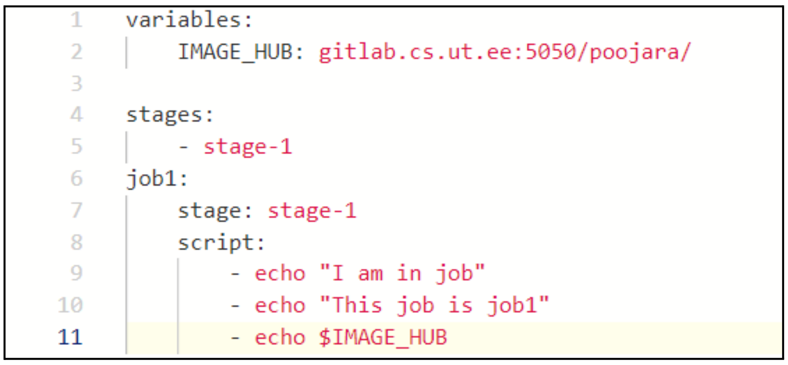
```
stages:
    - stage-1
job1:
    stage: stage-1
    script:
        - echo "I am in job"
        - echo "This job is job1"
        - echo $CI_JOB_STAGE
```

- Modify the `script` section and `echo` the following predefined variables:
  - CI_JOB_STAGE
  - CI_COMMIT_BRANCH
  - CI_COMMIT_AUTHOR
  - CI_COMMIT_DESCRIPTION
  - CI_COMMIT_MESSAGE
  - CI_CONFIG_PATH
  - CI_JOB_NAME
  - CI_JOB_ID
  - CI_JOB_STATUS
  - CI_PIPELINE_ID
  - CI_RUNNER_ID
- To print a predefined variable, you can use the following command in `script` section:
  - `echo $<variable_name>`
  - E.g. `echo $CI_JOB_STAGE`
  - Commit with the message “Printing predefined variables” and push the above changes.
- To see all the available variables, you can use `- export` OR `- env` in the script section.



```
script:
  - echo "I am inside the job1 job."
  - echo "This job is inside stage-1 stage."
  - export
```

- Define your own variables in the configuration file as below. This variable is accessible to all jobs. Create the `IMAGE_HUB` as your custom variable (sample given in the below figure)

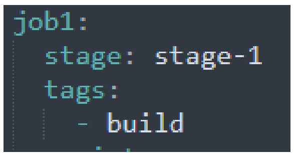


```
variables:
    IMAGE_HUB: gitlab.cs.ut.ee:5050/poojara/

stages:
    - stage-1
job1:
    stage: stage-1
    script:
        - echo "I am in job"
        - echo "This job is job1"
        - echo $IMAGE_HUB
```

- Add an `echo` statement to print the above `IMAGE_HUB` custom variable.
- Commit with the message "Printing the user defined variables inside .gitlab-ci.yml" and push the above changes.
- The other way to define variables is by the use of the `Variables` feature in GitLab. The `Variables` feature can be found in `settings` → `CI/CD` → Expand `Variables`. These variables can be used in other pipelines as well.
  - Add a variable '`my_project_wide_variable`' with value `<your_name>` in `settings` → `CI/CD` → Expand `Variables`.
  - `echo` the variable in the pipeline under the `script` section.
- Create a variable to store the GitLab access token using Variable feature
  - Add a variable `gitlabpassword` with value of access token created in `Exercise 1`
- Commit with the message "Printing the user variables" and push the above changes.

2.3 : Let the pipeline run on a specific gitlab-runner.

- At this moment, you have already registered your specific gitlab-runner with the tags `build` and `deploy`.
- You will use the above tags to run your jobs in your runner in the **k8s-controller** VM.
- Modify the configuration file again, so that all the jobs will run in your specific runner.

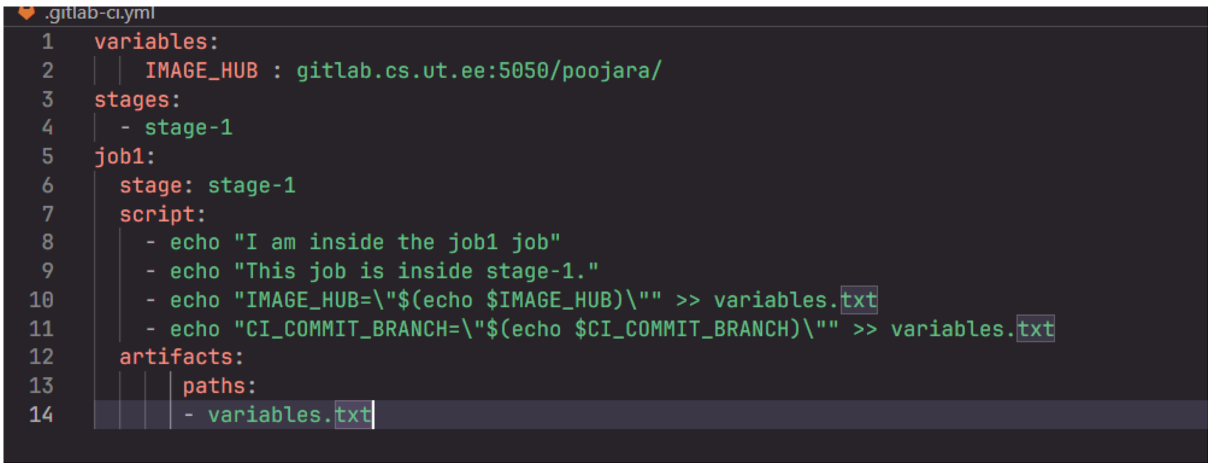


```
job1:
  stage: stage-1
  tags:
    - build
```

- Commit with the message "Added the tags" and push the above changes.
- See the newly created pipeline and job.

2.4: Working with GitLab job artifacts, “before_script”, “script”, and “after_script”

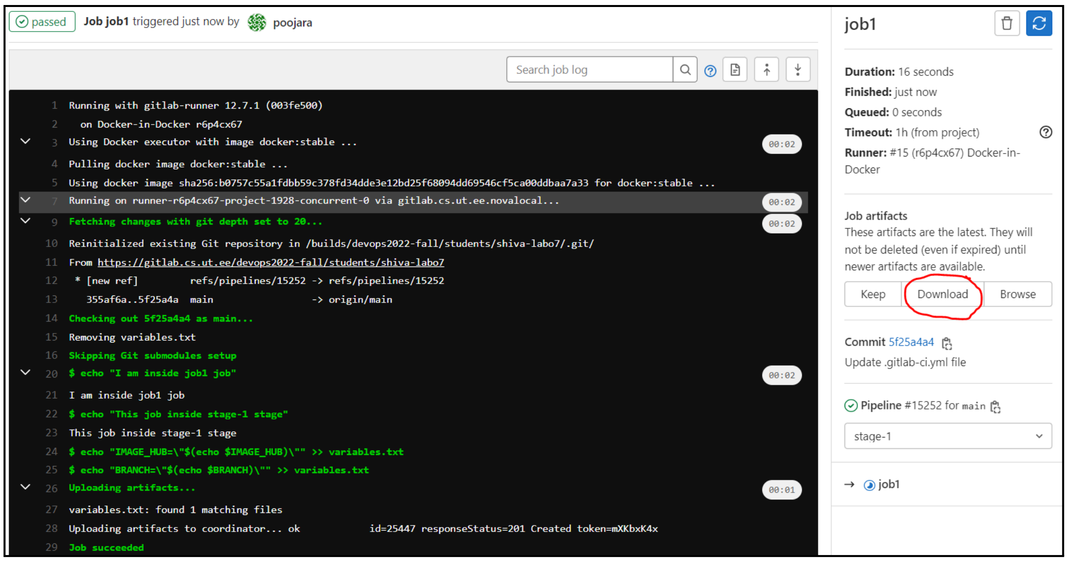
- Artifacts are used to specify which files to save as job artifacts. Jobs can output an archive of files and directories. This output is known as a job artifact.
  - Artifacts can be mentioned as shown below. Update your `.gitlab-ci.yml` file as shown below



.gitlab-ci.yml

```yaml
variables:
    IMAGE_HUB : gitlab.cs.ut.ee:5050/poojara/
stages:
  - stage-1
job1:
  stage: stage-1
  script:
    - echo "I am inside the job1 job"
    - echo "This job is inside stage-1."
    - echo "IMAGE_HUB=\"$(echo $IMAGE_HUB)\"" >> variables.txt
    - echo "CI_COMMIT_BRANCH=\"$(echo $CI_COMMIT_BRANCH)\"" >> variables.txt
  artifacts:
      paths:
       - variables.txt
```

  - Commit with the message “Added artifacts for job1” and push the above changes.
  - You can download your job artifacts, after the pipeline is executed



passed **Job job1** triggered just now by poojara

```
Running with gitlab-runner 12.7.1 (003fe500)
  on Docker-in-Docker r6p4cx67
Using Docker executor with image docker:stable ...                                        00:02
Pulling docker image docker:stable ...
Using docker image sha256:b0757c55a1fdbb59c378fd34dde3e12bd25f68094dd69546cf5ca00ddbaa7a33 for docker:stable ...
Running on runner-r6p4cx67-project-1928-concurrent-0 via gitlab.cs.ut.ee.novalocal...      00:02
Fetching changes with git depth set to 20...                                               00:02
Reinitialized existing Git repository in /builds/devops2022-fall/students/shiva-labo7/.git/
From https://gitlab.cs.ut.ee/devops2022-fall/students/shiva-labo7
 * [new ref]         refs/pipelines/15252 -> refs/pipelines/15252
   355af6a..5f25a4a  main                 -> origin/main
Checking out 5f25a4a4 as main...
Removing variables.txt
Skipping Git submodules setup
$ echo "I am inside job1 job"                                                              00:02
I am inside job1 job
$ echo "This job inside stage-1 stage"
This job inside stage-1 stage
$ echo "IMAGE_HUB=\"$(echo $IMAGE_HUB)\"" >> variables.txt
$ echo "BRANCH=\"$(echo $BRANCH)\"" >> variables.txt
Uploading artifacts...                                                                     00:01
variables.txt: found 1 matching files
Uploading artifacts to coordinator... ok           id=25447 responseStatus=201 Created token=mXKbxK4x
Job succeeded
```

job1

Duration: 16 seconds
Finished: just now
Queued: 0 seconds
Timeout: 1h (from project)
Runner: #15 (r6p4cx67) Docker-in-Docker

Job artifacts
These artifacts are the latest. They will not be deleted (even if expired) until newer artifacts are available.

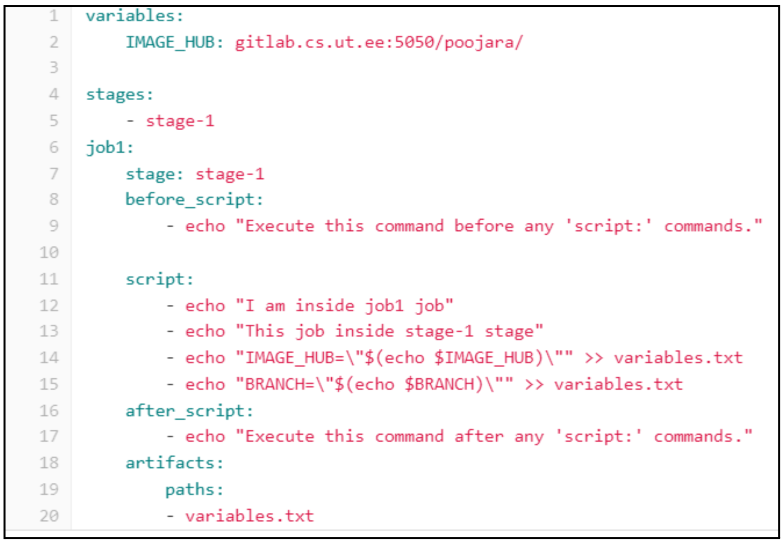
Commit 5f25a4a4
Update .gitlab-ci.yml file

Pipeline #15252 for `main`

stage-1

→ job1

- The `before_script`, and `after_script` are used to define an array of commands that should run before and after all the jobs under `script` tag,
  - Add the code as shown below



```
variables:
    IMAGE_HUB: gitlab.cs.ut.ee:5050/poojara/

stages:
    - stage-1
job1:
    stage: stage-1
    before_script:
        - echo "Execute this command before any 'script:' commands."

    script:
        - echo "I am inside job1 job"
        - echo "This job inside stage-1 stage"
        - echo "IMAGE_HUB=\"$(echo $IMAGE_HUB)\"" >> variables.txt
        - echo "BRANCH=\"$(echo $BRANCH)\"" >> variables.txt
    after_script:
        - echo "Execute this command after any 'script:' commands."
    artifacts:
        paths:
        - variables.txt
```

  - Commit with the message “Added before and after script references” and push the above changes.

Exercise 3: Working with gitlab container registry

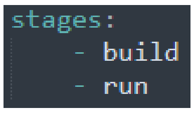
In this task, you modify the flask web application, create its docker image, and push it to the gitlab container registry. Further, you are going to deploy it on k8s cluster.

In this step, we will prepare a flask web application. So at this point, you have a `flask-app/Dockerfile` and `flask-app/templates/home.html` files in the gitlab project.

- Modify an `home.html` file inside `/templates` directory.
  - Add (inside <body> tag) the following content to your html file to display the message

    “`Hi I am <your_name> !! This is to demonstrate the gitlab image build`”

- Now, you need to update the `.gitlab-ci.yml` file with two stages: `build` and `run`. At this point, we can remove the previous stage `stage-1`



```
stages:
    - build
    - run
```

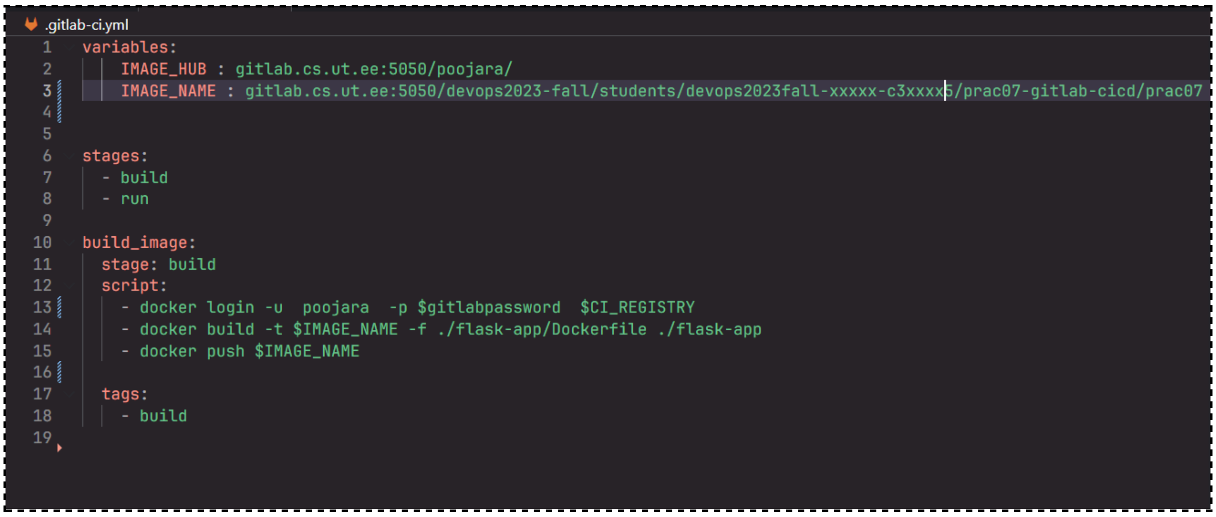
- Create an `IMAGE_NAME` variable with the value under variables.
  - Here, `name_space` could be your gitlab project path
    - Example, `devops2023-fall/students/devops2023fall-<lastname>-<studyCode>`
  - `image_name` could be prac07
  - `project_name` could be `prac07-gitlab-cicd`

`IMAGE_NAME: gitlab.cs.ut.ee:5050/<name_space>/<project_name>/<image_name>`

Ex: `IMAGE_NAME: gitlab.cs.ut.ee:5050/devops2023-fall/students/devops2023fall-oxxn-c18xx39/prac07-gitlab-cicd/prac7`

- Now let's create two jobs:
  - `build_image` to build the image
  - `run_image` to run the image
- Define `build_image` job in `.gitlab-ci.yml` file
  This job should run in the `build` stage.
  Under `script`:
  - Login to gitlab. Make sure that the `gitlabpassword` variable is defined in your Gitlab project. (Change username poojara to your github username)
    `docker login -u  poojara -p $gitlabpassword  $CI_REGISTRY`
  - Build the docker image using `Dockerfile` present in the project directory.
    `docker build -t $IMAGE_NAME -f ./flask-app/Dockerfile ./flask-app`
  - Push the docker image to the container registry.
    `docker push $IMAGE_NAME`
  - Add the tags

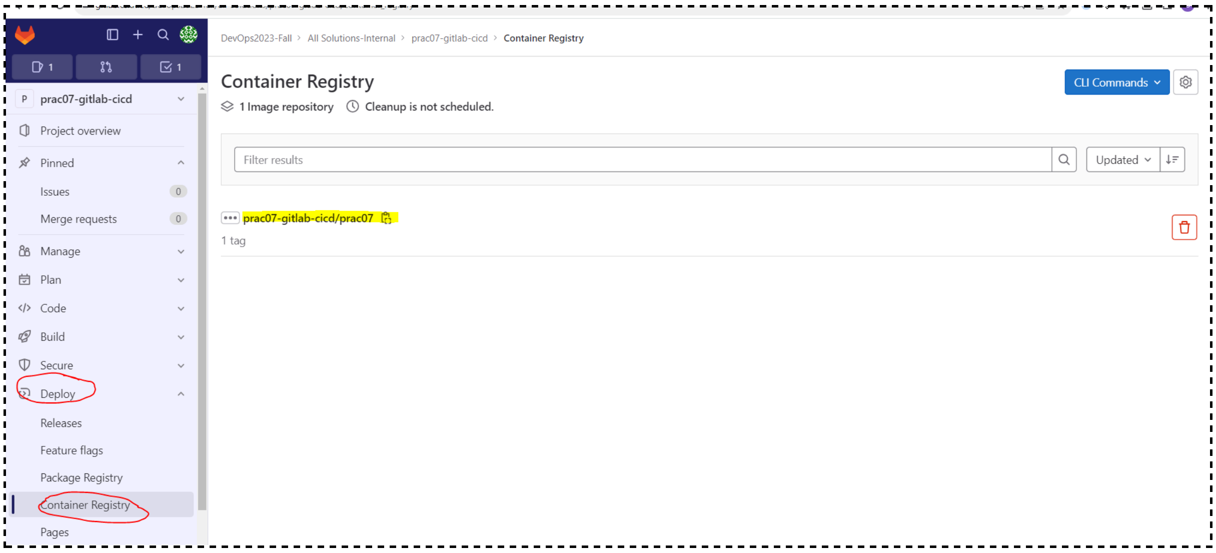
The update `.gitlab-ci.yml` file should look similar to below. Below image is just for reference purpose.



.gitlab-ci.yml

```
variables:
  IMAGE_HUB : gitlab.cs.ut.ee:5050/poojara/
  IMAGE_NAME : gitlab.cs.ut.ee:5050/devops2023-fall/students/devops2023fall-xxxxx-c3xxxx5/prac07-gitlab-cicd/prac07


stages:
  - build
  - run

build_image:
  stage: build
  script:
    - docker login -u  poojara  -p $gitlabpassword  $CI_REGISTRY
    - docker build -t $IMAGE_NAME -f ./flask-app/Dockerfile ./flask-app
    - docker push $IMAGE_NAME

  tags:
    - build

```

- Commit with the message “Added build_image job” and push the above changes.
- Once the pipeline is executed, you should see the container image in the gitlab registry.



Container Registry

1 Image repository   Cleanup is not scheduled.

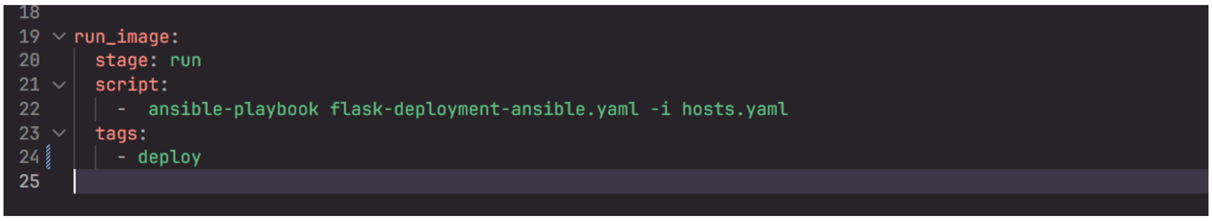
prac07-gitlab-cicd/prac07

1 tag

  - If you have errors in the job execution related to docker and gitlab runner, than please make sure you executed the following commands in the k8s-controller node
    - `sudo usermod -aG docker gitlab-runner`
    - `sudo service docker restart`
  - If your pipeline failing due to error like this "dial tcp: lookup docker on 193.40.5.39:53: no such host", than please disable the shared runner in Settings-->CICD-->Runners

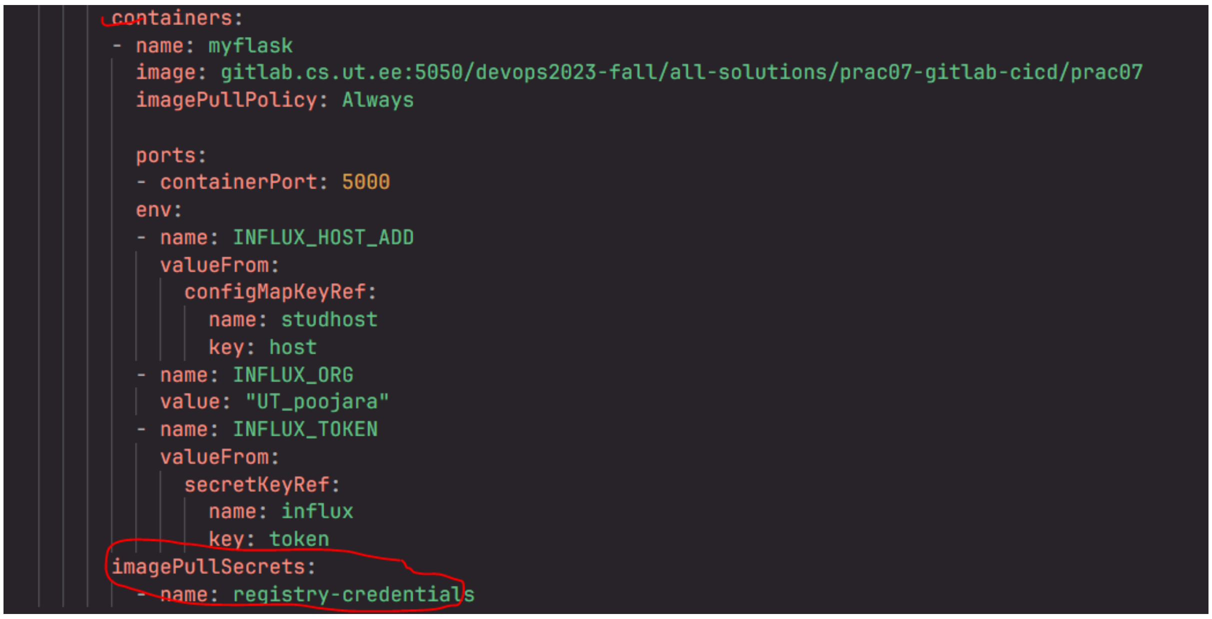
- Defining `run_image` job in `.gitlab-ci.yml` file .
  - This job should run in the `run` stage.
    Under `script`:
    - Add a command to run the ansible playbook with host.yaml file
    `ansible-playbook flask-deployment-ansible.yaml -i hosts.yaml`

The updated `.gitlab-ci.yml` file after defining `run_image` job should look like below:



```
run_image:
  stage: run
  script:
    - ansible-playbook flask-deployment-ansible.yaml -i hosts.yaml
  tags:
    - deploy
```

- Modify the `flask_deployment.yml` with two things mentioned as below
  - Update `image` with value of `$IMAGE_NAME` Ex:`image: gitlab.cs.ut.ee:5050/devops2022-fall/students/shiva-labo7/prac07`
  - Add property [`imagePullSecrets`]() under container `spec` which is required to pull the image from gitlab repo and it should look like



```
containers:
- name: myflask
  image: gitlab.cs.ut.ee:5050/devops2023-fall/all-solutions/prac07-gitlab-cicd/prac07
  imagePullPolicy: Always

  ports:
  - containerPort: 5000
  env:
  - name: INFLUX_HOST_ADD
    valueFrom:
      configMapKeyRef:
        name: studhost
        key: host
  - name: INFLUX_ORG
    value: "UT_poojara"
  - name: INFLUX_TOKEN
    valueFrom:
      secretKeyRef:
        name: influx
        key: token
imagePullSecrets:
  - name: registry-credentials
```

- Commit with message “added the run_image stage”and **push** above changes.
- See the newly created pipeline and jobs.
- Login and check the deployment running in **microk8scontroller** node using command `microk8s kubectl get po,svc -o wide -n ex3`
- Get the nodePort address under service
- At the end, you should be able to see the web page at http://microk8scontroller _EXT_IP:NODEPORT_ADDRESS
- If you don't see the change in the flask deployment with new pods and service, then you can add command in ansible script to delete the existing deployment and re-run the pipeline.(**flask-deployment-ansible.yaml)**

```
tasks:
  - name: Delete the deployment by running the kubectl command if it exists
    command: "microk8s kubectl delete -f flask-deployment.yaml"

  - name: create the deployment by running the kubectl command
    command: "microk8s kubectl create -f flask-deployment.yaml"
```

Screenshot - 1

Take a screenshot of a webpage and IP address are clearly seen.

Exercise 4: Working with image build versioning and updating the application

In this task, you're going to update the flask web application to display the minimum and maximum temperature values on the web page. Further, you learn about container image versioning.Its not good practice to tag the images always with tag “latest” for every pipeline execution. This is because, you may run in to the following problems

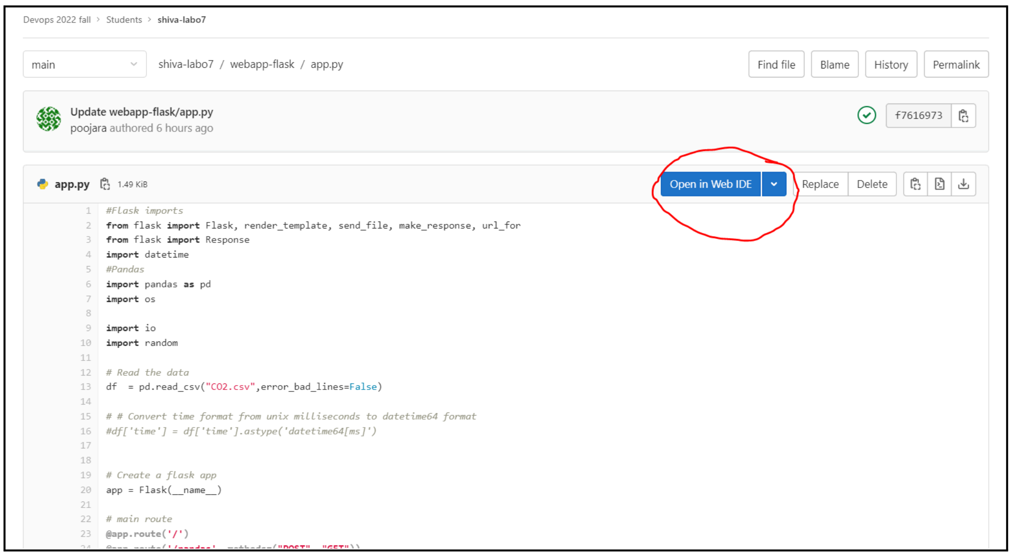
- If you re-execute an older CI job (or if you run the same CI job in multiple testing / feature Git branches), the CI jobs will keep overwriting the latest tag. “latest” loses its meaning. Your production environment will most likely become unstable if you configure it to use the latest tag of your image.

- It would become impossible to use some older version of your image on purpose in some of your deployments.
- To overcome this issue, you can use or tag the build version using GitLab CI/CD environment variables mentioned below:

Git tag	`CI_COMMIT_TAG`
Git commit SHA-256 hash	`CI_COMMIT_SHA`
Shortened Git commit hash	`CI_COMMIT_SHORT_SHA`
Git branch name	`CI_COMMIT_BRANCH`
date + timestamp	`CI_JOB_STARTED_AT`
unique build number	`CI_JOB_ID`

4.1 : Modify the flask application

Now, let us modify the flask application code under directory `flask-app` in the gitlab project and to modify/edit the code,you may use the gitlab Web IDE.

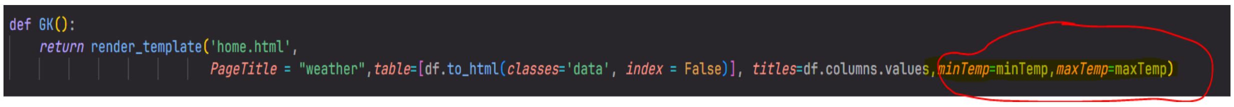


Devops 2022 fall > Students > shiva-labo7
main
shiva-labo7 / webapp-flask / app.py
Find file
Blame
History
Permalink
Update webapp-flask/app.py
poojara authored 6 hours ago
f7616973
app.py
1.49 KiB
Open in Web IDE
Replace
Delete

```python
#Flask imports
from flask import Flask, render_template, send_file, make_response, url_for
from flask import Response
import datetime
#Pandas
import pandas as pd
import os

import io
import random

# Read the data
df  = pd.read_csv("CO2.csv",error_bad_lines=False)

# # Convert time format from unix milliseconds to datetime64 format
#df['time'] = df['time'].astype('datetime64[ms]')


# Create a flask app
app = Flask(__name__)

# main route
@app.route('/')
```

- Let us modify the app.py to calculate today's minimum and maximum temperature and add the code snippet should be added in `app.py`
    - Import the datetime `import datetime`
    - You can add the following code soon after this line `df=result[result.columns[4:]]`
        - Get today's date `today = datetime.date.today()`
        - Extract the rows belongs to today `data = (df['_time'].dt.date == today)`
        - Get the today's data `today_data = df[data]`
        - Get minimum temperature data `minTemp = today_data['_value'].min()`
        - Get maximum temperature data `maxTemp = today_data['_value'].max()`

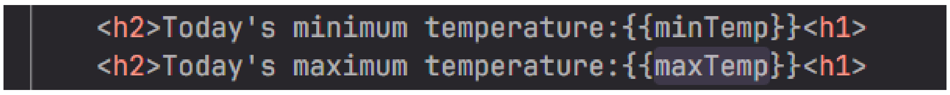
- Finally need to pass the minTemp and maxTemp values to html as shown below (minTemp=minTemp, maxTemp=maxTemp)



```
def GK():
    return render_template('home.html',
                           PageTitle = "weather",table=[df.to_html(classes='data', index = False)], titles=df.columns.values,minTemp=minTemp,maxTemp=maxTemp)
```

- Add the duration, min value, and max value to the `home.html` file present in `/templates/home.html` directory.
  The final code look like

- Now, modify the `templates/home.html` to display min and max temp as shown below:



```
<h2>Today's minimum temperature:{{minTemp}}<h1>
<h2>Today's maximum temperature:{{maxTemp}}<h1>
```

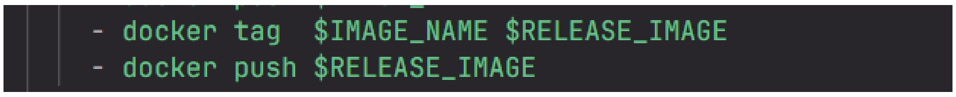
4.2 : Edit .gitlab-ci.yml file

Here basically you will update .gitlab-ci.yml to tag a container image while building.

- Update IMAGE_NAME: as `IMAGE_NAME: gitlab.cs.ut.ee:5050/<name_space>/<project_name>/<image_name>:$CI_JOB_ID`
- Example: `gitlab.cs.ut.ee:5050/devops2023-fall/students/devops2023fall-xx-c1xx9/prac07-gitlab-cicd/prac7:$CI_JOB_ID`
  - You may use different variables as image tag, e.g. $CI_COMMIT_SHORT_SHA
  - You can use any of the gitlab environment variables to tag the image as mentioned in the introduction of Exercise 4.
- Add one more variable as

RELEASE_IMAGE: `gitlab.cs.ut.ee:5050/<name_space>/<project_name>/<image_name>`:latest

- Add two more lines under build_image's script

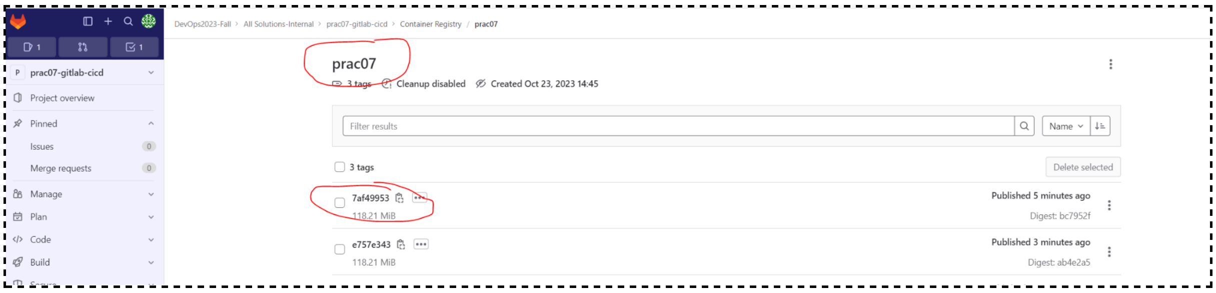


```
- docker tag  $IMAGE_NAME $RELEASE_IMAGE
- docker push $RELEASE_IMAGE
```

- Commit the project with message as "Updated the app.py and home.html to display min, max temp values"

4.3 : Checking the output of the flask application and container registry

- Once, you commit in 4.2 the pipeline is executed and you can see the *build_image* job output in container registry as shown below



DevOps2023-Fall > All Solutions-Internal > prac07-gitlab-cicd > Container Registry / prac07

prac07

3 tags Cleanup disabled Created Oct 23, 2023 14:45

Filter results Name

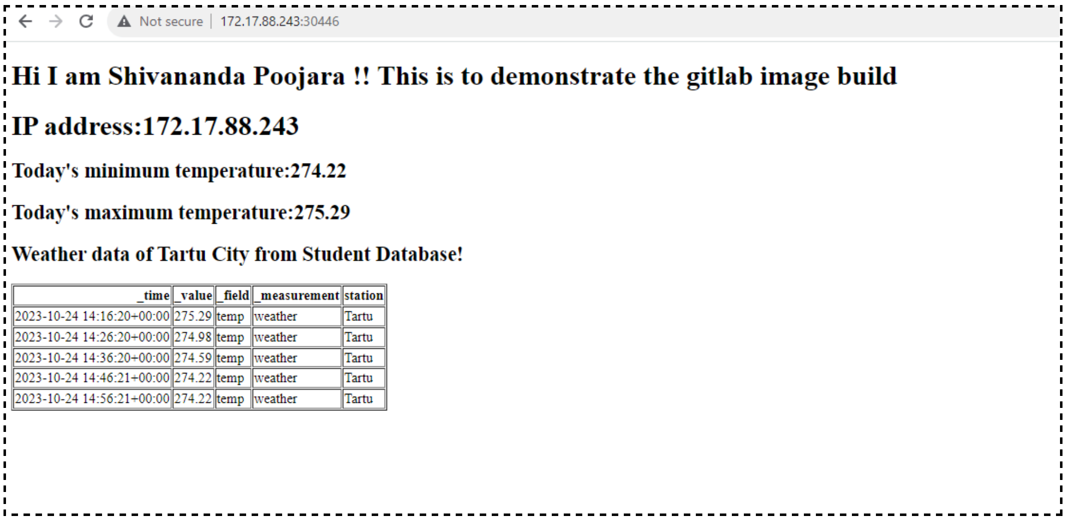
3 tags Delete selected

7af49953 118.21 MiB Published 5 minutes ago Digest: bc7952f

e757e343 118.21 MiB Published 3 minutes ago Digest: ab4e2a5

- You can check flask application running by using microk8scontroller external IP address and NodePort address

- The final flask application output is displayed as:



Not secure | 172.17.88.243:30446

Hi I am Shivananda Poojara !! This is to demonstrate the gitlab image build

IP address:172.17.88.243

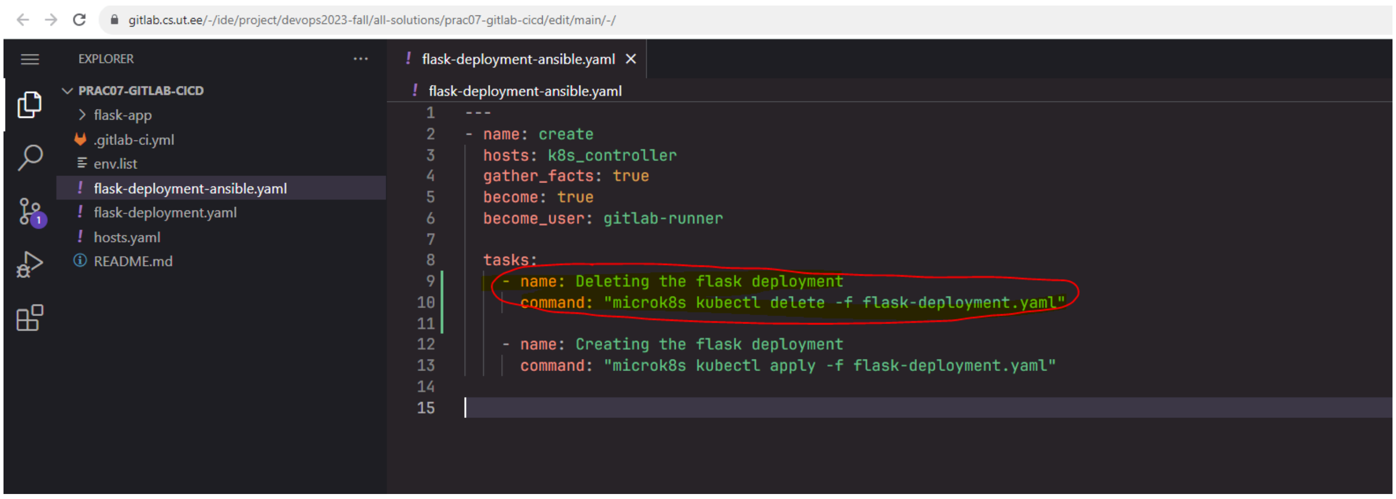
Today's minimum temperature:274.22

Today's maximum temperature:275.29

Weather data of Tartu City from Student Database!

_time	_value	_field	_measurement	station
2023-10-24 14:16:20+00:00	275.29	temp	weather	Tartu
2023-10-24 14:26:20+00:00	274.98	temp	weather	Tartu
2023-10-24 14:36:20+00:00	274.59	temp	weather	Tartu
2023-10-24 14:46:21+00:00	274.22	temp	weather	Tartu
2023-10-24 14:56:21+00:00	274.22	temp	weather	Tartu

If your unable to see the changes in the flask web application, than you can add the command in flask-deployment-ansible.yaml to delete the deployment, for example:



gitlab.cs.ut.ee/-/ide/project/devops2023-fall/all-solutions/prac07-gitlab-cicd/edit/main/-/

```
---
- name: create
  hosts: k8s_controller
  gather_facts: true
  become: true
  become_user: gitlab-runner

  tasks:
    - name: Deleting the flask deployment
      command: "microk8s kubectl delete -f flask-deployment.yaml"

    - name: Creating the flask deployment
      command: "microk8s kubectl apply -f flask-deployment.yaml"

```

Screenshot - 2

Take a screenshot of a webpage and IP address are clearly seen.

Try out!! (Not compulsory task):

You may also want to consider experimenting with the deployment of the InfluxDB, InfluxDBData, and Flask microservice application for personal exploration and learning. This isn't a formal requirement for the lab; rather, it's an opportunity to explore these technologies for your own interest and enjoyment.

Deliverables

1- Gather all the screenshots

- Screenshot 1
- Screenshot 2

2- Download code of your GitLab project

3- Zip the code, screenshot and Upload the zip file to the course wiki page.

4- You may **Stop** the Virtual Machines and you can start using the same in the next **practice session.**

Don't delete your VMs