gitlab CI/CD for team project: masterclass

Team Project 2022-23

Continuous Integration (CI) and Continuous Deployment (CD) are pivotal to modern application development. In this session, we will setup a CI pipeline using gitlab CI. You are encouraged to follow along and try the instructions with your own repository.

This session will use the terminal and gitlab web UI however the principles are the same as if using an IDE.

This session is not exhaustive. Similar to git, many of the commands are rarely used and it is completely fine to consult the documentation for gitlab CI when needed.

Terminology

Cloud Virtual Machine (VM) - a computer that can be accessed from the internet, usually over SSH, usually running a flavour of Linux, users of the machine usually have root access. A Virtual Machine can run multiple software packages, and various services to help with software development and/or deployment of a web application.
In this masterclass: we will see how a VM can simultaneously be a web (HTTP), Database (SQL) and CI (gitlab runner) server to support our CI.

gitlab runner - a Continuous Integration (CI) software system used to execute the CI pipeline of a gitlab repository. A gitlab runner needs to be registered with a gitlab repository to work. Gitlab runners are typically run on VMs in the cloud, but can be run on any machine with a terminal and internet access (even your own laptop- but we dont typically do this - why?). Gitlab runners usually run CI commands in a predicable environment like Docker. If your gitlab project has no runners enabled, the CI pipeline will fail.
In this masterclass: we will install gitlab runner on a VM and link it to our repository.

Docker - an Operating System containment layer. Containers are lightweight alternatives to full installations of an operating system. Containers usually have some software pre-installed. This makes a Docker image a predictable starting point and allows us to minimise the amount of configuration done manually. We will build a container of our own application to ease deployment.
In this masterclass: we will install a gitlab runner that uses Docker images to run our CI pipeline.

shell - an interactive terminal like bash (Ubuntu) or zsh (macos). Without GUIs, shell commands are the only way to control a VM or Docker container. We use the shell (often SSH- Secure SHell) to login to remote computers. CI pipelines are made of a sequence of shell commands, called a shell script. Docker is used to provide a repeatable lightweight shell.
In this masterclass: we will use shell commands to configure a remote VM and review the shell commands in our CI pipeline.

Container image registry - a service to store light-weight docker container definitions. We can publish the container for our own app to a image registry to simplify deployment.

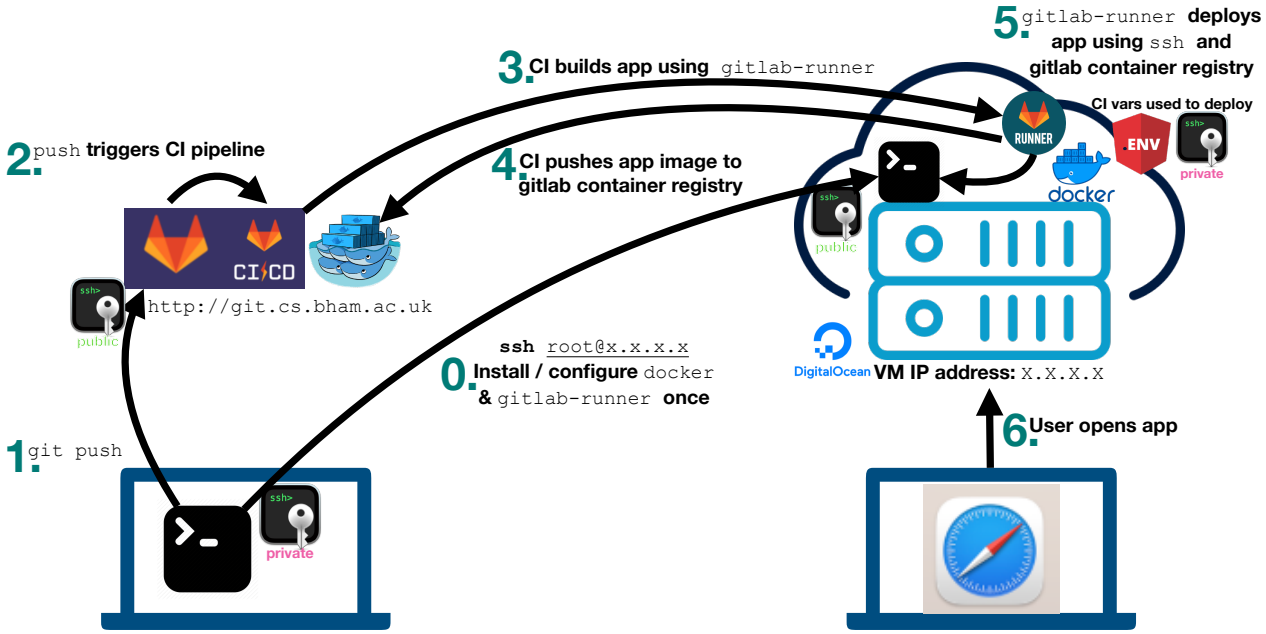
In this masterclass: We will use docker container images from the most used public container registry (docker hub) and our own internal registry (git.cs.bham.ac.uk).

.gitlab-ci.yml - The definition of a CI/CD pipeline used in a git repository to define the steps to build, test and deploy the software in a repository. The yml format means Yet Another Markup Language- yml is space sensitive. The CI pipeline has various sections including the docker image used to run the pipeline and CI variables that may be used in the pipeline.
In this masterclass: We try to get the given CI pipeline working.

environment variables and CI variables - we must never commit secret or variable information directly to a git repository (why?). Part of a CI pipeline often requires automatically SSH-ing into a remote VM to deploy software. Instead of storing the secrets directly in the git repository, we instead store these in a private variables.
In this masterclass: we will use variables to store project secrets and variables so we can securely login to our VMs and run the deployment stage.

ssh key pairs - SSH key pairs consist of a private and public key. We often install our public key on a remote server e.g. git.cs.bham.ac.uk or a VM. Using the private key we can log into the remote server securely without interactively entering a password. Key pairs help us to securely login to a remote server automatically, as we might want to login to a VM in a CI pipeline.
In this masterclass: we have to configure CI variables and upload the private key as a CI variable to our repository.

Diagram



Setup a gitlab-runner CI server for our repository

Get a virtual machine

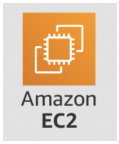
The department will kindly provide a VM and a private SSH key to an Amazon EC2 server. Please ensure you can ssh into the school machines in order to access this.

n.b. the provided machines will have `docker` and `gitlab-runner` pre-installed, but will be firewalled by default - you will need university VPNs to SSH into the VM.

Free Cloud VM options

Cloud VMs are provided by companies such as Google (Google Cloud Platform), Amazon (Elastic Compute Cloud), Microsoft (Azure VM) and many, more.

The following are 'free' options to get VMs:



wait for IT to provide the Amazon EC2 VM. n.b. the firewall restriction noted above.

Azure: Start with $100 Azure credit, no credit card required, credit lasts 1 year, OK user interface https://azure.microsoft.com/en-gb/free/students/

Google cloud platform $300 credit, credit card required, credit lasts 90 days, complex user interface https://cloud.google.com/free/docs/free-cloud-features

Google cloud platform compute engine free tier, not time-limited, credit card required https://cloud.google.com/free/docs/free-cloud-features#free-tier-usage-limits (see section "Compute Engine")

Digital Ocean: 60 day limit on credit, $200 free credit, credit card required, simpler UI. referral link for $200 credit: https://m.do.co/c/aa45eafdb905

n.b. if you decide to use a credit card to sign up for a VM add a calendar reminder to delete the VMs. **YOU ARE NOT REQUIRED TO SIGN UP FOR PAID SERVICES.**

SSH into the Virtual Machine

IT support provide a private SSH file in `.pem` format. Azure is similar, providing a private `.pem` SSH key. Digital Ocean allows you to upload a public key when you create a VM.

If you have ssh keys keys installed and loaded in the default location:

```
ssh $USER_NAME@$SERVER
```

if you need to specify the private key file:

```
ssh -i $KEY_FILE $USER_NAME@$SERVER
```

where:

`$USER_NAME` is the name of the user on the VM, e.g. `root`, `ec2-user` etc the user name is specific to that VM

`$SERVER` is the DNS name or public IP address of the server e.g. `ec2-52-56-105-171.eu-west-2.compute.amazonaws.com` or `139.59.163.214` the IP address is specific to your VM

`$KEY_FILE` is the local `.pem` private key file to access the server e.g. `~/.ssh/TeamProject2022.pem`. This file must exist on your local machine. Keep the file and the content private e.g. `cat ~/.ssh/TeamProject2022.pem` should result in a string starting `-----BEGIN OPENSSH PRIVATE KEY-----`

Optional: install Docker on the VM

The IT provided VM has Docker pre-installed. DigitalOcean provides a VM image with Ubuntu+Docker. On other VMs you may need to install Docker:

Instructions for Ubuntu here: https://docs.docker.com/engine/install/ubuntu/

Instructions for Amazon EC2 here: https://www.cyberciti.biz/faq/how-to-install-docker-on-amazon-linux-2/

To confirm successful installation, run the command:

```
docker version
Client: Docker Engine - Community
 Version:           23.0.0 ...
```

Optional: install `gitlab-runner` on the VM

The IT provided VM has `gitlab-runner` pre-installed. On other platforms you may need to install it:

https://docs.gitlab.com/runner/install/

There are also very useful install instructions on the git repository - 'settings' - 'CI/CD' - 'Runners' - 'Expand' - 'Show runner install instructions'.

Link the `gitlab-runner` to our own gitlab repository

Run the commands to link your gitlab runner to you repository:

```
sudo gitlab-runner register --url https://git.cs.bham.ac.uk/ --registration-token $REGISTRATION_TOKEN
```

where: `$REGISTRATION_TOKEN` is found on your git repository - 'Settings' - 'CI/CD' - 'Runners' - 'Expand'

This command will ask you several configuration questions. For most of the questions, use the default except:

```
Enter an executor: docker-ssh+machine, custom, virtualbox, parallels, shell, ssh,
docker+machine, instance, kubernetes, docker, docker-ssh:
docker
Enter the default Docker image (for example, ruby:2.7):
jhipster/jhipster:v7.9.3
```

Optional, if needed: set the repo to run on non untagged commits:

On your git go to - 'Settings' - 'CI/CD' - 'Runners' - 'Expand' - 'edit' - 'Run untagged jobs'

Deployment

In order to deploy the application to the Virtual Machine, we will need to set some variables in our CI pipeline- we need to provide a `$VM` to deploy on and authentication details `$VM_USER` and `$RSA`.

On your git repository - 'Settings' - 'CI/CD' - 'Variables' - 'Expand' - 'Add Variable' - uncheck 'Protect variable' Add the following three variables:

- `$RSA` - content of the private key - Type: File
- `$VM_USER` - the username of the VM - Type: Variable
- `$VM` - the IP address or domain name of the VM - Type: Variable

un-comment the `deploy-git:` section of the `,gitlab-ci.yml`

```
deploy-git:
  image: alpine:latest
  stage: deploy
  when: on_success
  before_script:
    - chmod og= $RSA
    - apk update && apk add openssh-client
    - ssh -o StrictHostKeyChecking=no -i $RSA $VM_USER@$VM "docker compose -f ~/team-project-deployment/src/main/docker/app.yml down || true"
    - ssh -o StrictHostKeyChecking=no -i $RSA $VM_USER@$VM "docker rm -f $(docker ps -a -q) || true"
    - ssh -o StrictHostKeyChecking=no -i $RSA $VM_USER@$VM "docker volume rm $(docker volume ls -q) || true"
    - ssh -o StrictHostKeyChecking=no -i $RSA $VM_USER@$VM "rm -rf ~/team-project-deployment || true"
    - ssh -o StrictHostKeyChecking=no -i $RSA $VM_USER@$VM "docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY"
    - ssh -o StrictHostKeyChecking=no -i $RSA $VM_USER@$VM "docker pull ${CI_REGISTRY_IMAGE}:latest"
  script:
    - scp -o StrictHostKeyChecking=no -i $RSA -r . $VM_USER@$VM:~/team-project-deployment
    - ssh -o StrictHostKeyChecking=no -i $RSA $VM_USER@$VM "sed -i '5s|teamproject|$CI_IMG|' ~/team-project-deployment/src/main/docker/app.yml"
    - ssh -o StrictHostKeyChecking=no -i $RSA $VM_USER@$VM "docker compose -f ~/team-project-deployment/src/main/docker/app.yml up -d"
```

Deployed app from the masterclass

The deployed app from the masterclass can be found here: http://138.68.150.235:8080

Note: lifting `ssh` firewall connection `LIMIT` on Ubuntu VMs

On Ubuntu VMs the default firewall settings limit the number of SSH connections to 6 within 30 seconds, whether the connections are successful or not. This can affect the deploy pipeline if you have many connections one after the other, as we do. Use the command `ufw allow ssh` to lift the restriction:

```
root@athens:~# ufw status |grep 22
22/tcp                     LIMIT       Anywhere
22/tcp (v6)                LIMIT       Anywhere (v6)
root@athens:~# ufw allow ssh
Rule updated
Rule updated (v6)
root@athens:~# ufw status |grep 22
22/tcp                     ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
```

This issue affects Ubuntu VMs, other VMs e.g. Amazon EC2 Linux may not have the same issue.

1 Hotfixes

Hotfix 1: install `docker-compose` on the VM

Using the IT supplied VMs (AWS). The IT provided VM are *supposed* to have `docker-compose` pre-installed. If there is an error related to `docker-compose` in your CI, manually install `docker-compose` on the VM:

```
sudo mkdir -p /usr/local/lib/docker/cli-plugins/
sudo curl -SL https://github.com/docker/compose/releases/latest/download/docker-compose-linux-aarch64 -o /usr/local/lib/docker/cli-plugins/docker-compose
sudo chmod +x /usr/local/lib/docker/cli-plugins/docker-compose
```

Then, update any instances of `docker compose` to `docker-compose` in the `.gitlab-ci.yml`.

Hotfix 2: install `chromium` on the Jhipster image

The `maven-package` stage may fail due to an issue with `chromium`. If you get an error in the pipeline related to `chromium` command not found, edit the `.gitlab-ci.yml` `maven-package` section:

```
maven-package:
  stage: package
  script:
    - echo "jhipster" | sudo -S apt-get update
    - echo "jhipster" | sudo -S apt-get install -y chromium-browser
    - echo 'whereis chromium-browser'
    - ./mvnw -ntp verify -Pprod -DskipTests -Dmaven.repo.local=$MAVEN_USER_HOME
```

Hotfix 3: cannot deploy using SSH in `git-deploy`

Using the IT supplied VMs (AWS, firewalled). If your pipeline fails on the deploy stage with `ssh: connect to host <ip> port 22: Operation timed out`, when using setting the `$VM` CI variable to the IP in the announcement, then try this:

On the VM, run `ip a`

In `eth0`, use the `inet ip` (e.g. 172.x.x.x, n.b. the last .x should **not** be .255) and paste that in the `$VM` CI variable on GitLab.

(Credit to Francis O' Brien on the CSS discord)

IT says the reason is the firewall:

"13.x.x.x is your VM's public ip address i.e. it is routeable from the internet. The ssh (22), http (80) and https (443) ports are open to traffic from the university's network only (147.188.0.0/16) hence the need to be on campus or connected to the VPN to access the VM.

172.x.x.x is your VM's private ip address i.e. it is not routeable from the internet and only routeable from within the same network. Ssh'ing from your container to the host VM using the public ip address means the traffic is being sent from the VM out to the internet and back to the VM which will be blocked by the firewall (because only ssh from 147.188.0.0/16 is allowed)."