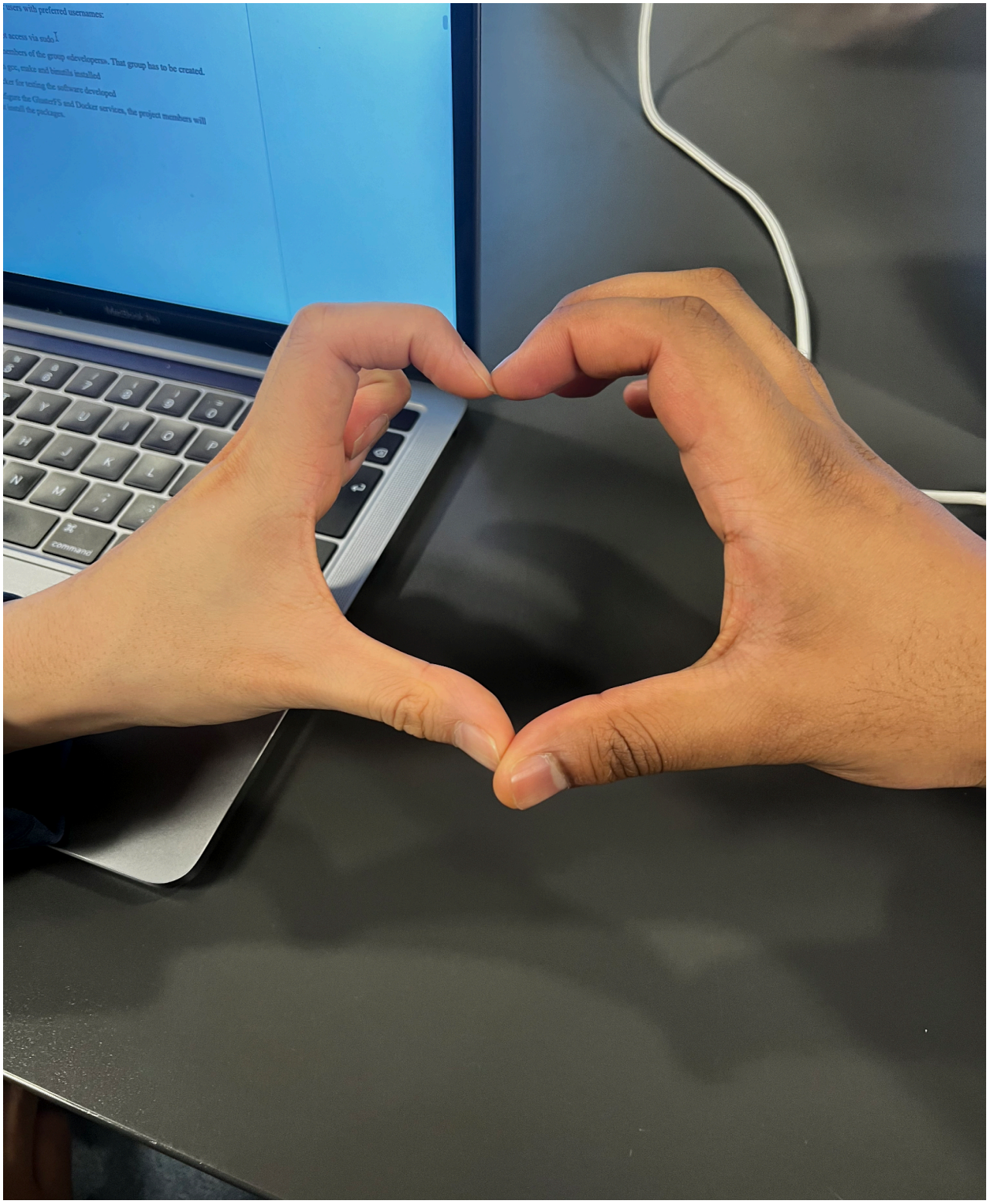
A development environment



ACIT4430 - Biraveen Nedunchelian & Kim Phan

Table of content

Task:

Alternative 1 - A development environment

A development project is about to start programming. They ask you to deploy a development environment for them. They have the following requirements:

- Two storage servers with the GlusterFS server installed, each with a single CPU.
- Two servers the developers will use to write code on, each with a single CPU. They have to have emacs, jed and git installed
- All machines have to have four users with preferred usernames:
- bob - janet - alice - tim
- All four users should get root access via sudo
- Tim and Janet have to be members of the group «developers». That group has to be created.
- Two compile servers with gcc, make and binutils installed
- One server running Docker for testing the software developed
- You don’t have to configure the GlusterFS and Docker services, the project members will do that themselves. Just install the packages.

Workflow Design

Access structure:

As part of the deployment process, it has been determined to grant the junior system administrator limited access to specific resources or areas.

Openstack:

What is openstack?

Openstack is an open-source cloud computing platform. It's an Infrastructure as a service or IAAS and it enables us to manage private and public clouds. Having access to Openstack is essential for our deployment because we often need to know the specific IP addresses of various instances, especially when executing commands via SSH.

Why did we choose openstack?

By choosing OpenStack as our cloud computing platform is a strategic decision. The number one reason was its cost-effectiveness, begin an open source platform. The reason begin it is Oslo Metropolitan university´s primary cloud solution utilized across our educational institution. This ensures streamlined support for our cloud-based application.(as our project)

GITLAB - Version control

- We use gitlab for versioncontrol and use it to manage new iterations. This has helped us during testing, when automating the entire project with a python file

Trello - Kanban

During the development of our infrastructure we have decided to use trello to map/divide tasks among the **two senior developers.** We also used if to write down our deployment guide for our juniors

Creating users

For creating users and developer groups,we will use ansible to configure it on all instances

Creating instances

For deploying our instances and installing all the necessary packages. We will use terraform and its remote exec to configure and set them up.

Expectations:

What we expect to create is a fully automatic deployment which will be easy to understand and straight forwards to deploy for our juniors. The idea is to make the deployment so easy that anyone, even a person who has no experience with cloud can use it.

Deployment guide:

General important information about the setup

Terraform:
What will be needed to deploy our infrastructure
The majority of our architecture can be deployed through terraform. We have decided to simplify the deployment process for our beloved juniors/interns. The deployment for them mis quite simple, they can open the terminal, go into our project map and can perform a simple “terraform init” to initialize, then a “terraform plan” to see what will be deployed. They will then have to run a last command which is “terraform apply” and then write yes to confirm the application of deployment through terraform.
Below i'll explain how our project folder is setup for our development environment

Folder hierarchy

```
.
└── Project - A developer environment/
    ├── Main.tf
    ├── Playbooks/
    │   └── users.yml
    ├── Developer_server_emacs_jed_git/
    │   └── DevelopmentServerproject1.tf
    ├── Docker_server2/
    │   └── project1_docker.tf
    ├── Storage_server_GlusterFS/
    │   └── Gluster.tf
    ├── Compile_server_gcc_make_binutils/
    │   └── compileserver.tf
    ├── Master VM/
    │   └── MasterVM.tf
    └── Python/
        ├── linux_commands.py
        └── main.py
```

The folder is created in a way to separate each entity to modulate the project making it easier for future testing. it's also seen as good/best practice when it comes to terraform deployment. The hierarchy represents how important the file is.

Each of the .tf files in the folder represents a VM, with the exception of "`DevelopmentServerproject1.tf`" which deploys two vm's because of the "Count = 2" in this part:

```
resource "openstack_compute_instance_v2" "dev_server" {
  count           = 2 # Creates two instances instead of just 1
  name            = "DevServer-${count.index}"
  image_name      = "ubuntu-22.04-LTS"
  flavor_name     = "css.1c1r.10g"
  key_pair        = "key"
  security_groups = ["default"]
```

The main.tf file is where the **combined magic** happens. It only consists of code which calls out the other files within the folder. Down below, you can see how we are only calling on each module in our main.tf file.

Benefits:

- Systematize the project
- Easier to develop further
- easier to manage:
  If the cloud provider updates a part of the infrastructure in use so that it becomes unsupported of incompatible with the existing infrastructure, we don't have to take down the entire infrastructure, we can only take down one part, modify the tf file to fit our infrastructure again and then use terraform apply to set it up again.
- modularity is best practice
- easy to reuse

Downsides:

- Can be time consuming the first time it's done(overhead
- Refactoring, meaning as our infrastructure grows and evolves, you may need to refactor modules. This can be as simple as changing variable names or as complex as redesigning entire modules, which can be time-consuming.

How will our juniors deploy our infrastructure?

Since the project does not specifically say so, but seems to hint at three possible solutions I've decided to write out how both solutions are deployed.

Deployment Alternativ A:

Since every instance is written in terraform, we have made it so every terraform file in each folder can be set up and run by just applying the main.tf-file. The main.tf file consists of just commands calling on the infrastructures. Below is a part of script calling on one of our terraform files:

```
module "Docker_server2" {
   source = "./Docker_server2"
 }
```

The main.tf file has to be at the root directory for this approach to be possible. What happens here is that the main.tf file calls on each of the terraform files in every folder systematically starting from which script is first called in the main.tf file and downwards

NOTE: THIS IDEA WAS SCRAPPED BECAUSE IT DIDN'T HAVE REDUNDANCY

Deployment Alternativ B:

Similar to alternative A, the only difference is that instead of giving our juniors access to the files through and deploys via our/his computers, we are instead giving him access to a Master-virtual machine(MasterVM). He can then simply use a “terraform apply” command and set up the entire infrastructure in one click. If there is only a part of the infrastructure which needs to be redeployed. The junior can simply navigate to the correct folder the affected vm is in and then perform either a “terraform destroy” and then a “terraform apply” or just a simple “Terraform apply” if the affected infrastructure depending on the severity of the affected virtual machine(VM)

NOTE: THIS IDEA WAS SCRAPPED BECAUSE IT SEEMED TO UNNECESSARILY COMPLICATED TO DEPLOY FOR JUNIORS

Deployment Alternativ C: THE ONE WE WENT FOR

Is also similar to alternative A and more closely to B, the only difference is that instead of giving our juniors access to the files through and deploys via our/his computers, we are instead giving him access to a Master-virtual machine(MasterVM). He then goes into the A-development-environment folder which was cloned from our public github repo. He will then have into our Python folder and run our python main.py file. with the following command:

- python3 main.py

The infrastructure will then after a few minutes deploy itself and add users to each instance. The master vm uses our computer's ssh keys so that our computer can get access to it. after that every other vm get the ssh key to out masterVM so that out master vm can access our other instances

Terraform files - explained

MasterVM.tf - The one to rule them all

This IS the masterVM our juniors will get access to. This is the VM which will deploy all out all our other vm´s. By doing it this way our juniors dont need access to every vm. He only needs one which has access to the mall.This one is installed with the following packages:

- Puppet
- Openstack
- Ansible
- Terraform
- A git clone command to clone the github repo from us

Main.tf - The one to deploy them all

This is the one terraform file which calls on every of the terraform files below. This makes it possible for us to deploy all our vm´s in one go instead of deploying every instance manually. This terraform file simplifies the juniors setup process.

DevelopmentServerproject1.tf - 2 development servers to write code on in 1 terraform file

The DevelopmentServerproject1.tf file is the one responsible for creating 2 development servers. This is the one the developers will use to write code on. These ones is installed with the following packages:

- emacs
- jed
- git

project1_docker.tf - Dockerization

This terraform file is responsible for creating a docker server. The service is enabled from default and can be run straight out the gate. This one is installed with the following packages:

- Docker

Gluster - 2 storage servers in 1 terraform file

This terraform file deploys our 2 glusterFS servers. The gluster server comes preinstalled with gluster service and is enabled from the start as the terraform remote exec does it for them. There is no further action needed to enable/activate it. These ones is installed with the following packages:

- GlusterFS

Compile Server - Compilation

This terraform file(named: —------- is the one used to create the compile instances. Everything is setup from the start, and can be used right away These ones is installed with the following packages:

- gcc
- make
- binutils

Ansible file - explained

users.yml

The users.yml have two jobs, those are the following:

- Create the users
- Also create groups and put the respected people in our groups

Commands to check if the group developers got created:

- getent group grep developers

Commands to see if all the users got created:

- getent passwd | grep -E 'bob|janet|alice|tim' - you basically ask to get the passwords

Since i was more comfortable with terraform than ansible i used terraform to install the software packages(such as gcc,git and so on) , but in a ideal situation i would rather use an ansible playbook to install all the packages as i would mitigate the issue of redundancy, and make it so that if a junior accidentally deletes a packages, i can just run the ansible playbook again and get it up again

Testing Workflow

Pre-testing

We ran our deployment three before it was the juniors turn, we mapped the time for deployments and they were:

- took us 5 min and 3 sec minutes
- took us 2min and 28sec
- took us 2 min and 34 sec

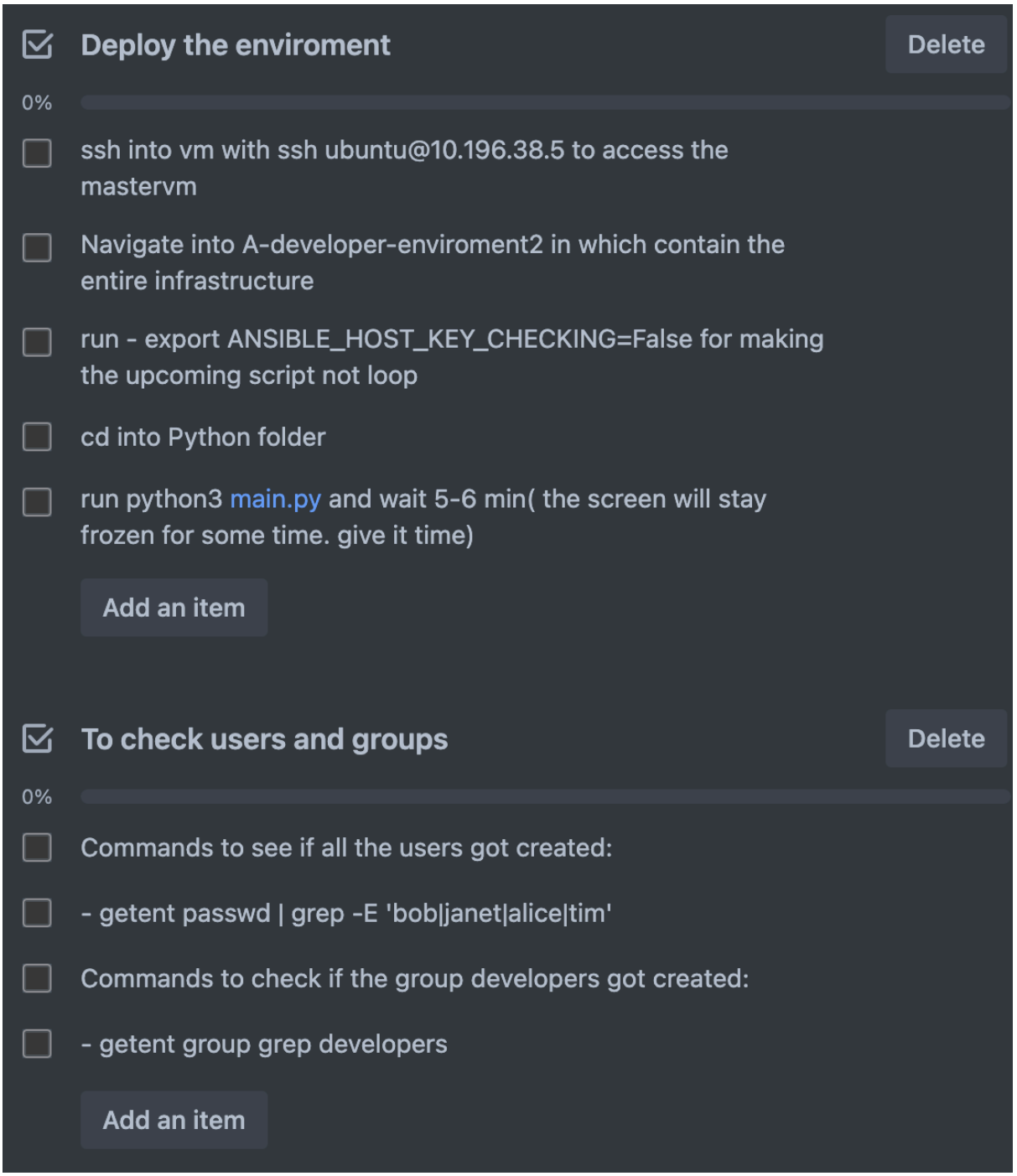
We can see it took less time to deploy it the second and third time compared to the first time. The reason begins when running terraform apply for the first time. It performs several steps which can take a lot of time, as such depending on the complexity of our infrastructure. The time take during the first apply includes:

- Initialization: terraform initializes the project, which includes downloading and installing the required providers, initializing modules, and setting up the backend for state management
- Planning:Terraform generates an execution plan, determining what actions are necessary to achieve the desired state specified in the configuration.
- State Management: Terraform updates the state file to reflect the changes made to the infrastructure. The state file keeps track of all resources managed by Terraform.

when you destroy and apply it again, it will be quicker because: Terraform keeps track of your infrastructure in the state file. When you run terraform destroy, Terraform updates the state file to reflect that the resources no longer exist. Upon the next terraform apply, Terraform has a clear picture of what needs to be created from scratch. The efficiency here doesn't necessarily reduce the time cloud providers take to provision resources but can make the overall Terraform execution slightly faster by avoiding unnecessary checks or updates.

Juniors deployment

For the juniors to deploy the infrastructure we made them follow our deployment plan named “for juniors” in canvas. Below is a picture of the deployment plan in a checklist for the juniors to easily know where they are in the deployment stage if they forget:



Questions asked before deployment start:

There were no questions before deployment, so this part will remain unanswered.

Questions asked during deployment:

Since we tried to automate most of the project and make it as seamless as possible, so that even a beginner could understand it. The only question we got during deployment was:

How long will the deployment take and why doesn't it give feedback while deploying:

- For this deployment it will take approximately 3 minutes. We should have mentioned in our deployment plan that we won't get feedback until the infrastructure is fully deployed, this will be added for the next iteration

Questions asked after deployment:

How do I delete the instances?
For our next iteration we will implement a destroy section in our deployment plan

Conclusion

As mentioned previously the installation of packages could be done through ansible instead, since we decided to install the packages through terraform, we have limited ourselves in a way where we have to install the packages on each instance manually instead of just using an ansible playbook as we could have. For the next iteration, this will be implemented instead.

The automation to deploy the infrastructure as well as destroying the infrastructure is easy and nice to deploy according to our tester.

As this is our first automation project, there are certainly challenges that I have not yet known about and therefore not addressed. what i do know is that for the future we will have to improve more on better deployment plan making it easier to juniors to operate our configuration