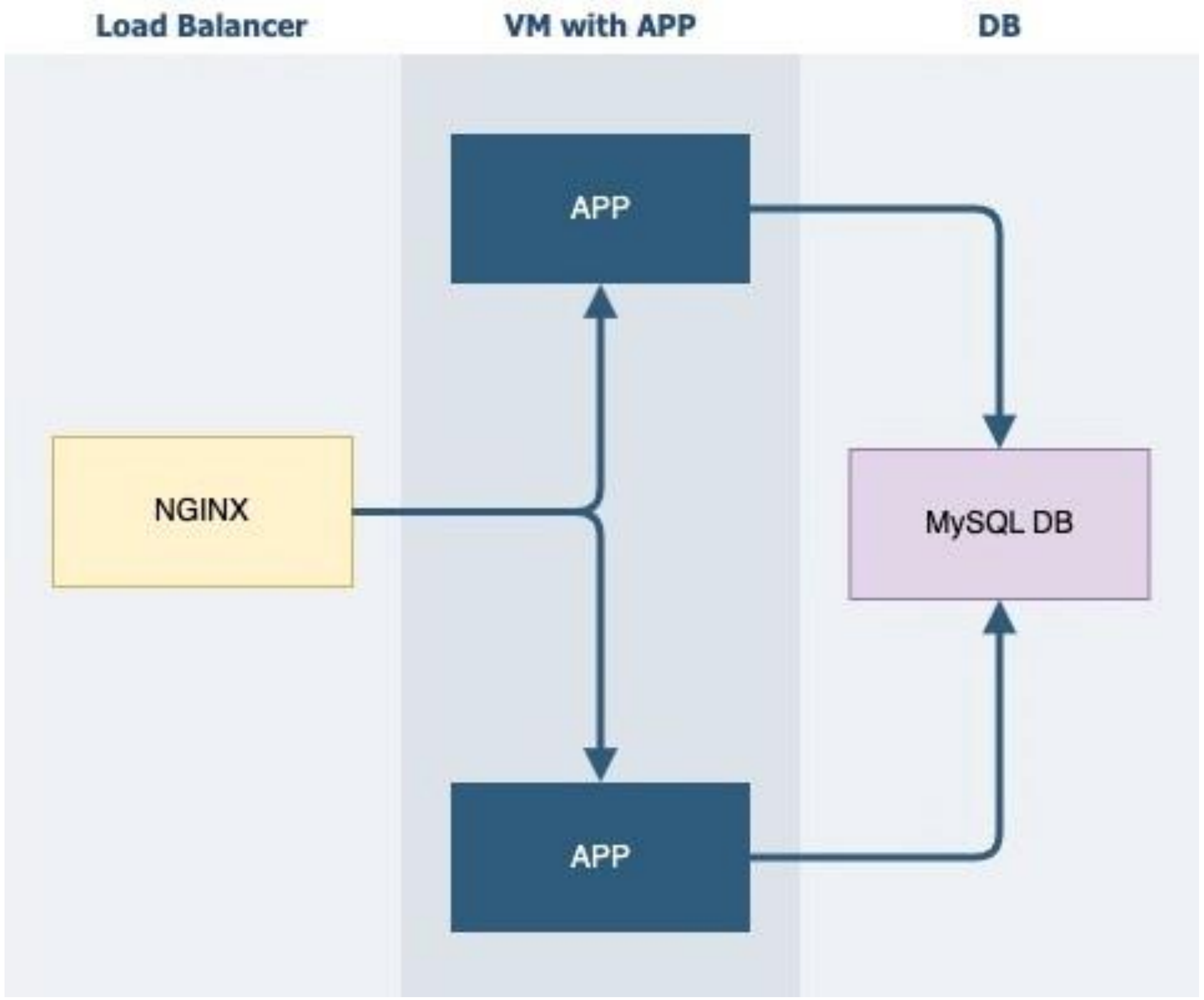
# TERRAFORM WITH AWS

Kamil Dudzicz



## ENVIROMENT

## PRIMARY GOALS

Run few VM to present how works

load ballancer

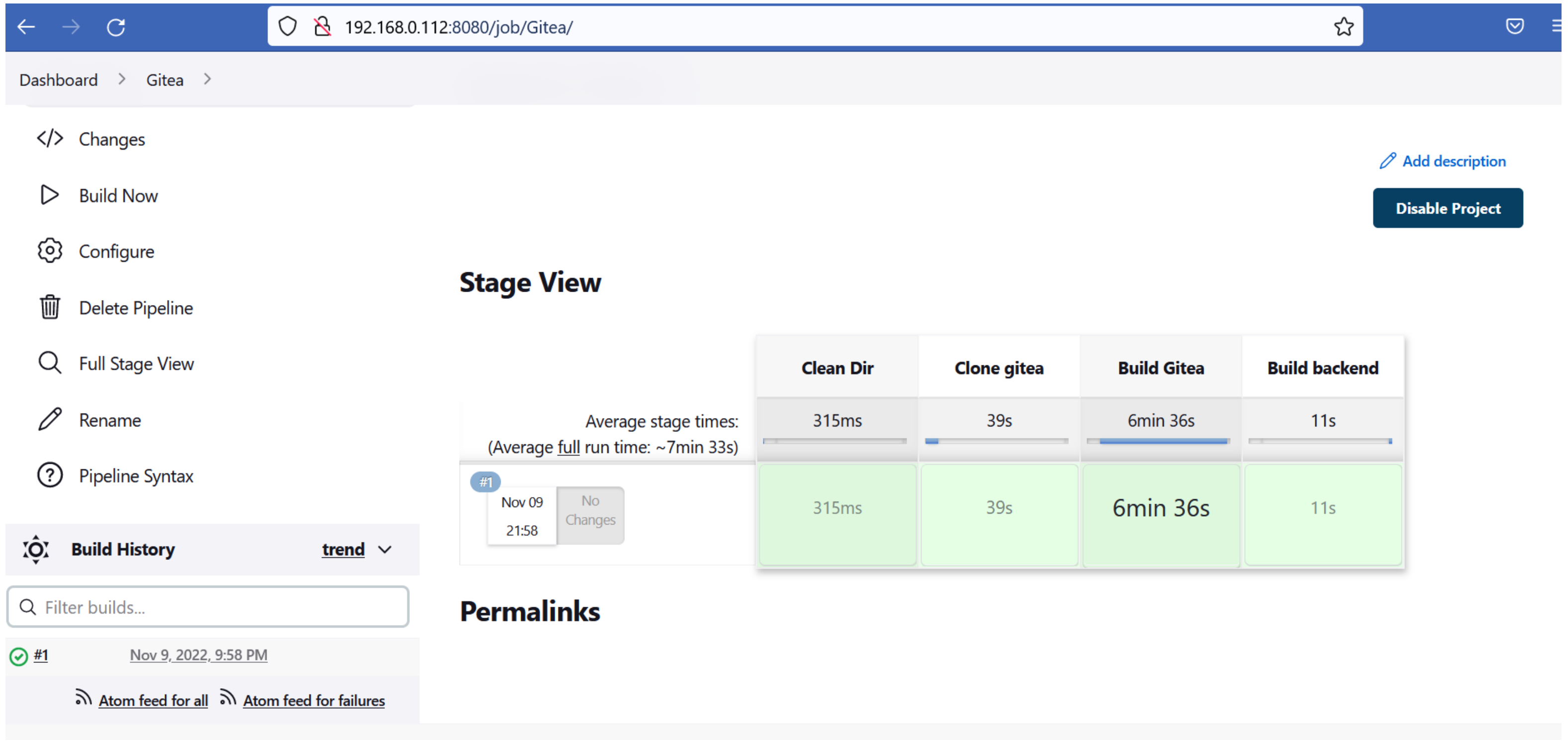
**Preparation to main task** was in stages:
- Building and testing application
- Automatic creation VM and entire environment

**Build and test –version 1**

With Vagrant i created a Virtual Machine with Oracle Linux and install Maven on it.
Next steps was configure Maven to build application and test it.

## Build and test –version 2

This task was more difficult than previous one. I was created pipeline of Gitea Application. Thanks this i learn about Jenkins and how to create automatic process to build and testing application located in github.



192.168.0.112:8080/job/Gitea/

Dashboard > Gitea >

- Changes
- Build Now
- Configure
- Delete Pipeline
- Full Stage View
- Rename
- Pipeline Syntax

Add description

Disable Project

**Stage View**

| | Clean Dir | Clone gitea | Build Gitea | Build backend |
|---|---|---|---|---|
| Average stage times: (Average full run time: ~7min 33s) | 315ms | 39s | 6min 36s | 11s |
| #1 Nov 09 21:58 No Changes | 315ms | 39s | 6min 36s | 11s |

**Permalinks**

**Build History** trend

Filter builds...

#1 Nov 9, 2022, 9:58 PM

Atom feed for all Atom feed for failures

## Automatic creation VM and entire environment – version 1

With Vagrant i created automatic 4 VM.

- VM1: for database (MariaDB)
- VM2: for nginx (as a load ballancer)
- VM3: first copy off Gitea Application
- VM4: second copy off Gitea Application

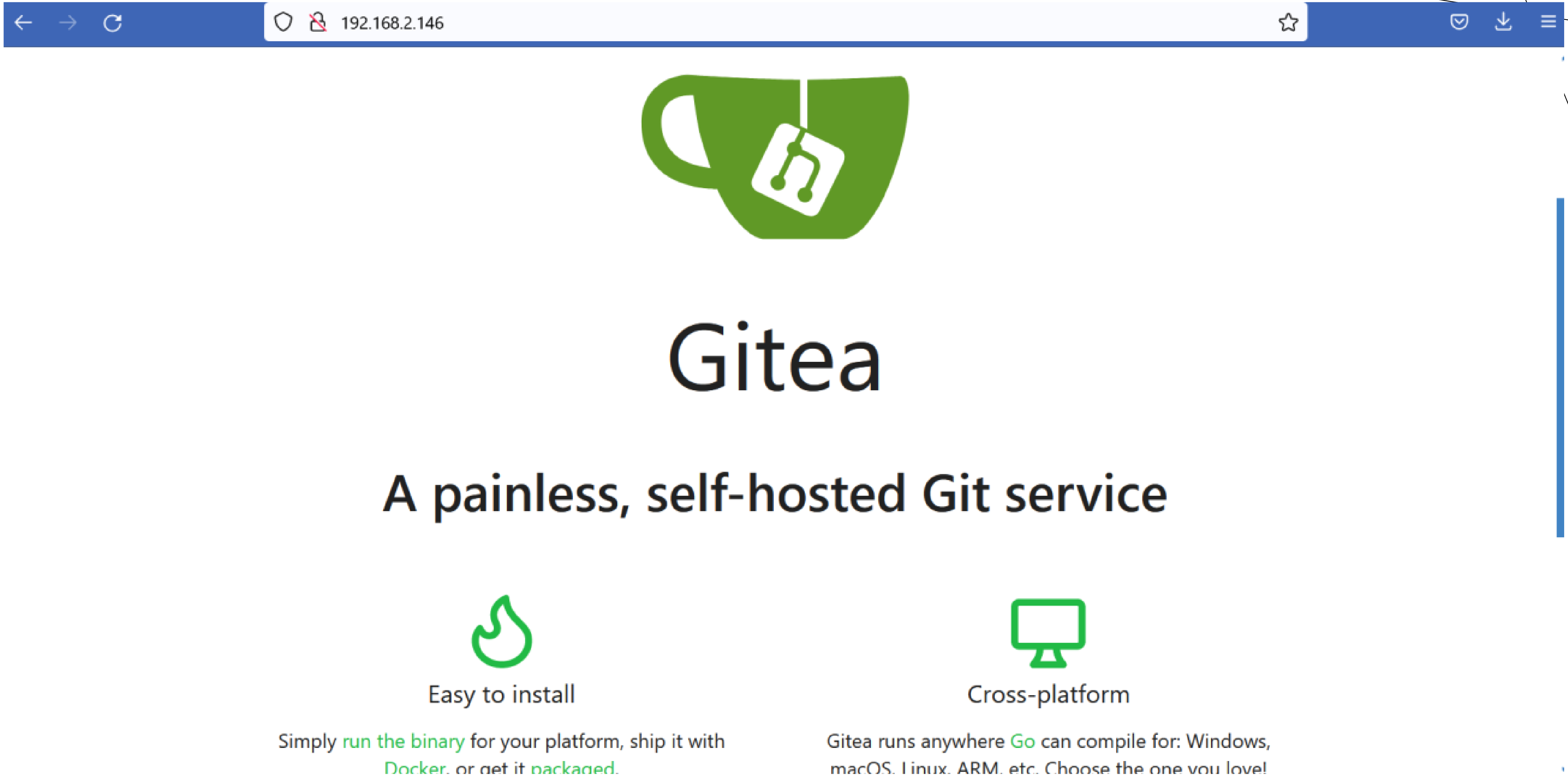
The scheme of operation of this environment is the same as my main task - the user communicates with the machine with nginx and works with Gitea application. Nginx choose one of the VM with apps and comunicate with it until connection is available. When some error appears nginx reconnect to different VM with apps.
Each copy of VM with Gitea Application comunicate with database located on different VM.

## Automatic creation VM and entire environment – version 2

Using Ansible, I automatically created a virtual environment, thanks to which I could present nginx as a load balancer. Similar as in the previous case, we have VM: (APP1, APP2, DB+LB, Ansible controller). In this case i use the same VM to nginx and database.

As in the previous cases, the user calls the main address (nginx) in the browser, which redirects to the appropriate place.

```
- name: load balancer and MySQL db preparing
  hosts: nginx-db
  roles:
  - python
  - mysql_db
  - nginx
 - name: download gitea and start service
  hosts: vm1, vm2
  roles:
  - gitea
  - gitea-st
```



Gitea

A painless, self-hosted Git service

Easy to install

Simply run the binary for your platform, ship it with Docker, or get it packaged.

Cross-platform

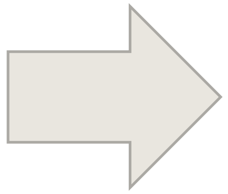
Gitea runs anywhere Go can compile for: Windows, macOS, Linux, ARM, etc. Choose the one you love!

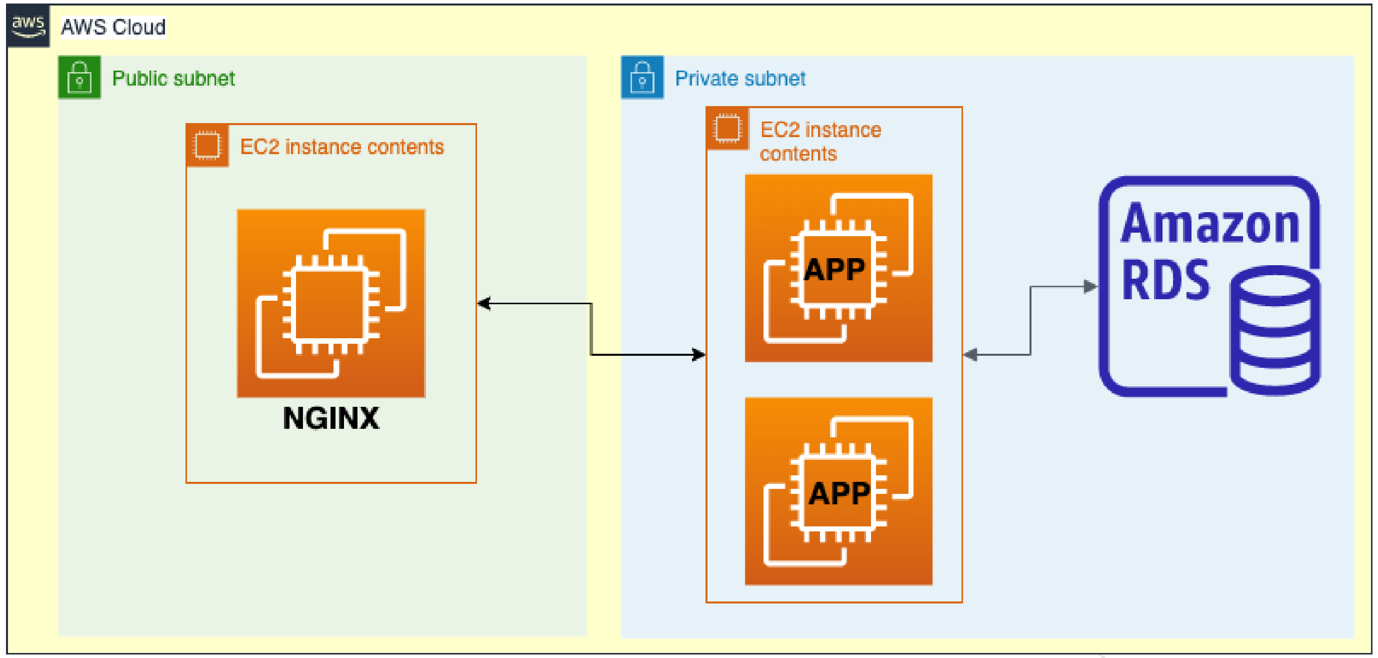
## TASK: TERRAFORM WITH AWS

Goals: Run few VM to present how works  load ballancer

All the previous tasks properly showed the dependencies between the VMs to show the proper working the load balancer. To complete the main task, I had to do:

- Prepare VM to install Terraform and AWS plugin
- Create free account on Amazon to get access to AWS with free tier
- Connect Terraform with AWS
- Prepare Terraform file to create whole struture





HashiCorp Terraform

When building this type of structure, we must remember about dependencies, because some modules are dependent on others. For example, an VM application needs have a configured application and database connection to function properly. To handling MySQL Database i choose RDS and configured it to get a credential data ( endpoint and port ) to get access to it.

```
provider "aws" {
    region = "eu-west-2"
}

module "db" {
    source = "./RDS"
}

module "sg" {
    source = "./SG"
}

module "sg2" {
    source = "./SG2"
}
```

```
resource "aws_instance" "app" {
    ami = "ami-0f540e9f488cfa27d"
    instance_type = "t2.micro"
    count = 2
    security_groups = [module.sg.internal]
    user_data = file("app.sh")
    tags = {
        Name = "app"
    }
}

resource "aws_instance" "lb" {
    ami = "ami-0f540e9f488cfa27d"
    instance_type = "t2.micro"
    security_groups = [module.sg2.external]
    user_data = file("lb.sh")
    depends_on = [aws_instance.app]
    tags = {
        Name = "lb"
    }
}
```

```
resource "aws_eip" "elasticeip" {
    instance = aws_instance.lb.id
}

output "EIP" {
    value = aws_eip.elasticeip.public_ip
}
```

THANK YOU FOR ATTENTION