Infrastructure as Mess

Infrastructure as Code

Infrastructure as Code

Mircea Lungu
DevOps, Maintenance, and Software Evolution @ ITU

Infrastructure?

“Dependencies” that need to be in place for your code to run

- Machines
- Networks
- IP addresses
- Operating systems
- Libraries

Infrastructure as Code

Automating the creation of all dependencies with ‘code’

- Repeatable & reliable builds
- A way of documenting architecture

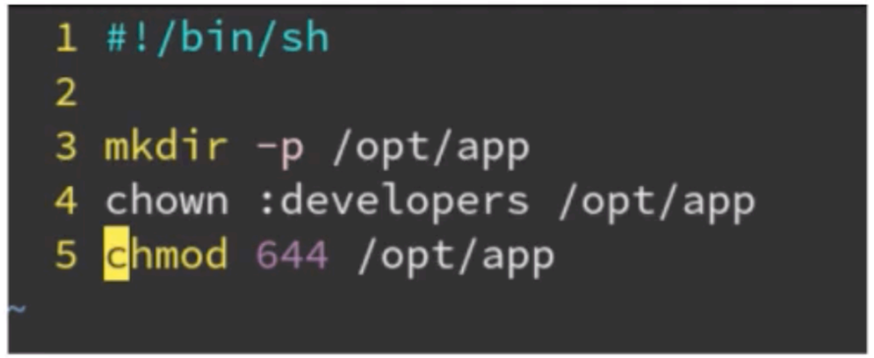
Alternatives?

- **Computers** in your server room + manual installation
- **Clicking around** in the web UI of your cloud provider

as *Imperative* Code

Imperative (e.g. Bash)

- step by step instructions of what to do
- if script fails in the middle have to restart everything from scratch…



As *Declarative* Code

Declarative (e.g. Puppet)

- specify: desired state of the infra
- should continue where it's left off if failure in the middle of the process

```
1 file { '/opt/app':
2    ensure => 'directory',
3    group  => 'developers',
4    mode   => '0644',
5 }
```

Comparing the Two Approaches

A Case Study

1. **Imperative Bash script**
   - Create machine in GCP, and
   - Provision with docker-compose
2. **Declarative using Terraform (from your exercises)**
   - Create multiple machines, and
   - Deploy MiniTwit swarm

Note: not an exact comparison… scripts are doing different things

provisioning > gcloud > create.sh

```
#!/bin/bash

if [ "$#" -ne 3 ]; then
    echo "Illegal number of parameters"
    exit
fi


export MACHINE_NAME=$1
export IPADDRESS=$2
export MACHINE_TYPE=$3


gcloud beta compute --project=mirceas-project \
    instances create $MACHINE_NAME \
    --zone=europe-west4-a \
    --machine-type=$MACHINE_TYPE \
    --subnet=default \
    --address=$IPADDRESS \
    --network-tier=STANDARD \
    --maintenance-policy=MIGRATE \
    --service-account=mirceas-account@developer.gserviceaccount.com \
    --scopes=https://www.googleapis.com/auth/devstorage.read_only,https
    --tags=http-server,https-server \
    --image=debian-10-buster-v20210217 \
    --image-project=debian-cloud \
    --boot-disk-size=60GB \
    --boot-disk-type=pd-ssd \
    --boot-disk-device-name=$MACHINE_NAME \
    --no-shielded-secure-boot \
    --shielded-vtpm \
    --shielded-integrity-monitoring \
    --reservation-affinity=any
```

Bash: Creating the machine

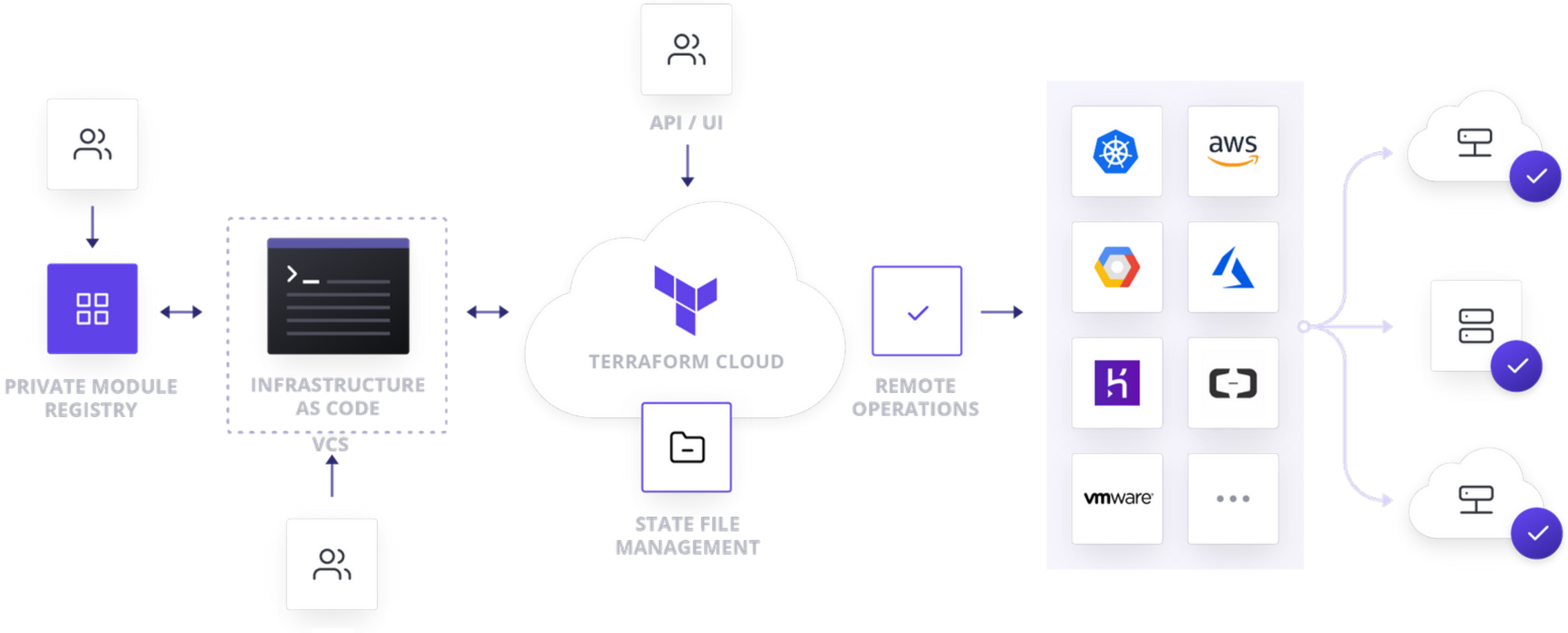
The Good: no need to click around in the UI

The Bad: only works on google cloud

The Ugly: that compute command

Terraform

A “Proxy” for interacting with cloud providers



https://morioh.com/p/e782d802941d

minitwit_swarm_cluster.tf > resource "digitalocean_droplet" "minitwit-swarm-leader"

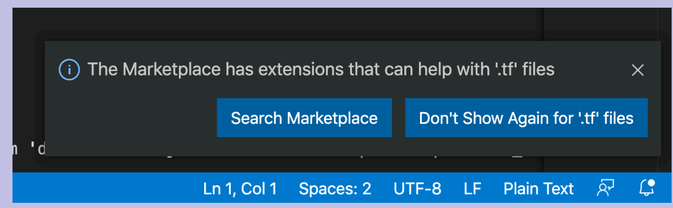
```
1
2   #  _                 _
3   # | | ___  __ _  __| | ___ _ __
4   # | |/ _ \/ _` |/ _` |/ _ \ '__|
5   # | |  __/ (_| | (_| |  __/ |
6   # |_|\___|\__,_|\__,_|\___|_|
7
8   # create cloud vm
9   resource "digitalocean_droplet" "minitwit-swarm-leader" {
10    image = "docker-18-04"
11    name = "minitwit-swarm-leader"
12    region = var.region
13    size = "s-1vcpu-1gb"
14    # add public ssh key so we can access the machine
15    ssh_keys = [digitalocean_ssh_key.minitwit.fingerprint]
16
17    # specify a ssh connection
18    connection {
19      user = "root"
20      host = self.ipv4_address
21      type = "ssh"
22      private_key = file(var.pvt_key)
23      timeout = "2m"
24    }
25
26    provisioner "file" {
27      source = "stack/minitwit_stack.yml"
28      destination = "/root/minitwit_stack.yml"
29    }
30
```

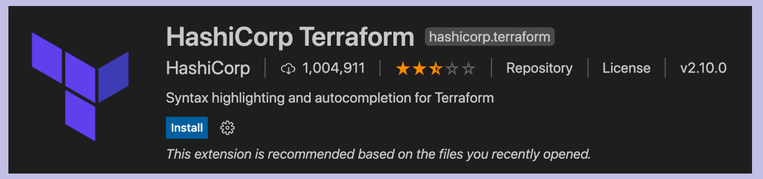
(from your exercises)

Terraform: Creating the machine

DSL: HCL (Hashicorp Config Lang)

- declarative
- “stringly” typed
- greatly improved with IDE tools





Variables in Bash

```
8
9   export MACHINE_NAME=$1
10  export IPADDRESS=$2
11  export MACHINE_TYPE=$3
12
13
14  gcloud beta compute --project=mirceas-project \
15      instances create $MACHINE_NAME \
16      --zone=europe-west4-a \
17      --machine-type=$MACHINE_TYPE \
18      --subnet=default \
19      --address=$IPADDRESS \
20      --network-tier=STANDARD \
21      --maintenance-policy=MIGRATE \
22      --service-account=mirceas-account@developer.gserviceaccount.com \
23      --scopes=https://www.googleapis.com/auth/devstorage.read_only,http
24      --tags=http-server,https-server \
25      --image=debian-10-buster-v20210217 \
26      --image-project=debian-cloud \
27      --boot-disk-size=60GB \
28      --boot-disk-type=pd-ssd \
29      --boot-disk-device-name=$MACHINE_NAME \
30      --no-shielded-secure-boot \
31      --shielded-vtpm \
32      --shielded-integrity-monitoring \
33      --reservation-affinity=any
```

parameterizing our script!

it is still a pain to always remember to type the CLI arguments; might be nicer if we could just load these variables from a file

minitwit_swarm_cluster.tf M | minitwit.auto.tfvars

minitwit.auto.tfvars

```
1   # do region
2   region = "fra1"
3
4   # ssh key
5   pub_key = "ssh_key/terraform.pub"
6   pvt_key = "ssh_key/terraform"
7
```

minitwit_swarm_cluster.tf > resource "digitalocean_droplet" "minitwit-swarm-l

```
1
2   #  _                 _
3   # | | ___  __ _  __| | ___ _ __
4   # | |/ _ \/ _` |/ _` |/ _ \ '__|
5   # | |  __/ (_| | (_| |  __/ |
6   # |_|\___|\__,_|\__,_|\___|_|
7
8   # create cloud vm
9   resource "digitalocean_droplet" "minitwit-swarm-leader" {
10    image = "docker-18-04"
11    name = "minitwit-swarm-leader"
12    region = var.region
13    size = "s-1vcpu-1gb"
14    # add public ssh key so we can access the machine
15    ssh_keys = [digitalocean_ssh_key.minitwit.fingerprint]
16
```

Variables in Terraform

- **Envvars** prefixed with TF_VAR_
- Automatically loads a number of variable definitions files if they are present:
  - Files named exactly **terraform.tfvars**
  - Any files with names ending in **.auto.tfvars**

https://www.terraform.io/docs/language/values/variables.html

Bash: Waiting for the machine to boot up

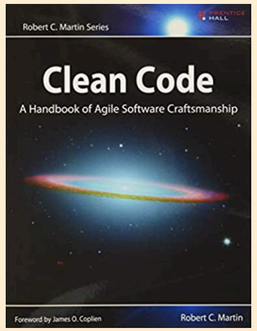
```
# https://serverfault.com/questions/152795/linux-command-to-wait-for-a-ssh-server-to-be-up
while ! gcloud beta compute ssh mlun@$MACHINE_NAME --command="pwd"
do
    echo "Waiting to be able to ssh..."
done
```

If code, then comments…

Intention and Clarification

A comment is always useful when intention is expressed. It is not important to comment what we have done in the code, because the reader can see the code itself. It is more important to explain what we wanted to do in the code.

There are cases when we cannot express exactly what our intention is. Because of this we need to add comments that add more clarification and explain why we didn't take a specific action. Maybe there is a bug in an external library that we had to avoid or the client had an odd request.



Terraform: Waiting for the other machine to come up

Dependencies between resources result in a partial order of creation

```
#  _ __ ___   __ _ _ __   __ _  __ _  ___ _ __
# | '_ ` _ \ / _` | '_ \ / _` |/ _` |/ _ \ '__|
# | | | | | | (_| | | | | (_| | (_| |  __/ |
# |_| |_| |_|\__,_|_| |_|\__,_|\__, |\___|_|
#                                 |___/

# create cloud vm
resource "digitalocean_droplet" "minitwit-swarm-manager" {
  # create managers after the leader
  depends_on = [digitalocean_droplet.minitwit-swarm-leader]
```

Bash: Little adjustments to the newly created machine

```
# required on a default debian 10 image from google cloud
# because the locale is broken otherwise
echo gcloud beta compute ssh mlun@$MACHINE_NAME --command="\"sudo apt-get install locales-all -y\""
gcloud beta compute ssh mlun@$MACHINE_NAME --command="sudo apt-get install locales-all -y"
```

- this is cool! otherwise, six months later when we have to do this, we'll have to google again how to do it
- **How come the script can ssh to the machine?**
  - I've already uploaded the public key in the GCP UI
  - by clicking and typing…

Uploading ssh keys so we can later connect

minitwit_swarm_cluster.tf > ...

```
1
2   #  _                 _
3   # | | ___  __ _  __| | ___ _ __
4   # | |/ _ \/ _` |/ _` |/ _ \ '__|
5   # | |  __/ (_| | (_| |  __/ |
6   # |_|\___|\__,_|\__,_|\___|_|
7
8   # create cloud vm
9   resource "digitalocean_droplet" "minitwit-swarm-leader" {
10    image = "docker-18-04"
11    name = "minitwit-swarm-leader"
12    region = var.region
13    size = "s-1vcpu-1gb"
14    # add public ssh key so we can access the machine
15    ssh_keys = [digitalocean_ssh_key.minitwit.fingerprint]
16
```

2

ssh_key.tf > ...

1

```
1
2   # add the ssh key
3   resource "digitalocean_ssh_key" "minitwit" {
4     name = "minitwit"
5     public_key = file(var.pub_key)
6   }
7
```

Note: the ssh keys must be already created (follow your instructions and you'll be fine)

https://registry.terraform.io/providers/digitalocean/digitalocean/latest/docs/resources/ssh_key

Bash: Provisioning Docker

```
# docker tools

  # 1. Add Docker's official GPG key:
gcloud beta compute ssh mlun@$MACHINE_NAME --command='curl -fsSL https://download.docker.com/linux/debian/gpg |


  # 2. set up the stable repository
gcloud beta compute ssh mlun@$MACHINE_NAME --command='echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docke

  # 3. Update the apt package index, and install the latest version of Docker
gcloud beta compute ssh mlun@$MACHINE_NAME --command='sudo apt-get update'
gcloud beta compute ssh mlun@$MACHINE_NAME --command='sudo apt-get install docker-ce docker-ce-cli containerd.i


# docker-compose
gcloud beta compute ssh mlun@$MACHINE_NAME --command='sudo curl -L "https://github.com/docker/compose/releases/
gcloud beta compute ssh mlun@$MACHINE_NAME --command='sudo chmod +x /usr/local/bin/docker-compose'


# add my user to docker group
gcloud beta compute ssh mlun@$MACHINE_NAME --command='sudo usermod -aG docker mlun'
```

Line of text that is too long…

Boilerplate?

As a programmer … I do not like boilerplate

```
17    # specify a ssh connection
18    connection {
19      user = "root"
20      host = self.ipv4_address
21      type = "ssh"
22      private_key = file(var.pvt_key)
23      timeout = "2m"
24    }
25
26    provisioner "file" {
27      source = "stack/minitwit_stack.yml"
28      destination = "/root/minitwit_stack.yml"
29    }
30
31    provisioner "remote-exec" {
32      inline = [
33        # allow ports for docker swarm
34        "ufw allow 2377/tcp",
35        "ufw allow 7946",
36        "ufw allow 4789/udp",
37        # ports for apps
38        "ufw allow 80",
39        "ufw allow 8080",
40        "ufw allow 8888",
41
42        # initialize docker swarm cluster
43        "docker swarm init --advertise-addr ${self.ipv4_address}"
44      ]
45    }
```

Provisioners

actions on the local machine or on a remote machine in order to prepare servers or other infrastructure objects for service

file: copying things from local to remote

remote-exec: running things remotely

- in this case, setting up the firewall
- and initializing the swarm

```
# save the worker join token
provisioner "local-exec" {
  command = <<EOS
    ssh -o 'StrictHostKeyChecking no'
        root@${self.ipv4_address}
        -i ssh_key/terraform
        'docker swarm join-token worker -q' > temp/worker_token
    EOS
}

# save the manager join token
provisioner "local-exec" {
  command = <<EOS
    ssh -o 'StrictHostKeyChecking no'
        root@${self.ipv4_address}
        -i ssh_key/terraform
        'docker swarm join-token manager -q' > temp/manager_token
    EOS
}
```

local-exec: run things locally

- we can use it to get data from the remote to a local folder
- also you can use heredoc in it!

Heredoc String: https://en.wikipedia.org/wiki/Here_document

Terraform: Provisioners

Last resort solution: not “first class” in the platform

Provisioner *tools*

- cloud-init — standard provisioning tool (example script)
- packer — by hashicorp - for creating machine images (example script)
- ansible

More on Variables: self

An HCL Keyword

- **Expressions in provisioner** blocks cannot **refer to their parent resource** by name.
- Instead, they can use the special *self* object.

```
# create cloud vm
resource "digitalocean_droplet" "minitwit-swarm-leader" {
  image = "docker-18-04"
  name = "minitwit-swarm-leader"
  region = var.region
  size = "s-1vcpu-1gb"
  # add public ssh key so we can access the machine
  ssh_keys = [digitalocean_ssh_key.minitwit.fingerprint]

  # specify a ssh connection
  connection {
    user = "root"
    host = self.ipv4_address
    type = "ssh"
    private_key = file(var.pvt_key)
    timeout = "2m"
  }
```

```
#  _ __ ___   __ _ _ __   __ _  __ _  ___ _ __
# | '_ ` _ \ / _` | '_ \ / _` |/ _` |/ _ \ '__|
# | | | | | | (_| | | | | (_| | (_| |  __/ |
# |_| |_| |_|\__,_|_| |_|\__,_|\__, |\___|_|
#                                 |___/

# create cloud vm
resource "digitalocean_droplet" "minitwit-swarm-manager" {
  # create managers after the leader
  depends_on = [digitalocean_droplet.minitwit-swarm-leader]

  # number of vms to create
  count = 2

  image = "docker-18-04"
  name = "minitwit-swarm-manager-${count.index}"
  region = var.region
  size = "s-1vcpu-1gb"
  # add public ssh key so we can access the machine
  ssh_keys = [digitalocean_ssh_key.minitwit.fingerprint]
```

More Terraform

- dependencies between resources
- creating multiple resources

Swarm manager nodes use the Raft Consensus Algorithm to manage the swarm state.
Raft is leader-based, which means only the leader node has the ability to change the state of the swarm. If you send a request that requires a state change to a follower node, this request will be forwarded to the leader node.

Joining the swarm as a manager

```
provisioner "file" {
  source = "temp/manager_token"
  destination = "/root/manager_token"
}

provisioner "remote-exec" {
  inline = [
    # allow ports for docker swarm
    "ufw allow 2377/tcp",
    "ufw allow 7946",
    "ufw allow 4789/udp",
    # ports for apps
    "ufw allow 80",
    "ufw allow 8080",
    "ufw allow 8888",

    # join swarm cluster as managers
    "docker swarm join --token $(cat manager_token) ${digitalocean_droplet.minitwit-swarm-leader.ipv4_address}"
  ]
}
```

```
#  _                 _
# | | ___  __ _  __| | ___ _ __
# | |/ _ \/ _` |/ _` |/ _ \ '__|
# | |  __/ (_| | (_| |  __/ |
# |_|\___|\__,_|\__,_|\___|_|

# create cloud vm
resource "digitalocean_droplet" "minitwit-swarm-leader" {
  image = "docker-18-04"
  name = "minitwit-swarm-leader"
```

Referring to resources by ${TYPE.NAME.PROPERTY}

Terraform: Planning

- Terraform
  - Tracks current state of the infra
  - **Plans** to bring the actual state of the infra in sync with the desired state
- If you remove a device from the template, terraform will remove it
- Nice for incremental execution of plans
- Can’t easily do this with shell scripting

Planning: So what if we want to manage the infra with the team?

Everybody needs to know the state of the plan and infra

- You can save the state … in the cloud!
- Backend configuration
- By default state is saved locally

bootstrap.sh

```
15
16  echo -e "\n--> Initializing terraform\n"
17  # initialize terraform
18  terraform init \
19      -backend-config "bucket=$SPACE_NAME" \
20      -backend-config "key=$STATE_FILE" \
21      -backend-config "access_key=$AWS_ACCESS_KEY_ID" \
22      -backend-config "secret_key=$AWS_SECRET_ACCESS_KEY"
23
```

secrets_template

```
1  export TF_VAR_do_token=
2  export SPACE_NAME=
3  export STATE_FILE=
4  export AWS_ACCESS_KEY_ID=
5  export AWS_SECRET_ACCESS_KEY=
6
```

https://www.terraform.io/docs/language/settings/backends/s3.html

Other Resources that Are not in The Example

Setting up DNS records

```
resource "digitalocean_record" "www" {
  domain = var.domain_name
  type   = "A"
  name   = "@"
  value  = digitalocean_droplet.web.ipv4_address
}
```

```
gcloud dns record-sets \
    transaction add  "104.196.108.178"\
    --name "example.com."\
    --ttl 60 --type "A"\
    --project ruptureofthemundaneplane\
    --zone zone-cubalibre
```

Terrafrom: Modules

- Help with reusing configurations
- Can be parameterized
- When you terraform apply you read all the .tf files in the current folder (module)
- Too advanced for our purposes

Identity and Access Management (IAM)

- Securely control access to cloud services and resources
- For users, groups, and applications
- Different cloud providers provide different IAM solutions

Principles for IaC

- **Version** your infra code
- **Test** your infra code
- Write **beautiful** infra code
  - No write huge plan files
  - Comments explaining the “why”

To think about

Simulator stops several weeks before the exam

Can you also stop your systems so you don’t pay…

… while being able to demo your system at the exam?

Can you easily bring up your infra & system on the day of the exam?