Infrastructure as Code

Mircea Lungu
DevOps, Maintenance, and Software Evolution @ ITU

Lecture Feedback

https://forms.gle/fsoFxjwNg4ZTNXvt5

What is infrastructure?

- **all the “stuff” that need to be in place for your code to run**
- these things are dependecies of your system
  - machine
  - networks
  - ip addresses
  - operating system
  - libraries

Infrastructure as …

(a brief history in three steps)

- **computers** (in your basement or data center)
- **clicks** (in the ui of your cloud provider)
- **code** (in the VCS)

Infrastructure as Code

- a way to obtain repeatable + reliable builds
- alternative to: manual (optional with “paper” notes)
    - not automated (obviously!)
    - not repeatable
    - prone to error
    - https://youtu.be/so-xL0nmn0o?t=341

Code: Imperative vs. Declarative

- **Imperative (e.g. Bash)**
  - step by step instructions of what to do
  - if script fails in the middle have to restart everything from scratch…
- **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

```
#!/bin/sh

mkdir -p /opt/app
chown :developers /opt/app
chmod 644 /opt/app
```

```
file { '/opt/app':
  ensure => 'directory',
  group  => 'developers',
  mode   => '0644',
}
```

A Case Study

of comparing two approaches

- **Imperative script (simple deployment by your's truly)**
  - Create machine in GCP and provision with docker-compose

- **Declarative using Terraform (from your exercises)**
  - Create multiple machines and deploy MinitTwit swarm

- Note: not exactly a comparison… the two scripts are not doing the same thing

provisioning > gcloud > create.sh

```bash
#!/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 Nice: no need to click around in the UI
- That’s a mouthfull of a command
- con: Only works on google cloud

gcloud beta compute instances create

Send feedback

NAME

gcloud beta compute instances create - create Compute Engine virtual machine instances

SYNOPSIS

```
gcloud beta compute instances create INSTANCE_NAMES [INSTANCE_NAMES ...]
  [--accelerator =[count = COUNT],[type = TYPE]] [--async] [--no-boot-disk-auto-delete]
  [--boot-disk-device-name = BOOT_DISK_DEVICE_NAME]
  [--boot-disk-provisioned-iops = BOOT_DISK_PROVISIONED_IOPS]
  [--boot-disk-size = BOOT_DISK_SIZE] [--boot-disk-type = BOOT_DISK_TYPE] [--can-ip-forward]
  [--confidential-compute] [--create-disk =[PROPERTY = VALUE,...]] [--csek-key-file = FILE]
  [--deletion-protection] [--description = DESCRIPTION] [--disk =[auto-delete = AUTO-DELETE],
  [boot = BOOT],[device-name = DEVICE-NAME],[mode = MODE],[name = NAME],[scope = SCOPE]]
  [--enable-display-device] [--[no-]enable-nested-virtualization]
  [--erase-windows-vss-signature] [--hostname = HOSTNAME] [--labels =[KEY = VALUE,...]]
  [--local-ssd =[device-name = DEVICE-NAME],[interface = INTERFACE]]
  [--machine-type = MACHINE_TYPE] [--maintenance-policy = MAINTENANCE_POLICY]
  [--metadata = KEY = VALUE,[KEY = VALUE,...]] [--metadata-from-file = KEY = LOCAL_FILE_PATH,[...]]
  [--min-cpu-platform = PLATFORM] [--min-node-cpu = MIN_NODE_CPU] [--network = NETWORK]
  [--network-interface =[PROPERTY = VALUE,...]] [--network-performance-configs =[PROPERTY = VALUE,...]]
  [--network-tier = NETWORK_TIER] [--post-key-revocation-action-type = POLICY] [--preemptible]
  [--private-ipv6-google-access-type = PRIVATE_IPV6_GOOGLE_ACCESS_TYPE]
  [--private-network-ip = PRIVATE_NETWORK_IP] [--no-require-csek-key-create]
  [--resource-policies =[RESOURCE_POLICY,...]] [--no-restart-on-failure]
  [--shielded-integrity-monitoring] [--shielded-secure-boot] [--shielded-vtpm]
  [--source-instance-template = SOURCE_INSTANCE_TEMPLATE]
  [--source-machine-image = SOURCE_MACHINE_IMAGE] [--source-machine-image-csek-key-file = FILE]
  [--subnet = SUBNET] [--tags = TAG,[TAG,...]] [--zone = ZONE] [--address = ADDRESS | --no-address]
  [--boot-disk-kms-key = BOOT_DISK_KMS_KEY : --boot-disk-kms-keyring = BOOT_DISK_KMS_KEYRING
   --boot-disk-kms-location = BOOT_DISK_KMS_LOCATION
   --boot-disk-kms-project = BOOT_DISK_KMS_PROJECT] [--custom-cpu = CUSTOM_CPU
   --custom-memory = CUSTOM_MEMORY : --custom-extensions --custom-vm-type = CUSTOM_VM_TYPE]
  [--image-family-scope = IMAGE_FAMILY_SCOPE --image-project = IMAGE_PROJECT --image = IMAGE |
   --image-family = IMAGE_FAMILY | --source-snapshot = SOURCE_SNAPSHOT] [--node = NODE |
   --node-affinity-file = NODE_AFFINITY_FILE | --node-group = NODE_GROUP] [--public-ptr |
   --no-public-ptr] [--public-ptr-domain = PUBLIC_PTR_DOMAIN | --no-public-ptr-domain]
  [--reservation = RESERVATION --reservation-affinity = RESERVATION_AFFINITY; default="any"]
  [--scopes =[SCOPE,...] | --no-scopes] [--service-account = SERVICE_ACCOUNT |
   --no-service-account] [GCLOUD_WIDE_FLAG ...]
```

You can imagine that taking all these as command line arguments is fun… up to a point.

Does it remind you of a bad smell form the famous Clean Code book?

gcloud beta compute instances create

Send feedback

NAME

gcloud beta compute instances create - create Compute Engine virtual machine instances

SYNOPSIS

```
gcloud beta compute instances create INSTANCE_NAMES [INSTANCE_NAMES ...]
  [--accelerator =[count = COUNT],[type = TYPE]] [--async] [--no-boot-disk-auto-delete]
  [--boot-disk-device-name = BOOT_DISK_DEVICE_NAME]
  [--boot-disk-provisioned-iops = BOOT_DISK_PROVISIONED_IOPS]
  [--boot-disk-size = BOOT_DISK_SIZE] [--boot-disk-type = BOOT_DISK_TYPE] [--can-ip-forward]
  [--confidential-compute] [--create-disk =[PROPERTY = VALUE,...]] [--csek-key-file = FILE]
  [--deletion-protection] [--description = DESCRIPTION] [--disk =[auto-delete = AUTO-DELETE],
  [boot = BOOT],[device-name = DEVICE-NAME],[mode = MODE],[name = NAME],[scope = SCOPE]]
  [--enable-display-device] [--[no-]enable-nested-virtualization]
  [--erase-windows-vss-signature] [--hostname = HOSTNAME] [--labels =[KEY = VALUE,...]]
  [--local-ssd =[device-name = DEVICE-NAME],[interface = INTERFACE]]
  [--machine-type = MACHINE_TYPE] [--maintenance-policy = MAINTENANCE_POLICY]
  [--metadata = KEY = VALUE,[KEY = VALUE,...]] [--metadata-from-file = KEY = LOCAL_FILE_PATH,[...]]
  [--min-cpu-platform = PLATFORM] [--min-node-cpu = MIN_NODE_CPU] [--network = NETWORK]
  [--network-interface =[PROPERTY = VALUE,...]] [--network-performance-configs =[PROPERTY = VALUE,...]
  [--network-tier = NETWORK_TIER] [--post-key-revocation-action-type = POLICY] [--preemptible]
  [--private-ipv6-google-access-type = PRIVATE_IPV6_GOOGLE_ACCESS_TYPE]
  [--private-network-ip = PRIVATE_NETWORK_IP] [--no-require-csek-key-create]
  [--resource-policies =[RESOURCE_POLICY,...]] [--no-restart-on-failure]
  [--shielded-integrity-monitoring] [--shielded-secure-boot] [--shielded-vtpm]
  [--source-instance-template = SOURCE_INSTANCE_TEMPLATE]
  [--source-machine-image = SOURCE_MACHINE_IMAGE] [--source-machine-image-csek-key-file = FILE]
  [--subnet = SUBNET] [--tags = TAG,[TAG,...]] [--zone = ZONE] [--address = ADDRESS | --no-address]
  [--boot-disk-kms-key = BOOT_DISK_KMS_KEY : --boot-disk-kms-keyring = BOOT_DISK_KMS_KEYRING
   --boot-disk-kms-location = BOOT_DISK_KMS_LOCATION
   --boot-disk-kms-project = BOOT_DISK_KMS_PROJECT] [--custom-cpu = CUSTOM_CPU
   --custom-memory = CUSTOM_MEMORY : --custom-extensions --custom-vm-type = CUSTOM_VM_TYPE]
  [--image-family-scope = IMAGE_FAMILY_SCOPE --image-project = IMAGE_PROJECT --image = IMAGE |
   --image-family = IMAGE_FAMILY | --source-snapshot = SOURCE_SNAPSHOT] [--node = NODE |
   --node-affinity-file = NODE_AFFINITY_FILE | --node-group = NODE_GROUP] [--public-ptr |
   --no-public-ptr] [--public-ptr-domain = PUBLIC_PTR_DOMAIN | --no-public-ptr-domain]
  [--reservation = RESERVATION --reservation-affinity = RESERVATION_AFFINITY; default="any"]
  [--scopes =[SCOPE,...] | --no-scopes] [--service-account = SERVICE_ACCOUNT |
   --no-service-account] [GCLOUD_WIDE_FLAG ...]
```

You can imagine that taking all these parameters as command line parameters is fun only up to a point.

provisioning > gcloud > create.sh

```bash
#!/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
```

Different Commands for Different Providers

- If I want to deploy to another provider I need to install their CL tool
- Also they'll have different parameters

Terraform

A “Proxy” for interacting with cloud providers

https://morioh.com/p/e782d802941d

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

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

# 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"
  }

  provisioner "file" {
    source = "stack/minitwit_stack.yml"
    destination = "/root/minitwit_stack.yml"
  }

```

Terraform: A declarative alternative

(from your exercises)

- Dedicated DSL: HCL
  - “stringly” typed
  - greatly improved with IDE tools

modern IDEs can make up for the limits of the language quite a bit…

Even so…

IDE support can only do so much…

- e.g. unfinished string => all file is highlighted…

minitwit_swarm_cluster.tf 9+, M

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
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
```

From Uncle Mircea's "Imperative" IaC

```
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

Variables in Terraform

minitwit_swarm_cluster.tf M | minitwit.auto.tfvars

minitwit.auto.tfvars

```
# do region
region = "fra1"

# ssh key
pub_key = "ssh_key/terraform.pub"
pvt_key = "ssh_key/terraform"

```

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

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

# 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]

```

- **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

```
# 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
```

Waiting for the server to come up

Also, where there is code, there will be 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.

Waiting for the other server to come up

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

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

- dependencies between resources result in a partial order of creation

```
# 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"
```

Little fixes … that we have to run on the newly created machine

- 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 that 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

Boilerplate code!

Line of text that is too long…

```
# 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'
```

Provisioning Docker

by now you realize that there has to be a better way to execute things remotely that requires less 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
27
28
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  }
```

Connection info

Provisioners

- **Specifying the Connection Info**
  - **Expressions in provisioner** blocks cannot **refer to their parent resource** by name.
  - Instead, they can use the special self object.
- **First Type of Provisioner (remote-exec): Running things remotely**
  - in this case, setting up the firewall
  - and initializing the swarm

```
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
32
33
34
35
36
37
38
39
40
41
42
43
44
45
```

Provisioners

- **Second Type: file**
  - Copying things from local to remote

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

provisioner "file" {
  source = "stack/minitwit_stack.yml"
  destination = "/root/minitwit_stack.yml"
}
```

Provisioners

- **Second Type: file**
  - Copying things from local to remote
- Missing Type: *reverse-provisioner* - getting data from the resource to the local machine

```
# 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
}
```

Provisioners

- **Third Type: local-exec**
  - that we can use to 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

Provisioners

- Terraform recommends that you don't use provisioners :)
- Because they're not "first class" in the platform
- Instead use specific provisioner tools (e.g.
  - 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 Specifications

- **dependencies between resources**
- **creating multiple resources**
- (reminder leader is the manager's manager ;)

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 their TYPE.NAME.PROPERTY

Planning >> more than shell scripting…

- 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

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

Everybody neds to know the state of the plan and infra

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

bootstrap.sh

```
echo -e "\n--> Initializing terraform\n"
# initialize terraform
terraform init \
    -backend-config "bucket=$SPACE_NAME" \
    -backend-config "key=$STATE_FILE" \
    -backend-config "access_key=$AWS_ACCESS_KEY_ID" \
    -backend-config "secret_key=$AWS_SECRET_ACCESS_KEY"
```

secrets_template

```
export TF_VAR_do_token=
export SPACE_NAME=
export STATE_FILE=
export AWS_ACCESS_KEY_ID=
export AWS_SECRET_ACCESS_KEY=
```

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

Miscelaneous Notes

- When you terraform apply you read all the .tf files in the current folder (module)
- There’s a concept of modules
  - too advanced for our purpose here
  - it helps with reusing configurations
  - they can be parameterized

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
```

No Silver Bullet
—Essence and Accident in Software Engineering

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity.

Abstract[1]

All software construction involves essential tasks, the fashioning of the complex conceptual structures that compose the abstract software entity, and accidental tasks, the representation of these abstract entities in programming languages and the mapping of these onto machine languages within space and speed constraints. Most of the big past gains in software productivity have come from removing artificial barriers that have made the accidental tasks inordinately hard, such as severe hardware constraints, awkward programming languages, lack of machine time. How much of what software

Further Topic: IAM

- IAM - **Identity and Access Management**
  - securely control access to cloud services and resources
  - for users, groups, and applications
- Different cloud providers provide different IAM solutions!
  - e.g. comparison of Google vs. AWS

Thought Experiment

- We are going to stop the simulator for several weeks before the exam
- It would make sense to stop also your systems, so you don’t pay
- It would be nice if you could demo your system at the exam
- so….
- Could you bring up your infra on the day of the exam?

Some Principles for IaC

- Version your IaC
- Don’t write a huge plan file
- You still need comments/documentation for the “why”
- Continuously test your IaC

Production-grade infrastructure checklist, part 1/4

Task	Description	Example tools
Install	Install the software binaries and all dependencies.	Bash, Chef, Ansible, Puppet
Configure	Configure the software at runtime: e.g., configure port settings, file paths, users, leaders, followers, replication, etc.	Bash, Chef, Ansible, Puppet
Provision	Provision the infrastructure: e.g., EC2 Instances, load balancers, network topology, security groups, IAM permissions, etc.	Terraform, CloudFormation
Deploy	Deploy the service on top of the infrastructure. Roll out updates with no downtime: e.g., blue-green, rolling, canary deployments.	Scripts, Orchestration tools (ECS, K8S, Nomad)

Production-grade infrastructure checklist, part 2/4

Task	Description	Example tools
Security	Encryption in transit (TLS) and on disk, authentication, authorization, secrets management, server hardening.	ACM, EBS Volumes, Cognito, Vault, CiS
Monitoring	Availability metrics, business metrics, app metrics, server, metrics, events, observability, tracing, alerting.	CloudWatch, DataDog, New Relic, Honeycomb
Logs	Rotate logs on disk. Aggregate log data to a central location.	CloudWatch Logs, ELK, Sumo Logic, Papertrail
Backup and restore	Make backups of DBs, caches, and other data on a scheduled basis. Replicate to separate region/account.	RDS, ElastiCache, ec2-snapper, Lambda

Production-grade infrastructure checklist, part 3/4

Task	Description	Example tools
Networking	VPCs, subnets, static and dynamic IPs, service discovery, service mesh, firewalls, DNS, SSH access, VPN access.	EIPs, ENIs, VPCs, NACLs, SGs, Route 53, OpenVPN
High availability	Withstand outages of individual processes, EC2 Instances, services, Availability Zones, and regions.	Multi AZ, multi-region, replication, ASGs, ELBs
Scalability	Scale up and down in response to load. Scale horizontally (more servers) and/or vertically (bigger servers).	ASGs, replication, sharding, caching, divide and conquer
Performance	Optimize CPU, memory, disk, network, GPU and usage. Query tuning. Benchmarking, load testing, profiling.	Dynatrace, valgrind, VisualVM, ab, Jmeter

Production-grade infrastructure checklist, part 4/4

Task	Description	Example tools
Cost optimization	Pick proper instance types, use spot and reserved instances, use auto scaling, nuke unused resources	ASGs, spot instances, reserved instances
Documentation	Document your code, architecture, and practices. Create playbooks to respond to incidents.	READMEs, wikis, Slack
Tests	Write automated tests for your infrastructure code. Run tests after every commit and nightly.	Terratest

Closing Thoughts

- DevOpser requires a lot of knowledge / handles a lot of essential complexity => A lot of ground to cover and knowledge to gather

- Know the rules so you know when to break them
  - e.g. machines cattle not pets vs. lichess
    - billion matches / >20k concurrent users, etc.
    - list of lichess machines,
    - lichess talk