# Solution M4: Terraform Fundamentals

The following is just one possible solution. There may be other more effective or faster solutions

## Task 1

Let us assume that we have a **Docker** host with IP address set to **192.168.99.100**

You can check the provided **Vagrantfile**

First, we must make sure that the project files are copied on the **Docker** host. For this solution, the files will in the **/project/site** folder of the **Docker** host

We could prepare a **main.tf** file with the following content:

terraform {

  required\_providers {

    docker = {

      source  = "kreuzwerker/docker"

    }

  }

}

provider "docker" {

  host = "tcp://192.168.99.100:2375/"

}

# One way to deliver the project's files to the Docker host

resource "null\_resource" "files" {

  triggers = {

    web\_image\_id = docker\_image.img-web.id

  }

  provisioner "remote-exec" {

    inline = [

      "sudo rm -rf /project || true",

      "sudo mkdir /project || true",

      "cd /project",

      "sudo git clone https://github.com/shekeriev/bgapp",

    ]

    connection {

      type     = "ssh"

      user     = "vagrant"

      password = "vagrant"

      host     = "192.168.99.100"

    }

  }

}

resource "docker\_network" "net-docker" {

  name = "app-net"

}

resource "docker\_image" "img-web" {

  name = "shekeriev/bgapp-web"

}

resource "docker\_image" "img-db" {

  name = "shekeriev/bgapp-db"

}

resource "docker\_container" "con-web" {

  name  = "web"

  image = docker\_image.img-web.latest

  ports {

    internal = 80

    external = 80

  }

  # Files are on the Docker host - either copied manually or by using some sort of automation

  volumes {

    host\_path      = "/project/bgapp/web"

    container\_path = "/var/www/html"

    read\_only      = true

  }

  networks\_advanced {

    name = "app-net"

  }

  depends\_on = [

    null\_resource.files,

  ]

}

resource "docker\_container" "con-db" {

  name  = "db"

  image = docker\_image.img-db.latest

  env   = ["MYSQL\_ROOT\_PASSWORD=12345"]

  networks\_advanced {

    name = "app-net"

  }

}

Then we may follow the standard procedure

*Keep in mind that you should have* ***git*** *binary installed on the machine where* ***Docker*** *instance is*

*Should we want to use* ***Docker*** *in a virtual machine, we can use the provided* ***Vagrantfile*** *and spin up the machine*

***vagrant up***

First initialize the provider with

**terraform init**

Then check the plan with

**terraform plan**

And finally, run the solution with

**terraform apply**

*If our* ***Docker*** *host is based on earlier versions of CentOS 8.x, we may have to take care of the added network. We may have to add it to the trusted zone of the firewall*

*Alternatively, we can create the network upfront (not in the* ***Terraform*** *file) and then just link it to the containers*

Once we are done, we can clean up with

**terraform destroy**

*And if we used the* ***Docker*** *from the* ***Vagrantfile****, we should remove the instance as well*

***vagrant destroy --force***

## Task 2 (AWS)

Our AMI of choice will be **Amazon Linux 2023** based image

Knowing the specifics of the image and that we must create two VMs, we can prepare two shell provision scripts

The first one will be named **provision-web.sh** and will have the following content:

#!/bin/bash

# Install the required packages

sudo dnf install -y httpd php8.4 php8.4-mysqlnd git

# Switch the SELinux to permissive mode

sudo setenforce 0

# Enable and start the Apache2 service

sudo systemctl enable --now httpd

# Download the project

git clone https://github.com/shekeriev/bgapp

# Copy the files related to the web application

sudo cp ~/bgapp/web/\* /var/www/html

# Substitute MariaDB related connection parameters

sudo sed -i 's/db/10.10.10.101/g' /var/www/html/config.php

The second one will be named **provision-db.sh** and will have the following content:

#!/bin/bash

# Install the required packages

sudo dnf install -y mariadb1011-client-utils mariadb1011-server git

# Enable and start the MariaDB service

sudo systemctl enable --now mariadb

# Download the project

git clone https://github.com/shekeriev/bgapp

# Install the database required for the web application

sudo mysql -u root < ~/bgapp/db/db\_setup.sql

The last file in the solution is the **main.tf** file which will have the following content:

provider "aws" {

  access\_key = "<ACCESS-KEY>"

  secret\_key = "<SECRET-KEY>"

  region     = "eu-central-1"

}

resource "aws\_vpc" "hw-vpc" {

  cidr\_block           = "10.10.0.0/16"

  enable\_dns\_hostnames = true

  enable\_dns\_support   = true

  tags = {

    Name = "HW-VPC"

  }

}

resource "aws\_internet\_gateway" "hw-igw" {

  vpc\_id = aws\_vpc.hw-vpc.id

  tags = {

    Name = "HW-IGW"

  }

}

resource "aws\_route\_table" "hw-prt" {

  vpc\_id = aws\_vpc.hw-vpc.id

  route {

    cidr\_block = "0.0.0.0/0"

    gateway\_id = aws\_internet\_gateway.hw-igw.id

  }

  tags = {

    Name = "HW-PUBLIC-PRT"

  }

}

resource "aws\_subnet" "hw-snet" {

  vpc\_id                  = aws\_vpc.hw-vpc.id

  cidr\_block              = "10.10.10.0/24"

  map\_public\_ip\_on\_launch = true

  tags = {

    Name = "HW-SUB-NET"

  }

}

resource "aws\_route\_table\_association" "hw-prt-assoc" {

  subnet\_id      = aws\_subnet.hw-snet.id

  route\_table\_id = aws\_route\_table.hw-prt.id

}

resource "aws\_security\_group" "hw-pub-sg" {

  name        = "hw-pub-sg"

  description = "HW Public SG"

  vpc\_id      = aws\_vpc.hw-vpc.id

  ingress {

    from\_port   = 22

    to\_port     = 22

    protocol    = "tcp"

    cidr\_blocks = ["0.0.0.0/0"]

  }

  ingress {

    from\_port   = 80

    to\_port     = 80

    protocol    = "tcp"

    cidr\_blocks = ["0.0.0.0/0"]

  }

  ingress {

    from\_port   = 3306

    to\_port     = 3306

    protocol    = "tcp"

    cidr\_blocks = ["10.10.10.0/24"]

  }

  egress {

    from\_port   = 0

    to\_port     = 0

    protocol    = "-1"

    cidr\_blocks = ["0.0.0.0/0"]

  }

}

resource "aws\_network\_interface" "hw-web-net" {

  subnet\_id       = aws\_subnet.hw-snet.id

  private\_ips     = ["10.10.10.100"]

  security\_groups = [aws\_security\_group.hw-pub-sg.id]

  tags = {

    Name = "HW-WEB-PRIVATE-IP"

  }

}

resource "aws\_network\_interface" "hw-db-net" {

  subnet\_id       = aws\_subnet.hw-snet.id

  private\_ips     = ["10.10.10.101"]

  security\_groups = [aws\_security\_group.hw-pub-sg.id]

  tags = {

    Name = "HW-DB-PRIVATE-IP"

  }

}

resource "aws\_instance" "hw-web" {

  ami           = "ami-0229b8f55e5178b65"

  instance\_type = "t2.micro"

  key\_name      = "iac-keypair"

  network\_interface {

    network\_interface\_id = aws\_network\_interface.hw-web-net.id

    device\_index         = 0

  }

  provisioner "file" {

    source      = "./provision-web.sh"

    destination = "/tmp/provision-web.sh"

    connection {

      type        = "ssh"

      user        = "ec2-user"

      private\_key = file("~/.ssh/iac-keypair.pem")

      host        = self.public\_ip

    }

  }

  provisioner "remote-exec" {

    inline = [

      "chmod +x /tmp/provision-web.sh",

      "/tmp/provision-web.sh"

    ]

    connection {

      type        = "ssh"

      user        = "ec2-user"

      private\_key = file("~/.ssh/iac-keypair.pem")

      host        = self.public\_ip

    }

  }

}

resource "aws\_instance" "hw-db" {

  ami           = "ami-0229b8f55e5178b65"

  instance\_type = "t2.micro"

  key\_name      = "iac-keypair"

  network\_interface {

    network\_interface\_id = aws\_network\_interface.hw-db-net.id

    device\_index         = 0

  }

  provisioner "file" {

    source      = "./provision-db.sh"

    destination = "/tmp/provision-db.sh"

    connection {

      type        = "ssh"

      user        = "ec2-user"

      private\_key = file("~/.ssh/iac-keypair.pem")

      host        = self.public\_ip

    }

  }

  provisioner "remote-exec" {

    inline = [

      "chmod +x /tmp/provision-db.sh",

      "/tmp/provision-db.sh"

    ]

    connection {

      type        = "ssh"

      user        = "ec2-user"

      private\_key = file("~/.ssh/iac-keypair.pem")

      host        = self.public\_ip

    }

  }

}

output "web\_app\_public\_ip" {

  value = aws\_instance.hw-web.public\_ip

}

Don't forget to adjust settings like the access and secret keys, names and paths to files, IP addresses, etc.

Once done, we must initialize the provisioner plugin with

**terraform init**

Then, we may ask for the plan with:

**terraform plan**

And finally, we can spin up the infrastructure with:

**terraform apply**

Copy the public IP address, open a browser tab, and paste it

You should see the working web application

Clean up everything with:

**terraform destroy**

## Task 2 (KVM)

Will be working with CentOS Stream 8

Knowing the specifics of the image and that we must create two VMs, we can prepare two shell provision scripts

The first one will be named **provision-web.sh** and will have the following content:

#!/bin/bash

# Install the required packages

sudo yum install -y httpd php php-mysqlnd git

# Switch the SELinux to permissive mode

sudo setenforce 0

# Enable and start the Apache2 service

sudo systemctl enable --now httpd

# Download the project

git clone https://github.com/shekeriev/bgapp

# Copy the files related to the web application

sudo cp ~/bgapp/web/\* /var/www/html

The second one will be named **provision-db.sh** and will have the following content:

#!/bin/bash

# Install the required packages

sudo yum install -y mariadb mariadb-server git

# Enable and start the MariaDB service

sudo systemctl enable --now mariadb

# Download the project

git clone https://github.com/shekeriev/bgapp

# Install the database required for the web application

mysql -u root < ~/bgapp/db/db\_setup.sql

In addition, we will prepare a **cloud init** file (**cloud\_init.cfg**) with the following content:

#cloud-config

ssh\_pwauth: True

users:

  - name: user

    sudo: ['ALL=(ALL) NOPASSWD:ALL']

    shell: /bin/bash

    groups: wheel

chpasswd:

  list:

    - root:RootParolka

    - user:UserParolka

  expire: False

The last file in the solution is the **main.tf** file which will have the following content:

terraform {

  required\_providers {

    libvirt = {

      source = "dmacvicar/libvirt"

    }

  }

}

provider "libvirt" {

  # for remote KVM

  # uri = "qemu+ssh://<username>@<kvm-host>/system"

  # for local KVM this is enough

  uri = "qemu:///system"

}

# a base/master image to be used for both machines

resource "libvirt\_volume" "almalinux-cloud-image" {

  name = "almalinux-cloud-image.qcow2"

  pool = "default"

  # Use either a remote image

  # source = "https://repo.almalinux.org/almalinux/8/cloud/x86\_64/images/AlmaLinux-8-GenericCloud-latest.x86\_64.qcow2"

  # Or first download it and use it as a local one

  source = "/var/lib/libvirt/images/AlmaLinux-8-GenericCloud-latest.x86\_64.qcow2"

  format = "qcow2"

}

resource "libvirt\_volume" "almalinux-db" {

  name = "almalinux-db.qcow2"

  pool = "default"

  base\_volume\_id = libvirt\_volume.almalinux-cloud-image.id

}

resource "libvirt\_volume" "almalinux-web" {

  name = "almalinux-web.qcow2"

  pool = "default"

  base\_volume\_id = libvirt\_volume.almalinux-cloud-image.id

}

data "template\_file" "user-data" {

  template = "${file("${path.module}/cloud\_init.cfg")}"

}

resource "libvirt\_cloudinit\_disk" "commoninit" {

  name = "commoninit.iso"

  pool = "default"

  user\_data      = "${data.template\_file.user-data.rendered}"

}

resource "libvirt\_domain" "almalinux-db" {

  name   = "almalinux-db"

  memory = "1024"

  vcpu   = 1

  # using the default (NAT) network

  network\_interface {

    network\_name = "default"

    hostname = "db"

    wait\_for\_lease = true

  }

  disk {

    volume\_id = "${libvirt\_volume.almalinux-db.id}"

  }

  cloudinit = "${libvirt\_cloudinit\_disk.commoninit.id}"

  console {

    type = "pty"

    target\_type = "serial"

    target\_port = "0"

  }

  graphics {

    type = "spice"

    listen\_type = "address"

    autoport = true

  }

  provisioner "file" {

    source      = "./provision-db.sh"

    destination = "/tmp/provision-db.sh"

    connection {

      type        = "ssh"

      user        = "user"

      password    = "UserParolka"

      #host        = libvirt\_domain.almalinux-db.network\_interface[0].addresses[0]

      host        = self.network\_interface[0].addresses[0]

    }

  }

  provisioner "remote-exec" {

    inline = [

      "chmod +x /tmp/provision-db.sh",

      "/tmp/provision-db.sh"

    ]

    connection {

      type        = "ssh"

      user        = "user"

      password    = "UserParolka"

      #host        = libvirt\_domain.almalinux-db.network\_interface[0].addresses[0]

      host        = self.network\_interface[0].addresses[0]

    }

  }

}

resource "libvirt\_domain" "almalinux-web" {

  name   = "almalinux-web"

  memory = "1024"

  vcpu   = 1

  # using the default (NAT) network

  network\_interface {

    network\_name = "default"

    hostname = "web"

    wait\_for\_lease = true

  }

  disk {

    volume\_id = "${libvirt\_volume.almalinux-web.id}"

  }

  cloudinit = "${libvirt\_cloudinit\_disk.commoninit.id}"

  console {

    type = "pty"

    target\_type = "serial"

    target\_port = "0"

  }

  graphics {

    type = "spice"

    listen\_type = "address"

    autoport = true

  }

  provisioner "file" {

    source      = "./provision-web.sh"

    destination = "/tmp/provision-web.sh"

    connection {

      type        = "ssh"

      user        = "user"

      password    = "UserParolka"

      #host        = libvirt\_domain.almalinux-web.network\_interface[0].addresses[0]

      host        = self.network\_interface[0].addresses[0]

    }

  }

  provisioner "remote-exec" {

    inline = [

      "chmod +x /tmp/provision-web.sh",

      "/tmp/provision-web.sh"

    ]

    connection {

      type        = "ssh"

      user        = "user"

      password    = "UserParolka"

      #host        = libvirt\_domain.almalinux-web.network\_interface[0].addresses[0]

      host        = self.network\_interface[0].addresses[0]

    }

  }

}

output "ip-db" {

  value = "${libvirt\_domain.almalinux-db.network\_interface[0].addresses[0]}"

}

output "ip-web" {

  value = "${libvirt\_domain.almalinux-web.network\_interface[0].addresses[0]}"

}

Once done, we must initialize the provisioner plugin with

**terraform init**

Then, we may ask for the plan with:

**terraform plan**

And finally, we can spin up the infrastructure with:

**terraform apply**

Copy the web machine's IP address, open a browser tab, and paste it

You should see the working web application

Clean up everything with:

**terraform destroy**