**What is Amazon Elastic Container Service?**

- Amazon Elastic Container Service What is Amazon Elastic Container Service? Amazon Elastic Container Service (Amazon ECS) is a highly scalable, fast container management service that makes it easy to run, stop, and manage containers on a cluster.

**What are AWS containers?**

The containers are, in effect, a new base-level building block in the same vein as EC2 and S3. Secure– Your tasks run on EC2 instances within an Amazon Virtual Private Cloud (VPC). The tasks can take advantage of IAMroles, security groups, and other AWS security features.

**Why Container Cluster Management System is needed?**

Provides clustering layer for controlling the deployment of your containers onto the underlying hosts

Manages container lifecycle within the cluster

Scheduling Containers across the cluster

Scaling containers

**What is AWS ECS (EC2 Container Service)?**

Amazon EC2 Container Service (ECS) is a highly scalable, fast and high performance container management service.

Easily run, stop and manage Docker containers on cluster of Amazon EC2 instances.

Schedules the placement of Docker containers across your cluster based on resource needs, availability and requirements.

**Components of ECS**

Cluster – Logical group of container instances

Container Instance – EC2 instance in which ECS agents runs and is registered to cluster.

Task Definition – Description of application to be deployed

Task – An instantiation of task definition running on container instance

Service – Runs and maintains predefined tasks simultaneously

Container – Docker Container created during task instantiation

**Why we use ECS services**

Allows you to run and maintain a specified/desired number of tasks.

If any task fails or stop for any reason, ECS service scheduler launches another task of your task definition to maintain desired task count.

**Key Advantages of ECS Service**

Easy Cluster Management – ECS sets up and manages clusters made up of Docker containers. It launches and terminates the containers and maintains complete information about the state of your cluster.

Auto Scaling – Instance as well as Service level.

Zero-downtime deployment – service updation follows Blue-Green deployments.

Resource Efficiency – A containerized application can make very efficient use of resources. You can choose to run multiple, unrelated containers on the same EC2 instance in order to make good use of all available resources.

AWS Integration – Your applications can make use of AWS features such as Elastic IP addresses, resource tags, and Virtual Private Cloud (VPC)

Service Discovery – used for internal Service to service communication.

Fargate technology – automatically scale, load balance, and manage scheduling of your containers.

Secure – Your tasks run on EC2 instances within an The tasks can take advantage of IAM roles, security groups, and other AWS security features.

**Key Challenges of ECS Service**

Supported by only AWS.

Application level custom monitoring is not available.

**Terraform**

Terraform is one of the very popular Infrastructure as a code tools.

**Infrastructure-as-code** enables you to provision, maintain and define your cloud infrastructure using code. IaC is nothing but defining your infrastructure using configuration files (code). Rather than manually initiating and maintaining infrastructure, both administrators and developers can instantiate infrastructure using the configuration files. It is one of the products of Hashicorp which is Open-source software. Terraform is a service that is similar to Amazon’s cloud formation and have few advantages over it. Since Terraform code is open source code where its code resides in Github and more than 1200 contributors contributed to terraform code. You can find the official terraform code in GitHub using the following link

https://github.com/hashicorp/terraform

Terraform is built using simple language known as HCL( Hashicorp configuration Language ) where we can use simple CLI commands to deploy the resources.It also supports JSON. It provides a consistent workflow for operators to provision infrastructure across Public cloud, Private cloud and external services.

**The terraform includes following steps while creating resources**

Write ( Infrastructure as code )

Plan ( Preview changes before applying )

Create ( creating Reproducible infrastructure )

In Terraform the configuration files usually named with main.tf where you can write the code for your desired infrastructure. You can also declare the variables in the file variables.tf where you can declare your variables those can be latter used in main.tf. Terraform also consists of various modules which can be used in building your codes.

**Infrastructure as Code (IaC)**

Infrastructure as code is nothing but building the desired infrastructure at once using automation using a collection of codes namely code build. Infrastructure as code (IaC) gives us the greater advantage of scaling up and scaling down the infrastructure at our will in minimal time which is the greatest advantage of IaC . Another Great advantage of IaC is that once you build the code for your infrastructure you can run it for any number of times to replicate the same architecture. In case of adding the additional features to your infrastructure it is enough to add the corresponding code to your code build and update it thus, it makes any complex infrastructure with more flexible.

**Advantages of using Terraform**

Terraform is a tool for building, Replicating and modifying the infrastructure safely and more efficiently in minimal time.

Terraform is the open source tool which supports a wide variety of cloud vendor’s tool such as AWS, Microsoft Azure, Digital Ocean, Google cloud etc.

Since it is the open source code any one can contribute to the code to the terraform. If you are expert in terraform you too can contribute to terraform.

Terraform contributors keeps on implementing the additional features to the existing code which meets the requirement of many of the users.

With the Reduced cost you can replicate any complex infrastructure within few hours or even in few minutes.

Since no human intervention is needed other than building code the possibilities of occurring error is very negligible.

Terraform uses simple commands which are easy to understand

Ex: terraform init

terraform refresh

terraform plan

terraform apply

**Project**

Terraform is an infrastructure orchestration tool (also known as “infrastructure as code(IaC)”). Using Terraform, you declare every single piece of your infrastructure once, in static files, allowing you to deploy and destroy cloud infrastructure easily, make incremental changes to the infrastructure, do rollbacks, infrastructure versioning, etc.

Amazon created an innovative solution for deploying and managing a fleet of virtual machines — AWS ECS. Under the hood, ECS utilizes AWSs’ well-known concept of EC2 virtual machines, as well as CloudWatch for monitoring them, auto scaling groups (for provisioning and deprovisioning machines depending on the current load of the cluster), and most importantly — Docker as a containerization engine.

**We need to provision a some building blocks:**

a VPC with a public subnet as an isolated pool for our resources

Internet Gateway to contact the outside world

Security groups for RDS MySQL and for EC2s

Auto-scaling group for ECS cluster with launch configuration

RDS MySQL instance

ECR container registry

ECS cluster with task and service definition

**Steps**

**1. The Terraform Part**

To start with Terraform we need to install it. Just go along with the steps in this document: https://www.terraform.io/downloads.html

Verify the installation by typing:

$ terraform --version

Terraform v0.13.4

**2. Terraform state**

Before writing the first line of our code let’s focus on understanding what the Terraform state is.

The state is a kind of a snapshot of the architecture. Terraform needs to know what was provisioned, what are the resources that were created, track the changes, etc.When working with AWS, this destination is s3.

This is the first thing that we need to code — tell terraform that the state location will be remote and kept is s3 (terraform.tf):

terraform {

backend "s3" {

bucket = "terraformeksproject"

key = "state.tfstate"

}

}

Terraform will keep the state in an s3 bucket under a state.tfstate key. In order that to happen we need to set up three environment variables:

$ export AWS\_SECRET\_ACCESS\_KEY=...

$ export AWS\_ACCESS\_KEY\_ID=..

$ export AWS\_DEFAULT\_REGION=...

**3.The resource block type aws vpc with name vpc creates Virtual Private Cloud** — a logically isolated virtual network. When creating VPC we must provide a range of IPv4 addresses. It’s the primary CIDR block for the VPC and this is the only required parameter.

Parameters enable\_dns\_support and enable\_dns\_hostnames are required if we want to provision database in our VPC that will be publicly accessible (and we do).

provider "aws" {}

resource "aws\_vpc" "vpc" {

cidr\_block = "10.0.0.0/24"

enable\_dns\_support = true

enable\_dns\_hostnames = true

tags = {

Name = "Terraform VPC"

}

}

**4. Internet gateway**

In order to allow communication between instances in our VPC and the internet we need to create Internet gateway.

resource "aws\_internet\_gateway" "internet\_gateway" {

vpc\_id = aws\_vpc.vpc.id

}

**5. Subnet**

Within the VPC let’s add a public subnet:

resource "aws\_subnet" "pub\_subnet" {

vpc\_id = aws\_vpc.vpc.id

cidr\_block = "10.1.0.0/22"

}

To create a subnet we need to provide VPC id and CIDR block. Additionally we can specify availability zone, but it’s not required.

**6. Route Table**

Route table allows to set up rules that determine where network traffic from our subnets is directed. Let’s create new, custom one, just to show how it can be used and associated with subnets.

What we did is created a route table for our VPC that directs all the traffic (0.0.0.0/0) to the internet gateway and associate this route table with both subnets. Each subnet in VPC have to be associated with a route table.

resource "aws\_route\_table" "public" {

vpc\_id = aws\_vpc.vpc.id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.internet\_gateway.id

}

}

resource "aws\_route\_table\_association" "route\_table\_association" {

subnet\_id = aws\_subnet.pub\_subnet.id

route\_table\_id = aws\_route\_table.public.id

}

**7. Security Groups**

resource "aws\_security\_group" "ecs\_sg" {

vpc\_id = aws\_vpc.vpc.id

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

from\_port = 443

to\_port = 443

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

from\_port = 0

to\_port = 65535

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

}

resource "aws\_security\_group" "rds\_sg" {

vpc\_id = aws\_vpc.vpc.id

ingress {

protocol = "tcp"

from\_port = 3306

to\_port = 3306

cidr\_blocks = ["0.0.0.0/0"]

security\_groups = [aws\_security\_group.ecs\_sg.id]

}

egress {

from\_port = 0

to\_port = 65535

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

}

First security group is for the EC2 that will live in ECS cluster. Inbound traffic is narrowed to two ports: 22 for SSH and 443 for HTTPS needed to download the docker image from ECR.

Second security group is for the RDS that opens just one port, the default port for MySQL — 3306. Inbound traffic is also allowed from ECS security group, which means that the application that will live on EC2 in the cluster will have permission to use MySQL.

Inbound traffic is allowed for any traffic from the Internet (CIDR block 0.0.0.0/0). In real life case there should be limitations, for example, to IP ranges for a specific VPN.

This ends setting up the networking park of our architecture.

**8. Database Instance**

Having prepared subnet and security group for RDS we need one more thing to cover before launching the database instance. To provision a database we need to follow some rules:

Our VPC has to have enabled DNS hostnames and DNS resolution (we did that while creating VPC).

Our VPC has to have a DB subnet group (that is about to happen).

Our VPC has to have a security group that allows access to the DB instance.

Let’s create the missing piece:

resource "aws\_db\_subnet\_group" "db\_subnet\_group" {

subnet\_ids = [aws\_subnet.pub\_subnet.id]

}

And database instance itself:

resource "aws\_db\_instance" "mysql" {

identifier = "mysql"

allocated\_storage = 5

backup\_retention\_period = 2

backup\_window = "01:00-01:30"

maintenance\_window = "sun:03:00-sun:03:30"

multi\_az = true

engine = "mysql"

engine\_version = "5.7"

instance\_class = "db.t2.micro"

name = "worker\_db"

username = "worker"

password = "worker"

port = "3306"

db\_subnet\_group\_name = aws\_db\_subnet\_group.db\_subnet\_group.id

vpc\_security\_group\_ids = [aws\_security\_group.rds\_sg.id, aws\_security\_group.ecs\_sg.id]

skip\_final\_snapshot = true

final\_snapshot\_identifier = "worker-final"

publicly\_accessible = true

}

All the parameters are more less self explanatory. If we want our database to be publicly accessible you have to set the publicly\_accessible parameter as true.

**9. Elastic Container Service**

ECS is a scalable container orchestration service that allows to run and scale dockerized applications on AWS.

To launch such an application we need to download image from some repository. For that we will use ECR. We can push images there and use them while launching EC2 instances within our cluster:

resource "aws\_ecr\_repository" "worker" {

name = "worker"

}

And the ECS itself:

resource "aws\_ecs\_cluster" "ecs\_cluster" {

name = "my-cluster"

}

**10. Applying the changes**

First we need to initialize a working directory that contains Terraform files by typing terraform init. This command will install needed plugins and provide a code validation.

Follow up with terraform plan.

Finding that you’re receiving an error?

You need to manually create the S3 bucket through the aws console, making sure to edit terraform.tf with the correct bucket name.

If everything is fine we can run terraform apply to finally provision the desired infrastructure.