

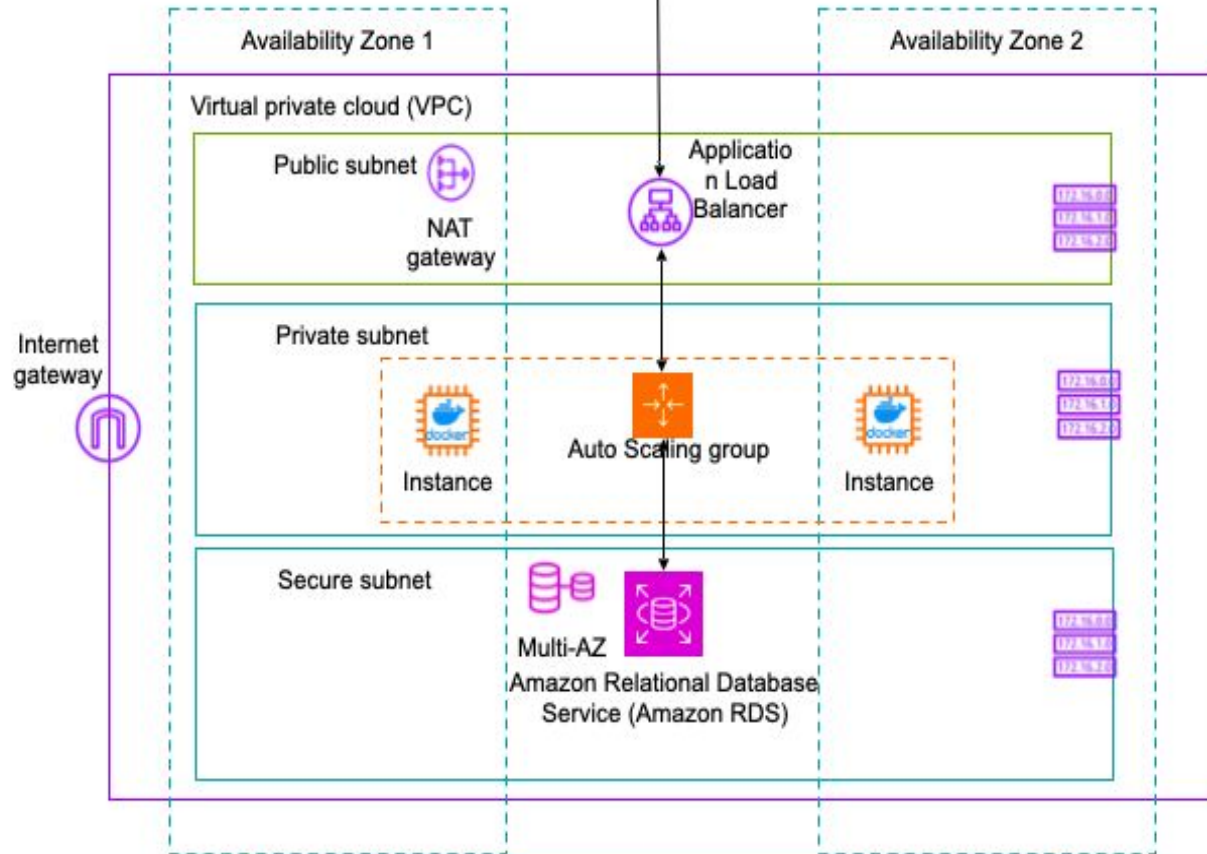
AWS Infrastructure provisioning using terraform

Creation of Infrastructure using Terraform

- Terraform is a mandatory skill. Used across the industry and in multi cloud scenario.
- It's easy to get started but can get really complicated later to manage.
- Best practices are a MUST.
- State must be in Cloud using S3 or Azure Blob Storage.
- For State file security enable S3 cross region replication, versioning and encryption
- Apply state locking using DynamoDB
- The CICD Pipeline to have three steps and checks:
 - **On commit to any Branch** perform terraform init, validate, fmt and plan operations, *checkov scan is optional*
 - **On pull request creation** — perform terraform init, validate fmt and plan so that reviewer can see what is the change this pipeline will perform
 - **On merge** the feature is merged to master/ main and terraform apply happens using GitOps approach.



Amazon Route 53



This is the Reference Architecture for infrastructure

Key requirements for Infrastructure

1- 3 Subnets ArchitecturePublic, Private and Secure Subnets.

Public Subnets should have routes to Internet Gateway. Private Subnets should have route to NAT Gateway. Secure subnets should not have route to IGW or NAT GW.

2- Load balancer in Public SubnetCreate Load Balancer in Public Subnet across 2 AZs. Create corresponding Listener and Target Groups.

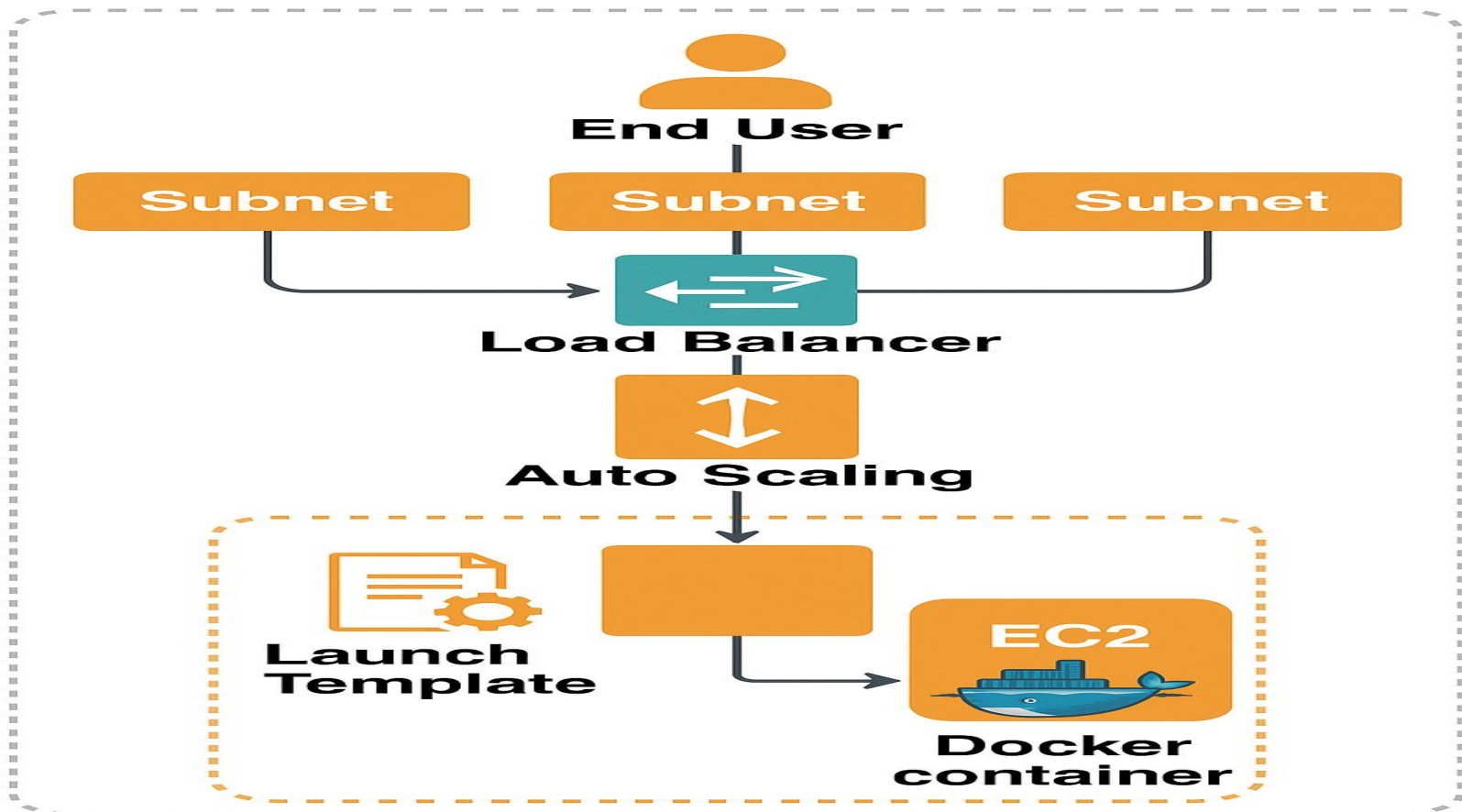
3- Create Auto Scaling GroupCreate the ASG across 2 AZs in private subnet. Attach the ASG to ALB. Desired=1, Min=1, Max=1

The instances should not have public IPs.

Instances should be connecting using SSM or EC2 Instance Connect Endpoint.

Security group should not open port 22.

EC2 should be using a user data script at startup and install the application [Docker Image or WAR JAR File]. This should be in sync with the ALB Target groups.



Step1: clone/fork this repository for infrastructure provisioning

1- https://github.com/ankit20000/springapp_terraform.git

2- Install terraform –

<https://developer.hashicorp.com/terraform/install>

3- Install aws cli –

<https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>

Step2 :configure aws cli with access keys

aws Services Search [Alt+S] Global Deepak Reddy

Access key created
This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.

[IAM](#) > [Security credentials](#) > [Create access key](#)

Step 1
[Alternatives to root user access keys](#)

Step 2
Retrieve access key

Retrieve access key [Info](#)

Access key

If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

Access key	Secret access key
AKIAZ6V5T3BQBDGZWG7E	***** Show

Access key best practices

- Never store your access key in plain text, in a code repository, or in code.
- Disable or delete access key when no longer needed.
- Enable least-privilege permissions.
- Rotate access keys regularly.

For more details about managing access keys, see the [best practices for managing AWS access keys](#).

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Step3 : apply command `Terraform init` to initialize terraform backend

The screenshot displays a VS Code workspace for a Terraform project named 'springapp_with-Terraform'. The Explorer panel on the left shows the project structure, with 'aws_petclinic_region1' selected. The main editor shows the 'auto_scaling_grp.tf' file, which contains Terraform code for an AWS launch template. The terminal at the bottom shows the execution of 'ls', 'cd', and 'terraform init' commands.

EXPLORER

- SPRINGAPP_WITH-TERAFORM [CODESPACES: CRIS...]
 - .devcontainer
 - aws_petclinic_region1
 - .terraform
 - .gitignore
 - .terraform.lock.hcl
 - alb.tf
 - auto_scaling_grp.tf
 - backend.tf
 - ec2.tf
 - main.tf
 - rds.tf
 - s3.tf
 - security_grp_alb.tf
 - terraform.tfstate
 - terraform.tfstate.backup
 - userdata.sh
 - variable.tf
 - aws_petclinic_region2
 - main
 - modules
 - README.md

auto_scaling_grp.tf

```
aws_petclinic_region1 > auto_scaling_grp.tf > resource "aws_launch_template" "ec2_asg" > user_data
1 data "aws_ami" "amazon_linux_2" {
2
3   owners      = ["amazon"]
4
5   filter {
6     name     = "owner-alias"
7     values   = ["amazon"]
8   }
9
10  filter {
11    name     = "name"
12    values   = ["amzn2-ami-hvm"]
13  }
14 }
```

TERMINAL

```
bash - aws_petclinic_region1
@deepak4566 →/workspaces/springapp_with-Terraform (main) $ ls
README.md  aws_petclinic_region1  aws_petclinic_region2  main  modules
@deepak4566 →/workspaces/springapp_with-Terraform (main) $ cd aws_petclinic_region1
@deepak4566 →/workspaces/springapp_with-Terraform/aws_petclinic_region1 (main) $ ls
alb.tf      backend.tf  main.tf    s3.tf      terraform.tfstate  userdata.sh
auto_scaling_grp.tf  ec2.tf      rds.tf    security_grp_alb.tf  terraform.tfstate.backup  variable.tf
@deepak4566 →/workspaces/springapp_with-Terraform/aws_petclinic_region1 (main) $ terraform init
```


EXPLORER

▼ SPRINGAPP_WITH-TERRAFORM [CODESPACES: CRIS...

> .devcontainer

▼ aws_petclinic_region1

> .terraform

◆ .gitignore

≡ .terraform.lock.hcl

🔒 alb.tf

🔒 auto_scaling_grp.tf

🔒 backend.tf

🔒 ec2.tf

🔒 main.tf

🔒 rds.tf

🔒 s3.tf

🔒 security_grp_alb.tf

(terraform.tfstate

≡ terraform.tfstate.backup

\$ userdata.sh

🔒 variable.tf

> aws_petclinic_region2

> main

> modules

① README.md

\$ userdata.sh M

🔒 auto_scaling_grp.tf X

```
aws_petclinic_region1 > 🔒 auto_scaling_grp.tf > resource "aws_launch_template" "ec2_asg" > 🔒 user_data
1  data "aws_ami" "amazon_linux_2" {
2
3      owners      = ["amazon"]
4
5      filter {
6          name     = "owner-alias"
7          values   = ["amazon"]
8      }
9
10     filter {
11         name     = "name"
12         values   = ["amzn2-ami-hvm*"]
13     }
14 }
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

CODE REFERENCE LOG

COMMENTS

Initializing the backend...

Initializing provider plugins...

- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.41.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see any changes that are required for your infrastructure. All Terraform commands should now work.

I

If you ever set or change modules or backend configuration for Terraform,

Step4 : apply command `Terraform plan` to check what resources are created

The screenshot shows a VS Code editor interface with a dark theme. The Explorer panel on the left displays a file tree for a project named 'SPRINGAPP_WITH-TERAFORM [CODESPACES: CRIS...]'.

The Explorer panel shows the following structure:

- SPRINGAPP_WITH-TERAFORM [CODESPACES: CRIS...]
 - .devcontainer
 - aws_petclinic_region1
 - .terraform
 - .gitignore
 - .terraform.lock.hcl
 - alb.tf
 - auto_scaling_grp.tf (selected)
 - backend.tf
 - ec2.tf
 - main.tf
 - rds.tf
 - s3.tf
 - security_grp_alb.tf
 - terraform.tfstate
 - terraform.tfstate.backup
 - userdata.sh
 - variable.tf
 - aws_petclinic_region2
 - main
 - modules
 - README.md

The main editor area shows the content of the selected file, `auto_scaling_grp.tf`. The code defines an `aws_launch_template` resource named `ec2_asg` with the following configuration:

```
1 data "aws_ami" "amazon_linux_2" {
2
3   owners      = ["amazon"]
4
5   filter {
6     name     = "owner-alias"
7     values   = ["amazon"]
8   }
9
10  filter {
11    name     = "name"
12    values   = ["amzn2-ami-hvm*"]
13  }
14 }
```

The bottom panel shows the output of the `terraform plan` command, displaying the resources that will be created or modified:

```
requester_managed = (known after apply)
route_table_ids   = (known after apply)
security_group_ids = (known after apply)
service_name      = "com.amazonaws.ap-south-1.ssmmessages"
state             = (known after apply)
subnet_ids        = (known after apply)
tags_all          = (known after apply)
vpc_endpoint_type = "Interface"
vpc_id            = (known after apply)
```

The plan summary at the bottom indicates: `Plan: 40 to add, 0 to change, 0 to destroy.`

Step5 : Terraform apply –auto approve` to create infra

The screenshot displays a VS Code editor interface with a dark theme. On the left, the Explorer sidebar shows a project structure for 'SPRINGAPP_WITH-TERRAFORM [CODESPACES: CRIS...]' containing files like .devcontainer, .aws_petclinic_region1, .terraform, .gitignore, .terraform.lock.hcl, .terraform.tfstate.lock.info, alb.tf, auto_scaling_grp.tf (selected), backend.tf, ec2.tf, main.tf, rds.tf, s3.tf, security_grp_alb.tf, terraform.tfstate, terraform.tfstate.backup, userdata.sh, variable.tf, and README.md. The main editor area shows the content of 'auto_scaling_grp.tf', which defines an 'aws_launch_template' resource named 'ec2_asg' using 'data.aws_ami.amazon_linux_2' and 'filter' blocks. The bottom panel features a 'TERMINAL' tab showing the output of a Terraform apply command, indicating successful creation of various AWS resources such as security groups, subnets, internet gateways, and route tables.

```
aws_petclinic_region1 > auto_scaling_grp.tf > resource "aws_launch_template" "ec2_asg" > user_data
1  data "aws_ami" "amazon_linux_2" {
2
3      owners      = ["amazon"]
4
5      filter {
6          name     = "owner-alias"
7          values   = ["amazon"]
8      }
9
10     filter {
11         name     = "name"
12         values   = ["amzn2-ami-hvm*"]
13     }
14 }
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS CODE REFERENCE LOG COMMENTS

```
terraform - aws_petclinic_region1 + v [] ... ^ x
aws_security_group.webSg: Creating...
aws_subnet.secure_subnet_az2: Creating...
aws_lb_target_group.alb_target_group: Creating...
aws_subnet.sub2: Creating...
aws_subnet.private_sub1: Creating...
aws_internet_gateway.igw: Creating...
aws_internet_gateway.igw: Creation complete after 1s [id=igw-069798d0d8895928c]
aws_security_group.alb_security_group: Creating...
aws_subnet.private_sub1: Creation complete after 1s [id=subnet-0b3ff73164a8199c1]
aws_subnet.private_sub2: Creation complete after 1s [id=subnet-04973f59c5ea913a6]
aws_subnet.secure_subnet_az1: Creation complete after 1s [id=subnet-069ca4da174bd845b]
aws_subnet.secure_subnet_az2: Creation complete after 1s [id=subnet-0c7f26abff7942c01]
aws_route_table.RT: Creating...
aws_db_subnet_group.database_subnet_group: Creating...
```

Step6: after creation, go load balancers in aws console and check whether application is running or not with dns name.

The screenshot displays the AWS Management Console interface for an Elastic Load Balancing (ELB) Application Load Balancer. The left sidebar shows the navigation menu with categories like Images, Elastic Block Store, Network & Security, Load Balancing, and Auto Scaling. The main content area shows the details of a specific load balancer.

Load Balancer Details:

- Load balancer type:** Application
- Status:** Active
- VPC:** vpc-0b940b93baa5530a2
- IP address type:** IPv4
- Scheme:** Internet-facing
- Hosted zone:** ZP97RAFLXTNZK
- Availability Zones:** subnet-04f1dd1e02304f1e7 (ap-south-1a (aps1-az1)), subnet-02704fa9c7334daf2 (ap-south-1b (aps1-az3))
- Date created:** March 25, 2024, 21:41 (UTC+05:30)
- Load balancer ARN:** arn:aws:elasticloadbalancing:ap-south-1:684371204192:loadbalancer/app/public-loadbalancer/6b54ffdb062f5c77
- Public DNS Name:** public-loadbalancer-243182553.ap-south-1.elb.amazonaws.com (A Record)

A tooltip indicates "DNS name copied".

Listeners and rules (1)

A listener checks for connection requests on its configured protocol and port. Traffic received by the listener is routed according to the default action and any additional rules.

Filter listeners:

Listeners table:

Protocol:Port	Default action	Rules	ARN	Security policy	Default SSL

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preference

We can successfully see that application is running

