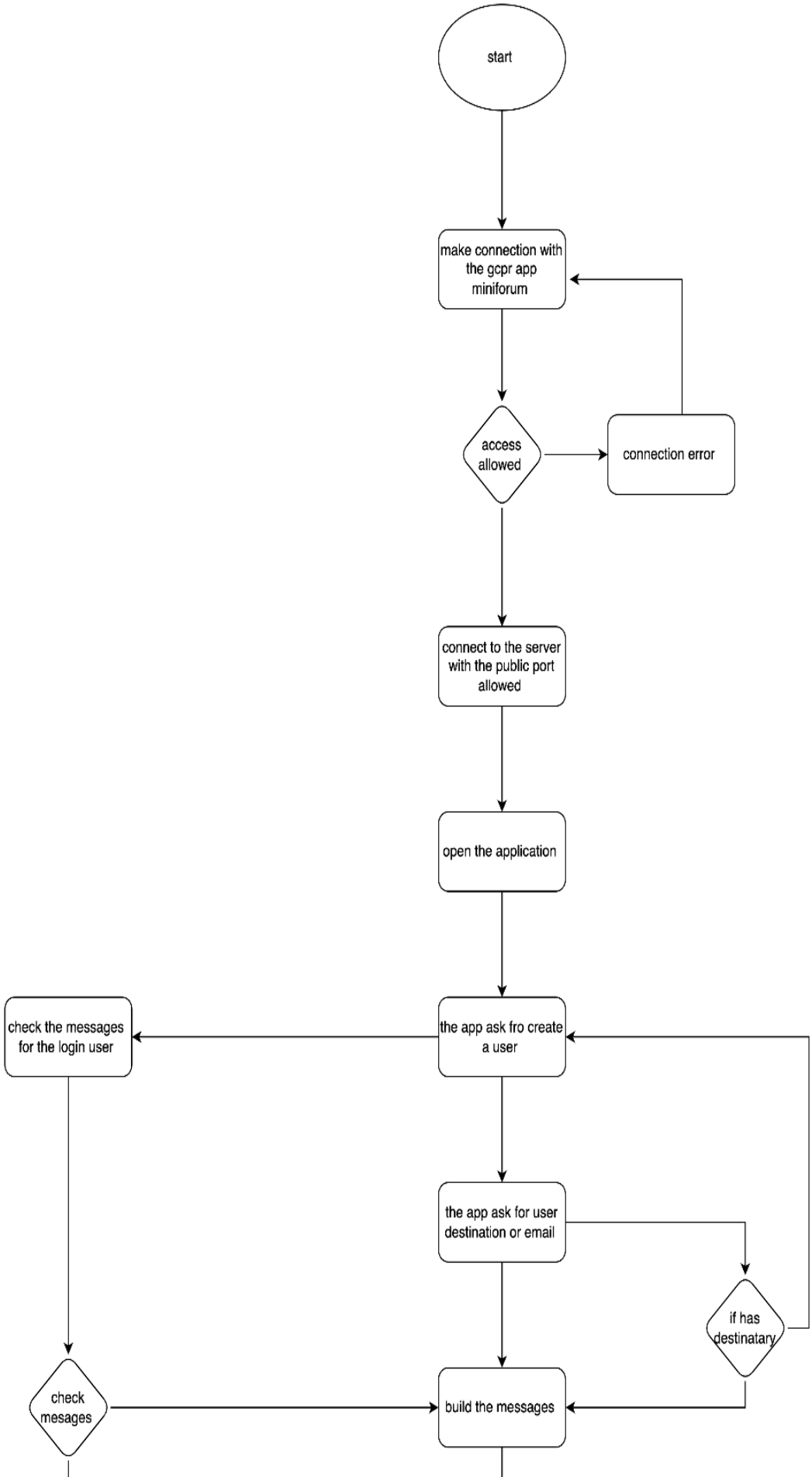
Contents

Concept

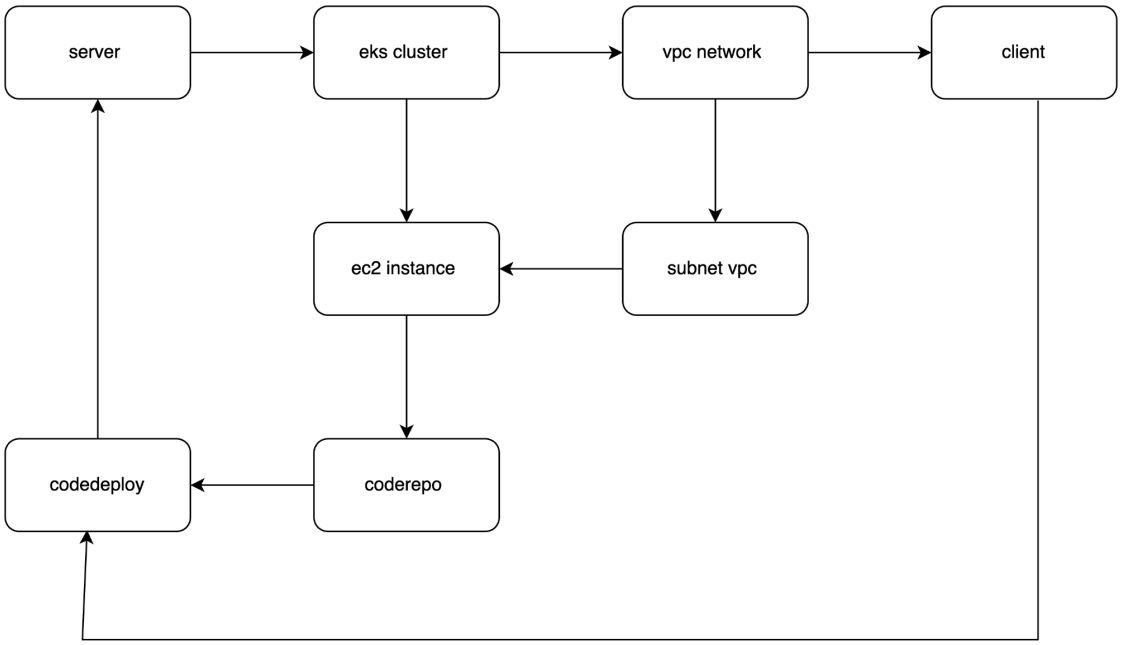
Based on the requirements of the test and use the technologies like terraform, python, and bash , and cloud services like aws, the propose of this test is show the abilities in create a project that can abble to connect to repository , consume the application project inside the repo, build that code use the IAS build by terraform and eks services crete a vpc connection for stablish public coms with cluster and internet , and make two networks , private for local communications inside the app and the cluster , and public for allow the app make see from internet when a client try to connect to the server for check if have some messages and next step if exist some new messages , that messages can receive by email

Python program

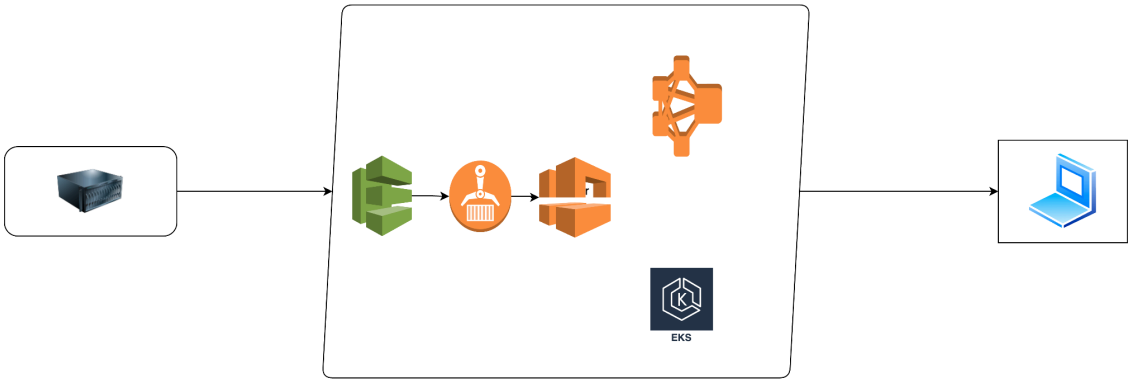

start
make connection with the gcpr app miniforum
access allowed
connection error
connect to the server with the public port allowed
open the application
check the messages for the login user
the app ask fro create a user
the app ask for user destination or email
if has destinatary
check mesages
build the messages

This is part of the flow in program related to the how can the grpc protocol enable for send , process and receive the messages between the users and process those messages inside once the complete process they anchor and attach inside the cluster in eks , the server and the client can enable to process the components inside the architecture , include a vpc network that allows the access to the app since public and private networks

Architecture Diagram



This is the functional diagram that describe how the services works, and how the lines the aws modules is connected to the application and link from node server to the vpc pass throught inside the vpc and process the request in cluster and send the request to the client



Contents

Concept

Based on the requirements of the test and use the technologies like terraform, python, and bash , and cloud services like aws, the propose of this test is show the abilities in create a project that can abble to connect to repository , consume the application project inside the repo, build that code use the IAS build by terraform and eks services crete a vpc connection for stablish public coms with cluster and internet , and make two networks , private for local communications inside the app and the cluster , and public for allow the app make see from internet when a client try to connect to the server for check if have some messages and next step if exist some new messages , that messages can receive by email

Python program

This is part of the flow in program related to the how can the grpc protocol enable for send , process and receive the messages between the users and process those messages inside once the complete process they anchor and attach inside the cluster in eks , the server and the client can enable to process the components inside the architecture , include a vpc network that allows the access to the app since public and private networks

Architecture Diagram

This is the functional diagram that describe how the services works, and how the lines the aws modules is connected to the application and link from node server to the vpc pass throught inside the vpc and process the request in cluster and send the request to the client

Summary
This project has a top level terraform module cluster which scaffolds vpc_and_subnets and eks modules to create following resources -

VPC
Subnets (3 public and 3 private)
1 NAT Gateway per AZ with corresponding Elastic IPs
Internet Gateway
Public and Private Route tables
EKS Cluster with OIDC Provider
EKS Managed AddOns
coredns
vpc-cni
kube-proxy
EKS Managed node group
Links to module documentation
Module Name Documentation Link
Cluster module with EKS, VPC and Subnets README
EKS README
VPC And Subnets README
Modularization of terraform
cluster is the top level module which is scaffold over vpc_and_subnets and eks modules.

Essentially you can imagine that your infrastructure team is building APIs by building opinionated vpc_and_subnets and eks modules in different repository. And your team's Platform team building cluster module which makes use of these APIs. And top level main.tf file invoking cluster module, this can be written by your developer team member who wants to use the EKS Cluster.

As you can see in the cluster module's main.tf we are invoking vpc_and_subnets and eks modules, and you can specify the source to remote github repository where your source of the modules are, you can read about module sources in the official terraform documentation.

Hope by looking at the module structure you will find some ideas to modularize and structure your terraform.

Prerequisites for this repository
Basic understanding of AWS, EKS, VPC, and Terraform.

AWS account with necessary permissions to create VPC, Subnets, EKS Cluster etc..

Configure aws cli to point to your aws account, you will need this to generate the kubeconfig to connect to the cluster.

Install kubectl compatible with the EKS version you are installing.

Try to work with Latest version of Terraform. I have used v1.5.2 on mac for this blog. If you want to manage multiple versions of Terraform use tfswitch,

If you want to learn how to generate the documentation from terraform files, install terraform-docs

Install helm a package manager for Kubernetes manifests, we will use it to install nginx helm chart once the cluster is created.

Prepare .tfvars file
.tfvars is a way to create the input files for terraform module. For example, you can create dev.tfvars for dev environment, test.tfvars for test environment and so on.

We have a sample.tfvars for reference, substitute values as per your need and play around.

Execute terraform to create the infrastructure

This section explains how to execute the terraform module cluster to create vpc, subnets and eks cluster.

main.tf
In main.tf tf file you will see that we are setting up the aws provider and calling cluster module. You will see that we are passing all the variables required by cluster modules.

variables.tf
In variables.tf tf file you will see declaration of all the variables we are taking input from .tfvars file and passing it to cluster module.

outputs.tf
In outputs.tf tf file you will see declaration of any output variables we might need for our usage after the resources are created. These values are being copied from cluster module's output, which accumulates from eks and vpc_and_subnets module outputs.

How to execute
Execute all the commands below from my-eks-tf root where the above explained files are -

Make sure your terminal is configured to talk to AWS Account, you can use one of the ways explained in this document. In my case I am using one of the profiles present in my ~/.aws/config file and setting the environment variable as below -
export AWS_PROFILE="my-aws-profile"
Another simple way is create an IAM User with AdminstratorAccess (only do for temporary purpose to quickly test, and make sure to delete the IAM User after usage) and set the following environment variables on the terminal.

export AWS_ACCESS_KEY_ID=<access key id copied after creating the IAM User>
export AWS_SECRET_ACCESS_KEY=<secret access key copied after creating the IAM User>
NOTE: if you choose the assumeRole option, you will need to modify the provider block in main.tf, I have kept a sample commented code. Please refer terraform documentation for more ways to set the provider.

Make sure the s3 bucket to store the tfstate file exists, if not please create. Following is an example how you can use aws cli to create the s3 bucket.
aws s3api create-bucket --bucket "your-bucket-name" --region "your-aws-region"

Initialize the module and set the backend of tfstate file which records the state of the resources created by terraform apply invocation.
tfstate file name

# tfstate s3 bucket name, this will have the tfstate file which you can use for further runs of this terraform module
# for example to upgrade k8s version or add new node pools etc.. The bucket name must be unique as s3 is a global service. Terraform will create the s3 bucket if it doesn't exist
tfstate_bucket_name="unique s3 bucket name you created above e.g. my-tfstate-<myname>"

initialize the terraform module
terraform init -backend-config "key=${tfstate_file_name}" -backend-config "bucket=${tfstate_bucket_name}" -backend-config "region=us-east-1"
After execution of above, you will observe that, an s3 bucket is created in aws account.

Retrieve the terraform plan, a preview of what will happen when you apply this terraform module. This is a best practice to understand the change.
terraform plan -var-file="path/to/your/terraform.tfvars"

terraform plan -var-file="sample.tfvars"
If you are satisfied with the plan above, this is the final step to apply the terraform and wait for the resources to be created. It will take about ~20 mins for all the resources to be created.
terraform apply -var-file="path/to/your/terraform.tfvars"

terraform apply -var-file="sample.tfvars"
After successful execution, go to next section on how to connect to the EKS Cluster and install nginx helm chart.

Connect to eks cluster and install nginx helm chart
In this section we will show how to connect to eks cluster and install nginx helm chart. This is just to prove that you have successfully created a function eks cluster. This is with the

assumption that you have installed all the cli tools mentioned in the pre-requisites section above.

Retrieve kubeconfig using aws cli, assuming you have configured the aws cli properly to point to the aws account which has the eks cluster. Please see aws cli documentation for configuration details.
aws eks update-kubeconfig --region "<aws region where we created the eks cluster>" --name "<eks cluster name>"

as per the sample.tfvars parameters
aws eks update-kubeconfig --region "us-east-2" --name "platformwale"
You can check if you are pointing to the right kubernetes cluster by running following kubectl command
kubectl config current-context
Install nginx helm chart
helm repo add bitnami https://charts.bitnami.com/bitnami
helm install -n default nginx bitnami/nginx
Check if all the pods are scheduled and running. Also validate that the load balancer is created, you can copy paste the EXTERNAL-IP and put it in browser, you should see the Welcome to nginx! page as shown in the screenshot below.
kubectl get pods -n default
kubectl get svc -n default
# example
$ kubectl get pods -n default
NAME READY STATUS RESTARTS AGE
nginx-7c8ff57685-ck9pn 1/1 Running 0 3m31s

$ kubectl get svc -n default nginx
NAME TYPE CLUSTER-IP EXTERNAL-IP
PORT(S) AGE
nginx LoadBalancer 172.20.214.161
a90947031083b48b9b61849f22cdceda-410225407.us-east-2.elb.amazonaws.com
80:32414/TCP 2m59s
Welcome to nginx!!

Cleanup
This is the most important step if you don't want any unexpected cloud costs on your account.

Make sure to uninstall the nginx helm chart to delete the loadbalancer before you start destroying the infrastructure in next step using terraform destroy. Make sure the nginx svc is deleted.

```
helm uninstall -n default nginx
# validate that the external service is deleted, it takes a few mins
$ kubectl get svc -n default nginx
Error from server (NotFound): services "nginx" not found
```

Destroy the infrastructure. It takes about ~15 mins to delete the infrastructure we created above.

```
terraform destroy -var-file="sample.tfvars"
```

Delete the s3 bucket created to store tfstate

```
# empty the bucket
aws s3 rm s3://<your-bucket-name> --recursive


# delete the bucket
aws s3api delete-bucket --bucket "your-bucket-name" --region "your-aws-region"
```

Terraform Documentation

Requirements

Name	Version
aws	5.6.2

Providers

Name	Version
aws	5.6.2

Modules

Name	Source	Version
cluster	./cluster	n/a

Inputs

Name	Description	Type	Default	Required
cluster_name	eks cluster name	string	"platformwale"	no
k8s_version	k8s version	string	"1.27"	no
region	aws region where the resources are being created	string	n/a	yes
vpc_cidr	vpc cidr block to be used	string	"10.0.0.0/16"	no
vpc_name	name of the vpc to be created	string	"platformwale"	no

Outputs

Name	Description
cluster_certificate_authority_data	Base64 encoded certificate data required to communicate with the cluster
cluster_endpoint	Endpoint for your Kubernetes API server
cluster_oidc_issuer_url	The URL on the EKS cluster for the OpenID Connect identity provider

Handy commands

Generate documentation by running terraform-docs command from the module directory. Now you can copy the documentation from stdout.

```
cd ./modules/eks
terraform-docs markdown .
```

Format hcl files.

```
# recursively format all the files
terraform fmt -recursive


# just want to format a file
terraform fmt "<file/path>"
```

Troubleshooting

If you see following error while executing terraform init command for the first time, this means the tfstate s3 bucket is not created, manually create the s3 bucket. You can read more details as mentioned in terraform s3 backend documentation.

```
╷
│ Error: Failed to get existing workspaces: S3 bucket does not exist.
│
│ The referenced S3 bucket must have been previously created. If the S3 bucket
│ was created within the last minute, please wait for a minute or two and try
│ again.
│
│ Error: NoSuchBucket: The specified bucket does not exist
│   status code: 404, request id: 2R4WDEWZZQGXT7YD, host id:
YHsfJYMpCvY5XcP+3rPzhpKl0kpmIku/VvSCjXfxHgskkTec7e0IPlm5PAjjCb3yUaKnlJ5HTMq3H
gByAepruXbT2MyQEf/J
│
│
│
```

You can also use the below AWS Cli command to create the aws s3 bucket, make sure your aws cli is configured to point to the aws account where you want to run the terraform.

```
aws s3api create-bucket --bucket "your-bucket-name" --region "your-aws-region"
```