# System design

1. AWS vs Azure vs GCP

<https://www.cloudzero.com/blog/aws-vs-azure-vs-google-cloud/>

* Availability Zones (regions, data centers, etc.)
* The existing services of choice include the cloud and on-premises (windows, K8S, etc.)
* AWS: Broad service and scalability for diverse projects
* Azure: Integrate with Microsoft products and emphasize security and compliance
* GCP: Data analytics, K8S and high-performance networking

1. Application solution

* Nature of the application (monolith or microservices, containerization (more flexibility to the application environment, better for long running service) vs serverless (cost effective for short running service, better on scaling))
* Cloud native solution (ECS) or cloud managed service (EKS)

1. Terraform vs CloudFormation vs eks CLI

* Cloud agnostic
* Deployment state management (flexibility, configuration drift)
* Technical familiarity

1. 3rd party TLS certification vs Cloud native solution

* TLS renewal
* Prod ready

1. Authentication and authorization

* Cloud native access control (IAM access entries (EKS), API server endpoint access (EKS), IAM role/policy, private subnets, firewalls, etc.)

1. CICD

* Push (CICD pipeline) vs pull (GitOps)

1. Scaling

* Cloud native solutions (autoscaling group) vs K8S (HPA, VPA, Karpenter)

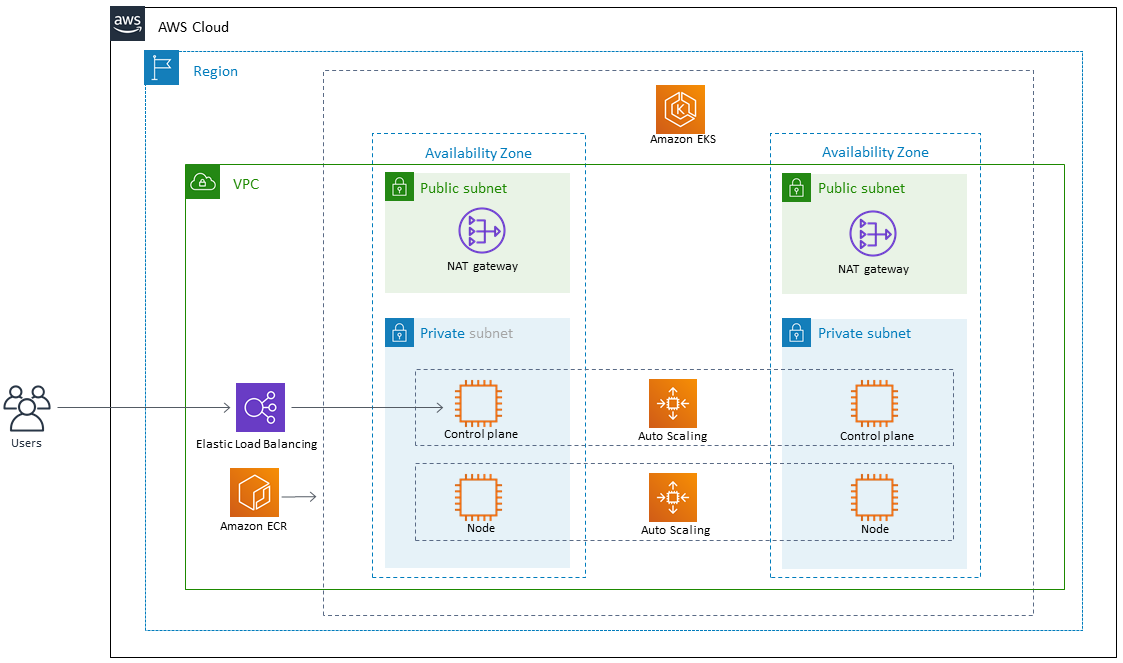
1. High Availability

* Cloud native HA (Availability zone, data centers, fargate)
* Cloud managed control plane, ALB health check, K8S replicas, probe

1. Monitoring

Prometheus/Grafana vs Splunk vs K8S Metrics server

1. System diagram



# Tools used

* Terraform is used to deploy the EKS cluster and the flask application
* GitHub is used as the code repository.
* GitHub registry is used to store the docker image
* Docker is used to build the flask image
* AWS is the cloud provider
* “aws-vault” tool (<https://github.com/99designs/aws-vault>) to access the EKS cluster via CLI
* Lens(<https://k8slens.dev/>) and kubectl CLI are used to access the EKS cluster for troubleshooting

# EKS cluster creation

The EKS cluster was created using Terraform.

1. Due to the security concern, an IAM user with permission to assume an IAM role will be used to deploy the cluster.
2. VPC, subnets and other network resources were created via the “vpc” module
3. EKS cluster and load balancer controller roles were created via the “eks” module

A computer screen with text

Description automatically generated

# Python flask application build and deployment

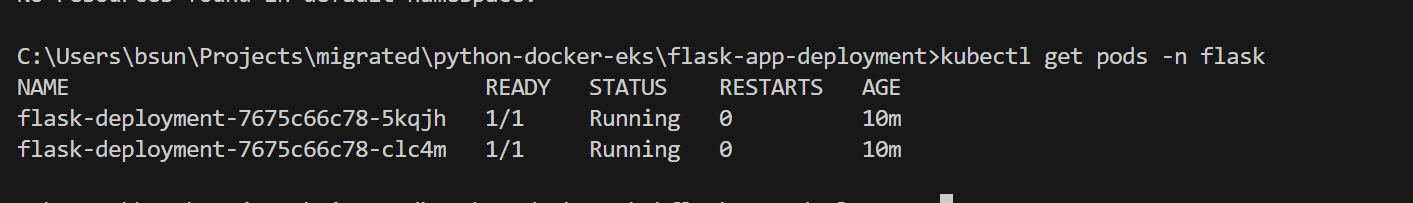
The flask application was built and pushed to GitHub as a package. A self-signed certificate was generated for the application server. (I roll back in my test with the AWS certificate as different certificates used by ALB and flask server will cause 502 Bad Gateway)A screenshot of a computer

Description automatically generated

The flask application was deployed to the EKS cluster using Terraform.

A screen shot of a computer

Description automatically generated



A TLS certificate was generated in AWS Certificate Manager to enable HTTPS connection.

A DNS CNAME was created in the company hosted zone in another AWS account via Terraform

A screen shot of a computer

Description automatically generated

# Metrics server and horizontal pod autoscaling

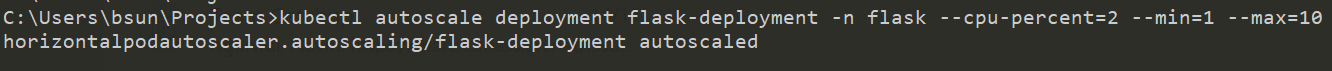
The metrics server was deployed via “kubectl”  
A computer screen with many white text

Description automatically generated

A screen shot of a computer

Description automatically generated

I run “while sleep 0.01; do wget -q -O- https://flask.neon.markets; done” in two sessions to trigger the horizontal autoscaling and monitor the resource usage



Before scaling up:

A screen shot of a computer program

Description automatically generated

After scaling up:

A screen shot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

With an php-apache server however, I could see that the workload are evenly distributed

kubectl apply -f <https://k8s.io/examples/application/php-apache.yaml>

kubectl run -i --tty load-generator --rm --image=busybox --restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done" (this is to trigger the scaling)

A computer screen shot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

# CICD with Bitbucket pipeline

Note: The Bitbucket pipeline requires the Bitbucket environment

The build step is triggered when there is a file change in the folder below:

        filter:

          includes:

            - "flask-app-deployment/\*"

Bug: Pulling the image of the same version will not overwrite the existing one.

A screenshot of a computer

Description automatically generated

The deploy step:

A screenshot of a computer screen

Description automatically generated

After deployment:



# Argo CD

Argo CD is deployed to the EKS cluster by following the steps below:

1. Argo CD only works with yaml manifest files. I need to translate the terraform deployment code in the main.tf. The easiest way to do that is to run below command to export the flask deployment:

**kubectl get deploy -n flask -o yaml > flask-app-deployment/app\_manifests/flask-deployment.yaml**

Note: The file exported contains some additional status information, I need to do some cleanup so that the file can be deployed. Try running “**kubectl apply -f flask-deployment.yaml**” until you get “deployment.apps/flask-deployment configured” without error.

1. Install argocd to the EKS cluster:

**kubectl create namespace argocd**

**kubectl apply -n argocd -f** [**https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml**](https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml)

A screen shot of a computer

Description automatically generated

1. Start Argo CD on your local computer and get the passport of “admin” to login the console:

**kubectl port-forward -n argocd svc/argocd-server 8080:443**

**The URL is 127.0.0.1:8080**

**kubectl -n argocd get secret argocd-initial-admin-secret -o yaml**

**A computer screen with white text

Description automatically generated**

**echo ZkJJTEVQM3oxZDlwOXVzLQ== | base64 –decode**

**A black background with white letters and numbers

Description automatically generated**

1. Link the flask GitHub repository to argocd:

**kubectl apply -f flask-app-deployment/argocd/flask-argo-application.yaml**

**A screenshot of a computer

Description automatically generated**

1. Update the image version in the **flask-deployment.yaml** file and verify the change (tested twice with version 0.0.4 and 0.0.3):

**A screenshot of a chat

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

A screenshot of a chat

Description automatically generated

A screenshot of a computer

Description automatically generated

# Issue faced and solutions applied

## Can’t connect to the API group after creating the EKS cluster

**Error:**

C:\Users\bsun\Projects>kubectl get svc

E1124 17:59:53.250966 25816 memcache.go:265] couldn't get current server API group list: Get "https://90F2F70114A39F575C0197231A604818.gr7.eu-west-1.eks.amazonaws.com/api?timeout=32s": dial tcp 10.111.162.76:443: i/o timeout

E1124 18:00:23.260130 25816 memcache.go:265] couldn't get current server API group list: Get "https://90F2F70114A39F575C0197231A604818.gr7.eu-west-1.eks.amazonaws.com/api?timeout=32s": dial tcp 10.111.162.76:443: i/o timeout

**Troubleshooting:**

* Check the firewall (AWS security groups) and make sure they are not blocking my IP.
* Since the EKS cluster is deployed in private subnets, the endpoint access is set to “private” meaning that it could only be accessible from within the VPC.

**Solutions:**

Create an EC2 jumpbox in the same private subnet of the EKS cluster and confirmed that I could access the cluster

Per the document below, change the endpoint access from ”private” to “Public and private”.

A screenshot of a computer screen

Description automatically generated

## Load balancer is not created and terraform creating ingress timeout

**Error:**

A black screen with white text

Description automatically generated

**Troubleshooting:**

kubectl describe ingress sample-application-ingress -n sample-application



**Solutions:**

The errorindicates that the alb couldn’t find any subnet match. Per the document below, I choose to specify subnets as annotations

<https://docs.aws.amazon.com/eks/latest/userguide/alb-ingress.html>

A screenshot of a computer

Description automatically generated

<https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.2/guide/ingress/annotations/>

A screenshot of a computer

Description automatically generated

## EKS failed to pull the docker image from github

**Error:**

A screenshot of a computer program

Description automatically generated

**Troubleshooting:**

The error indicates that EKS failed to fetch the token and therefore not authorized to pull the image

**Solution:**

1. Change the flask image to be public:  
   A screenshot of a computer

   Description automatically generated
2. Create a personal access token and follow the steps in the document below to generate a dockerconfigjson secret. Apply the secret and add the “image\_pull\_secret” to the deployment terraform code:

A screenshot of a computer

Description automatically generated

## Failed with “502 Bad Gateway” when testing the endpoint:

**Error:**

A screenshot of a computer

Description automatically generated

**Troubleshooting:**

Check the status of the load balancer target group and find out that all instances are “unhealthy”. This means the health check failed.

A screenshot of a computer

Description automatically generated

In this case, the “Traffic port” of flask application is 5000. The deployment log confirms that:

kubectl logs -n sample-application sample-application-deployment-79c75f45b8-9rqv5

\* Debug mode: off

WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.

\* Running on all addresses (0.0.0.0)

\* Running on http://127.0.0.1:**5000**

\* Running on http://100.64.0.241:**5000**

Press CTRL+C to quit

**Solutions:**

I found from the terraform code that the container port and target\_port were both 80. I updated them to 5000 and the issue was fixed

A screen shot of a computer

Description automatically generated

A computer screen with text and numbers

Description automatically generated

## Pods reboot over and over again

**Error:**

A number on a black background

Description automatically generated

**Troubleshooting:**

Similar to Issue 4, the port 80 in the error got my attention. Plus, the “initial\_delay\_seconds” and “period\_seconds” setting are way too short.

**Solutions:**

**I updated the terraform code and apply the change**

~ liveness\_probe {

~ initial\_delay\_seconds = 3 -> 10

~ period\_seconds = 3 -> 10

# (3 unchanged attributes hidden)

~ http\_get {

~ port = "80" -> "5000"

# (2 unchanged attributes hidden)