# Zocket Web Application Deployment Guide

# Kubernetes deployment

## Overview

This guide provides detailed instructions for deploying the Zocket web application using Kubernetes. The application comprises a frontend, backend, and MongoDB database. We'll be using Docker for containerization, Terraform for infrastructure provisioning, and Kubernetes for orchestration.

## Prerequisites

AWS CLI configured with appropriate permissions

Terraform installed

Kubernetes cluster (EKS) up and running

kubectl CLI installed

docker

**Step -1** Docker Images:

So first things first, I built two Docker images. One for our backend and another for the frontend. I used these commands to build and push them to Docker Hub:

docker build -t itskumaran/backend:v1 .

docker push itskumaran/backend:v1

docker run -d -P itskumaran/frontend:v1 .

docker push itskumaran/frontend:v1

**Step-2** Creating EKS Cluster Setup using iac:

Now, moving on to our AWS setup. I set up an EKS cluster, and I named it EKS\_CLOUD. To do this, I used some Terraform modules that contain:

provider.tf

main.tf

variable.tf

var.tf

These files hold all our infrastructure code and configurations. The cool thing is, with these modules, we can easily create more clusters just by tweaking the variables a bit.

**Terraform Operations:**

For managing all this, I used Terraform. It's super handy for infrastructure as code. Here are the commands I ran:

terraform init

terraform apply

terraform plan

terraform destroy

**Secure State Storage:**

Last but not least, I made sure to store our infrastructure state securely. I saved it in an S3 bucket. And to prevent any conflicts or issues, I used DynamoDB as a locking mechanism. It's a neat way to make sure everything runs smoothly and safely.

**Step-3** Setting Up EKS Cluster:

First off, after setting up the EKS cluster using Terraform, the next step is configuring it with kubectl. This lets us interact with our cluster. I also created a specific namespace named zockets to keep things organized.

kubectl create namespace zockets

**Step-4** Deploying Applications:

Next, we deployed our applications. For the backend, frontend, and MongoDB, I used the respective YAML files:

its all inthe folder k8s manifests

backend\_deployment.yaml

frontend\_deployment.yaml

-mongo

deployment.yaml

secrets.yaml

**Here are the commands to deploy them:**

kubectl apply -f backend\_deployment.yaml -n zockets

kubectl apply -f frontend\_deployment.yaml -n zockets

cd mongo/

ls

kubectl apply -f deployment.yaml -n zockets

kubectl apply -f secrets.yaml -n zockets

**Ingress Configuration:**

To handle external traffic and route it to the right services, I set up an Ingress using papi\_ingress\_nginx.yaml. This makes sure our application is accessible from the outside.

kubectl apply -f papi\_ingress\_nginx.yaml -n zockets

kubectl create namespace zockets

kubectl apply -f nginx-ingress-hpa.yaml

I have added nginx-ingress-hpa.yaml to maintain scaling [assign req roles and permission in iam roles ec2 ]

The HorizontalPodAutoscaler (HPA) automatically adjusts the number of pods in a Kubernetes Deployment based on the observed CPU utilization (or other metrics) of the pods.

This means that the HPA will adjust the number of pods to try to keep the CPU usage of the existing pods at around 80%. If the CPU usage goes above 80%, the HPA will increase the number of pods, and if it goes below 80%, the HPA will decrease the number of pods.

**Step5: ArgoCD Installation**

For streamlined deployments and CI/CD purposes, I installed ArgoCD. It's a fantastic tool for managing Kubernetes deployments. ArgoCD is like a helpful dashboard for our Kubernetes setup. It gives us an easy way to see and manage our apps without typing lots of commands. Once we have mapped our repo to ArgoCD set up, we can access it either using a special web address (load balancer) or directly through our server's IP (node port). Inside ArgoCD, I've added a task to deploy our apps. This task checks our setup against the plan we've made (in the manifest file). If any updating or fixing is needed, ArgoCD does it automatically. It keeps our apps running smoothly without us having to do everything by hand.

I have installed Prometheus and Grafana using helm for monitoring purpose. Dashboard pod/cpu default metrics, application logs, events I used argocd.