End-to-End Kubernetes and AWS Infrastructure Deployment with Terraform: A Static Website on NGINX

Project Overview

This project demonstrates the deployment of a static website using AWS infrastructure provisioned via Terraform, containerized using Docker, and deployed on a Kubernetes cluster using Minikube. The NGINX web server is used to serve the static website, and the setup is exposed using Kubernetes Ingress for external access.

Goals

- Infrastructure:

Provision an AWS VPC, Subnets, EC2 instances, and an Elastic Load Balancer (ELB) using Terraform.

- Application:

Create a static website, serve it using NGINX, and dockerize the application.

- Deployment:

Deploy the containerized application using Kubernetes (Minikube) with 4 replicas, exposing the service using ClusterIP and Ingress.

Technologies Used

- Terraform: For Infrastructure as Code (IaC) on AWS.
- AWS: VPC, Subnets, EC2, ELB.
- Docker: Containerization of the static website.
- Kubernetes (Minikube): Container orchestration for managing the application.
- NGINX: Web server for serving the static content.

Infrastructure Setup with Terraform

Description:

We will provision an AWS Virtual Private Cloud (VPC) with two subnets spread across two Availability Zones. Additionally, EC2 instances and an Elastic Load Balancer (ELB) will be created to distribute traffic.

Steps:

1. Install Terraform: if you haven't already.
   - Terraform Version: 1.4+
   - AWS Provider Plugin
2. Terraform Code: Create a main.tf file for the following setup:
   - VPC Creation
   - Subnet Allocation (2 AZs)
   - EC2 Instances and Security Groups
   - Elastic Load Balancer (ELB) setup
   - Ensure to manage Terraform state locally using terraform.tfstate file.

Terraform Configuration (Sample):

```
provider "aws" {
    region = "us-east-1"
    access_key = "******"
    secret_key = "******"

}

# VPC
resource "aws_vpc" "main" {
```

```
    cidr_block = "10.0.0.0/16"

}

#subnets
resource "aws_subnet" "subnet_a" {
    vpc_id = aws_vpc.main.id
    cidr_block = "10.0.1.0/24"
    availability_zone = "us-east-1a"

}

resource "aws_subnet" "subnet_b" {
    vpc_id = aws_vpc.main.id
    cidr_block = "10.0.2.0/24"
    availability_zone = "us-east-1b"
}

#internet Gateway

resource "aws_internet_gateway" "igw" {
    vpc_id = aws_vpc.main.id

}

# Route table

resource "aws_route_table" "main_rt" {
    vpc_id= aws_vpc.main.id
    route {
        cidr_block= "0.0.0.0/0"
        gateway_id = aws_internet_gateway.igw.id
    }

}

# secutiry group
resource "aws_security_group" "web_sg" {
    vpc_id = aws_vpc.main.id
    ingress {
        from_port = 80
        to_port = 80
        protocol = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
```

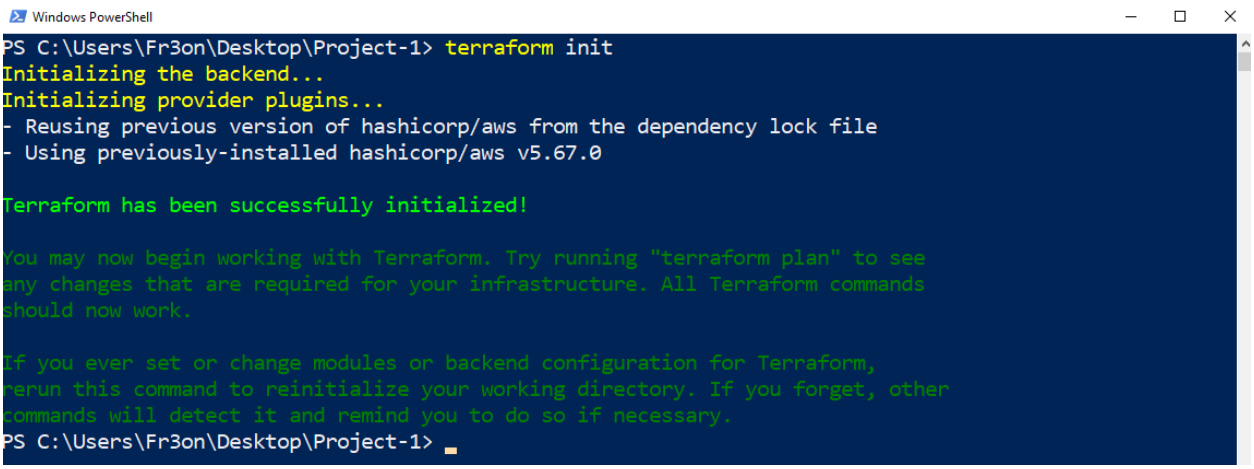
```
    }
    ingress {
        from_port = 22
        to_port = 22
        protocol = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
    }

}

# ELB
resource "aws_lb" "app_lb" {
    name = "app-load-balancer"
    internal = false
    load_balancer_type = "application"
    subnets = [aws_subnet.subnet_a.id, aws_subnet.subnet_b.id]
    security_groups = [aws_security_group.web_sg.id]


}
```

Commands:

- Initialize and apply Terraform configuration:

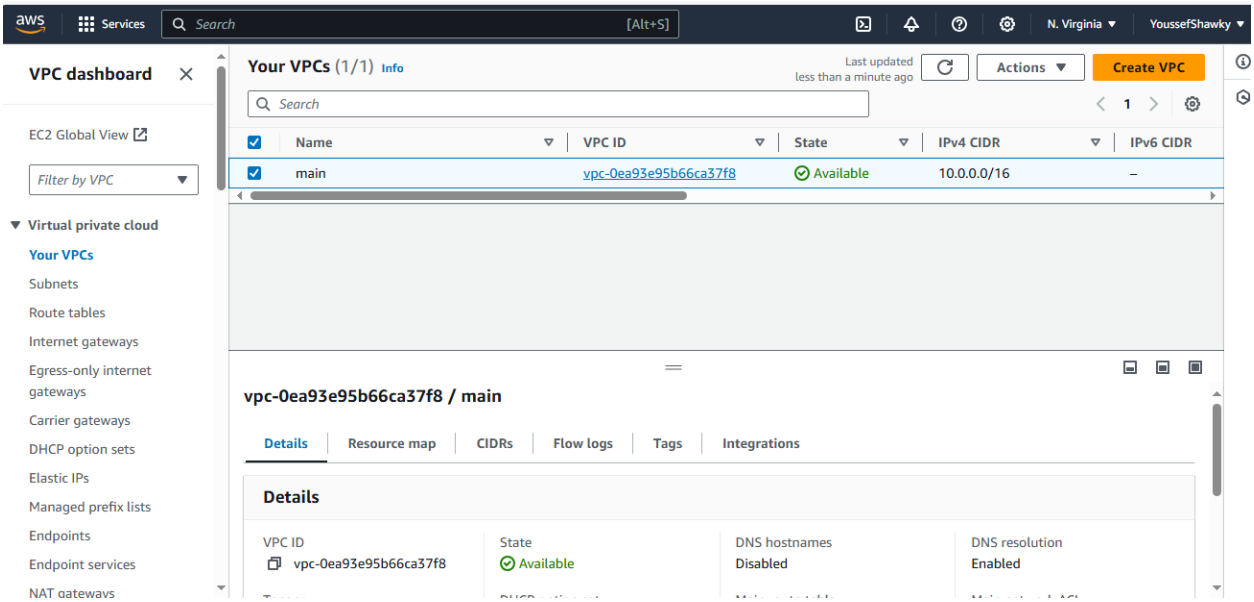
terraform init

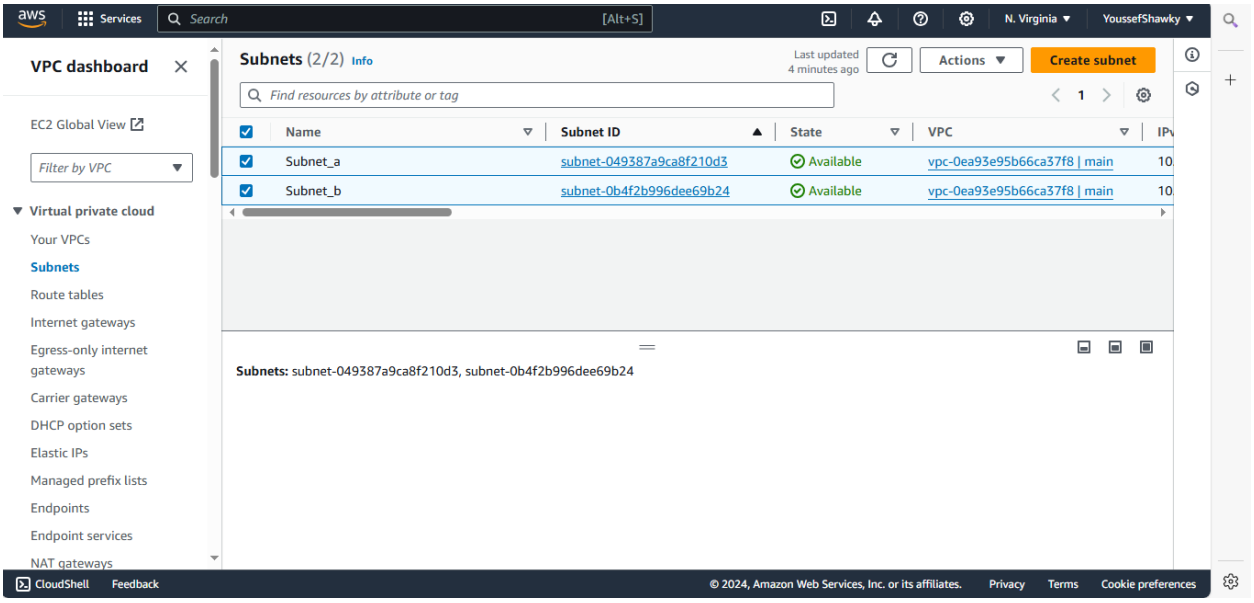
terraform apply

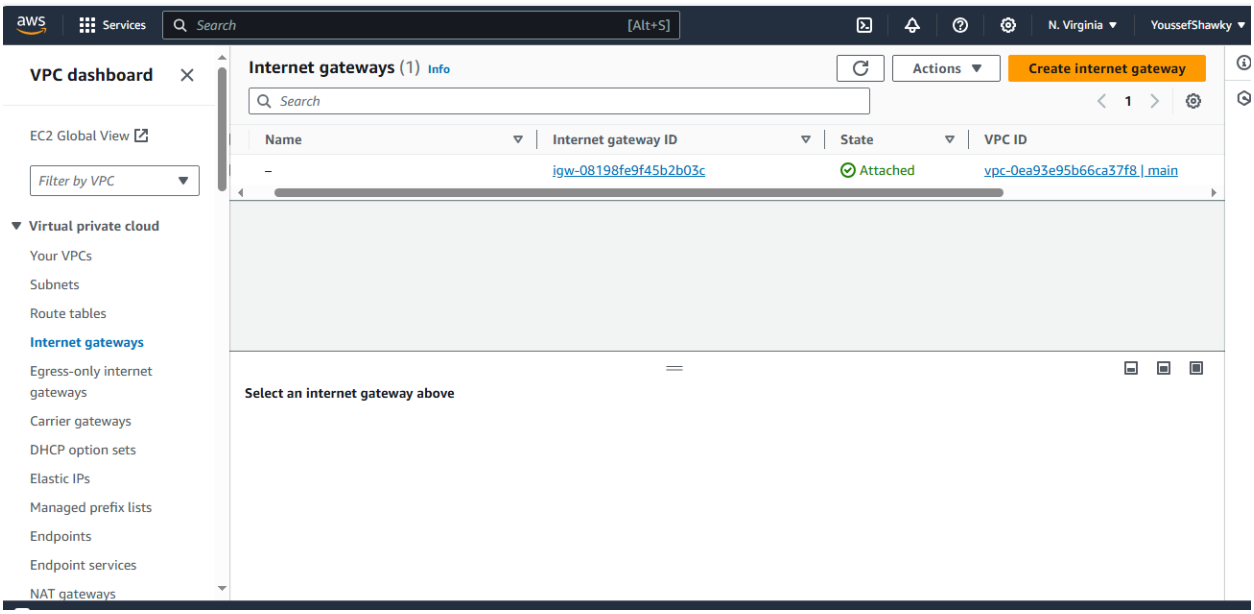


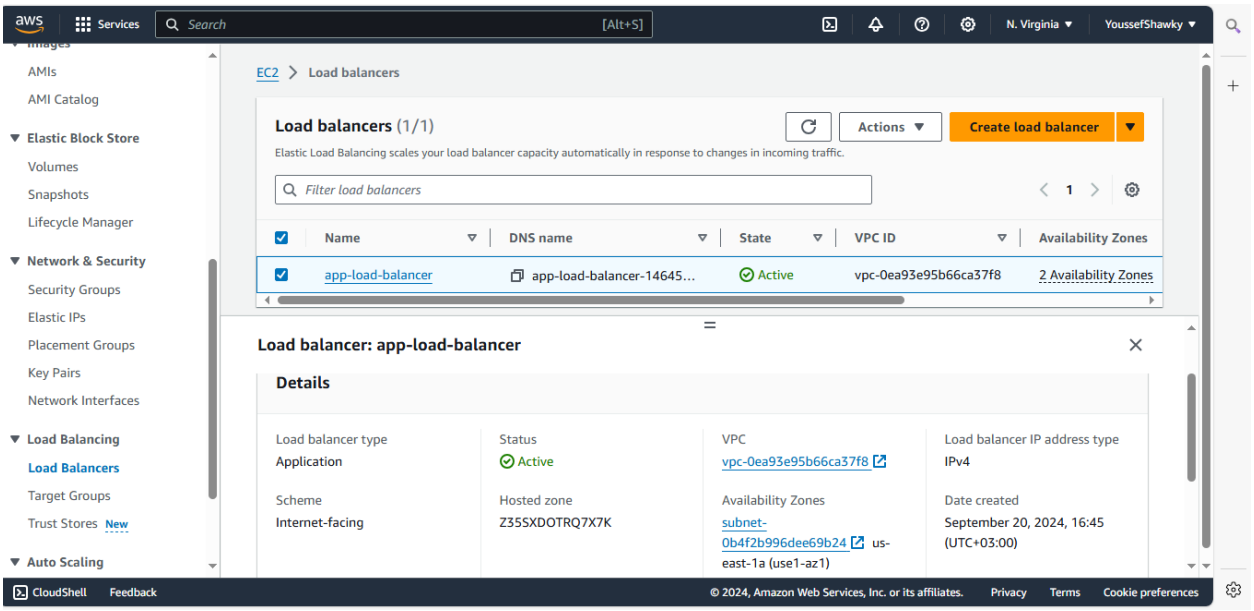
```
PS C:\Users\Fr3on\Desktop\Project-1> terraform apply

Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_internet_gateway.igw will be created
  + resource "aws_internet_gateway" "igw" {
      + arn      = (known after apply)
      + id       = (known after apply)
      + owner_id = (known after apply)
      + tags_all = (known after apply)
      + vpc_id   = (known after apply)
    }

  # aws_lb.app_lb will be created
  + resource "aws_lb" "app_lb" {
      + arn                                         = (known after apply)
      + arn_suffix                                  = (known after apply)
      + client_keep_alive                           = 3600
      + desync_mitigation_mode                      = "defensive"
      + dns_name                                    = (known after apply)
      + drop_invalid_header_fields                  = false
      + enable_deletion_protection                  = false
```

AWS console verification:




aws
Services
Search
[Alt+S]
N. Virginia
YoussefShawky
VPC dashboard
EC2 Global View
Filter by VPC
Virtual private cloud
Your VPCs
Subnets
Route tables
Internet gateways
Egress-only internet gateways
Carrier gateways
DHCP option sets
Elastic IPs
Managed prefix lists
Endpoints
Endpoint services
NAT gateways
Subnets (2/2) Info
Last updated 4 minutes ago
Actions
Create subnet
Find resources by attribute or tag
1
Name
Subnet ID
State
VPC
Subnet_a
subnet-049387a9ca8f210d3
Available
vpc-0ea93e95b66ca37f8 | main
Subnet_b
subnet-0b4f2b996dee69b24
Available
vpc-0ea93e95b66ca37f8 | main
Subnets: subnet-049387a9ca8f210d3, subnet-0b4f2b996dee69b24
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


aws
Services
Search
[Alt+S]
N. Virginia
YoussefShawky
VPC dashboard
EC2 Global View
Filter by VPC
Virtual private cloud
Your VPCs
Subnets
Route tables
Internet gateways
Egress-only internet gateways
Carrier gateways
DHCP option sets
Elastic IPs
Managed prefix lists
Endpoints
Endpoint services
NAT gateways
Internet gateways (1) Info
Actions
Create internet gateway
Search
1
Name
Internet gateway ID
State
VPC ID
–
igw-08198fe9f45b2b03c
Attached
vpc-0ea93e95b66ca37f8 | main
Select an internet gateway above


aws
Services
Search
[Alt+S]
N. Virginia
YoussefShawky
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
Load Balancing
Load Balancers
Target Groups
Trust Stores New
Auto Scaling
EC2 > Load balancers
Load balancers (1/1)
Actions
Create load balancer
Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.
Filter load balancers
Name
DNS name
State
VPC ID
Availability Zones
app-load-balancer
app-load-balancer-14645...
Active
vpc-0ea93e95b66ca37f8
2 Availability Zones
Load balancer: app-load-balancer
Details
Load balancer type
Application
Status
Active
VPC
vpc-0ea93e95b66ca37f8
Load balancer IP address type
IPv4
Scheme
Internet-facing
Hosted zone
Z35SXDOTRQ7X7K
Availability Zones
subnet-0b4f2b996dee69b24 us-east-1a (use1-az1)
Date created
September 20, 2024, 16:45 (UTC+03:00)
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Building the Docker Image (Static Website + NGINX)

Description:

This step involves creating a simple static website (with home, info, and content pages), serving it with NGINX, and building a Docker image.

Steps:

1. **Create a Static Website:**
   - Files:
     - index.html (Home page)
     - info.html (Info page)
     - content.html (Content page)
2. **Create a Dockerfile** for NGINX:
   - This will copy your static website into NGINX's web root.

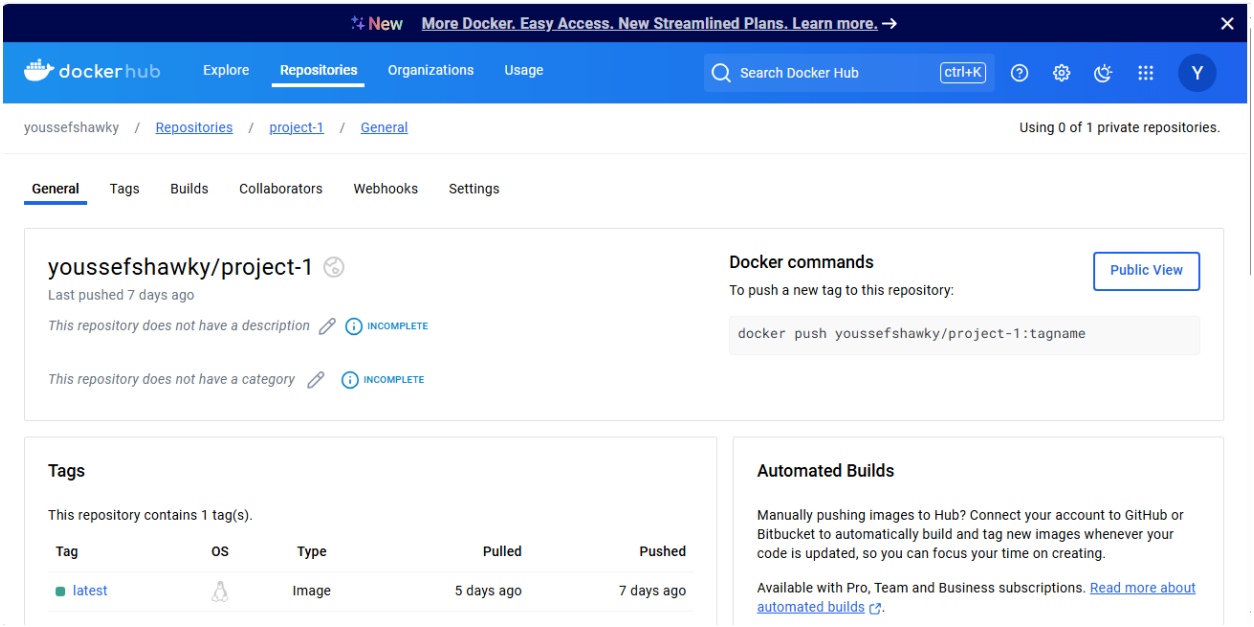
Dockerfile:

```
FROM nginx:alpine
COPY ./static-site /usr/share/nginx/html
EXPOSE 80
```

3. Build and Push the Docker Image:

docker build -t youssefshawky/project-1 .

docker push youssefshawky/project-1


New More Docker. Easy Access. New Streamlined Plans. Learn more. →
docker hub
Explore
Repositories
Organizations
Usage
Search Docker Hub
ctrl+K
Y
youssefshawky / Repositories / project-1 / General
Using 0 of 1 private repositories.
General
Tags
Builds
Collaborators
Webhooks
Settings
youssefshawky/project-1
Last pushed 7 days ago
This repository does not have a description
INCOMPLETE
This repository does not have a category
INCOMPLETE
Docker commands
Public View
To push a new tag to this repository:
docker push youssefshawky/project-1:tagname
Tags
This repository contains 1 tag(s).
Tag
OS
Type
Pulled
Pushed
latest
Image
5 days ago
7 days ago
Automated Builds
Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.
Available with Pro, Team and Business subscriptions. Read more about automated builds.

Kubernetes Setup (Minikube)

Description:

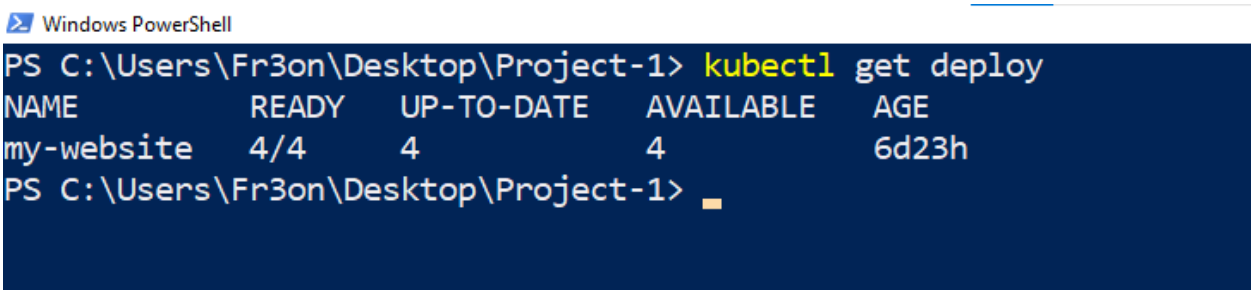
Deploy the static website Docker image on a Kubernetes cluster using Minikube. Create a deployment with 4 replicas, expose it using ClusterIP, and use Ingress to expose it externally.

Steps:

```yaml
# Creat a deployment with 4 replicas

apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-website
spec:
  replicas: 4
  selector:
    matchLabels:
      app: my-website
  template:
    metadata:
      labels:
        app: my-website
    spec:
      containers:
      - name: my-website
        image: youssefshawky/project-1:latest
        ports:
        - containerPort: 80

---

# Exposing the deployment

apiVersion: v1
kind: Service
metadata:
  name: my-website
spec:
  selector:
    app: my-website
  ports:
```

```
  - protocol: TCP
    port: 80
    targetPort: 80
  type: ClusterIP

---
# Set the ingress and expose the front end service to mapped to the backend
service

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-website-ingress
spec:
  rules:
    - host: mywebsite.io
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: my-website
                port:
                  number: 80
```

Minikube kubectl command verification:



```
PS C:\Users\Fr3on\Desktop\Project-1> kubectl get svc
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP   6d23h
my-website    ClusterIP   10.97.185.41    <none>        80/TCP    6d23h
PS C:\Users\Fr3on\Desktop\Project-1>
```

Windows PowerShell

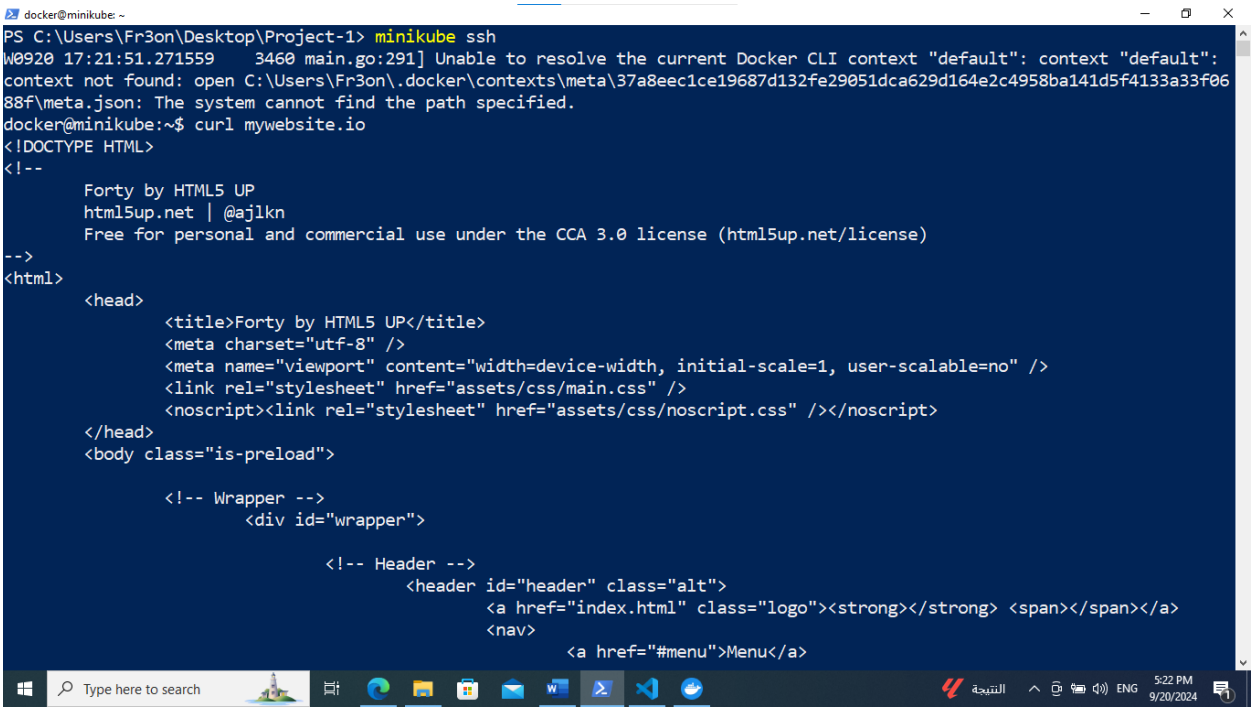
```
PS C:\Users\Fr3on\Desktop\Project-1> kubectl get ingress
NAME                 CLASS   HOSTS          ADDRESS        PORTS   AGE
my-website-ingress   nginx   mywebsite.io   192.168.49.2   80      6d23h
PS C:\Users\Fr3on\Desktop\Project-1>
```

Troubleshooting Section

Issues Faced:

- Ingress Not Working Externally:

Despite applying the Ingress configuration, the website wasn't accessible from the browser on the local machine. When running curl inside the Minikube VM (via SSH), the service was reachable, but not externally.



What I’ve Tried So Far:

- Verified the Ingress controller and services are running as expected.
- Enabled the Minikube Ingress addon.
- Tried using port-forwarding and `minikube tunnel`, but the Ingress remains inaccessible externally.
- Configured custom routes to the Minikube subnet, but no luck.

Interestingly, pinging from Minikube to my local machine works, but not the other way around.

My Thoughts:

Minikube seems to run as a "closed cluster," possibly by design. I suspect this networking isolation is why I can’t access the Ingress externally.

Future Work

CI/CD Pipelines:

- Jenkins Setup:

Future improvements include automating the build and deployment process using Jenkins. This would allow continuous integration of code changes and their deployment to the Kubernetes cluster.