Machine Learning Application Deployment using Cloud-Native Tools

Team: Jack, Pranav, Lasya, Meghshanth

Problem Statement

When it comes to deploying a Machine Learning Model, there are quite a few challenges. Some crucial ones include: how to scale the model, how the model can interact with different services within or outside the application, and how to programmatically execute repetitive operations.

Solution

In our project, we leveraged cloud ready technologies such as containerization, virtualization, orchestration, and cloud networking to build a customer facing scalable machine learning application.

Motive Behind the Project

- Implementation of System and OS-level Virtualization technologies in real-world applications.
- Build a Machine Learning Application on a Scalable Infrastructure.
- Explore the Cloud-Native Technologies and Tools to build, run, deploy, and manage the Machine Learning Application.

Technologies and Tools Used

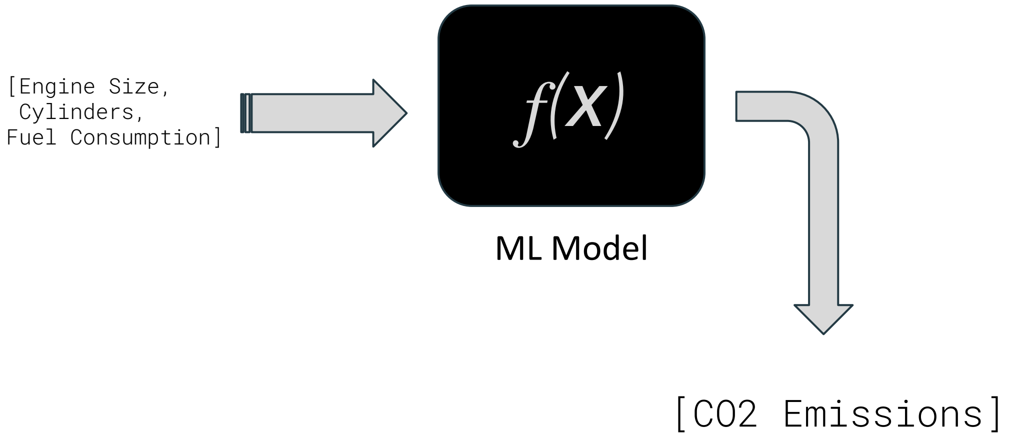
- Flask Web Framework
- Gunicorn Server
- Docker
- AWS Elastic Kubernetes Service (AWS EKS)
- Amazon EC2
- Amazon Elastic Load Balancer (AWS ELB)
- *Attempted* Google DNS “knowyourco2.com”

gunicorn

Flask

Machine Learning Model

- Simple Linear Regression Model to predict CO2 emission from vehicles
- The project uses LinearRegression() class from Sklearn library
- Implemented linear regression on 3 input variables



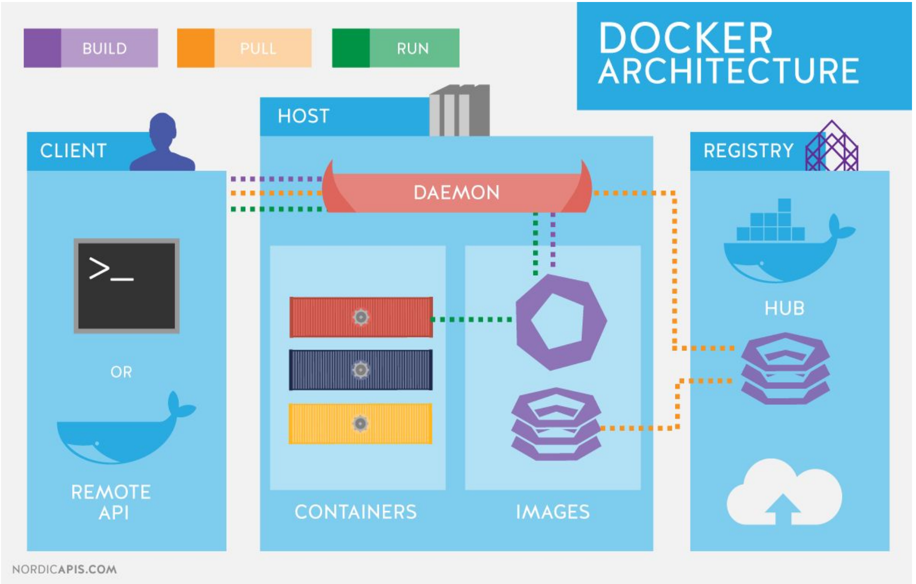
Flask and Gunicorn

- Flask-Simple, easy to use Micro web framework
- Python module used to develop web applications easily
- Flask creates a web server locally allowing for easy testing.
- Gunicorn - A scalable web application server is used to run the application into production

```python
#import libraries
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle

#Initialize the flask App
app = Flask(__name__)

#default page of our web-app
@app.route('/')
def home():
    return render_template('index.html')
```

Containerization using Docker

- Platform for building, running, and shipping applications with their dependencies.
- Dockerfile is used as a template to build the docker image.
- Built docker image of the flask-based application.
- Used Docker Hub as an image repository to push and store the docker image.



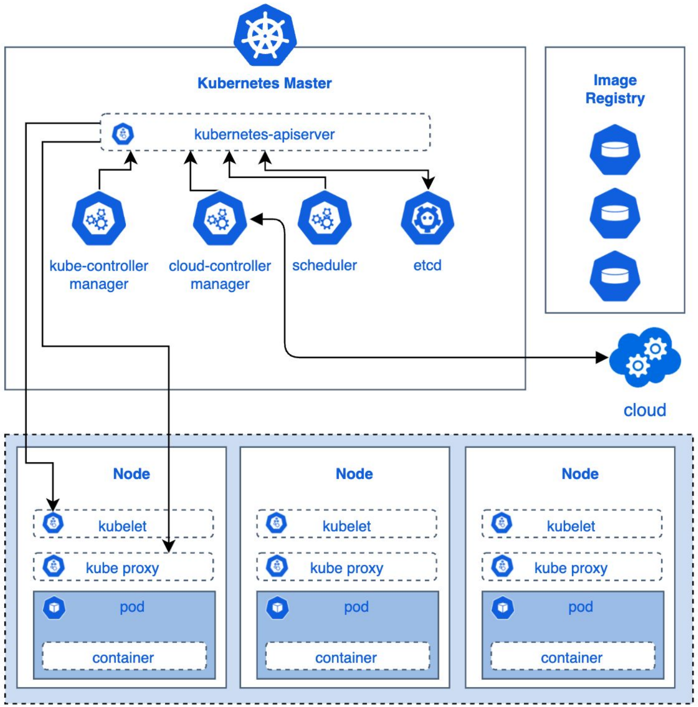
Dockerfile

```
FROM python:3.9.15-slim

RUN pip install pipenv

WORKDIR /app

COPY ["Pipfile", "Pipfile.lock", "./"]

RUN pipenv install --deploy --system

COPY . .

ENTRYPOINT ["gunicorn", "--bind=0.0.0.0:9696", "app:app"]
```

Kubernetes

- Kubernetes is an open source container orchestration engine
- Used for automating deployment, scaling, and management of containerized applications.
- Following are the few of the components of Kubernetes -

1. Kube-ApiServer
2. Etcd
3. Kube-Scheduler
4. Kube-Control-Manager
5. Cloud-Control-Manager



Kubernetes Deployment

- A *Deployment* provides declarative updates for Pods and ReplicaSets.
- *Pods* -Smallest deployable units of computing that you can create and manage in Kubernetes.
- A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time.

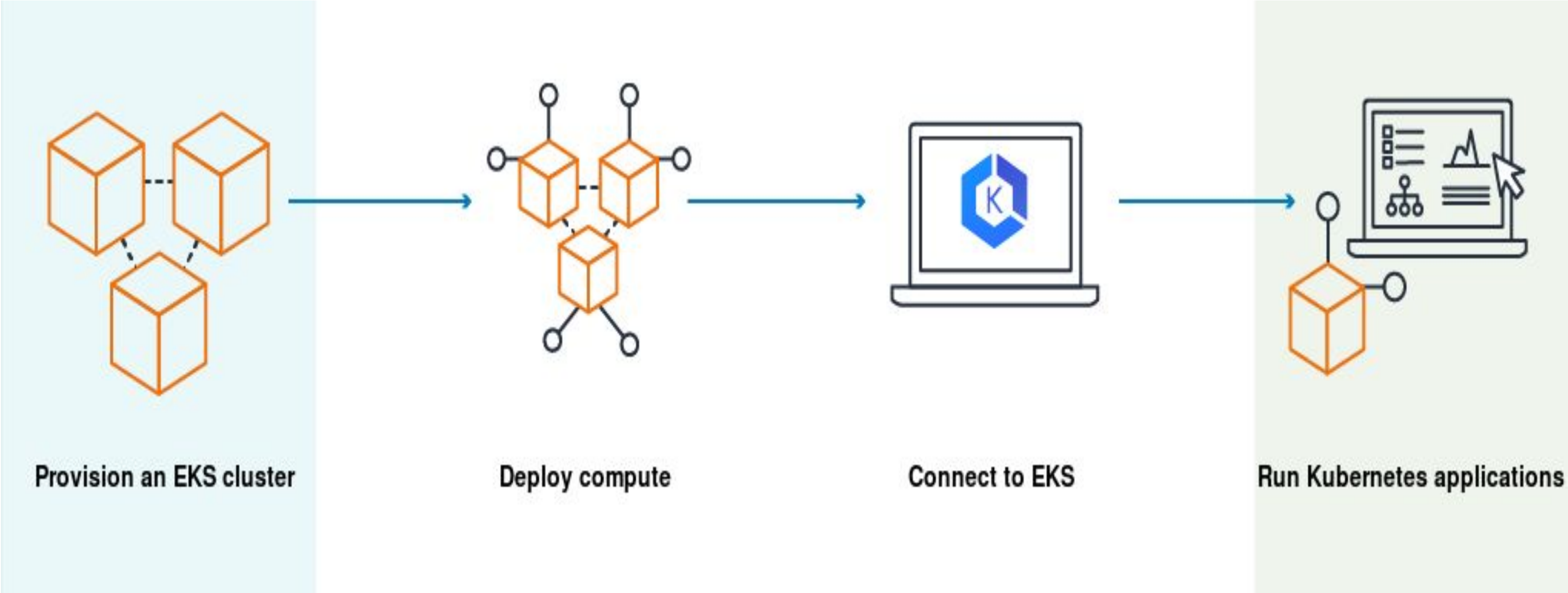
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-app-deployment
spec:
  selector:
    matchLabels:
      app: flask-ml-app
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: flask-ml-app
    spec:
      containers:
      - name: guniorn-server
        image: pranav2306/ml-model-app:1.0
        ports:
        - containerPort: 9696
```

Kubernetes Service - LoadBalancer

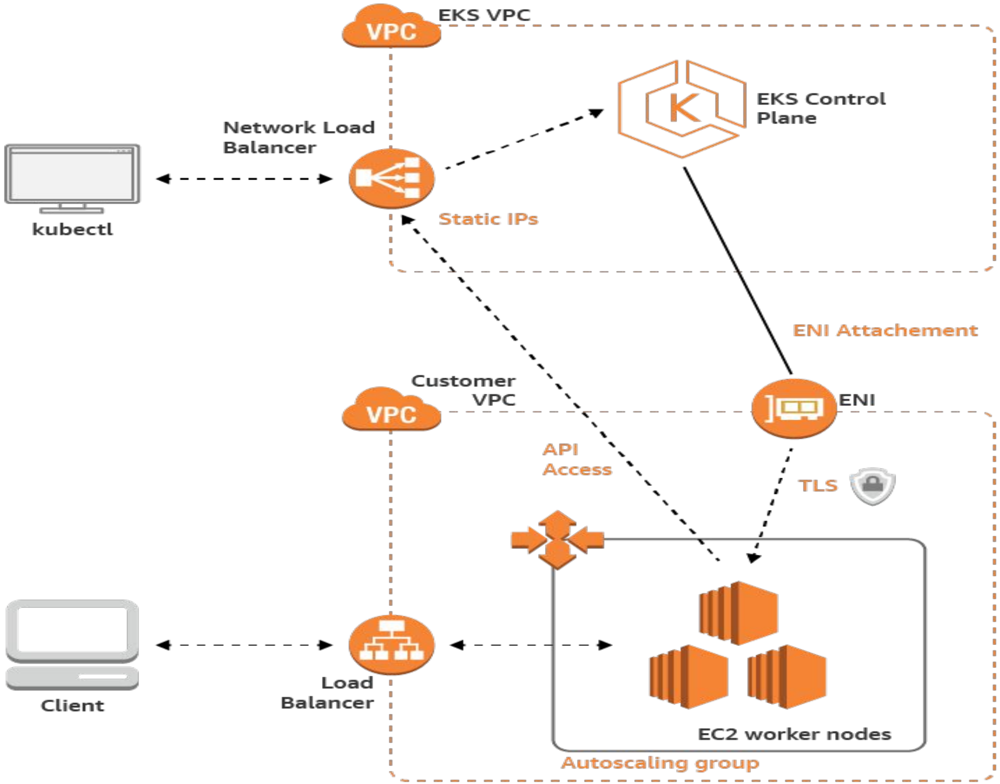
- When creating a Service, you have the option of automatically creating a cloud load balancer.
- This provides an externally-accessible IP address that sends traffic to the correct port on your cluster nodes, *provided your cluster runs in a supported environment and is configured with the correct cloud load balancer provider package.*

```yaml
apiVersion: v1
kind: Service
metadata:
  name: ml-model-service
spec:
  selector:
    app: flask-ml-app
  ports:
    - port: 80
      targetPort: 9696
  type: LoadBalancer
```

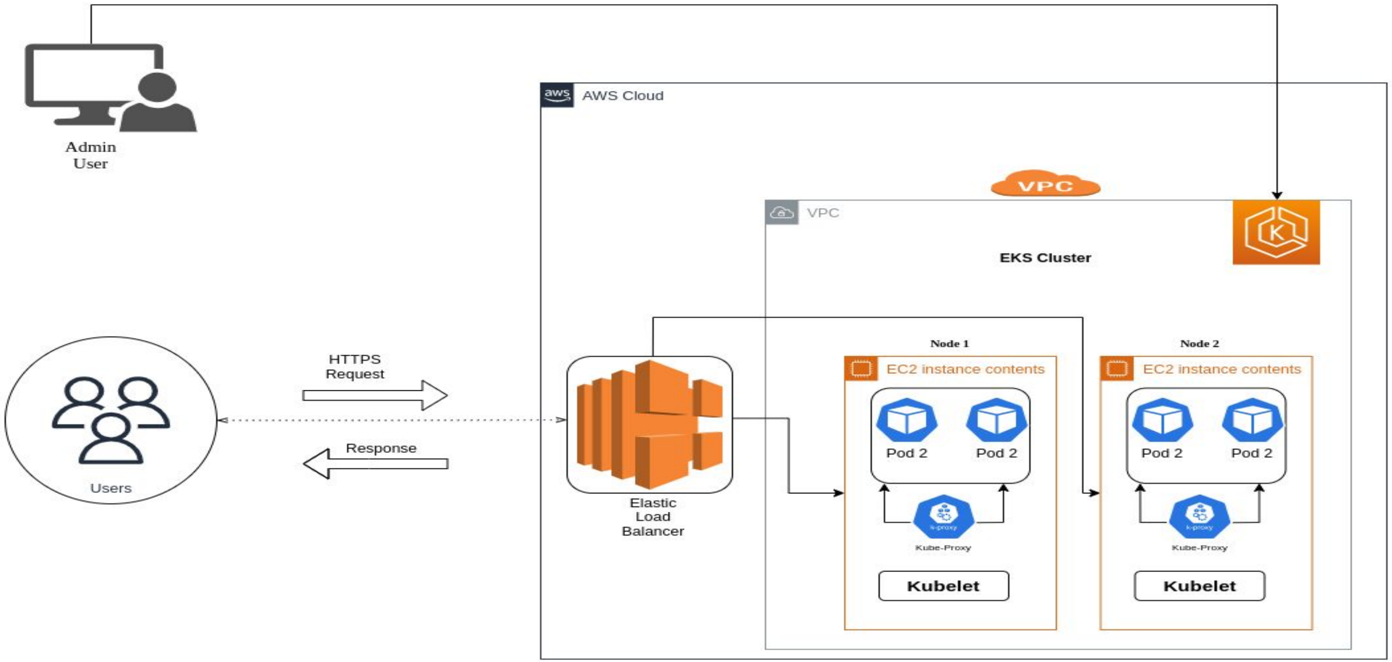
Elastic Kubernetes Service (EKS)



General EKS Cluster Architecture



Project Architectural Diagram



Future Scope

1. Configure Prometheus to monitor, trace, and analyze the kubernetes cluster.
2. Configure Grafana to visualize the charts, graphs, and alerts of the kubernetes cluster fetched by prometheus.
3. Provision and manage Kubernetes clusters on AWS EKS and interact with your cluster using the Kubernetes Terraform provider.