**Solution details**

This solution uses a scikit-learn model trained on the UCI Wine Quality dataset and served via FastAPI. This is a multivariate data set and is related to red and white variants of the Portuguese “Vinho Verde” wine. Input variables are physicochemicals such as pH, chlorides, residual sugar etc. Output variable is a quality score between 0 and 10. The services are deployed on a Kubernetes cluster. Metrics are collected using Prometheus and the results are visualised in a Grafana dashboard.

How it works at a hight level

1. Create a containerized REST service to expose the model via a prediction endpoint.
2. Instrument the server to collect metrics which are exposed via a separate metrics endpoint.
3. Deploy Prometheus to collect and store metrics.
4. Deploy Grafana to visualize the collected metrics.
5. Finally, we’ll simulate production traffic using Locust so that we have some data to see in our dashboard.

Deploying the model with FastAPI

1. All files contained within the model/ directory.
2. *Train.py* – simple script to produce a serialized model artifact.
3. *App/api*.py *–* defines a few routes for our model service including a model prediction endpoint and a health-check endpoint.
4. *App/schemas*.py *–* defines the expected schema for the request and response bodies in the model prediction endpoint.
5. *Dockerfile* lists the instructions to package our REST server as a container.
6. Server is deployed on our Kubernetes cluster using the manifest defined in Kubernetes/models/.

Instrumenting the service with metrics

1. Goal – capture metrics and expose this data via a /metrics endpoint on our server – this is done using *Prometheus-fastapi-instrumentator*, which is a library that includes FastAPI middleware that collects metrics for each request and exposes the metric data to a specified endpoint.
2. After deploying model service on the Kubernetes cluster, port forward to a pod running the server and check out the metrics endpoints running at [127.0.0.1:3000/metrics](http://127.0.0.1:3000/metrics).

Capturing metrics with Prometheus

1. Once metrics are exposed at the specified endpoint, Prometheus is used to collect and store this data.
2. *helm* is used to deploy Prometheus onto our Kubernetes cluster.
3. Prometheus will scrape the metric data at the endpoints at a specific interval (15 sec default).
4. This metric data can be queries by other services which make requests to the HTTP serve

Visualising results in Grafana

1. Grafana hosted on 127.0.0.1:8000 - pulls metric data from Prometheus and visualises it.
2. Queries are made to the Prometheus data source using query language called PromQL.
3. Pre-built dashboard is in dashboards/model.json.

Simulating production traffic with Locust

1. Use Locust, a python load testing framework, to make requests to the model service and simulate production traffic. This behaviour defined in *load\_tests/locustfile.py* , where we define three tasks.
   1. Request to our health check endpoint.
   2. Choose a random example from the wine quality dataset and make a request to our prediction service.
   3. Choose a random example from the wine quality dataset, corrupt the data, and make a bad request to our prediction service.

**Setup instructions**

1. Install kubectl, helm, docker, and minikube.
2. Spin up a Kubernetes cluster on local machine using minikube
   1. *minikube start -–driver=docker -–memory 4g –-nodes 2*
3. Deploy Prometheus and Grafan onto the cluster using helm
   1. *kubectl create namespace monitoring*
   2. *helm repo add prometheus-community* [*https://prometheus-community.github.io/helm-charts*](https://prometheus-community.github.io/helm-charts)
   3. *helm install prometheus-stack prometheus-community/kube-prometheus-stack -n monitoring*
4. Connect to the Grafana dashboard
   1. *kubectl port-forward svc/prometheus-stack-grafana 8000:80 -n monitoring*
5. Visit <http://127.0.0.1:8000/>
6. Login with credentials – username: admin, password: prom-operator
7. Import the model dashboard – click ‘+’ sign and select “import” – import the JSON file defined in dashboards/model.json.
8. Deploy the model
   1. *kubectl apply -f kubernetes/models/*
9. Begin the load test and simulate production traffic
   1. *kubectl apply -f kubernetes/load\_tests/*

**Tear down instructions**

1. Stop the model REST server
   1. *kubectl delete -f kubernetes/models/*
   2. *kubectl delete -f kubernetes/load\_tests/*
2. Remove the Prometheus stack
   1. *helm uninstall prometheus-stack -n monitoring*