Report with Software Requirement Specification

For

Inter Hall Open Soft

Version: 1.0.0

Prepared by:-
Team 333

Date:31-03-2022

Table of Contents

1. **Introduction**
   1.1 Purpose
   1.2 Intended Audience and Reading Suggestions
   1.3 Product Scope
   1.4 Document conventions
   1.5 References
2. **Overall Description**
   2.1 Product Perspective
   2.2 Product Functions
   2.4 Operating Environment
   2.6 User Documentation
3. **External Interface Requirements**
   3.1 User Interfaces
   3.2 Hardware Interfaces
   3.3 Software Interfaces
   3.4 Communications Interfaces
4. **System Features**
   4.1 Microservices
   4.2 Containerisation
   4.3 Container Management
   4.4 Load Balancing
   4.5 Monitoring
   4.6 Alerting
   4.7 Prediction
   4.8 User Functionalities

APPENDIX

APPENDIX I Data Flow Diagram
APPENDIX II Control Flow Diagram
APPENDIX III Test Cases and Results
APPENDIX IV Class Diagrams
APPENDIX V Cost Analysis

1. Introduction

1.1 Purpose

The purpose of this document is to create a list of detailed requirements for a hybrid cloud solution using open source technology. This document will capture the features of our solution and the technologies used to implement them.

1.2 Intended Audience a Reading Suggestions

This document will capture all stakeholders' preferences, different conflicts and their resolution. Also, it could be used by potential developers, design engineers, testers, project managers, etc. Eventually, this document can be used while preparing user documentation. This document will be proposed to different stakeholders for their approval and can be used as a reference guide in different phases of system development.

1.3 Product scope

The scope of the hybrid cloud solution is to create a backend service that is scalable, highly available and provides business continuity in the event of outages. This system will be an alternative to the traditional way of using a monolithic backend server to handle all the API requests.

1.4 Document Conventions

This document follows the following conventions:
X is a heading index
X - Font size - 18,
X.y - Font size - 14,
X.y.z - Font size - 12,
other - Fonts size - 12

1.5 References-

SRS IEEE Template to construct this document.

2. Overall Description

2.1 Product Perspective

This product provides a robust backend service to handle API requests of a nationwide fast-food chain AnyTimeFood. It is a hybrid cloud architecture using several multiple open source tools that carry out different functionalities like logging, visualisation, alerting and prediction effectively.

2.2 Product Functions

- Handling API requests including
    - Balancing the load when a large number of requests are received
    - Carrying out important functionalities even when the data centre is down.
- Maintaining database for
    - User Profiles
    - Orders and Deliveries
    - Loyalty points of users
- User's end
    - Get information about user profile
    - Get profile statistics like loyalty
    - Get a list of past orders
- For System Admin
    - Monitor the CPU utilisation of various nodes in the server
    - Get alerts when CPU utilisation is high
    - Get predictions of CPU utilisation for the next hour

2.3 Operating Environment

A Kubernetes cluster is hosted using Virtual machines taken on any public cloud service (here Azure Public Cloud). The server hosts an API, written in python using Flask Framework. To simulate the data centre, we are using a Virtual Machine which is an EC2 server from AWS Public Cloud.

2.4 User Documentation

Readme files will be there to guide the admin to install the system on the servers. These files also contain information on the API endpoints that the user needs to request to get the required information.

3. External Interface Requirements

3.1 User Interfaces

Any interface capable of making API requests to an endpoint can use the backend service.

3.2 Hardware Interfaces

The administration needs access to cloud-based computational services and a stable internet connection to connect to the monitoring services.

3.3 Software Interfaces

The cloud cluster has been managed by Kubernetes. It has been set up using Helm Charts. Prometheus and Grafana have been used for monitoring and alerts. Twilio API is used to send alert messages. Pytorch has been used to develop a deep-learning algorithm to predict CPU Utilisation. The backend has been developed in Flask. Docker has been used for containerisation. The database has been designed in Supabase, a free and open-source PostgreSQL database.

3.4 Communications Interfaces

The main communication interface is a REST API for the core application logic. There is also a dashboard to monitor the CPU metrics of the Kubernetes cluster nodes and the data centre. Also, an SMS-service has been used to send alerts to the user.

4. System features

4.1 Microservices

Microservices is an architectural style that structures the application as a collection of services that are highly maintainable and testable, independently deployable and loosely coupled. This enables rapid, frequent and reliable delivery of large applications.

Important functionalities in our backend like Orders and Deliveries have been converted into microservices.

4.2 Containerisation

All the services in our backend have been containerised. This involves encapsulation of the software code along with its dependencies. This has the following benefits

- Isolated dependencies
- Easy scaling
- Easily portable across different platforms

For the containerisation of our services, we have used Docker. The Flask backend has been dockerised and then hosted on the server.

4.3 Container Management

Due to the presence of a number of dockerized containers, an automated tool is required to create, deploy, destruct and scale the application.

Kubernetes, which is an open-source container orchestration system, has been used to manage the containers. Kubespray has been used to set up Kubernetes on our machines

4.4 Load Balancing

To mitigate the downsides of incoming traffic, we propose a load balancing solution for AnytimeFood. Load balancing ensures the availability, uptime and performance of your servers, websites and applications during traffic spikes

HAProxy, which stands for High Availability Proxy, is popular open-source software that has been used to set up load balancing. It has 3 primary purposes

- Load balancing using some scheduling algorithm like round-robin or random
- Conditional rerouting of all essential requests i.e /orders and /delivery and blocking of non-essential requests i.e. /loyalty during outages of the data centre.
- As a proxy service for all client requests which are then directed towards the public cloud or data centre as required

4.5 Monitoring

For monitoring of CPU metrics of the various nodes on our server, we have used Prometheus which is an open-source monitoring system. To visualise the data received from Prometheus and build intuitive charts and dashboards we have used Grafana which is again an open-source web application.

Various CPU metrics have been monitored in our system (Kubernetes cluster nodes and the datacentre)

- CPU Usage
- Memory Usage
- Uptime and Downtime
- Average Load
- Number of bytes received and transmitted
- Percentage availability of server
- Number of Requests

4.6 Alerting

To alert the user and system administration about high CPU usage and downtime of the system we have set up an alerting service in Grafana using webhooks. An SMS is sent to the admin whenever the CPU utilisation becomes high informing him of the same. Custom service has been set up in the Kubernetes cluster that gets informed whenever the data centre faces a downtime and an SMS is sent to the admin to notify him of the same.

4.7 Prediction

Just informing the admin when downtime occurs may not be that useful as in this case he/she will not be able to take any action. If the admin gets notified beforehand of a possible downtime then appropriate action can be taken to avoid the problem.

In this regard, we have trained an LSTM based Deep-Learning model to predict the CPU Utilisation for the next hour. If a high utilisation is predicted then an alert SMS is sent to the admin asking him to take appropriate action. Also, the predictions can be visualised in the Grafana dashboard.

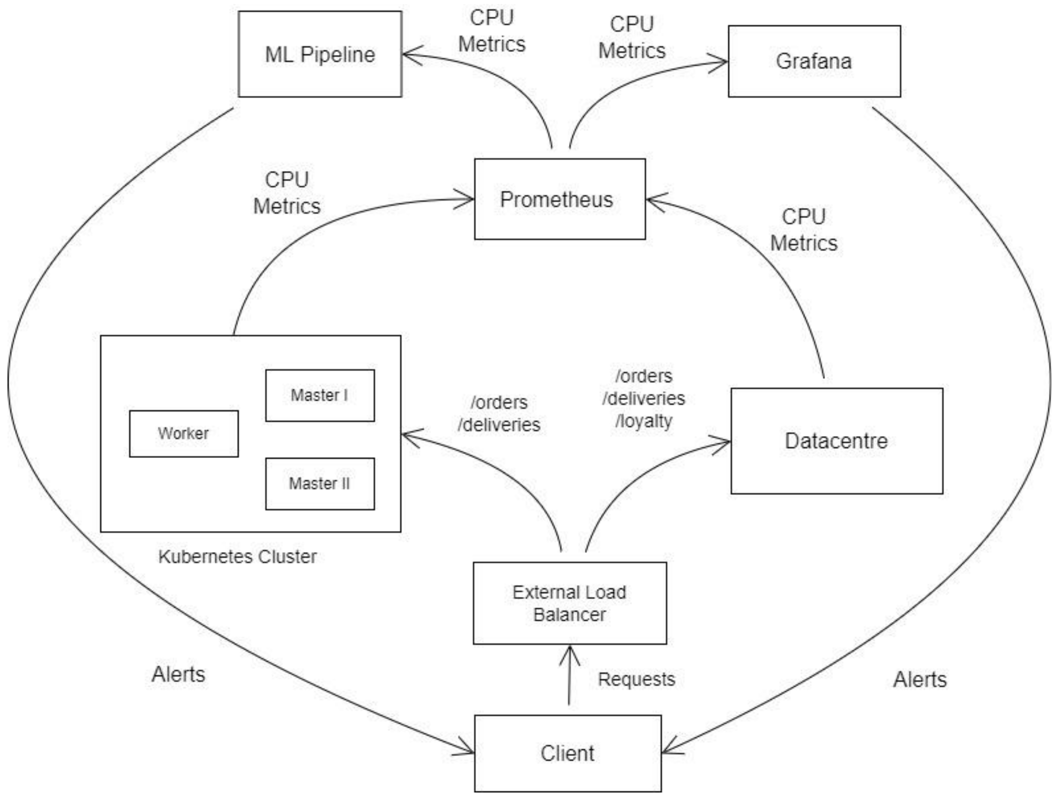
4.8 User Functionalities

Different functionalities have been provided in our backend for our users. These are as follows

- **Orders**: Get a list of orders for a particular user or get details of a particular order
- **User Details**: Get details of any particular user, create a user or get a list of users.
- **Deliveries**: Get the list of all the deliveries or details of any particular delivery
- **Loyalty**: Get the loyalty information of any particular user.
- **Login and Logout**: Log in and log out a particular user into the application

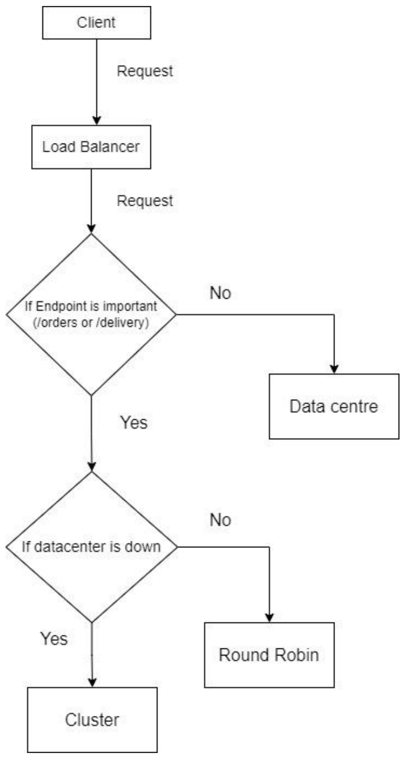
APPENDIX

APPENDIX - I Data Flow Diagram

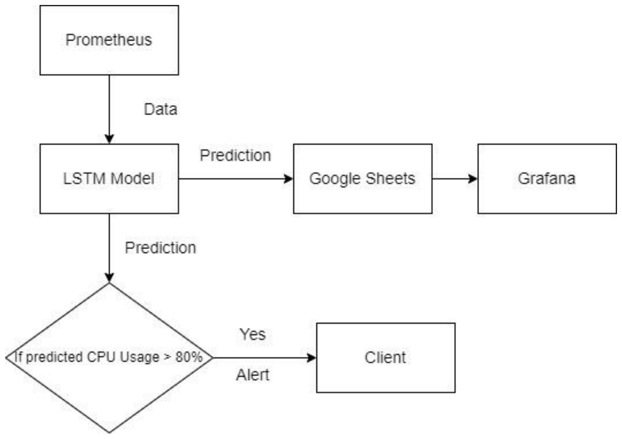


APPENDIX - II Control Flow Diagram

Request Handling Pipeline

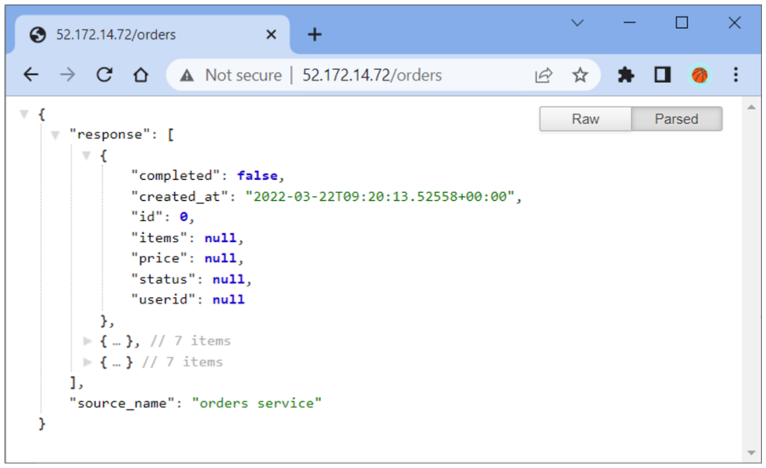


ML Pipeline

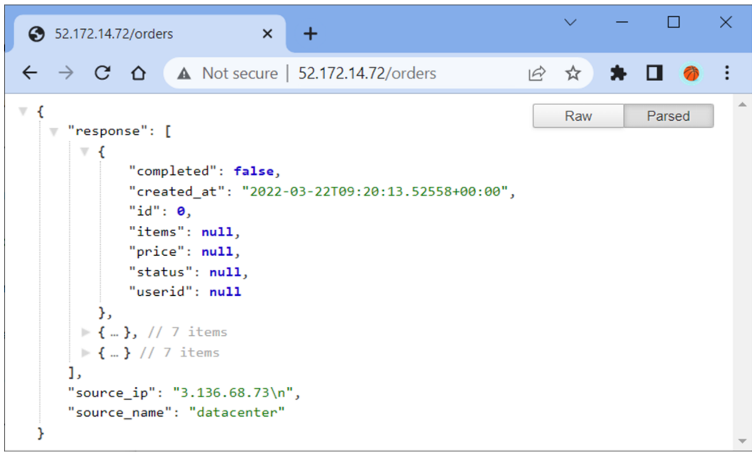


APPENDIX III Test Cases and Results

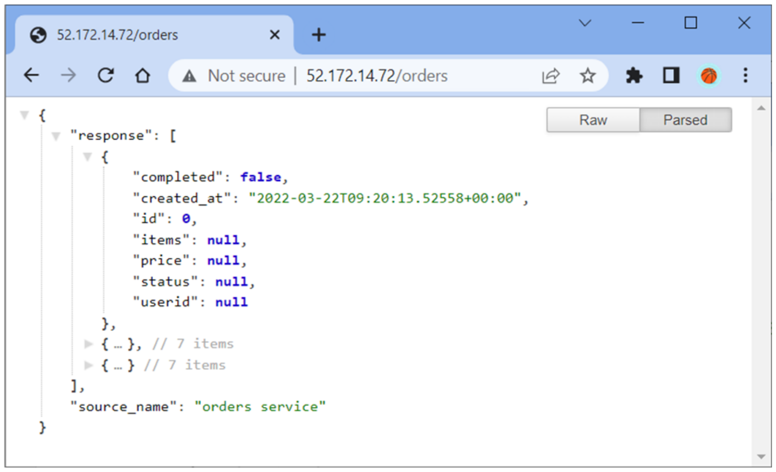
1. Load Balancing of /orders and /delivery



```
{
  "response": [
    {
      "completed": false,
      "created_at": "2022-03-22T09:20:13.52558+00:00",
      "id": 0,
      "items": null,
      "price": null,
      "status": null,
      "userid": null
    },
    { … }, // 7 items
    { … } // 7 items
  ],
  "source_name": "orders service"
}
```

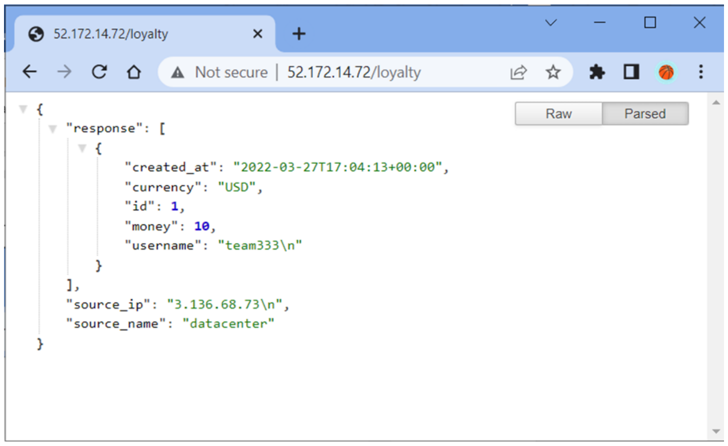


```
{
  "response": [
    {
      "completed": false,
      "created_at": "2022-03-22T09:20:13.52558+00:00",
      "id": 0,
      "items": null,
      "price": null,
      "status": null,
      "userid": null
    },
    { … }, // 7 items
    { … } // 7 items
  ],
  "source_ip": "3.136.68.73\n",
  "source_name": "datacenter"
}
```

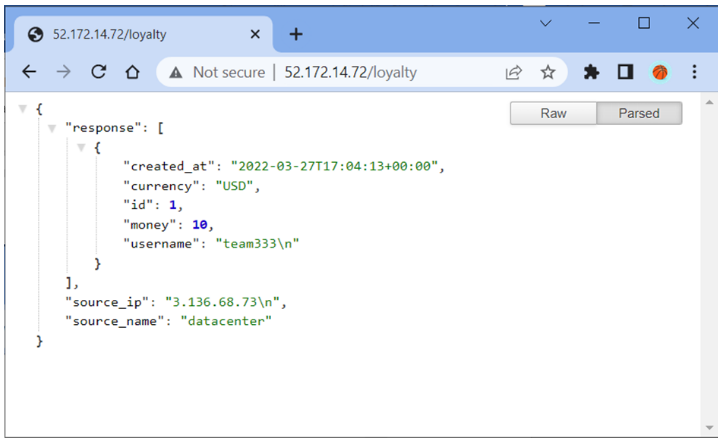


```
{
  "response": [
    {
      "completed": false,
      "created_at": "2022-03-22T09:20:13.52558+00:00",
      "id": 0,
      "items": null,
      "price": null,
      "status": null,
      "userid": null
    },
    { … }, // 7 items
    { … } // 7 items
  ],
  "source_name": "orders service"
}
```

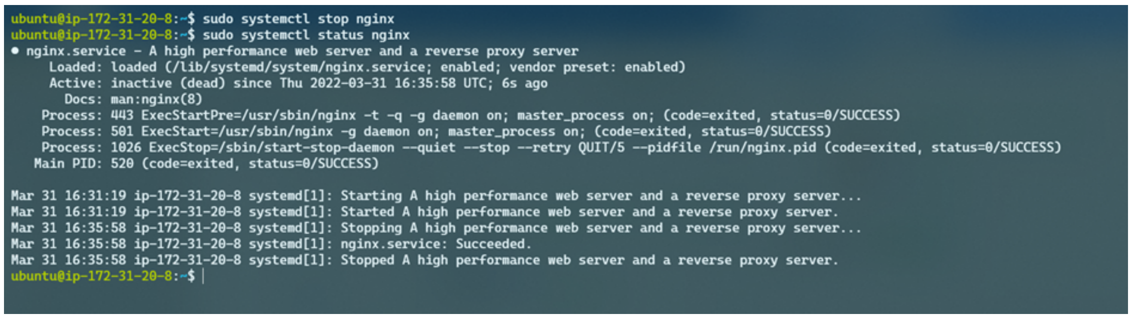
2. Only the Data centre is catering to /loyalty



```
{
  "response": [
    {
      "created_at": "2022-03-27T17:04:13+00:00",
      "currency": "USD",
      "id": 1,
      "money": 10,
      "username": "team333\n"
    }
  ],
  "source_ip": "3.136.68.73\n",
  "source_name": "datacenter"
}
```

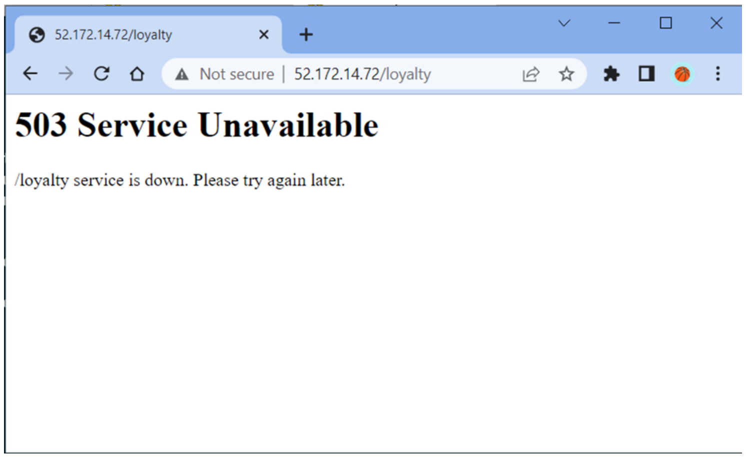


```
{
  "response": [
    {
      "created_at": "2022-03-27T17:04:13+00:00",
      "currency": "USD",
      "id": 1,
      "money": 10,
      "username": "team333\n"
    }
  ],
  "source_ip": "3.136.68.73\n",
  "source_name": "datacenter"
}
```



```
ubuntu@ip-172-31-20-8:~$ sudo systemctl stop nginx
ubuntu@ip-172-31-20-8:~$ sudo systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
     Active: inactive (dead) since Thu 2022-03-31 16:35:58 UTC; 6s ago
       Docs: man:nginx(8)
    Process: 443 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
    Process: 501 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
    Process: 1026 ExecStop=/sbin/start-stop-daemon --quiet --stop --retry QUIT/5 --pidfile /run/nginx.pid (code=exited, status=0/SUCCESS)
   Main PID: 520 (code=exited, status=0/SUCCESS)

Mar 31 16:31:19 ip-172-31-20-8 systemd[1]: Starting A high performance web server and a reverse proxy server...
Mar 31 16:31:19 ip-172-31-20-8 systemd[1]: Started A high performance web server and a reverse proxy server.
Mar 31 16:35:58 ip-172-31-20-8 systemd[1]: Stopping A high performance web server and a reverse proxy server...
Mar 31 16:35:58 ip-172-31-20-8 systemd[1]: nginx.service: Succeeded.
Mar 31 16:35:58 ip-172-31-20-8 systemd[1]: Stopped A high performance web server and a reverse proxy server.
ubuntu@ip-172-31-20-8:~$ |
```

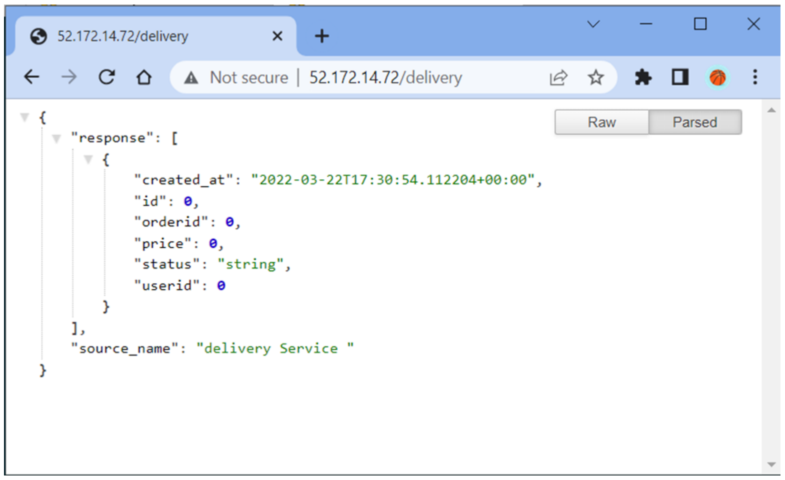

52.172.14.72/loyalty

Not secure | 52.172.14.72/loyalty

503 Service Unavailable

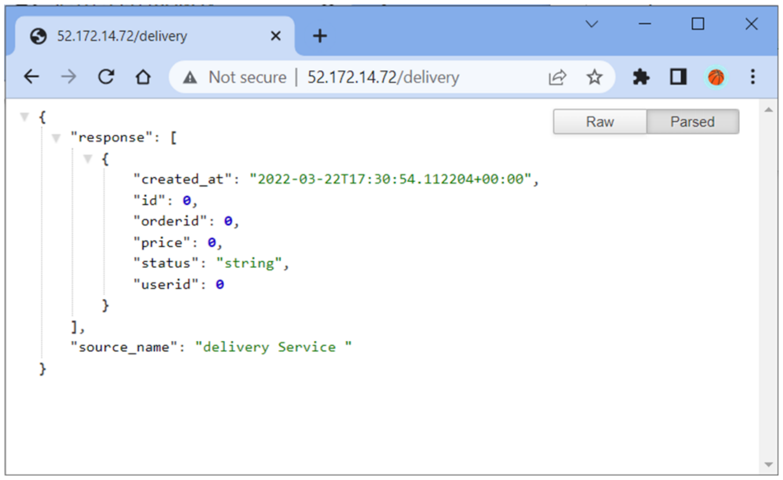
/loyalty service is down. Please try again later.

3. **Only /orders and /delivery are catered and are only catered by public cloud**


52.172.14.72/delivery

Not secure | 52.172.14.72/delivery

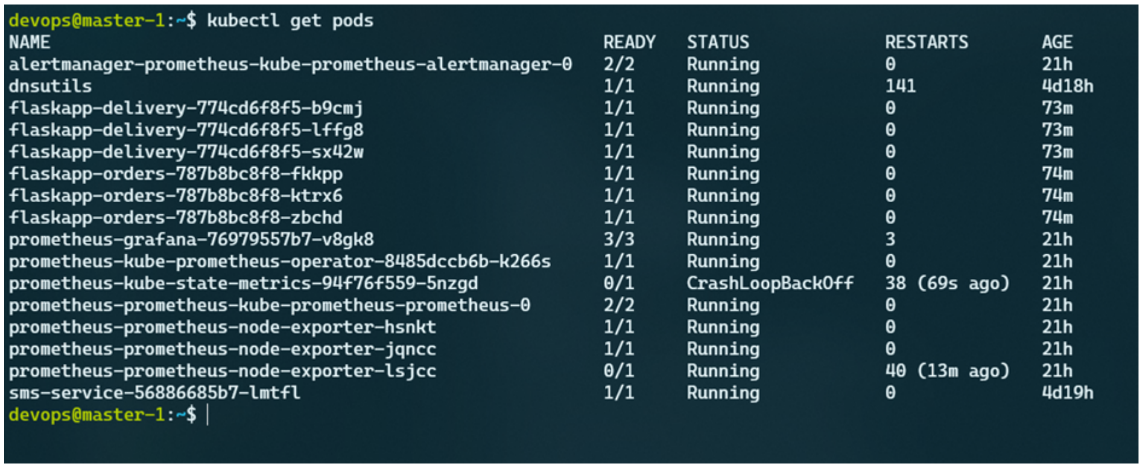
```
{
    "response": [
        {
            "created_at": "2022-03-22T17:30:54.112204+00:00",
            "id": 0,
            "orderid": 0,
            "price": 0,
            "status": "string",
            "userid": 0
        }
    ],
    "source_name": "delivery Service "
}
```


52.172.14.72/delivery

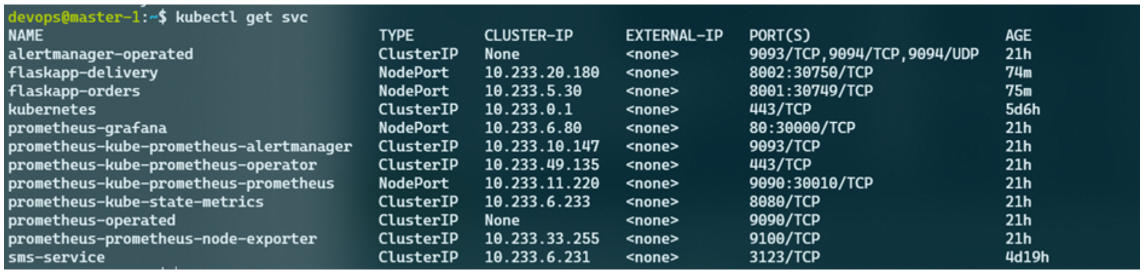
Not secure | 52.172.14.72/delivery

```
{
    "response": [
        {
            "created_at": "2022-03-22T17:30:54.112204+00:00",
            "id": 0,
            "orderid": 0,
            "price": 0,
            "status": "string",
            "userid": 0
        }
    ],
    "source_name": "delivery Service "
}
```

4. **List of Kubernetes pods and services**

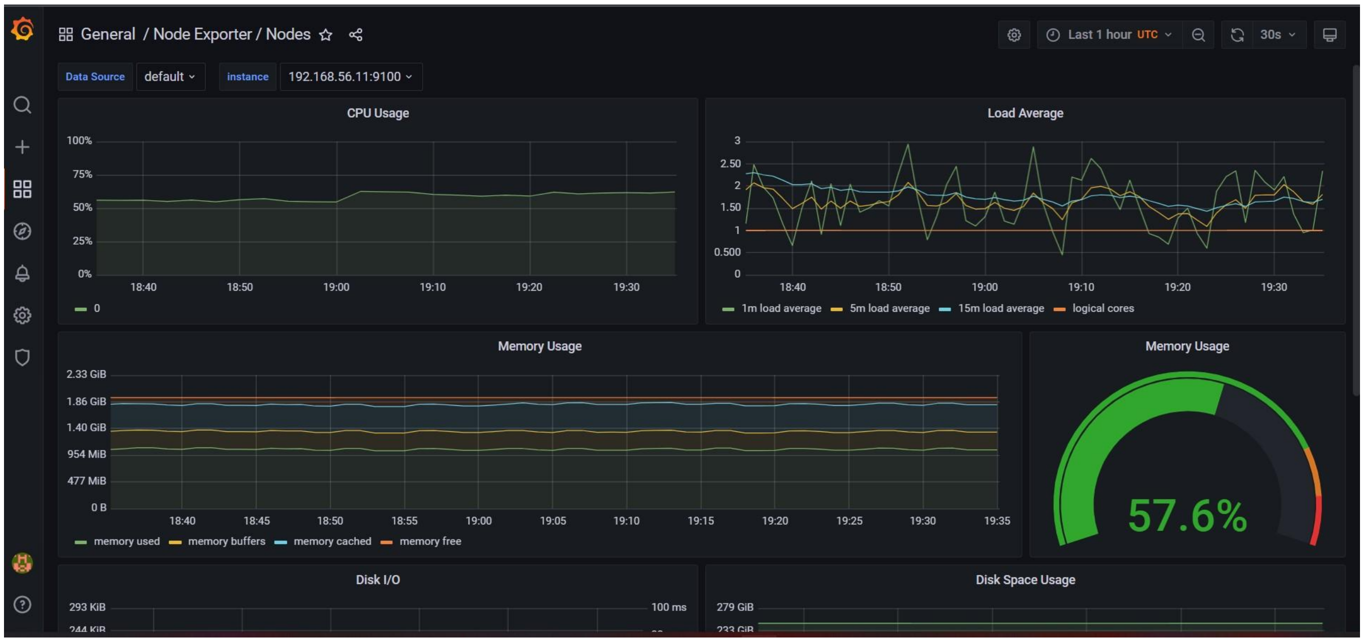


```
devops@master-1:~$ kubectl get pods
NAME                                                   READY   STATUS             RESTARTS       AGE
alertmanager-prometheus-kube-prometheus-alertmanager-0 2/2     Running            0              21h
dnsutils                                               1/1     Running            141            4d18h
flaskapp-delivery-774cd6f8f5-b9cmj                     1/1     Running            0              73m
flaskapp-delivery-774cd6f8f5-lffg8                     1/1     Running            0              73m
flaskapp-delivery-774cd6f8f5-sx42w                     1/1     Running            0              73m
flaskapp-orders-787b8bc8f8-fkkpp                       1/1     Running            0              74m
flaskapp-orders-787b8bc8f8-ktrx6                       1/1     Running            0              74m
flaskapp-orders-787b8bc8f8-zbchd                       1/1     Running            0              74m
prometheus-grafana-76979557b7-v8gk8                    3/3     Running            3              21h
prometheus-kube-prometheus-operator-8485dccb6b-k266s   1/1     Running            0              21h
prometheus-kube-state-metrics-94f76f559-5nzgd          0/1     CrashLoopBackOff   38 (69s ago)   21h
prometheus-prometheus-kube-prometheus-prometheus-0     2/2     Running            0              21h
prometheus-prometheus-node-exporter-hsnkt              1/1     Running            0              21h
prometheus-prometheus-node-exporter-jqncc              1/1     Running            0              21h
prometheus-prometheus-node-exporter-lsjcc              0/1     Running            40 (13m ago)   21h
sms-service-56886685b7-lmtfl                           1/1     Running            0              4d19h
devops@master-1:~$ |
```

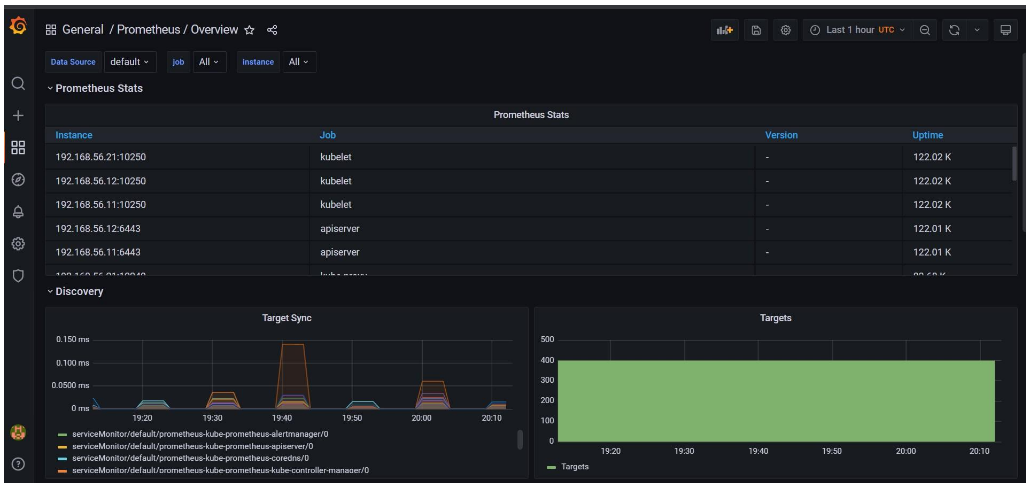


```
devops@master-1:~$ kubectl get svc
NAME                                     TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
alertmanager-operated                    ClusterIP   None            <none>        9093/TCP,9094/TCP,9094/UDP   21h
flaskapp-delivery                        NodePort    10.233.20.180   <none>        8002:30750/TCP               74m
flaskapp-orders                          NodePort    10.233.5.30     <none>        8001:30749/TCP               75m
kubernetes                               ClusterIP   10.233.0.1      <none>        443/TCP                      5d6h
prometheus-grafana                       NodePort    10.233.6.80     <none>        80:30000/TCP                 21h
prometheus-kube-prometheus-alertmanager  ClusterIP   10.233.10.147   <none>        9093/TCP                     21h
prometheus-kube-prometheus-operator      ClusterIP   10.233.49.135   <none>        443/TCP                      21h
prometheus-kube-prometheus-prometheus    NodePort    10.233.11.220   <none>        9090:30010/TCP               21h
prometheus-kube-state-metrics            ClusterIP   10.233.6.233    <none>        8080/TCP                     21h
prometheus-operated                      ClusterIP   None            <none>        9090/TCP                     21h
prometheus-prometheus-node-exporter      ClusterIP   10.233.33.255   <none>        9100/TCP                     21h
sms-service                              ClusterIP   10.233.6.231    <none>        3123/TCP                     4d19h
```

5. **Grafana Dashboard**
    a. **CPU Utilisation and Memory**



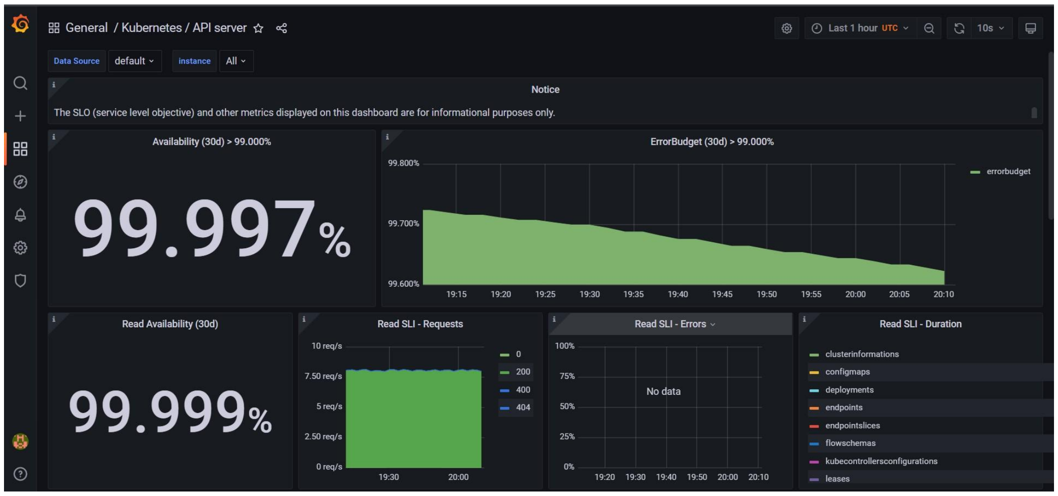
b. Overview of the status of Kubernetes and data centre


General / Prometheus / Overview
Last 1 hour UTC
Data Source default
job All
instance All
Prometheus Stats
Discovery
Target Sync
0.150 ms
0.100 ms
0.0500 ms
0 ms
19:20
19:30
19:40
19:50
20:00
20:10
serviceMonitor/default/prometheus-kube-prometheus-alertmanager/0
serviceMonitor/default/prometheus-kube-prometheus-apiserver/0
serviceMonitor/default/prometheus-kube-prometheus-coredns/0
serviceMonitor/default/prometheus-kube-prometheus-kube-controller-manager/0
Targets
500
400
300
200
100
0
Targets

Prometheus Stats

Instance	Job	Version	Uptime
192.168.56.21:10250	kubelet	-	122.02 K
192.168.56.12:10250	kubelet	-	122.02 K
192.168.56.11:10250	kubelet	-	122.02 K
192.168.56.12:6443	apiserver	-	122.01 K
192.168.56.11:6443	apiserver	-	122.01 K

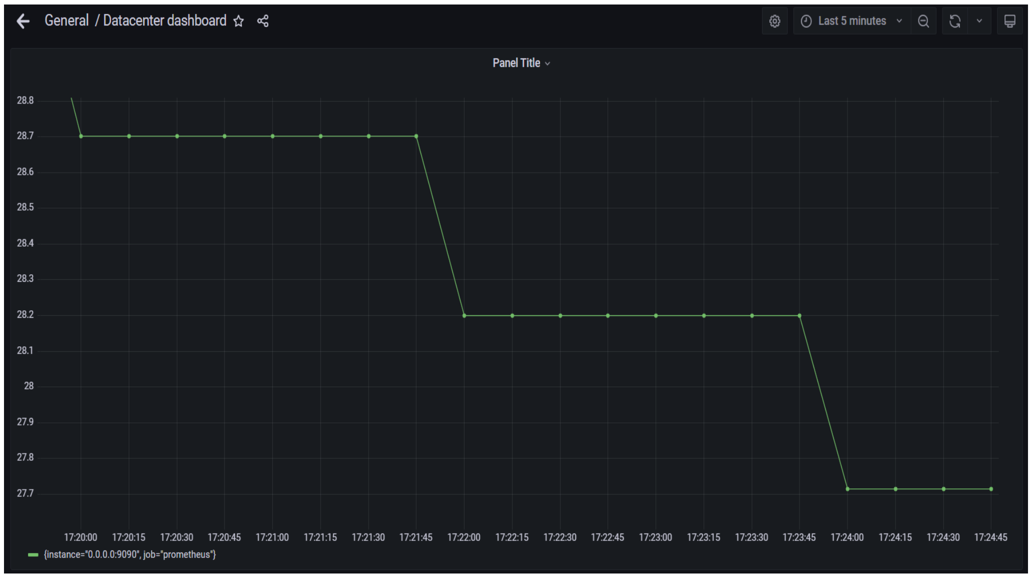
c. API Server



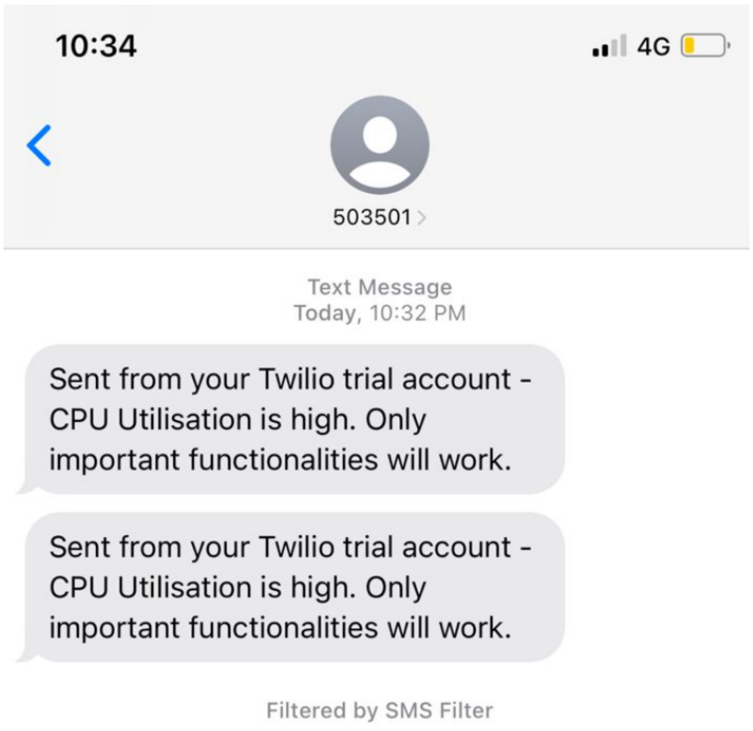
Notice

The SLO (service level objective) and other metrics displayed on this dashboard are for informational purposes only.

d. Data centre overview



6. **SMS Alerts through Grafana**



10:34

4G

503501

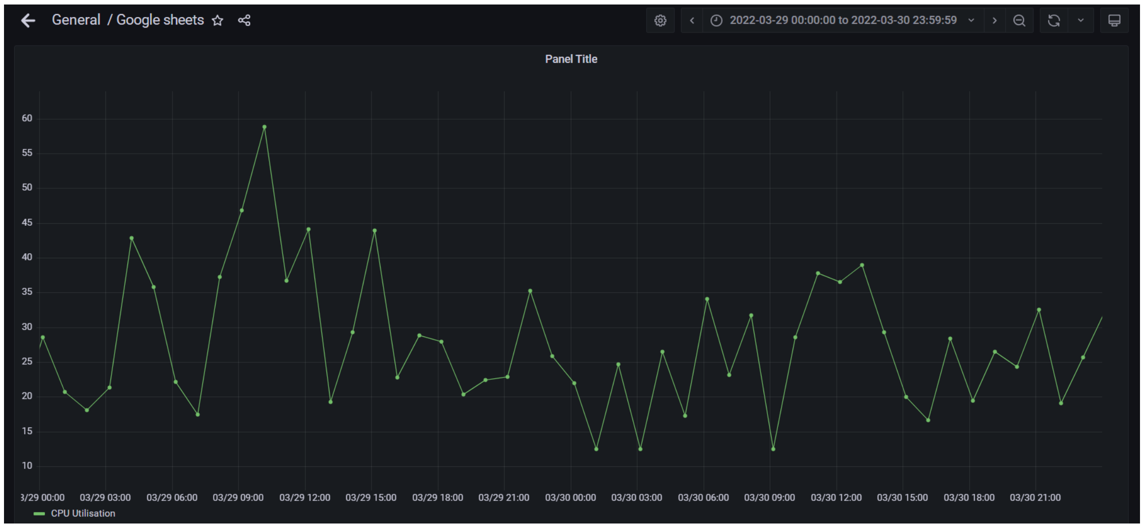
Text Message
Today, 10:32 PM

Sent from your Twilio trial account - CPU Utilisation is high. Only important functionalities will work.

Sent from your Twilio trial account - CPU Utilisation is high. Only important functionalities will work.

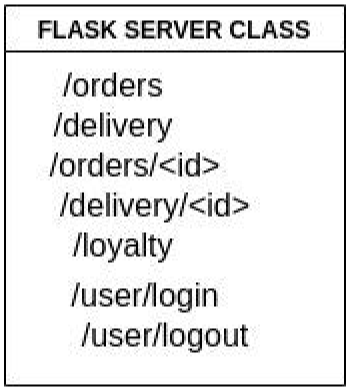
Filtered by SMS Filter

7. **ML Prediction in Grafana Dashboard**



APPENDIX IV Class Diagram

We have only one main class: The Flas Server whose functions are shown below


FLASK SERVER CLASS

/orders
/delivery
/orders/<id>
/delivery/<id>
/loyalty
/user/login
/user/logout

The other functionalities are mainly implemented using config files and therefore do not have any class structure.

APPENDIX V Cost Analysis

For the Kubernetes cluster, we have deployed 3 nodes: 2 master and 1 worker node. We will need another machine as a load balancer and one as a database. This adds to 5 machines. These have been deployed as Microsoft Azure B4ms Virtual Machines with 16GB RAM and 4 cores. The cost will be as shown in the table below:

Resource	Rate	Quantity	Total Price per day
B4ms	$ 0.166 / hr	5	$ 19.92
			Total: $ 19.92

Instead of this if we use Amazon EKS to deploy a Kubernetes cluster and use other AWS machines required to set up this complete system then the cost will be as shown in the table below.

Resource	Rate	Quantity	Total Price per day
t4g.xlarge	$ 0.134 / hr	3	$ 9.648
db.t4g.large	$ 0.167 / hr	1	$ 4.008
Egress	$ 0.1093 / GB	100 GB / day	$ 10.93
NAT Gateway	$ 0.056 / GB	100GB / day	$ 5.6
S3	$ 0.025 / GB	50 TB / month	$ 42.67
EKS Cluster	$ 0.1 / hr	1	$ 2.4
			Total: $ 75.256

Thus our approach can save cost by almost 378%!