LOW LATENCY WIRELESS EDGE CLUSTER SETUP

DESIGN PROJECT

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING, KURNOOL

MENTOR: Dr. R. ANIL KUMAR

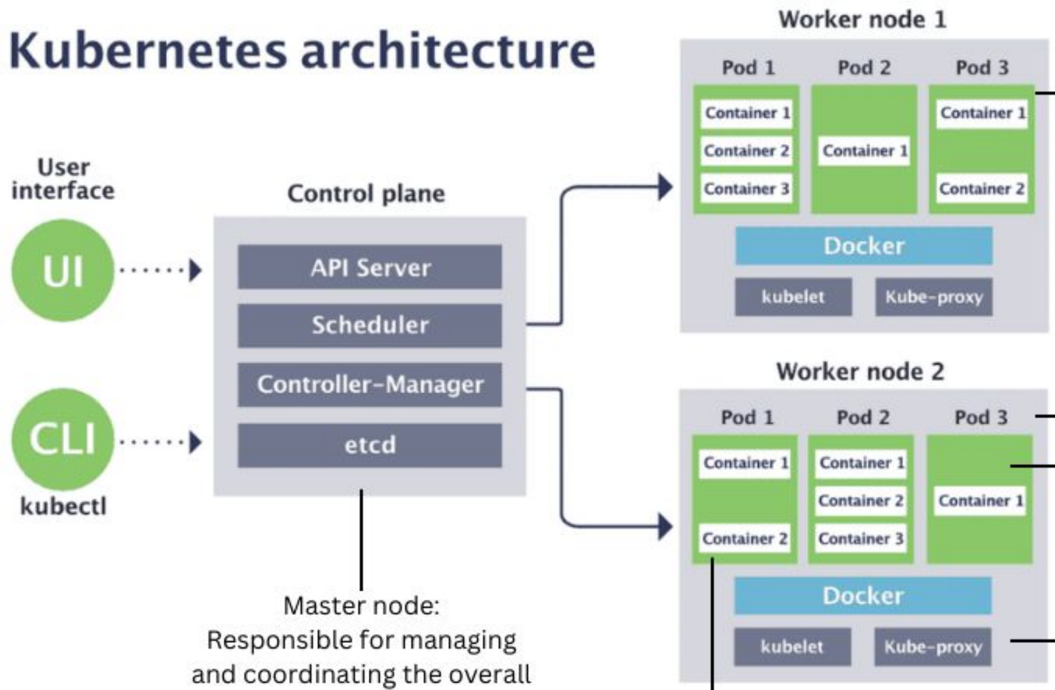
Team members:

1. P. Lakshmi Sujitha: 120CS0017
2. Y. Bhavishya: 120CS0025
3. K. R. Ramya Shri Shakthi: 120CS0046
4. S. Satakshi: 120CS0047

PROBLEM STATEMENT

Deploying smart applications over wireless nodes need dynamic edge computing platform for meeting the latency and reliability requirements.

Existing Kubernetes platform for edge computing



A node refers to a computing device that is part of the overall cluster infrastructure

A pod is the smallest execution unit which encapsulates one or more applications

Worker node:
Responsible for executing tasks and processing workloads assigned by the master/control plane.

Master node:
Responsible for managing and coordinating the overall operations of the edge cluster.

Packages of software that contain all the necessary elements to run in any environment.

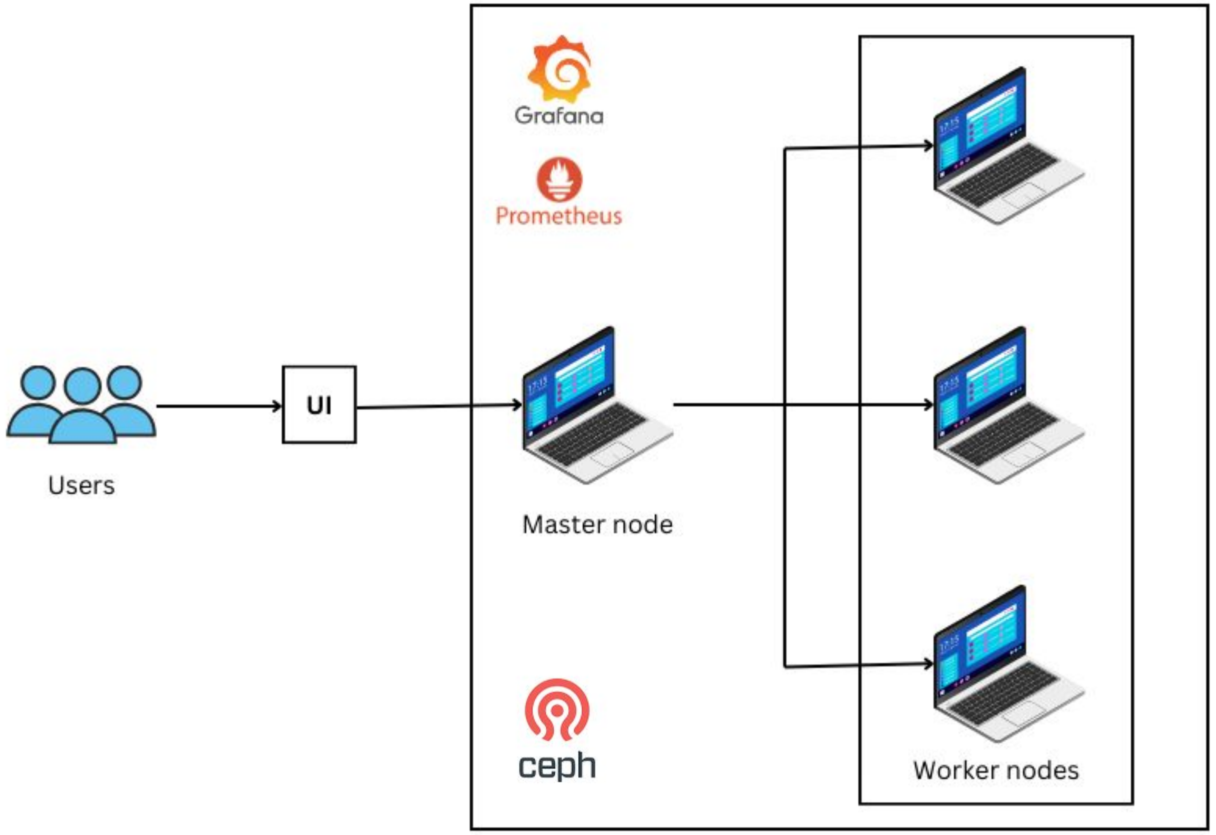
Why kubernetes platform?

- Kubernetes offers simple approach for setting up cluster environment.
- Kubernetes offers simplified application deployment process.
- Kubernetes supports basic scheduling and load balancing approaches.
- Kubernetes supports integration of popular resources monitoring and distributed storage applications.

Gaps in the existing kubernetes platform!

- Kubernetes does not offer event based vertical scaling.
- Kubernetes does not strategically assign applications to the nodes with minimal latency.
- Automatic deployment of the cluster is not possible.

To address these gaps, we have implemented some solutions, which are discussed in the upcoming slides.



Proposed cluster architecture diagram

Implementation

1) Event based vertical scaling

In our Kubernetes edge cluster with limited resources: efficient resource utilization is a priority. To implement this, we came up with this strategy:

- Only the required nodes are untainted at any given time.
- We incorporated Prometheus, to keep monitoring our edge cluster at all times.
- Grafana is connected to the prometheus data source of our cluster.

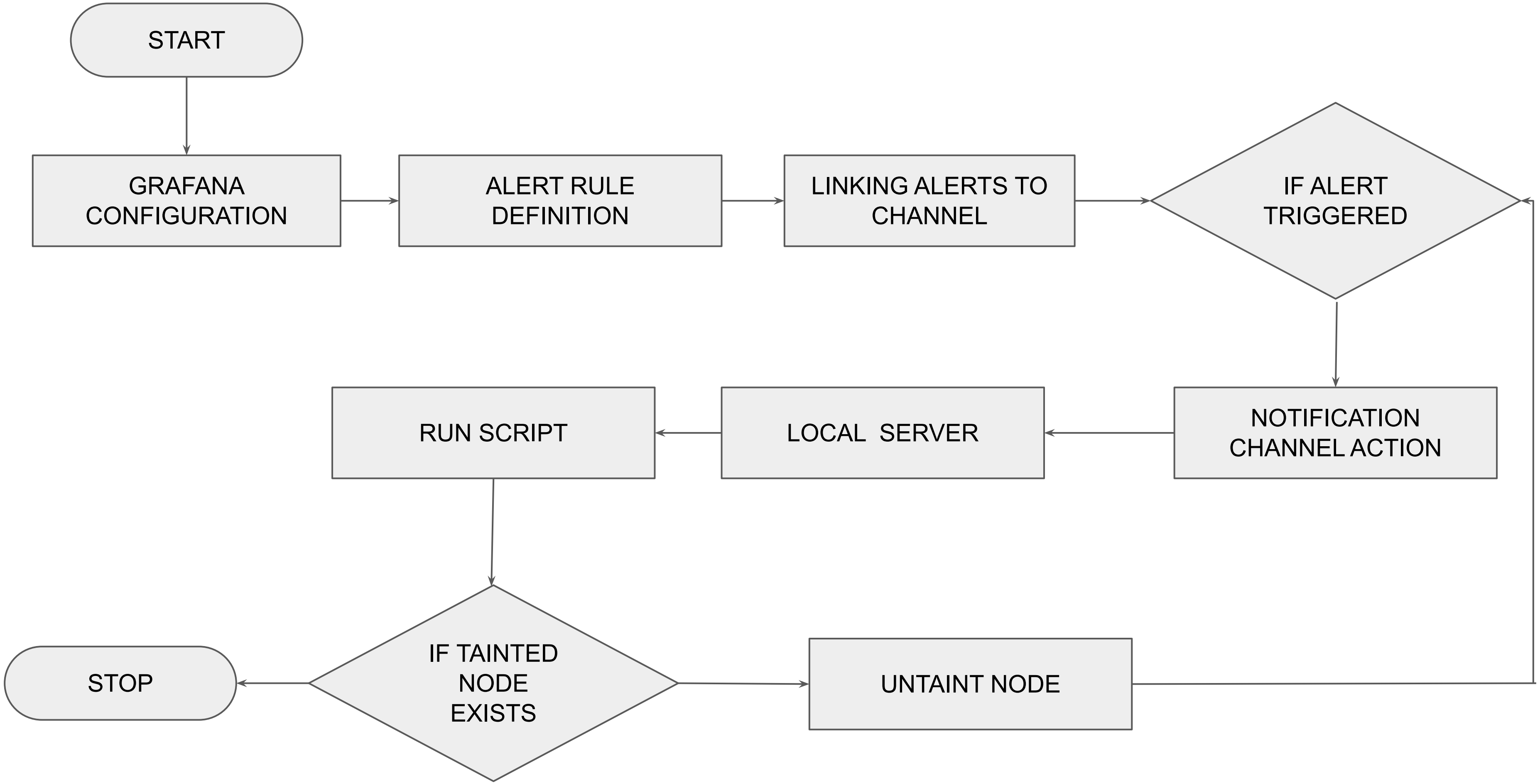
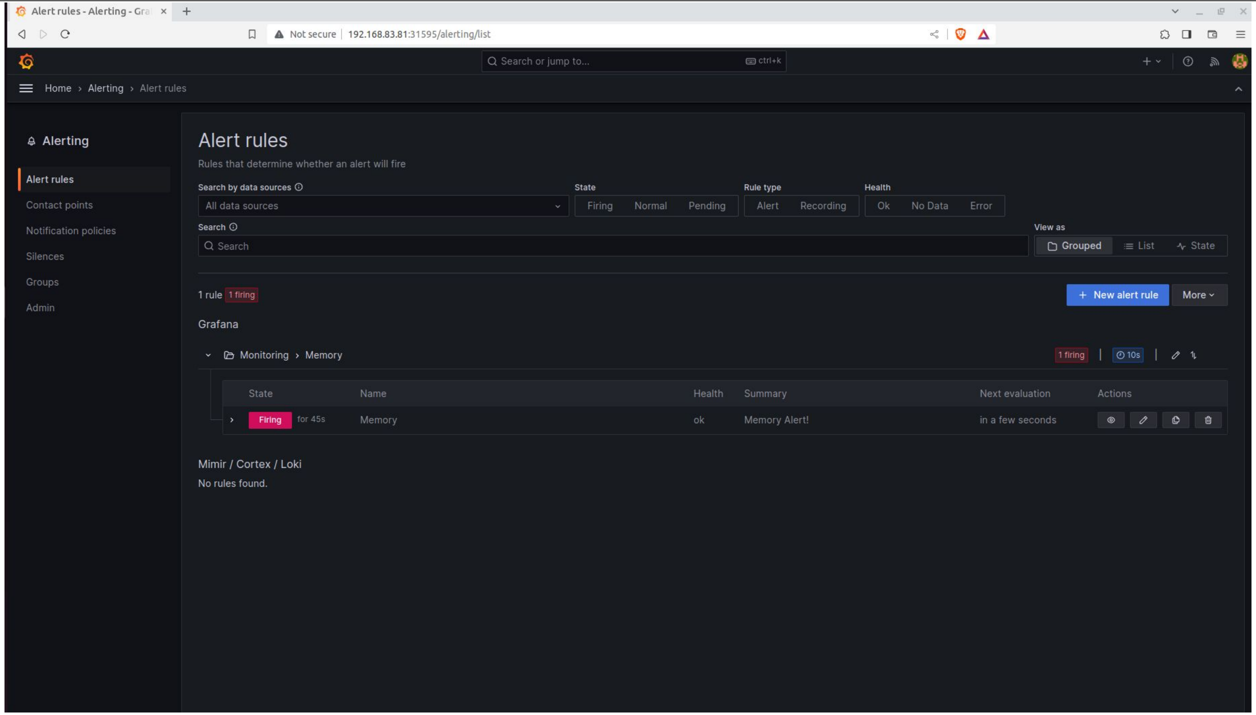


Fig 4: Flow of event based scaling

- We created a local server which listens to any alerts from Grafana.
- A notification channel is created in Grafana, specifying the method of communication for alert notification.
- Then, we configured alerts by writing multiple queries in grafana after which the alerts are linked to notification channel.
- As soon as an alert is triggered, the local server will recognise it and run the python script to automatically untaint the tainted nodes.



Alert rules

Rules that determine whether an alert will fire

1 rule 1 firing

Grafana

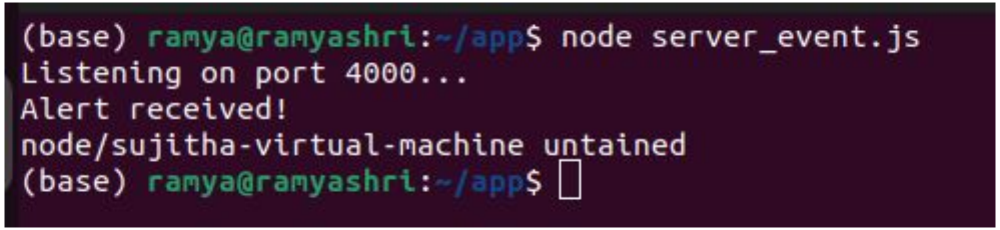
Monitoring › Memory 1 firing 10s

State	Name	Health	Summary	Next evaluation	Actions
Firing for 45s	Memory	ok	Memory Alert!	in a few seconds	

Mimir / Cortex / Loki

No rules found.

Figure 5: Screenshot of alerts firing in Grafana



```
(base) ramya@ramyashri:~/app$ node server_event.js
Listening on port 4000...
Alert received!
node/sujitha-virtual-machine untained
(base) ramya@ramyashri:~/app$
```

Figure 6: Screenshot of NodeJS server untainting the nodes

2) Wireless, distributed low latency cluster

- Our entire edge infrastructure operates wirelessly, utilizing laptops as nodes within the cluster.
- It is feasible for all nodes to be dispersed across different locations, as long as they are connected to the the same network. This mirrors the functionality of a distributed edge cluster.
- To guarantee swift responses for users, it is essential to minimize the latency within the cluster.

- So, we came up with an approach to minimise the latency in our cluster.
- Since the nodes are distributed, the worker node with minimal latency from the master node is to be found.
- An instance of the application is then created on that node.
- Heavy influx in requests results in exhaustion of node resources.
- So, as soon as that happens, our code checks for the node with next minimal latency and creates another instance of the application on that node.

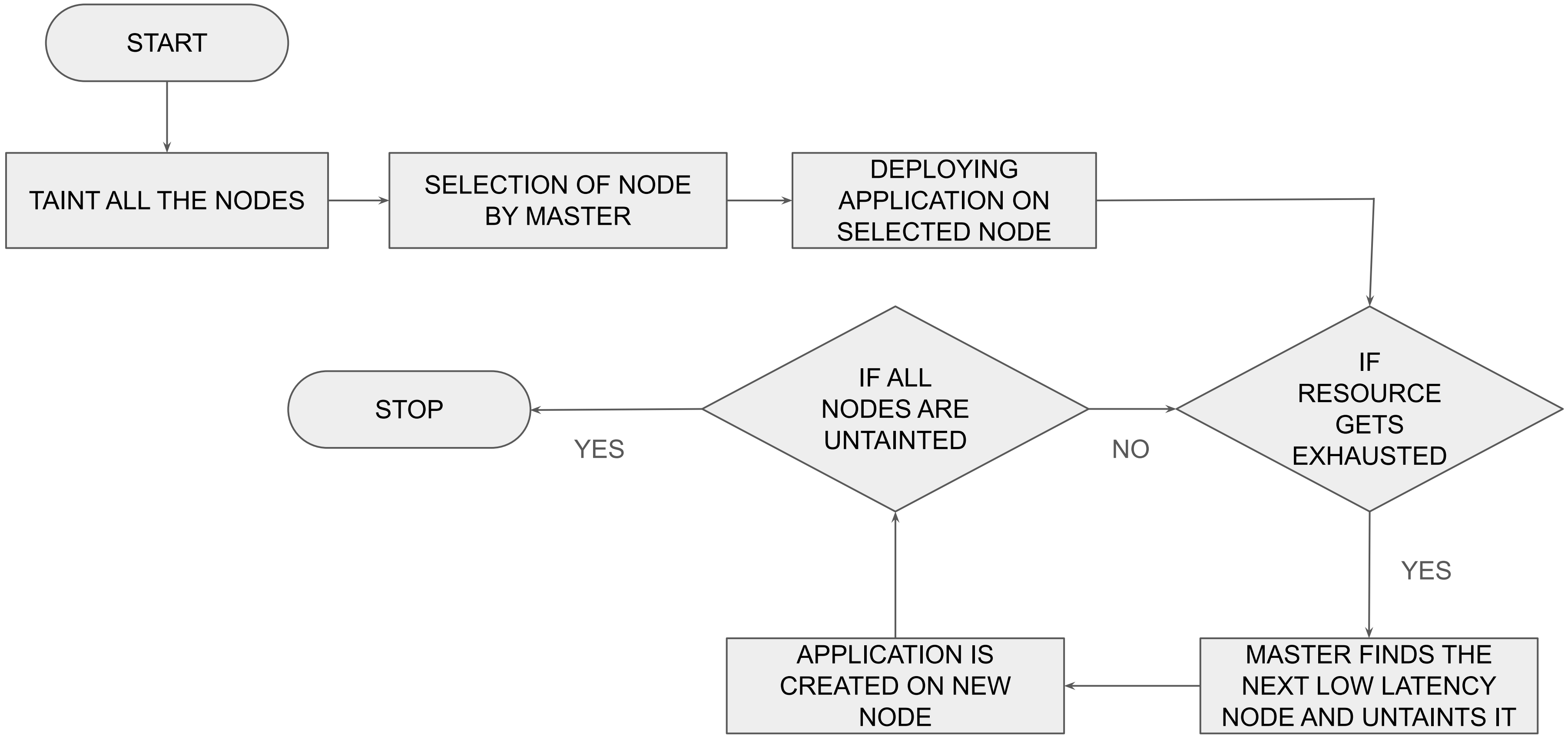

START
TAINT ALL THE NODES
SELECTION OF NODE BY MASTER
DEPLOYING APPLICATION ON SELECTED NODE
IF RESOURCE GETS EXHAUSTED
YES
MASTER FINDS THE NEXT LOW LATENCY NODE AND UNTAINTS IT
APPLICATION IS CREATED ON NEW NODE
IF ALL NODES ARE UNTAINTED
NO
YES
STOP

Figure 7: Flow of low latency cluster

- All the worker nodes in the cluster are initially tainted.
- The master node runs an algorithm to find a tainted worker node with the lowest latency among all the worker nodes.
- To keep track of resource utilization in the cluster, Grafana and Prometheus are used. If any resource utilization exceeds a certain threshold, alerts are sent to the local server.
- As soon as the alert is received, the master runs the algorithm to find the tainted node with next minimum latency.
- This process essentially schedules the applications in nodes with minimal latency.

3) Automatic Deployment:

- For a worker to join the cluster, it must have a token which is only generated after the master node is initialized. Kubernetes does not offer any in built communication channel to send the tokens to the worker nodes which makes it tedious.
- To make the addition of worker node more automated and efficient, we are running a code using client-server architecture for the distribution file containing necessary tokens and commands.

DISTRIBUTED STORAGE

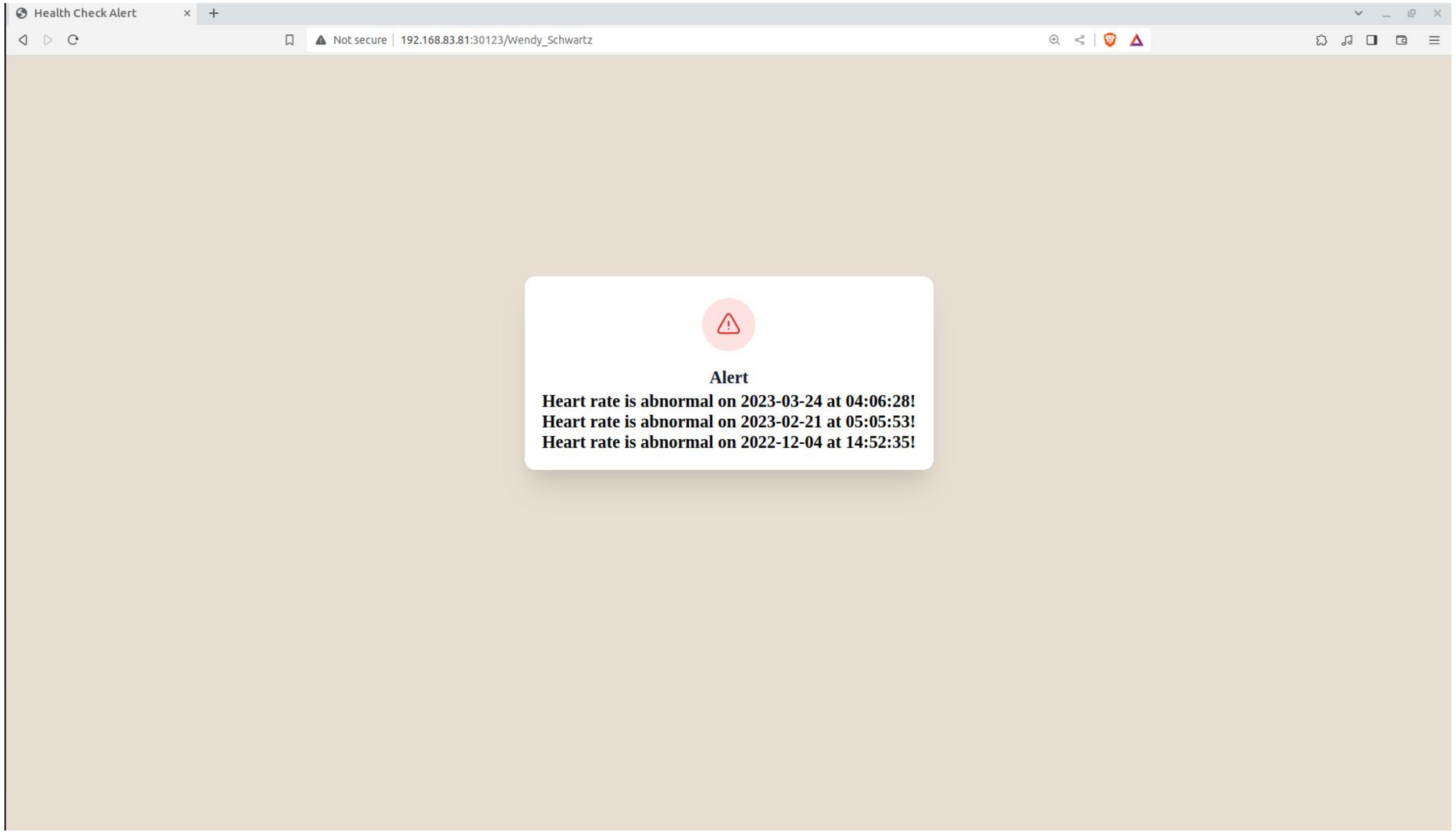
- Data inside a pod is lost forever when a pod gets deleted or gets crashed.
- To prevent this data loss persistent volume and persistent volume claims can be used.
- But when a node crashes the data cannot be retrieved.
- So we have implemented distributed storage using rook-ceph.

- Rook-ceph will create replicas of the data that is to be stored in multiple nodes ensuring reliability of data in cluster.
- This makes sure that even if one node crashes no data is lost because a copy of the data is available in the cluster.

TESTING USING SAMPLE APPLICATION

- We have created a sample application using NodeJS and MongoDB to test the features of our cluster.
- In this application, we are assuming that a sensor generates data encompassing fields such as name, date, time, and heart rate. This data is then transmitted to a MongoDB pod deployed within the edge cluster.
- To facilitate the heart rate monitoring process, we’ve implemented an Express.js code in a separate pod.

- The application pod is exposed to users through a NodePort service, providing external accessibility. A client can access their heart rate monitoring results using the url http://masterip:portnumber/patientname. Kubernetes services are employed to establish connectivity between the Express.js application pod and the MongoDB pod, enabling seamless access to health-related data stored in the ”healthcheck” database.



Alert

Heart rate is abnormal on 2023-03-24 at 04:06:28!
Heart rate is abnormal on 2023-02-21 at 05:05:53!
Heart rate is abnormal on 2022-12-04 at 14:52:35!

Figure 8: Screenshot of our application running

- We have deployed this application on our low latency edge cluster to test using the newly deployed features of the cluster.
- We deployed this application on the node with the lowest latency and we periodically sent n number of random requests to the application.
- We found that our cluster was able to handle all the traffic without any issues.

POSSIBLE USE CASES OF OUR CLUSTER:

DRONE APPLICATIONS:

- Given the wireless capability of our edge nodes, we have the potential for node mobility, allowing each node to function as a drone and execute computations.
- While we have not explored the mobility aspect of our wireless network, there's an opportunity to establish a low-latency cluster tailored for drone operations.

FUTURE WORKS

- While our current edge cluster has implemented the low-latency feature to some extent, our aim is to elevate this capability to a higher standard by fundamentally redefining the architecture itself by selecting nodes with minimal latency.
- In addition to it, we intend to employ more optimal algorithms for identifying nodes with minimal latency, moving beyond our existing methodology.

- Furthermore, we plan to add more resources to the cluster substantially.
- In addition to refining our low-latency implementation, we would like to explore the concept of mobility within the context of wireless nodes.

TECHNOLOGIES USED

1. Kubernetes
2. Docker
3. Prometheus
4. Grafana
5. NodeJS
6. MongoDB

THANK YOU!