B.M.S. COLLEGE OF ENGINEERING

Autonomous Institute, Affiliated to VTU

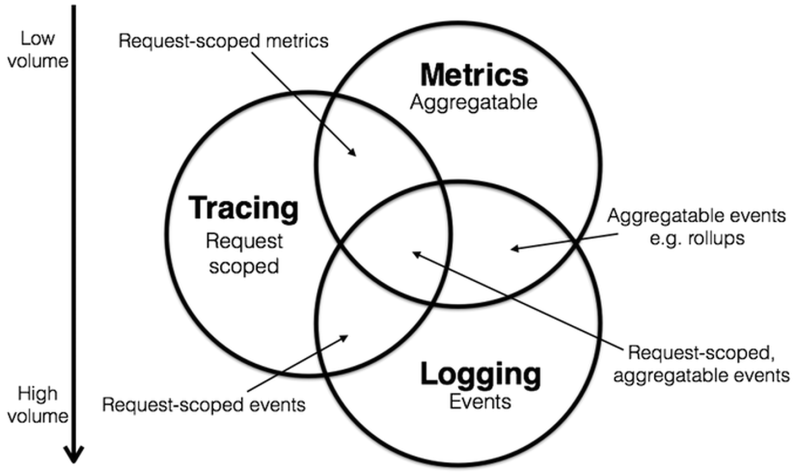
Estd. 1946

DEPARTMENT OF CSE

CTY Project Work In collaboration with HPE

Project Title	Open source monitoring and observability stack on Kubernetes		
Student Team	Name: Pooja Srinivasan USN:1BM18CS069 SEM:UG-6th SEM	Name: Anusree Manoj K USN:1BM18CS017 SEM:UG-6th SEM	
	Name: Shikha N USN:1BM18CS149 SEM:UG-6th SEM	Name: Niha USN:1BM18CS060 SEM:UG-6th SEM	
Faculty Mentor	Dr. Nandini Vineeth Associate Professor	**HPE Mentors**	Divakar Padiyar Sonu Sudhakaran
Review for the Period	09-03-2021	08-07-2021	
Task Given	Open source Monitoring and Observability Stack on Kubernetes Kubernetes is the go-to for container management, giving organizations superpowers for running container applications at scale. Kubernetes can simplify the management of your containerized applications and services across different cloud services. It can be a double-edged sword, though, as it also adds complexity to your system by introducing a lot of new layers and abstractions, which translates to more components and services that need to be monitored. This makes Kubernetes monitoring and overall observability even more critical. Aim of this project is to put together an open source monitoring and observability stack on kubernetes covering Monitoring, Alerting/Visualization, Log Aggregation/analytics, and Distributed systems tracing infrastructure which collectively make up observability.		
Difficulties Faced	1.Working remotely. 2. Insufficient hardware. 3. Ramp up time in working with various tools.		
Libraries Used	None		
Github Link:	https://github.com/PoojaSrinivasan18/HPE-CTY		
Code:	None		

Introduction	Kubernetes is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications. We started our project by exploring and understanding concepts of containerization and the role of Kubernetes. Understood Physical Server vs Virtual Machine vs Containers.We explored and understood the concepts of Observability and Monitoring with respect to K8.We went on to learn and do hands on Deployment of microK8s, minikube and k3s. We implemented Grafana and Prometheus stack on the clusters to observe their metrics. We then configured the alert manager and a notification channel on grafana to receive alerts on a slack channel. We also implemented logging and tracing using Loki and Jaeger.
Observability	Observability is a way to get insights into the whole infrastructure. It is essential for the operations team . Observability means assembling all fragments from logs, monitoring tools and organizing them in such a way which gives actionable knowledge of the whole environment, thus creating an insight. Observability is combining multiple items to create a deep understanding of the actual health, real issues, and what should be done to improve the environment and troubleshooting at a root level. Observability means service can explain any questions about what is happening on the inside of the system just by observing the outside of the system, without entering new code to answer further questions. Observability is a test of how well the inner states of a system can be assumed by knowledge of its external outputs. The three pillars of observability are logs, traces and metrics. ● **Logs** represent discrete events and data in a structured textual form.They contain both the log the message emitted by a component of the service or the code(payload) and metadata, such as the timestamp, label, tag, or other identifiers.In Kubernetes, every container writes its logs to standard output which can be aggregated by a centralized logging solution for future analysis. Some of the common options for consistently gathering and analyzing logs are: Collecting kubernetes logs with Fluentd**,**Storing and analyzing with ELK stack(Elasticsearch, Logstash, and Kibana). ● A **trace** is a representation of consecutive events which reflect an end-to-end request path in a distributed system.Traces helps us to understand the entire request flow, and hence it’s possible to troubleshoot most performance bottlenecks or analyze dependencies between different services. Tools used for Distributed traces are Istio,OpenCensus,Zipkin and Jaeger. ● A **metric** is a fundamental type of signal that can be emitted by a service or the infrastructure it's running on and numeric representation of data over intervals of time.It is used to understand the system health using

	telemetry signals.Metrics are stored in time series like Prometheus. Tools used for metrics are: Prometheus,Graphite,InfluxDB,OpenTSDB and OpenCensus. 
Observability vs Monitoring	Monitoring is something you do. Observability is something you have. Generally, when we are monitoring something, we need to have the insight into exactly what data to monitor. If/when something goes wrong, a good monitoring tool alerts us to what is happening based on what we are monitoring. We use monitoring to track performance, identify problems and anomalies, find the root cause of issues, and gain insights into physical and cloud environments. We need to be monitoring to have observability. Monitoring can view the external, logging can see the outputs over time, and with these combined, observability can be achieved by inferring the internal based on the historical. Observability and monitoring complement each other, with each one serving a different purposes. Monitoring tells you when something is wrong, while observability enables you to understand why. Monitoring is a subset of and key action for observability. You can only monitor a system that's observable. Monitoring tracks the overall health of an application. It aggregates data on how the system is performing in terms of access speeds, connectivity, downtime, and bottlenecks. Observability, on the other hand, drills down into the "what" and "why" of application operations, by providing granular and contextual insight into its specific failure modes.

<table>
<tr><td></td><td>
<table>
<tr><th>Observability</th><th>Monitoring</th></tr>
<tr><td>What is my system doing?</td><td>Is my system working?</td></tr>
<tr><td>Tells you why something goes wrong</td><td>Tells you when something went wrong</td></tr>
<tr><td>Proactive in nature</td><td>Reactive in nature</td></tr>
<tr><td>Reduces the duration and impact of incidents</td><td>Enables quick response when an incident occurs</td></tr>
<tr><td>Gain understanding actively</td><td>Consume information passively</td></tr>
<tr><td>Build to tame dynamic environments with changing complexity</td><td>Built to maintain static environments with little variation</td></tr>
<tr><td>Preferred by developers of systems with variability and unknown permutations</td><td>Used by developers of systems with little change and known permutation</td></tr>
</table>
</td></tr>
<tr><td>Observability Use Cases</td><td>

As an Infrastructure Admin, I would like the "Platform" services to be monitored constantly for application runtime errors, re-curing operational / API / UI failures and generate alerts so that assigned / scheduled Support team gets notified immediately and is able to troubleshoot the error through Operations Console.
As an Infrastructure Admin, I would configure monitoring of Server / Storage /Network infrastructure in the "Platform" such that pre-configured monitoring thresholds, conditions violation etc are sent to Operations Console for any Critical failures in the System.
As an Infrastructure Admin, I would like to configure Monitoring in the "Platform" to send Monitoring alerts to the Operations console for any persistent failures in the System for timely response from the Operations team.
As an IT Operator / SRE, I would like to monitor a gradual but consistent degradation of the Platform services performance / response times so that I can rectify any h/w resource related or scaling issues with the application and prevent missing any SLA(s).
As an IT Operator / SRE, I would like to monitor cluster kube state and node metrics of Platform
As an IT Operator / SRE, I would like to monitor all the namespaces of the Platform

</td></tr>
</table>

Installation

MicroK8s

MicroK8s is a powerful, lightweight, reliable production-ready Kubernetes distribution. It is an enterprise-grade Kubernetes distribution that has a small disk and memory footprint while offering carefully selected add-ons out-the-box, such as Istio, Knative, Grafana, Cilium and more.

Microk8s is a non-elastic, rails-based single-node Kubernetes tool that is focused primarily on offline development, prototyping, and testing.

- To install microk8s:

  sudo snap install microk8s --edge --classic

```
lab@lab-VirtualBox:~$ sudo snap install microk8s --edge --classic
[sudo] password for lab:
Sorry, try again.
[sudo] password for lab:
microk8s (1.20/edge) v1.20.6 from Canonical* installed
```

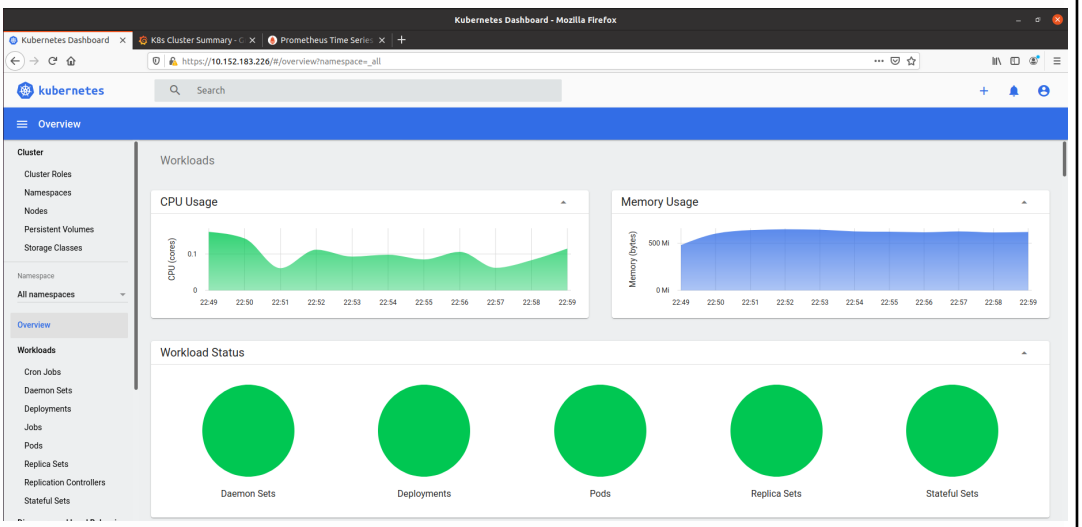
- To start the microk8s:

  sudo microk8s.start

```
microk8s (1.20/edge) v1.20.6 from Canonical* installed
lab@lab-VirtualBox:~$ sudo microk8s.start
Started.
```

The command should report that the service has started and pod scheduling has been enabled.

The kubernetes dashboard is as shown below:



Minikube

Minikube is local Kubernetes, focusing on making it easy to learn and develop for Kubernetes.

- Docker installation: following official docker installation documentation for ubuntu
  sudo apt-get install docker-ce docker-ce-cli containerd.io

- Minikube installation:

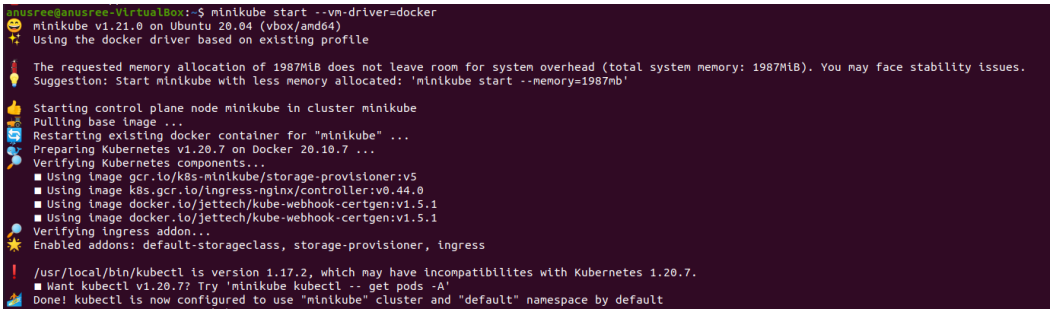
  curl -LO

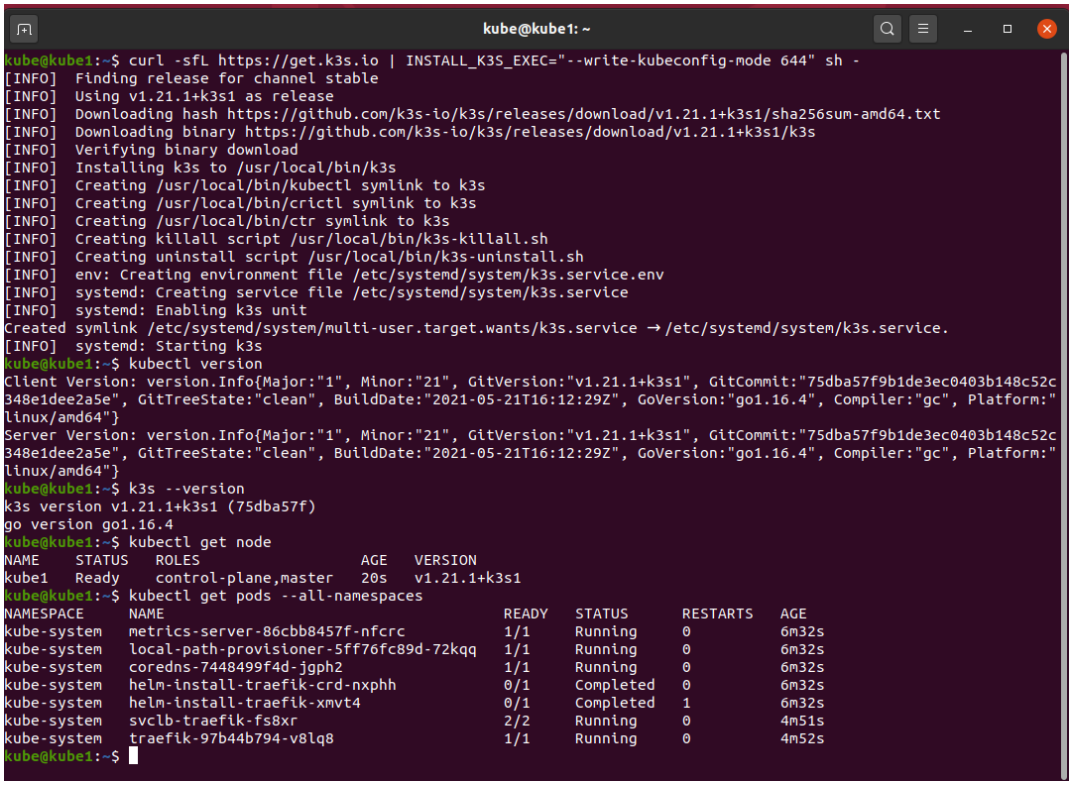
  https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

  sudo install minikube-linux-amd64 /usr/local/bin/minikube

- Start minikube cluster using docker as driver

  minikube start --vm-driver=docker



	K3s: Installing K3s using the script and we observed that seven pods were deployed in the kube-system namespace by default. 
Monitoring	Cluster resource monitoring is essential to follow in real time. In comparison with traditional infrastructure, cluster resources are constantly scaling and changing. You can never know where your pods will be launched on your cluster. For these reasons, we need to monitor both the underlying resources of the cluster and the inner cluster health. For monitoring we have used prometheus and grafana: Prometheus collects metrics from configured targets at given intervals, evaluates rule expressions, displays the results, and can trigger alerts when specific conditions are observed. Grafana is a multi-platform open source analytics and interactive visualization web application. It provides charts, graphs, and alerts for the web when connected to supported data sources. Prometheus and Grafana are installed using helm charts in Minikube and K3s and they are added to microK8s cluster with the help of built-in add-on. ● Enable Prometheus add-on to monitor metrics on MicroK8s Cluster. ● Enabling built-in Prometheus add-on on primary Node.

```
hpe@hpe-VirtualBox:~$ microk8s enable prometheus dashboard dns
Addon dns is already enabled.
Fetching kube-prometheus version v0.7.0.
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   143  100   143    0     0    240      0 --:--:-- --:--:-- --:--:--   240
100  287k    0  287k    0     0   198k      0 --:--:--  0:00:01 --:--:--  198k
kube-prometheus-0.7.0/
kube-prometheus-0.7.0/.github/
kube-prometheus-0.7.0/.github/ISSUE_TEMPLATE/
kube-prometheus-0.7.0/.github/ISSUE_TEMPLATE/bug.md
kube-prometheus-0.7.0/.github/ISSUE_TEMPLATE/feature.md
kube-prometheus-0.7.0/.github/ISSUE_TEMPLATE/support.md
kube-prometheus-0.7.0/.github/workflows/
kube-prometheus-0.7.0/.github/workflows/ci.yaml
kube-prometheus-0.7.0/.gitignore
kube-prometheus-0.7.0/DCO
kube-prometheus-0.7.0/LICENSE
kube-prometheus-0.7.0/Makefile
kube-prometheus-0.7.0/NOTICE
kube-prometheus-0.7.0/OWNERS
kube-prometheus-0.7.0/README.md
kube-prometheus-0.7.0/build.sh
kube-prometheus-0.7.0/code-of-conduct.md
kube-prometheus-0.7.0/docs/
kube-prometheus-0.7.0/docs/EKS-cni-support.md
kube-prometheus-0.7.0/docs/GKE-cadvisor-support.md
kube-prometheus-0.7.0/docs/community-support.md
kube-prometheus-0.7.0/docs/developing-prometheus-rules-and-grafana-dashboards.m
```

- Monitoring the number services and pods running to check whether prometheus and grafana are running or not:

```
hpe@hpe-VirtualBox:~$ microk8s kubectl get pod -n monitoring
NAME                                   READY   STATUS              RESTARTS   A
GE
prometheus-operator-7649c7454f-d9dqr   0/2     ContainerCreating   0          3
m7s
node-exporter-vmwcg                    0/2     ContainerCreating   0          2
m45s
kube-state-metrics-78dc55b74b-jrbfp    0/3     ContainerCreating   0          2
m47s
grafana-6b8df57c5b-z5qcj               0/1     ContainerCreating   0          2
m49s
prometheus-adapter-69b8496df6-pf56c    0/1     ContainerCreating   0          2
m23s
alertmanager-main-0                    0/2     ContainerCreating   0          1
07s
prometheus-k8s-0                       0/2     ContainerCreating   0          1
```

```
hpe@hpe-VirtualBox:~$ microk8s kubectl get services -n monitoring
NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
                AGE
prometheus-operator     ClusterIP   None             <none>        8443/TCP
                3m5s
alertmanager-main       ClusterIP   10.152.183.149   <none>        9093/TCP
                2m51s
grafana                 ClusterIP   10.152.183.245   <none>        3000/TCP
                2m24s
kube-state-metrics      ClusterIP   None             <none>        8443/TCP,944
3/TCP           2m22s
node-exporter           ClusterIP   None             <none>        9100/TCP
                2m21s
prometheus-adapter      ClusterIP   10.152.183.37    <none>        443/TCP
                2m5s
alertmanager-operated   ClusterIP   None             <none>        9093/TCP,909
4/TCP,9094/UDP  85s
prometheus-operated     ClusterIP   None             <none>        9090/TCP
                84s
prometheus-k8s          ClusterIP   10.152.183.135   <none>        9090/TCP
                79s
```

- Prometheus port-forwarding to enable external access:

```
hpe@hpe-VirtualBox:~$ microk8s kubectl port-forward -n monitoring services/prom
etheus-k8s --address 0.0.0.0 9090:9090
Forwarding from 0.0.0.0:9090 -> 9090
Handling connection for 9090
```

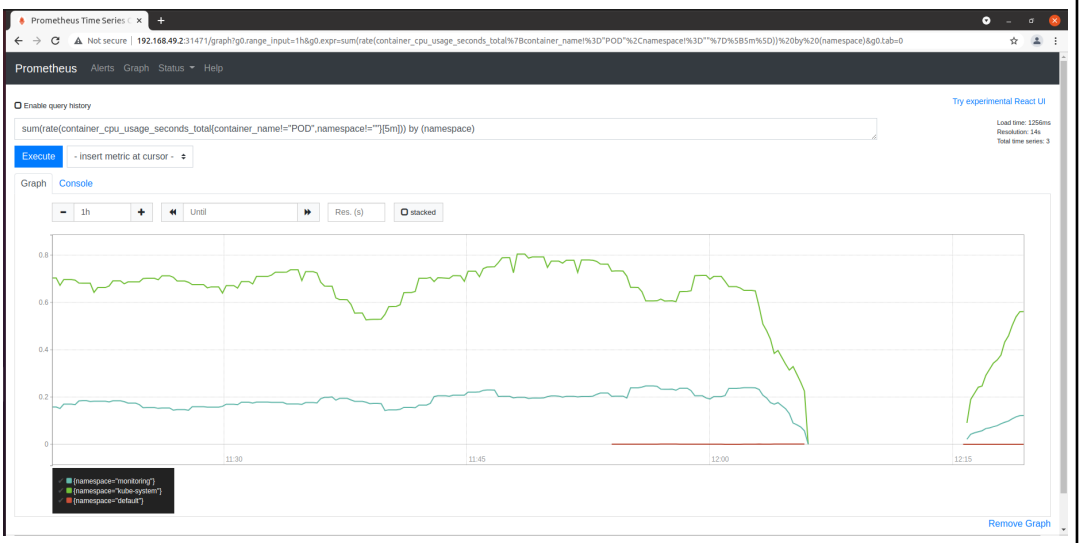
- Grafana port-forwarding to enable external access:

```
hpe@hpe-VirtualBox:~$ microk8s kubectl port-forward -n monitoring services/graf
ana --address 0.0.0.0 3000:3000
Forwarding from 0.0.0.0:3000 -> 3000
Handling connection for 3000
Handling connection for 3000
```

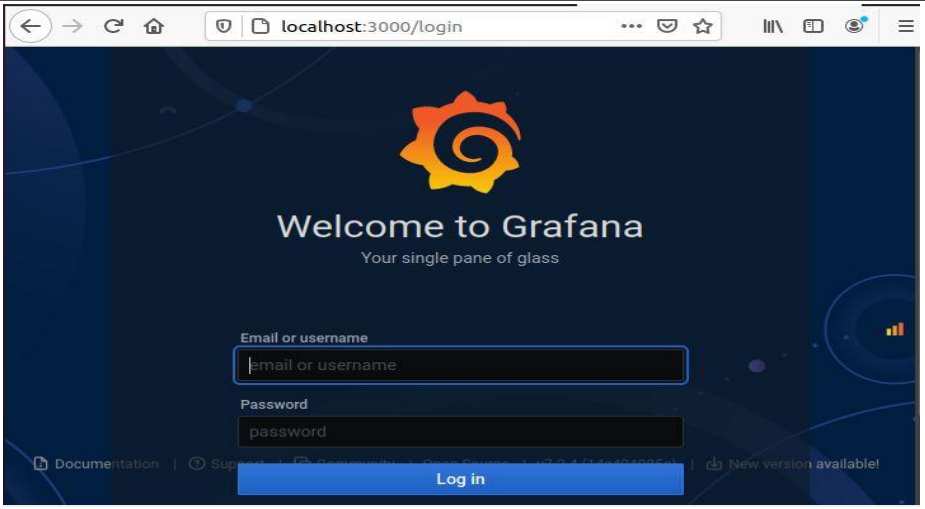
Connection between the prometheus and grafana for monitoring and observing is done via adding data source that is prometheus and importing the dashboard from Grafana.com using 8685. In the dashboard configuration select the Prometheus

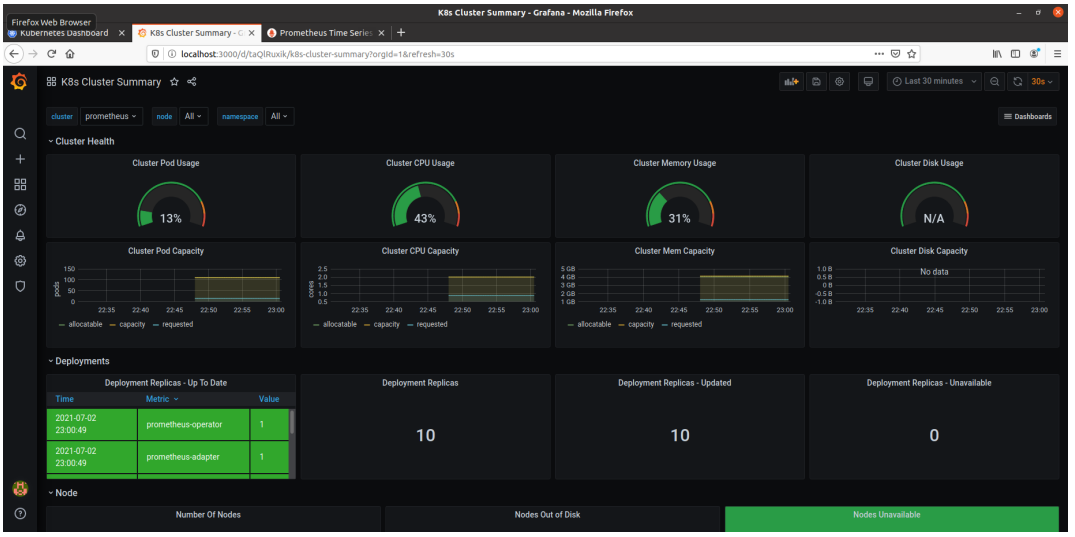
Datasource. And hence we can visualize using grafana.

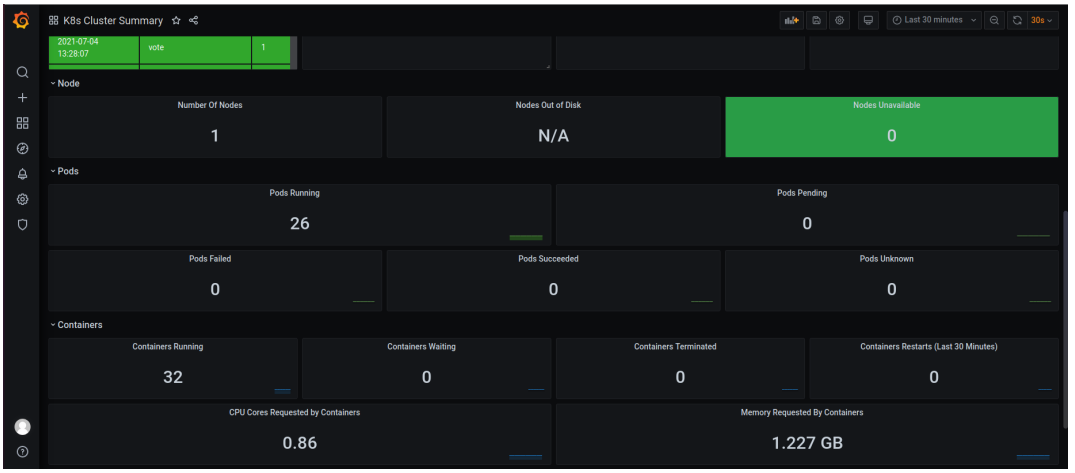
- Prometheus dashboard and graphs depicting the metrics:

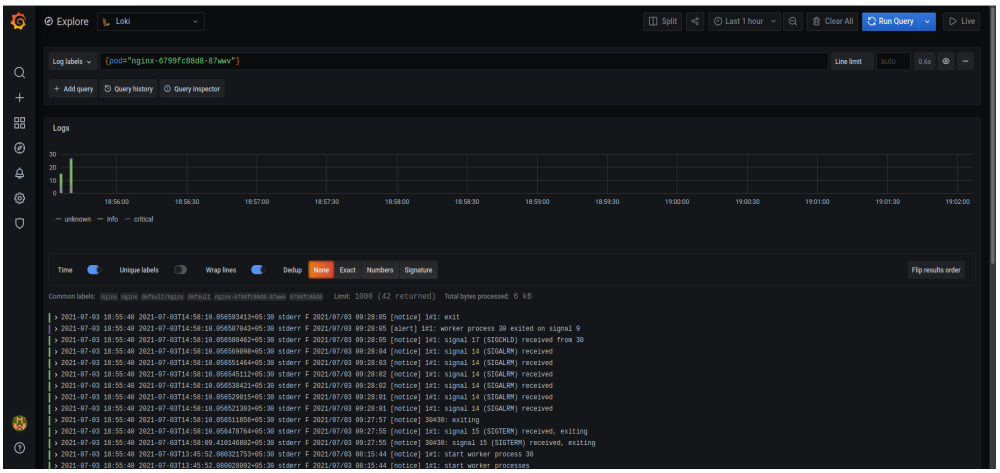
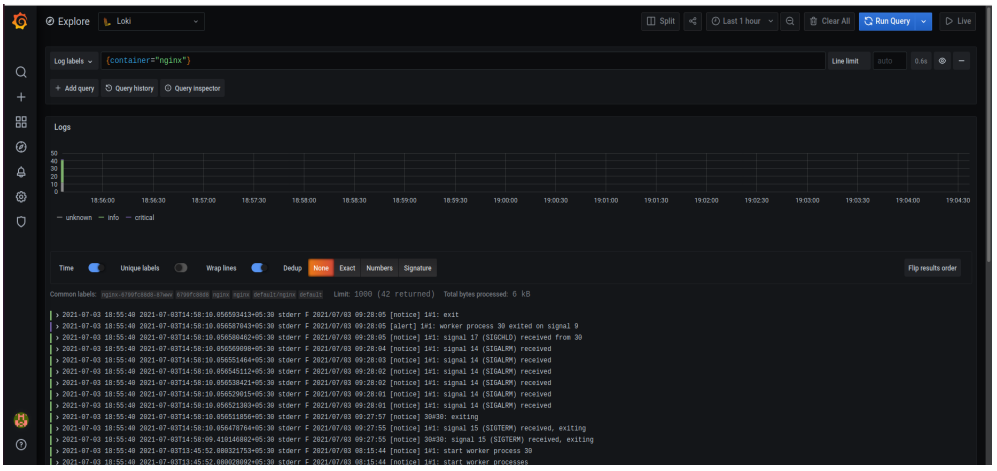
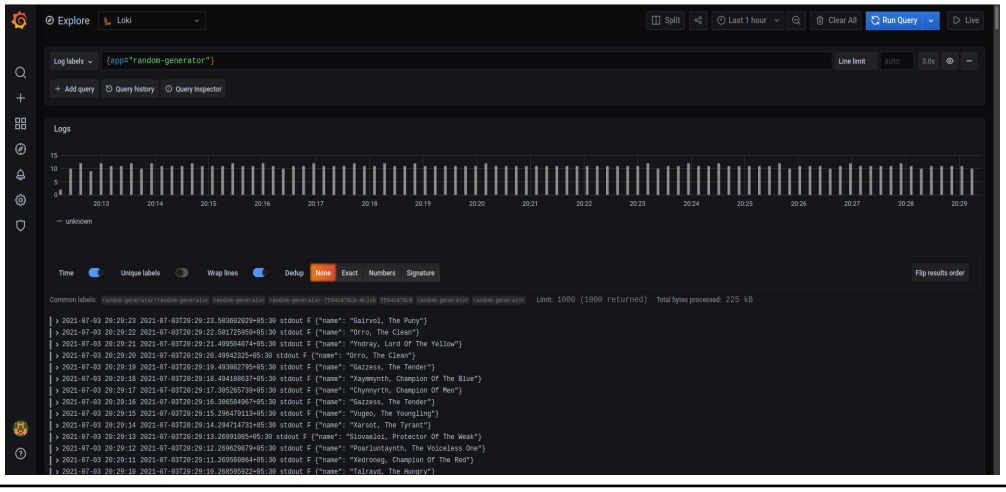


- Grafana dashboard: Visualization of cluster
  It shows the metrics for cluster, nodes, pods, jobs and deployments running on it.

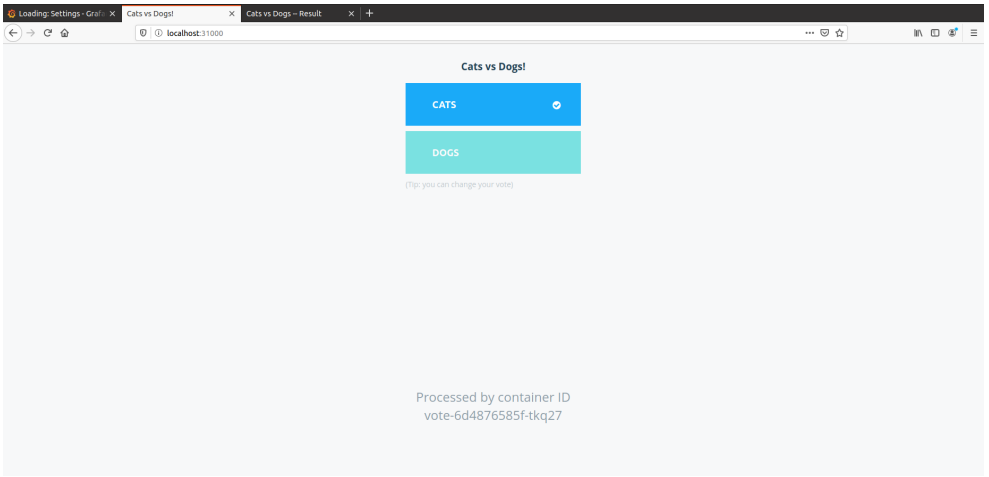

localhost:3000/login
Welcome to Grafana
Your single pane of glass
Email or username
email or username
Password
password
Log in
Documentation
New version available!

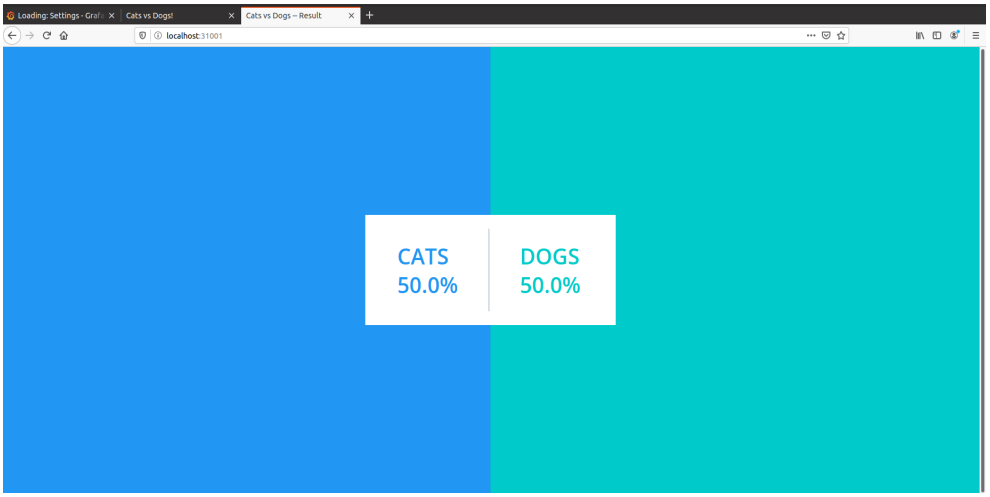

K8s Cluster Summary - Grafana - Mozilla Firefox
K8s Cluster Summary
Cluster Health
Cluster Pod Usage
13%
Cluster CPU Usage
43%
Cluster Memory Usage
31%
Cluster Disk Usage
N/A
Cluster Pod Capacity
Cluster CPU Capacity
Cluster Mem Capacity
Cluster Disk Capacity
No data
Deployments
Deployment Replicas - Up To Date
prometheus-operator
prometheus-adapter
Deployment Replicas
10
Deployment Replicas - Updated
10
Deployment Replicas - Unavailable
0
Node
Number Of Nodes
Nodes Out of Disk
Nodes Unavailable

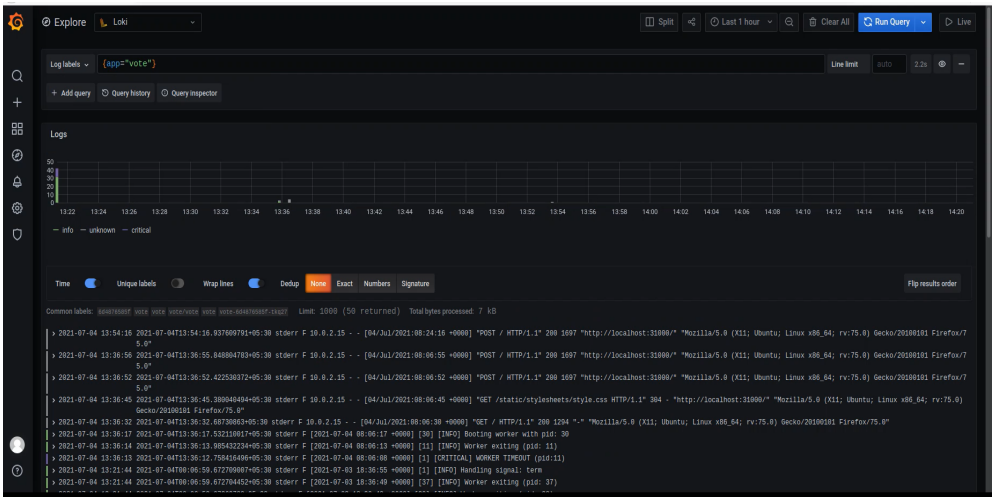

K8s Cluster Summary
vote
Node
Number Of Nodes
1
Nodes Out of Disk
N/A
Nodes Unavailable
0
Pods
Pods Running
26
Pods Pending
0
Pods Failed
0
Pods Succeeded
0
Pods Unknown
0
Containers
Containers Running
32
Containers Waiting
0
Containers Terminated
0
Containers Restarts (Last 30 Minutes)
0
CPU Cores Requested by Containers
0.86
Memory Requested By Containers
1.227 GB

Logging	The logs are particularly useful for debugging problems and monitoring cluster activity. Logs can explain latency and errors. We have used Loki data source to get logging information. • Logs for nginx app:   • We have deployed an app which randomly generates 30 names of dragons for every 30 seconds. The below figure shows the logs of random-generator app: 

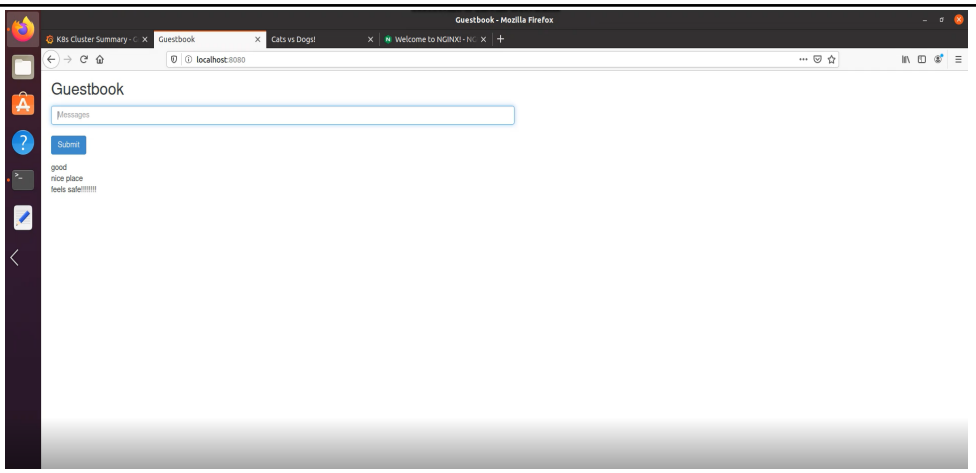
- We have deployed a voting app in which we can vote between cats and dogs; logs for the same are displayed below:

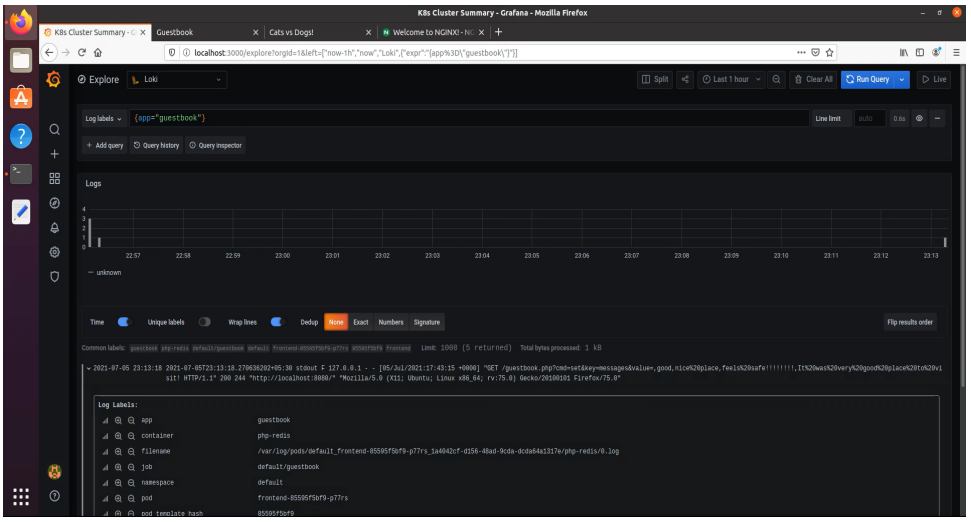




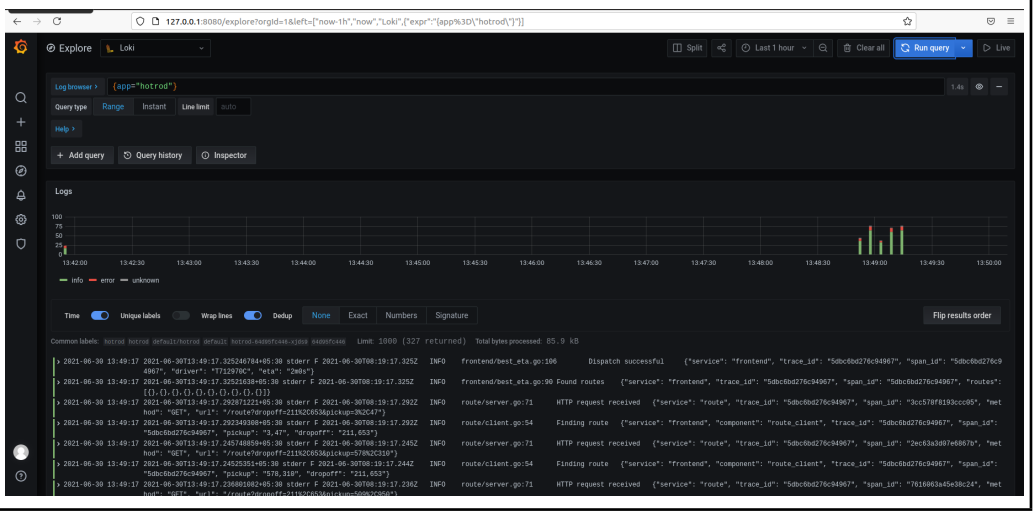


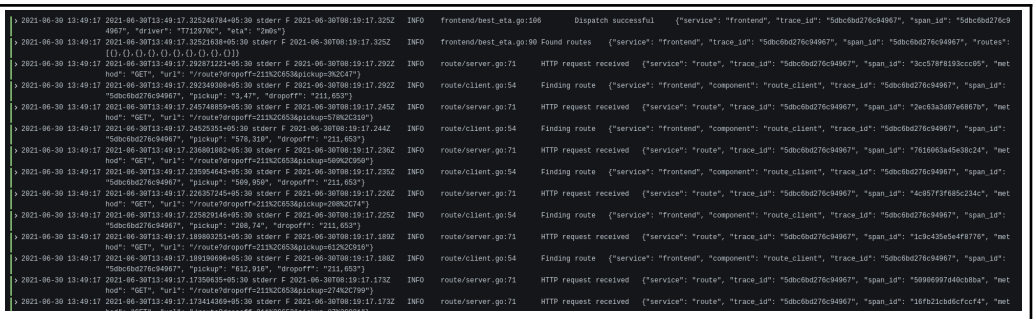
- We have deployed a PHP guestbook front-end application with redis as database. This application is a means of acknowledgement by the visitors where they can post their reviews about the place. The logs for the same are shown below:





- We have also deployed a hot r.o.d app in which there are four customers requesting for a car ride,this sends a get request to the server, the logs for the same are shown below:



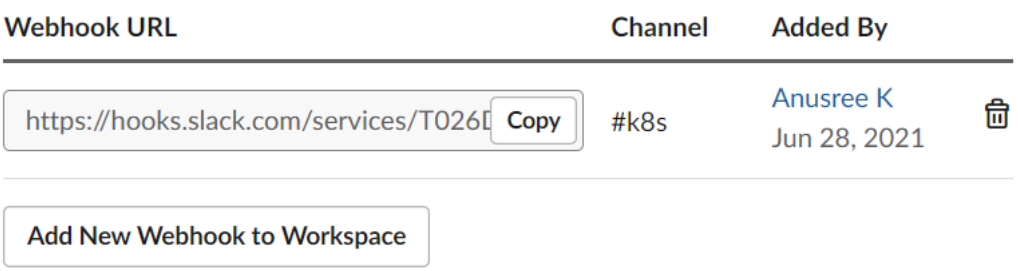


Alerting

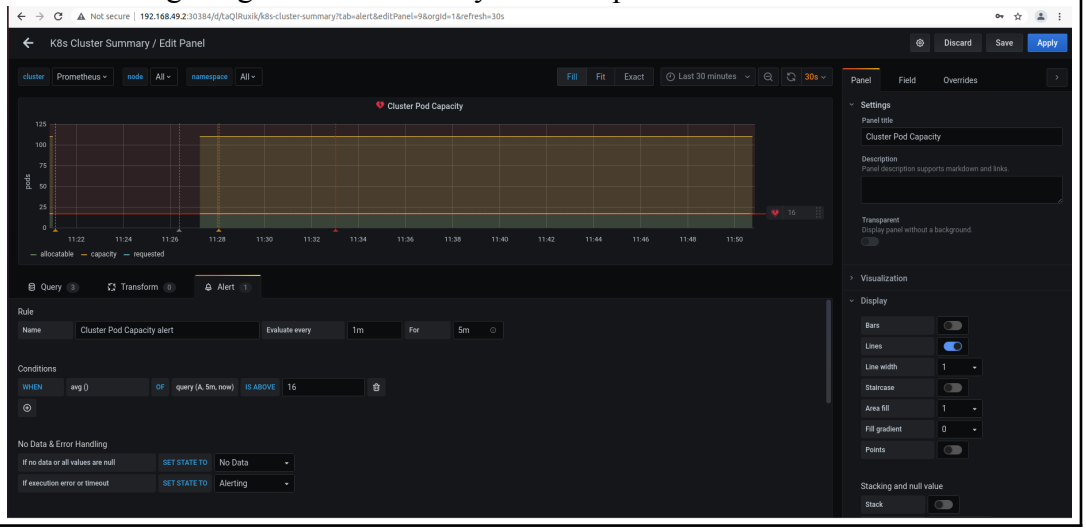
We understand that an Ops team supervising several clusters in production cannot keep constantly an eye on their dashboards. We therefore utilize alert features.
There are two ways to implement Alerting in our monitoring stack. We might use the built-in alerting function of Grafana or the prometheus AlertManager component (which is installed by the helm chart).
Grafana can send an alert on Slack, mail, webhook or other communication channels.

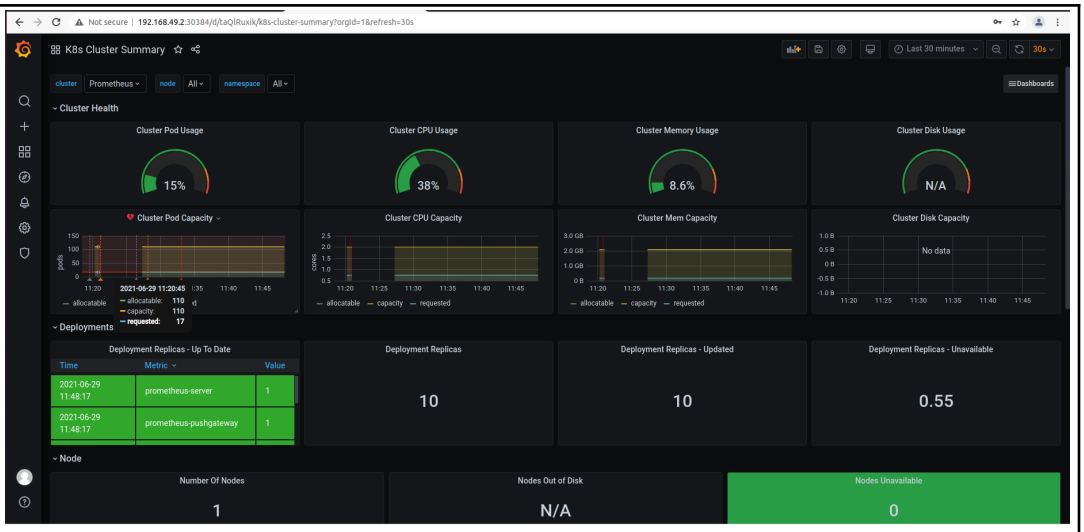
Alerting using built-in alerting function of Grafana:

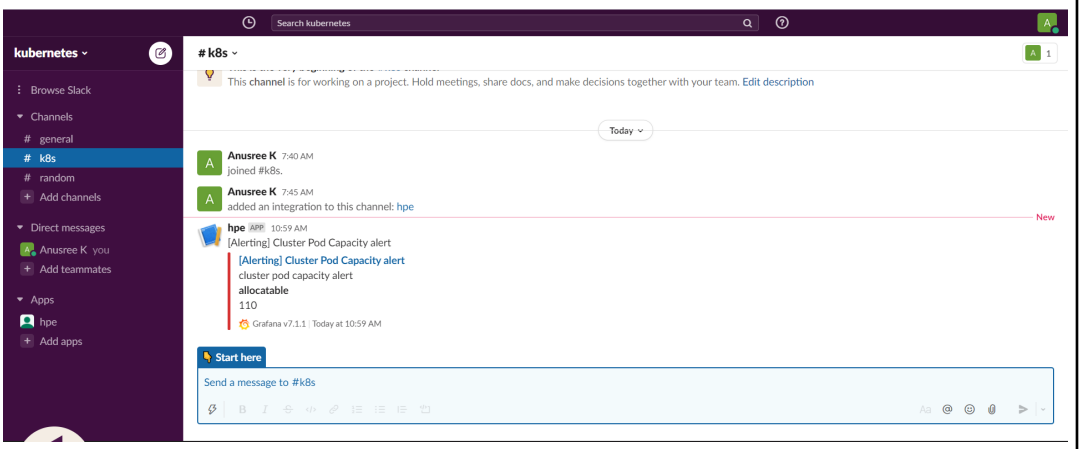
- Create a slack webhook URL : It will help us create an endpoint for sending messages to a specific channel of our Slack server.



- Add slack as a notification channel in Grafana
- Once the channel is set up, go to the “K8S Cluster Summary” dashboard, and click on the Cluster Pod Capacity title, to edit the panel.
- We set up an alert to monitor the pod capacity, which is limited at 110 by Kubernetes by default on each node. This limit is considered as the limit of reliability. If the number of nodes in the cluster is limited and if we reach this hard limit of 110, the remaining pods will enter in a pending state which could be very problematic in a production cluster when scaling during a high business activity for example.



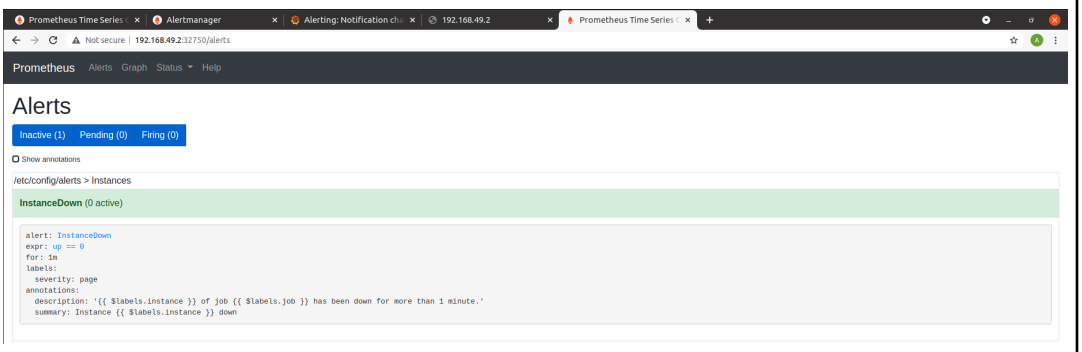




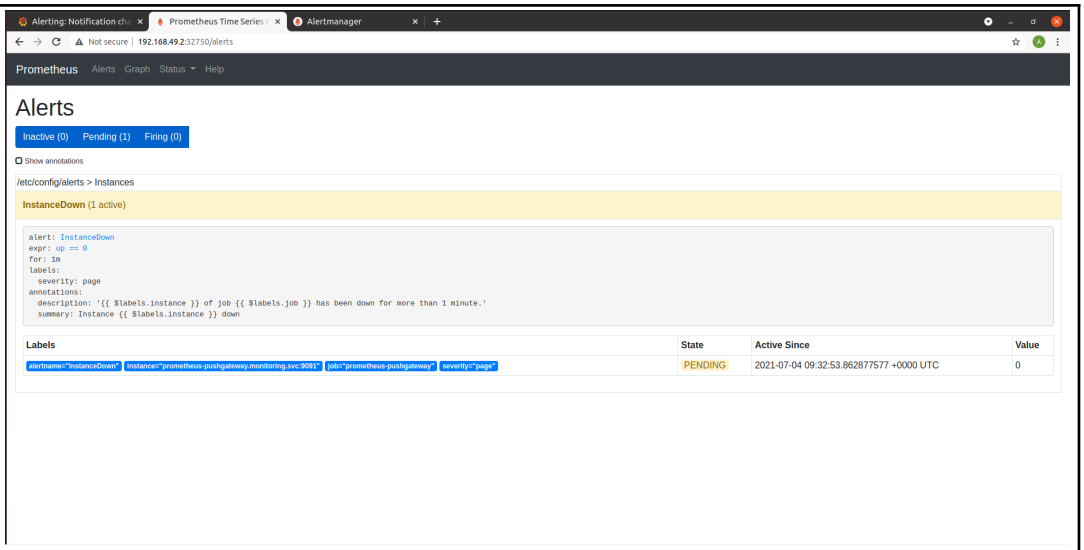
Alerting using Alertmanager:

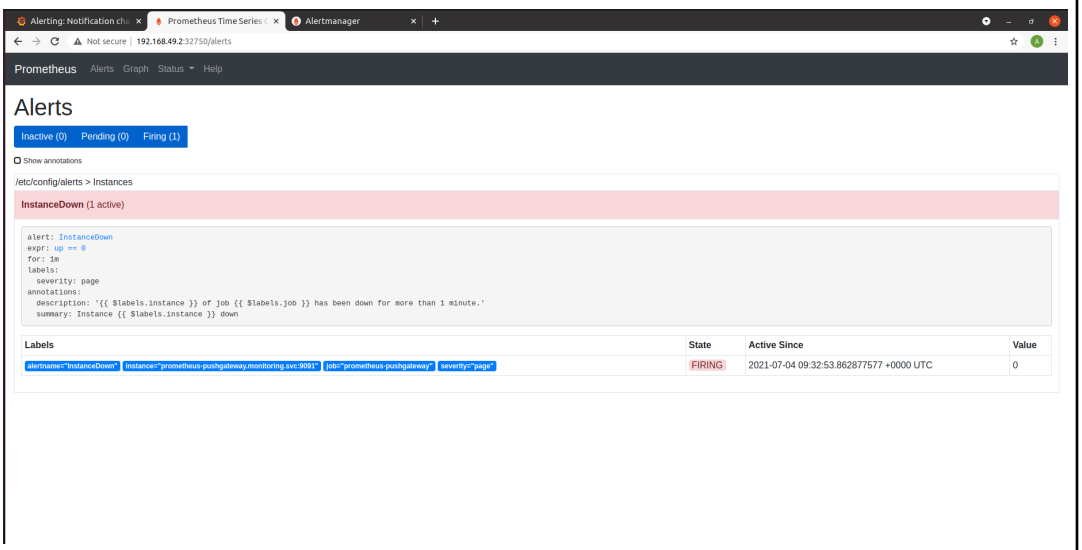
Prometheus is configured with alerting rules which define conditions our metrics should meet (along with some basic information about the alert).
Before we configure Alertmanager, add an alert to Prometheus; we create a values.yaml file to configure our helm chart and add a single alert to it which would trigger if a target is down for 1 minute.

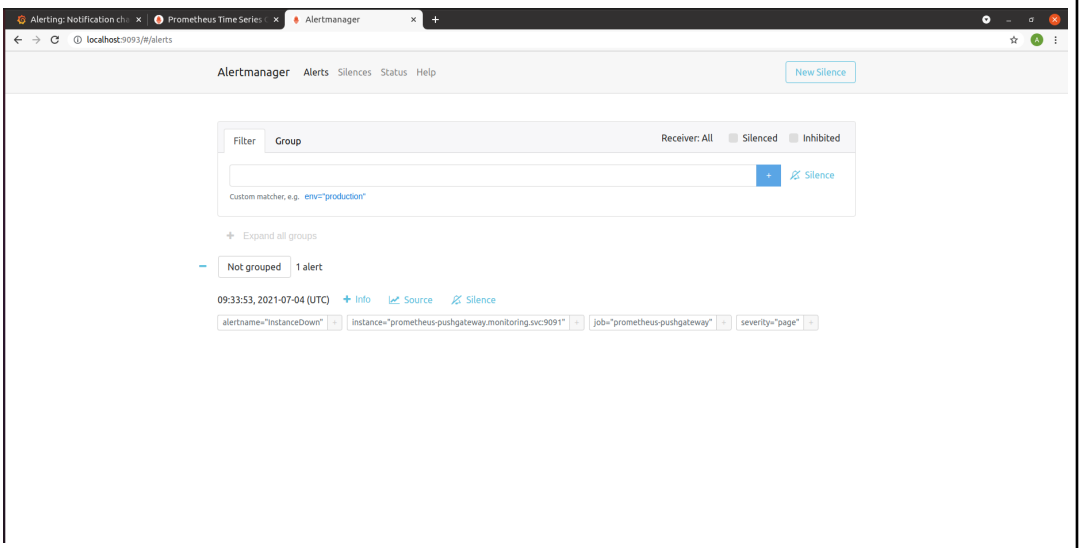
helm upgrade -f values.yaml prometheus stable/prometheus



We destroy one of our target endpoints to see this in action.
kubectl delete deployment -l app=prometheus,component=pushgateway

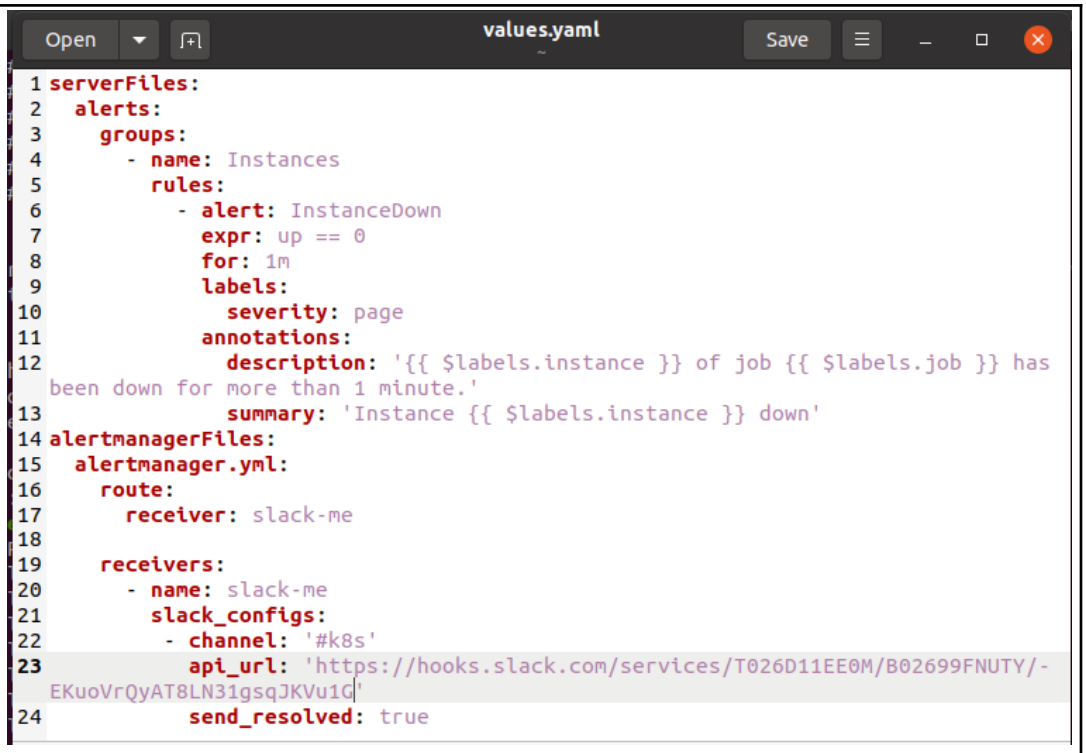






Forwarding alerts to slack:

To add a Slack receiver, we update our values.yaml chart configuration by adding a snippet containing the slack webhook url and channel.

<table>
<tr>
<td></td>
<td>


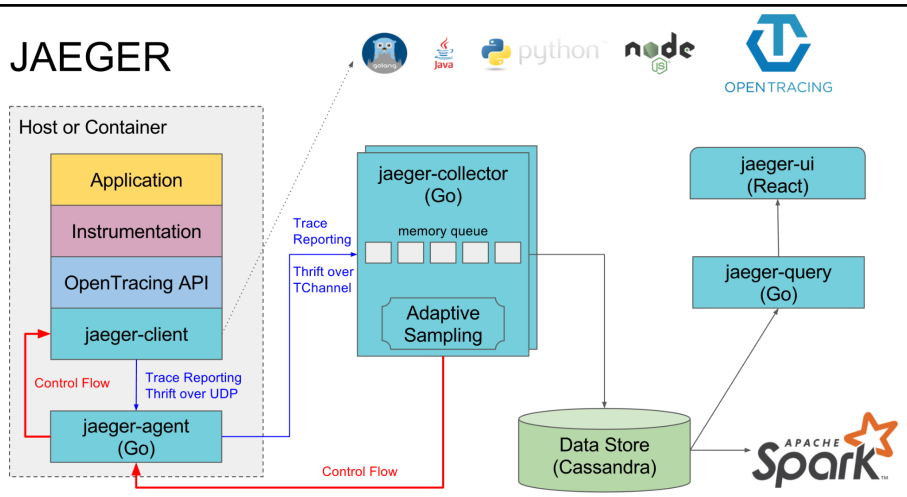

```yaml
serverFiles:
  alerts:
    groups:
      - name: Instances
        rules:
          - alert: InstanceDown
            expr: up == 0
            for: 1m
            labels:
              severity: page
            annotations:
              description: '{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 1 minute.'
              summary: 'Instance {{ $labels.instance }} down'
alertmanagerFiles:
  alertmanager.yml:
    route:
      receiver: slack-me

    receivers:
      - name: slack-me
        slack_configs:
          - channel: '#k8s'
            api_url: 'https://hooks.slack.com/services/T026D11EE0M/B02699FNUTY/-EKuoVrQyAT8LN31gsqJKVu1G'
            send_resolved: true
```

Now we upgrade the helm chart to get the alerts on slack

</td>
</tr>
<tr>
<td>Tracing</td>
<td>

A trace is a representation of consecutive events of a single operation which reflect an end-to-end request path in a distributed system.Unlike logging, which is event triggered and discrete, tracing provides a broader and continuous application view.
Tracing helps us understand the path of a process/transaction/entity while traversing the application stack and identifying the bottlenecks at various stages. This helps to optimize the application and increase performance.
Jaeger is open source software for tracing transactions between distributed services and is based on the vendor-neutral OpenTracing APIs and instrumentation and it comprises an additional agent on each host to aggregate data in batches before sending it to the collector.
The following shows the Jaeger architecture:

</td>
</tr>
</table>



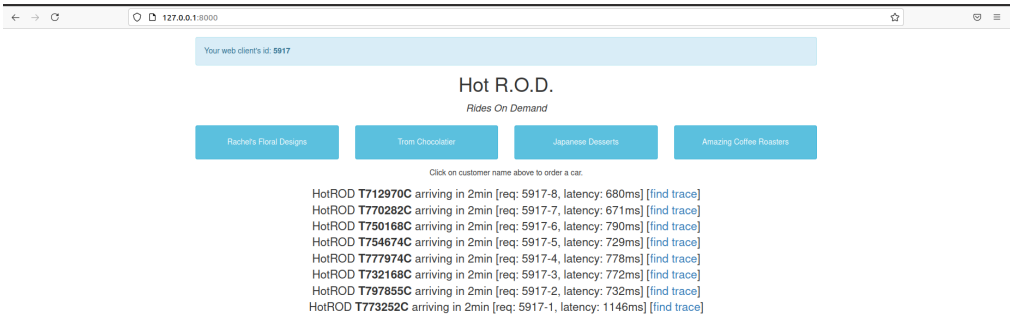
It comprises an additional agent on each host to aggregate spans from the Jaeger client(application) instrumented with OpenTracing , in batches before sending it to the collector.The collector stores the execution traces in a datastore for persistence.The traces can be visualized on Jaeger UI using Jaeger query.

Installing Jaeger:

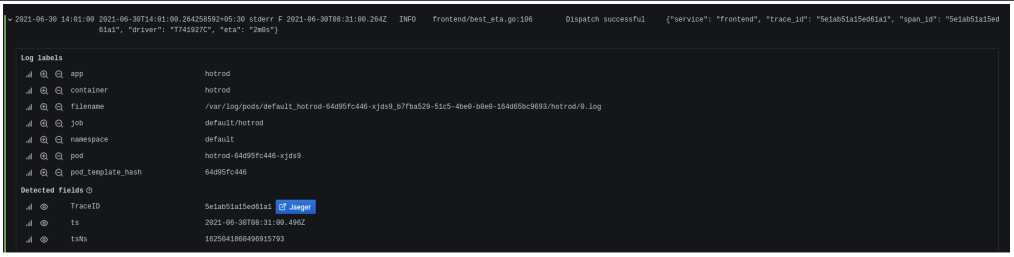
Jaeger was installed by cloning the jaegertracing github repo and converting docker-compose manifests to kubernetes using kompose.

```
kube@kube1:~$ kubectl get pod -o wide
NAME                                                READY   STATUS    RESTARTS   AGE     IP            NODE
prometheus-kube-state-metrics-5786c7446c-d5bxz      1/1     Running   3          5d19h   10.42.0.50    kube1
nginx-deployment-585449566-jwlj4                    1/1     Running   2          20h     10.42.0.53    kube1
prometheus-node-exporter-thjw2                      1/1     Running   3          5d19h   10.0.2.15     kube1
alertmanager-prometheus-kube-prometheus-alertmanager-0   2/2     Running   6          5d18h   10.42.0.46    kube1
prometheus-prometheus-kube-prometheus-prometheus-0  2/2     Running   7          5d18h   10.42.0.45    kube1
prometheus-kube-prometheus-operator-98d5bdcd6-2rwjl 1/1     Running   3          5d19h   10.42.0.52    kube1
grafana-6f67f447ff-59qnb                            1/1     Running   3          5d18h   10.42.0.56    kube1
loki-promtail-qnvqz                                 1/1     Running   0          17m     10.42.0.58    kube1
loki-0                                              1/1     Running   0          17m     10.42.0.59    kube1
hotrod-64d95fc446-5vjvv                             1/1     Running   0          6m49s   10.42.0.60    kube1
jaeger-7dd885bd4b-rvxv9                             1/1     Running   0          4m21s   10.42.0.61    kube1
```

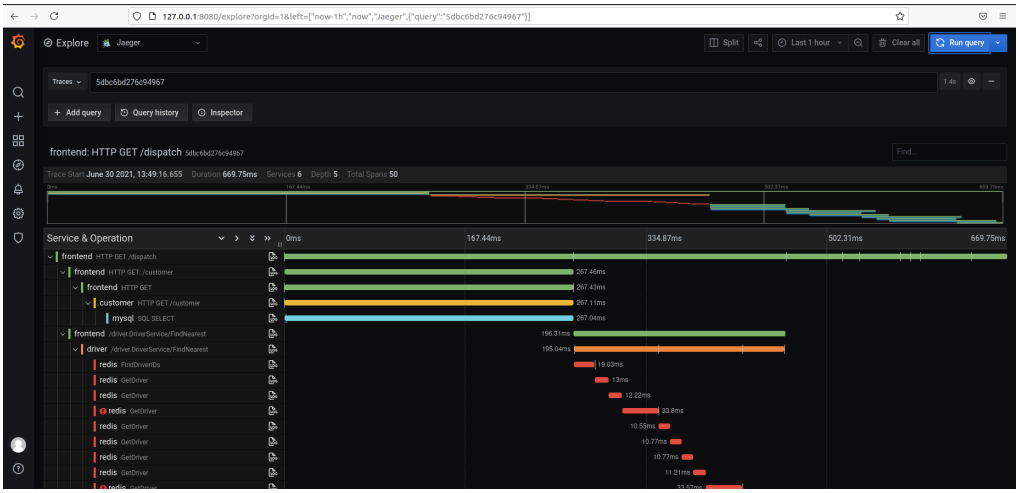
- The Following shows the HotRod UI with various requests with latency and driver ID:



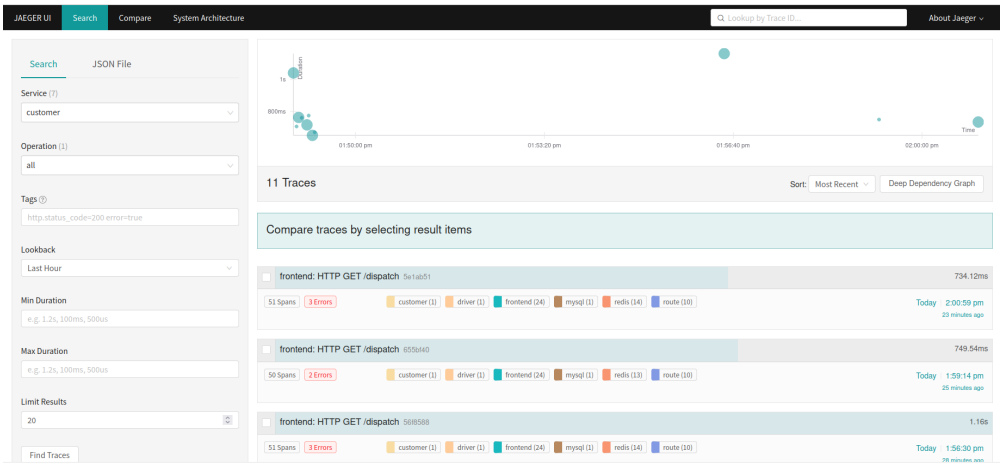
- In Grafana while adding the Loki datasource,we have added a derived field with internal link to Jaeger,which allows us to directly look at the traces for the event “dispatch successful” after entering the query {app=”hotrod”} in logQL.



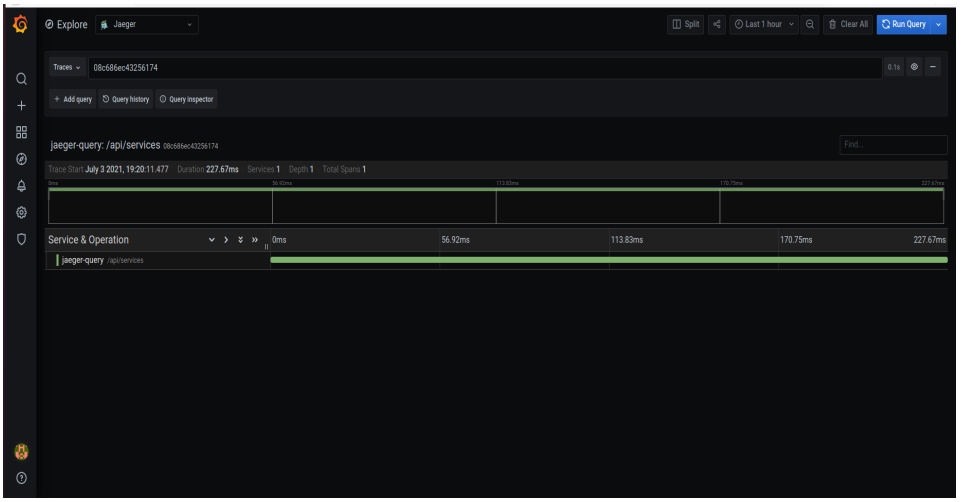
- These are the traces for the requests which helps to identify the bottlenecks in the services and rectify them.The spans in Red are the errors encountered while fulfilling the request by the server.

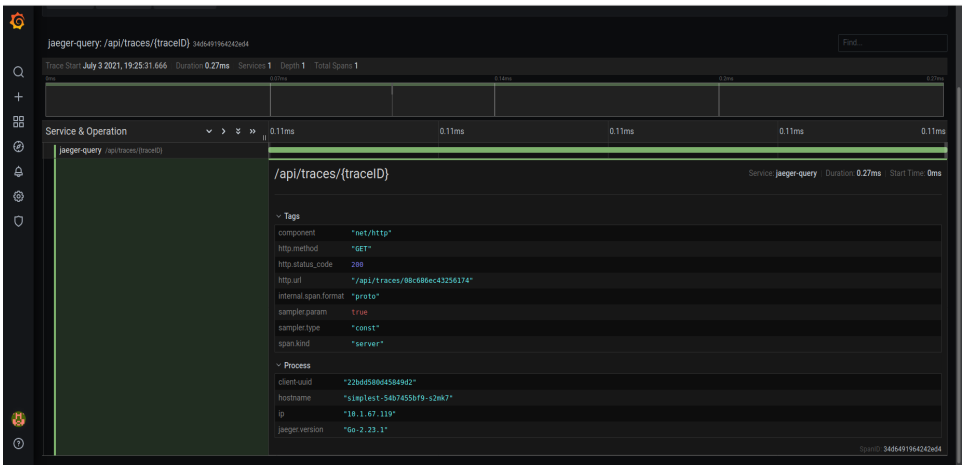


This shows the Jaeger UI which enables us to filter traces based on trace ID, services and operations and view them.



The following shows the trace and the span context for a particular log by specifying the Trace ID jaeger





Multi-Cluster	**What is Multi-cluster?** Multi-cluster is a strategy for deploying an application on or across multiple Kubernetes clusters with the goal of improving availability, isolation, and scalability. Multi-cluster can be important to ensure compliance with different and conflicting regulations, as individual clusters can be adapted to comply with geographic- or certification-specific regulations. The speed and safety of software delivery can also be increased, with individual development teams deploying applications to isolated clusters and selectively exposing which services are available for testing and release. **Why Multi-cluster?** 1. **Improved Operational Readiness**: By standardizing cluster creation, the associated operational runbooks, troubleshooting, and tools are simplified. This eliminates common sources of operational error while also reducing the cognitive load for support engineers and SREs, which ultimately leads to improved overall response time to issues. 2. **Isolation and Multi-Tenancy**: Strong isolation guarantees simplify key operational processes such as cluster and application upgrades. Moreover, isolation can reduce the blast radius of a cluster outage. Organizations with strong tenancy isolation requirements can route each tenant to their individual cluster. 3. **Increased Availability and Performance**: Multi-cluster enables applications to be deployed in or across multiple availability zones and regions, improving application availability and regional performance for global applications. 4. **Compliance**: Cloud applications today have to comply with a myriad of regulations and policies. A single cluster is unlikely to be able to comply with every regulation. A multi-cluster strategy reduces the scope of compliance for each individual cluster 5. **Eliminate Vendor Lock-In**: A multi-cluster strategy enables your organization to shift workloads between different Kubernetes vendors to take advantage of new capabilities and pricing offered by different vendors. **Multi-Cluster Application Architecture** 1. **Replicated**: In this model, each cluster runs a full copy of the application. This simple but powerful approach enables an application to scale globally, as the application can be replicated into multiple availability zones or data centers and user traffic routed to the closest or most appropriate cluster. Coupled with a health-aware global load balancer, this architecture also enables failover; if one cluster stops functioning or becomes unresponsive, user traffic is routed to another cluster. 2. **Split-by-Service:** In this model, the services of a single application or system are divided across multiple clusters. This approach provides

stronger isolation between parts of the application at the expense of greater complexity. This pattern is often used to ease compliance with regulatory requirements. For example, PCI DSS compliant services and supporting infrastructure can be localised into a single cluster, and the remaining application clusters can be operated outside of this scope. This pattern also facilitates speed and safety during application development and delivery, as individual development teams can deploy their specific services into their own cluster without impacting other teams.

Installation of Multicluster:

Step 1: Install the multicluster control plane
On each cluster, run:

```
linkerd multicluster install | \
 kubectl apply -f -
```

To verify that everything has started up successfully, run:

```
linkerd multicluster check
```

Step 2: Link the clusters

Each cluster must be linked. This consists of installing several resources in the source cluster including a secret containing a kubeconfig that allows access to the target cluster Kubernetes API, a service mirror control for mirroring services, and a Link custom resource for holding configuration. To link cluster west to cluster east, you would run:

```
linkerd --context=east multicluster link --cluster-name east |

kubectl --context=west apply -f -
```

To verify that the credentials were created successfully and the clusters are able to reach each other, run:

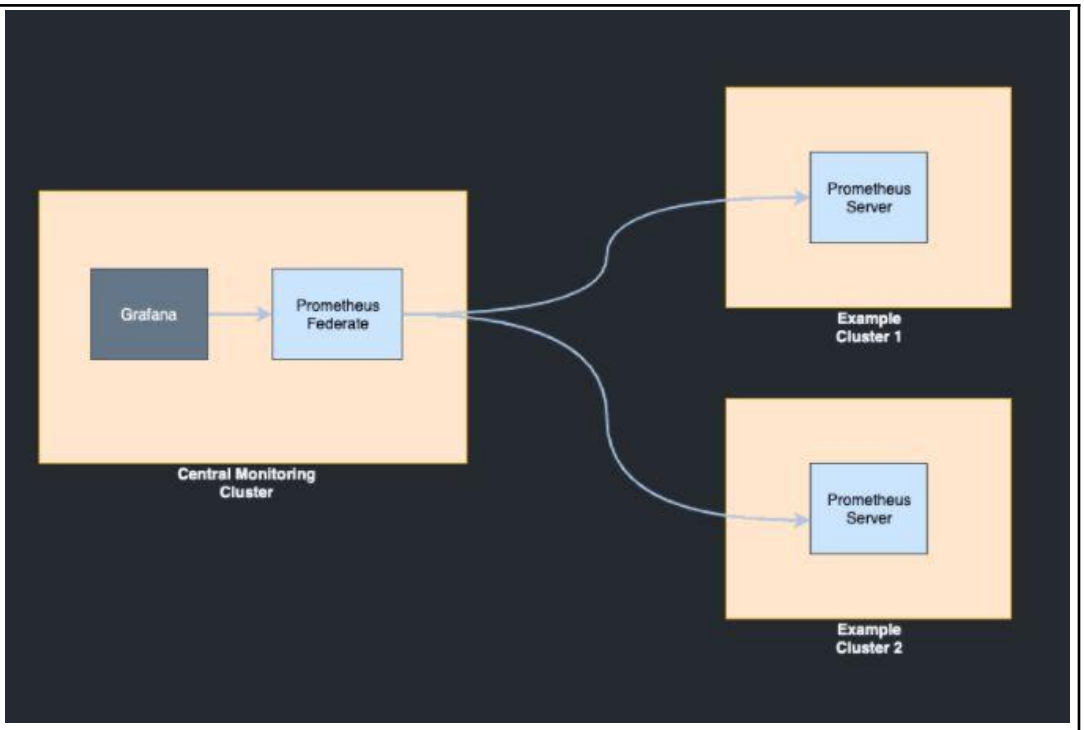
```
linkerd --context=west multicluster check
```

You should also see the list of gateways that show up by running. Note that you'll need Linkerd's Viz extension to be installed in the source cluster to get the list of gateways:

```
linkerd --context=west multicluster gateways
```

Step 3: Export services
Services are not automatically mirrored in linked clusters. By default, only services with the mirror.linkerd.io/exported label will be mirrored. For each service you would like mirrored to linked clusters, run:

	kubectl label svc foobar mirror.linkerd.io/exported=true **Monitoring Multi-cluster** For the monitoring tooling, we selected a combination of the open source Prometheus and Grafana tools. Prometheus acts as the storage and query engine while Grafana acts as the interface for visualization of the monitoring data via dashboards and panels. Grafana offers an easy to read interface that can be easily deployed and managed in a Kubernetes cluster. By supporting multiple data sources (Prometheus, MySQL, AWS CloudWatch, etc.), it can be used to monitor the whole infrastructure. Prometheus collects metrics by scraping data from the clusters and we selected it for its simplicity and support, as well as its Prometheus Federation service, which can be used to scale to hundreds of clusters. Prometheus federation is a Prometheus server that can scrape data from other Prometheus servers. It supports hierarchical federation, which in our case resembles a tree. At Mattermost, a default version of the Prometheus server is installed in each one of our clusters and a Prometheus federation server is deployed together with Grafana in a central monitoring cluster. Prometheus federation scrapes data from all the other Prometheus servers that run in our clusters. For future expansion, a central Prometheus federation can be used to scrape data from multiple Prometheus federation servers that scrape data from groups of tens of clusters.

	
Learning Outcomes	• Importance of containerization and container orchestration. • Importance of using Kubernetes. • Using different tools to set up kubernetes clusters. • Importance of observability and monitoring, • Exploring different tools for observability and monitoring. • Deploying multiple applications on a kubernetes cluster. • Practical implementation of three pillars of observability i.e metrics,logging,tracing. • Exploring multi-cluster observability.
Future Work	• Creating a central dashboard for visualizing metrics of multiple clusters. • Using Spark to process Big Data on Kubernetes. • Implementation of load balancers using traefik. • Monitoring and observability using EFK Stack. • Observability using Open Telemetry. • Setting up a High availability cluster using K3d. • AI/ML stack on Kubernetes.
References	https://kubernetes.io/docs/home/ https://lightstep.com/observability/ https://boxboat.com/2019/08/08/monitoring-kubernetes-with-prometheus/ https://blog.marcnuri.com/prometheus-grafana-setup-minikube

	https://gregoiredayet.medium.com/monitoring-and-alerting-on-your-kubernetes-cluster-with-prometheus-and-grafana-55e4b427b22d https://medium.com/codex/setup-grafana-loki-on-local-k8s-cluster-minikube-90450e9896a8 https://www.infracloud.io/blogs/tracing-grafana-tempo-jaeger/ https://www.jaegertracing.io/docs/1.23/architecture/ https://www.redhat.com/en/topics/microservices/what-is-jaeger https://www.getambassador.io/learn/multi-cluster-kubernetes/