



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. Nandhini V Associate Professor	HPE Mentors	Divakar Padiyar Sonu Sudhakaran		
Review for the Period	09-03-2021	08-07-2021			
Task Given	<p>Open source Monitoring and Observability Stack on Kubernetes</p> <p>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.</p> <p>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.</p>				
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	<p>Kubernetes is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications.</p> <p>We started our project by exploring and understanding concepts of containerization and the role of Kubernetes.</p> <p>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.</p>
Observability	<p>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.</p> <p>The three pillars of observability are logs, traces and metrics.</p> <ul style="list-style-type: none"> ● Logs represent discrete events and data in a structured textual form. They contain both the log 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

	<p>telemetry signals. Metrics are stored in time series like Prometheus. Tools used for metrics are: Prometheus, Graphite, InfluxDB, OpenTSDB and OpenCensus.</p> <p>The diagram illustrates the relationship between three monitoring paradigms: Tracing, Metrics, and Logging. It consists of three overlapping circles. The top circle is labeled "Metrics Aggregatable". The bottom-left circle is labeled "Tracing Request scoped". The bottom-right circle is labeled "Logging Events". Arrows point from the labels to their respective circles. A vertical arrow on the left indicates "Low volume" at the top and "High volume" at the bottom, pointing downwards. "Request-scoped metrics" points to the top circle. "Request-scoped events" points to the bottom-left circle. "Aggregatable events e.g. rollups" points to the right side of the top circle. "Request-scoped, aggregatable events" points to the bottom-right circle.</p>
Observability Monitoring	<p>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.</p> <p>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.</p> <p>Observability and monitoring complement each other, with each one serving a different purposes.</p> <p>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.</p> <p>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.</p>

	Observability	Monitoring
What is my system doing?	Is my system working?	
Tells you why something goes wrong	Tells you when something went wrong	
Proactive in nature	Reactive in nature	
Reduces the duration and impact of incidents	Enables quick response when an incident occurs	
Gain understanding actively	Consume information passively	
Build to tame dynamic environments with changing complexity	Built to maintain static environments with little variation	
Preferred by developers of systems with variability and unknown permutations	Used by developers of systems with little change and known permutation	
Observability Use Cases	<ul style="list-style-type: none"> ● 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 	

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
```

Installation

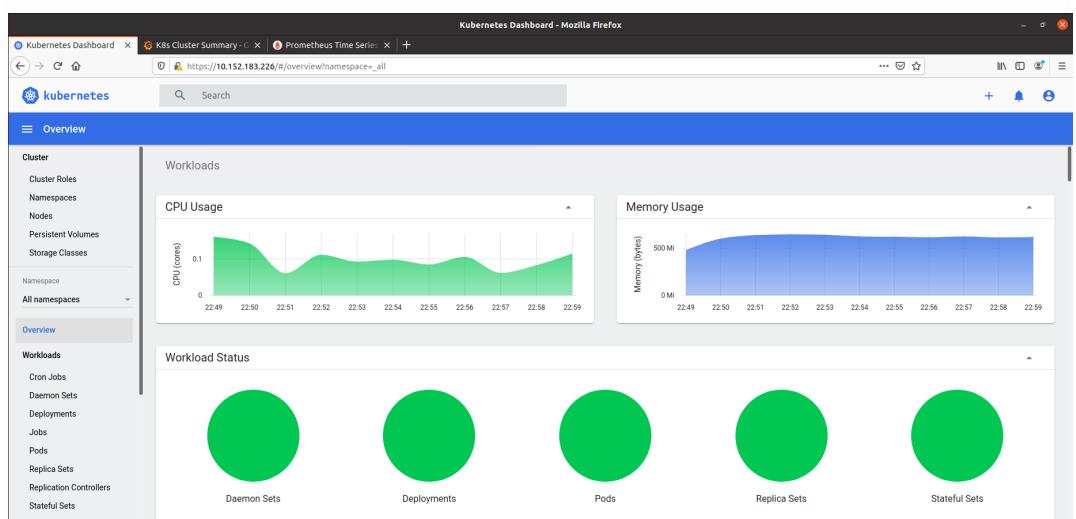
- To start the microk8s:

```
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

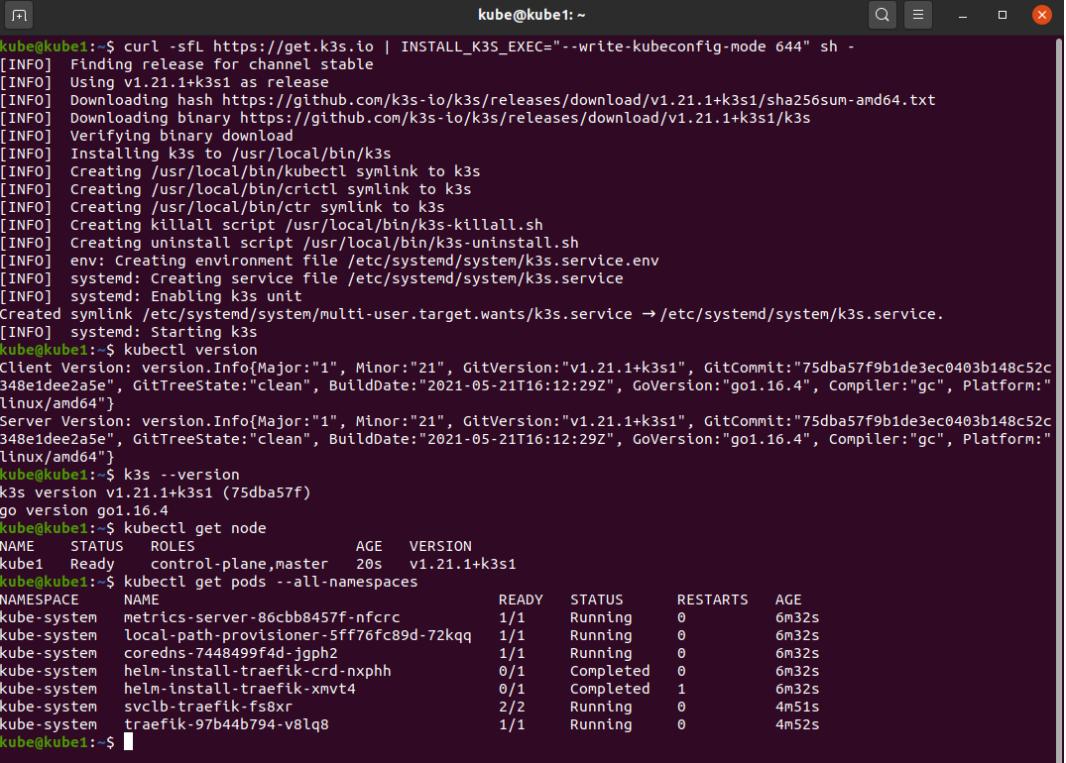
```
minikube start --vm-driver=docker
```

```
anusree@anusree-VirtualBox: ~ $ minikube start --vm-driver=docker
🌟 minikube v1.21.0 on Ubuntu 20.04 (vbox/amd64)
💡 Using the docker driver based on existing profile

❗ The requested memory allocation of 1987MiB does not leave room for system overhead (total system memory: 1987MiB). You may face stability issues.
💡 Suggestion: Start minikube with less memory allocated: 'minikube start --memory=1987mb'

👉 Starting control plane node minikube in cluster minikube
🌐 Pulling base image ...
🔄 Restarting existing docker container for "minikube" ...
🌐 Preparing Kubernetes v1.20.7 on Docker 20.10.7 ...
🌐 Verifying Kubernetes components...
    ■ Using Image gcr.io/k8s-minikube/storage-provisioner:v5
    ■ Using Image k8s.gcr.io/ingress-nginx/controller:v0.44.0
    ■ Using Image docker.io/jettech/kube-webhook-certgen:v1.5.1
    ■ Using Image docker.io/jettech/kube-webhook-certgen:v1.5.1
🌐 Verifying ingress addon...
⭐ Enabled addons: default-storageclass, storage-provisioner, ingress

❗ /usr/local/bin/kubectl is version 1.17.2, which may have incompatibilities with Kubernetes 1.20.7.
    ■ Want kubectl v1.20.7? Try `minikube kubectl -- get pods -A`
💡 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

	<p>K3s:</p> <p>Installing K3s using the script and we observed that seven pods were deployed in the kube-system namespace by default.</p>  <pre> kube@kube1:~\$ curl -sfL https://get.k3s.io INSTALL_K3S_EXEC="--write-kubeconfig-mode 644" sh - [INFO] Finding release for channel stable [INFO] Using v1.21.1+k3s1 as release [INFO] Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.21.1+k3s1/sha256sum-amd64.txt [INFO] Downloading binary https://github.com/k3s-io/k3s/releases/download/v1.21.1+k3s1/k3s [INFO] Verifying binary download [INFO] Installing k3s to /usr/local/bin/k3s [INFO] Creating /usr/local/bin/kubectl symlink to k3s [INFO] Creating /usr/local/bin/crtl symlink to k3s [INFO] Creating /usr/local/bin/ctr symlink to k3s [INFO] Creating killall script /usr/local/bin/k3s-killall.sh [INFO] Creating uninstall script /usr/local/bin/k3s-uninstall.sh [INFO] env: Creating environment file /etc/systemd/system/k3s.service.env [INFO] systemd: Creating service file /etc/systemd/system/k3s.service [INFO] systemd: Enabling k3s unit Created symlink /etc/systemd/system/multi-user.target.wants/k3s.service → /etc/systemd/system/k3s.service. [INFO] systemd: Starting k3s kube@kube1:~\$ kubectl version Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.1+k3s1", GitCommit:"75dba57f9b1de3ec0403b148c52c348e1dee2a5e", GitTreeState:"clean", BuildDate:"2021-05-21T16:12:29Z", GoVersion:"go1.16.4", Compiler:"gc", Platform:"linux/amd64"} Server Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.1+k3s1", GitCommit:"75dba57f9b1de3ec0403b148c52c348e1dee2a5e", GitTreeState:"clean", BuildDate:"2021-05-21T16:12:29Z", GoVersion:"go1.16.4", Compiler:"gc", Platform:"linux/amd64"} kube@kube1:~\$ k3s --version k3s version v1.21.1+k3s1 (75dba57f) go version go1.16.4 kube@kube1:~\$ kubectl get node NAME STATUS ROLES AGE VERSION kube1 Ready control-plane,master 20s v1.21.1+k3s1 kube@kube1:~\$ kubectl get pods --all-namespaces NAMESPACE NAME READY STATUS RESTARTS AGE kube-system metrics-server-86ccb8457f-nfcrc 1/1 Running 0 6m32s kube-system local-path-provisioner-5ff76fc89d-72kqq 1/1 Running 0 6m32s kube-system coredns-7448499f4d-jgph2 1/1 Running 0 6m32s kube-system helm-install-traefik-crd-nxphh 0/1 Completed 0 6m32s kube-system helm-install-traefik-xmvtf4 0/1 Completed 1 6m32s kube-system svclb-traefik-fs8xr 2/2 Running 0 4m51s kube-system traefik-97b44b794-v8lq8 1/1 Running 0 4m52s kube@kube1:~\$ </pre>
Monitoring	<p>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.</p> <p>For monitoring we have used prometheus and grafana:</p> <p>Prometheus collects metrics from configured targets at given intervals, evaluates rule expressions, displays the results, and can trigger alerts when specific conditions are observed.</p> <p>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.</p> <p>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.</p> <ul style="list-style-type: none"> ● Enable Prometheus add-on to monitor metrics on MicroK8s Cluster. ● Enabling built-in Prometheus add-on on primary Node.

```

token: R0GwAxEvINDC7XgZowExpt5nbsarLyB0QnNivodmOrZAxLzAv8ytpazOK
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:

NAME	READY	STATUS	RESTARTS	A
prometheus-operator-7649c7454f-d9dqr	0/2	ContainerCreating	0	3
node-exporter-vmwcg	0/2	ContainerCreating	0	2
kube-state-metrics-78dc55b74b-jrbfp	0/3	ContainerCreating	0	2
grafana-6b8df57c5b-z5qcj	0/1	ContainerCreating	0	2
prometheus-adapter-69b8496df6-pf56c	0/1	ContainerCreating	0	2
alertmanager-main-0	0/2	ContainerCreating	0	1
prometheus-k8s-0	0/2	ContainerCreating	0	1

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
prometheus-operator	ClusterIP	None	<none>	8443/TCP
alertmanager-main	ClusterIP	10.152.183.149	<none>	9093/TCP
grafana	ClusterIP	10.152.183.245	<none>	3000/TCP
kube-state-metrics	ClusterIP	None	<none>	8443/TCP,944
node-exporter	ClusterIP	None	<none>	9100/TCP
prometheus-adapter	ClusterIP	10.152.183.37	<none>	443/TCP
alertmanager-operated	ClusterIP	None	<none>	9093/TCP,909
prometheus-operated	ClusterIP	None	<none>	9090/TCP
prometheus-k8s	ClusterIP	10.152.183.135	<none>	9090/TCP

- Prometheus port-forwarding to enable external access:

```

hpe@hpe-VirtualBox:~$ microk8s kubectl port-forward -n monitoring services/prometheus-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/grafana --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:

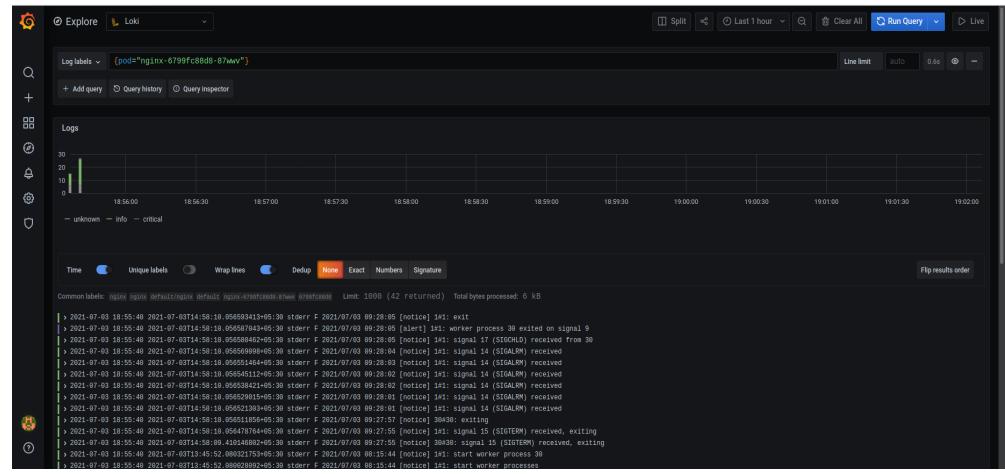


- Grafana dashboard: Visualization of cluster

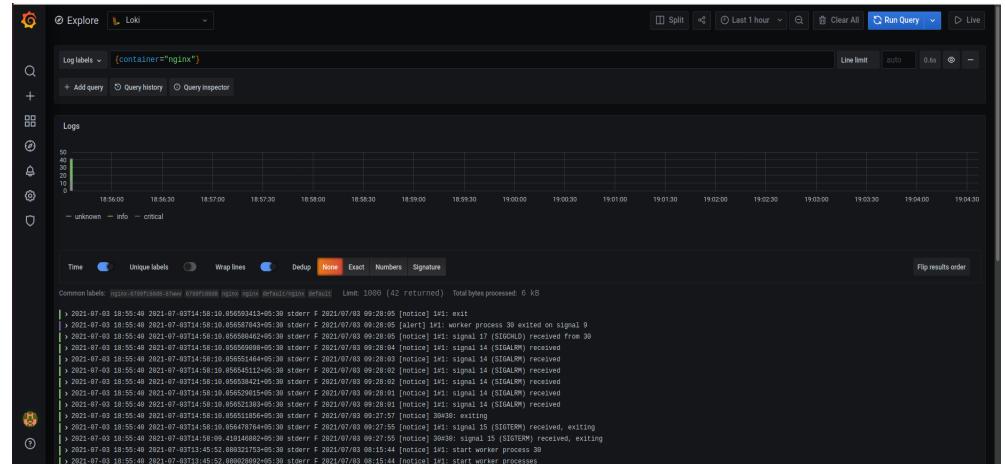
It shows the metrics for cluster, nodes, pods, jobs and deployments running on it.

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.

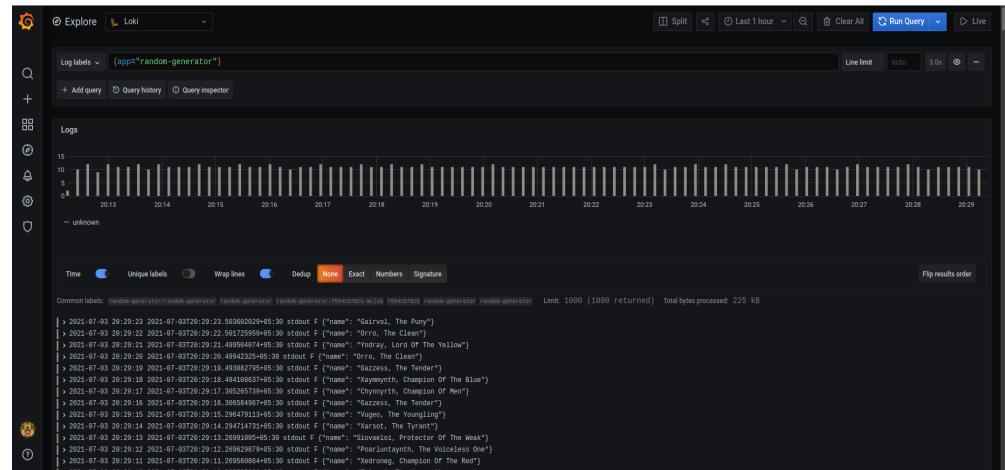
- Logs for nginx app:



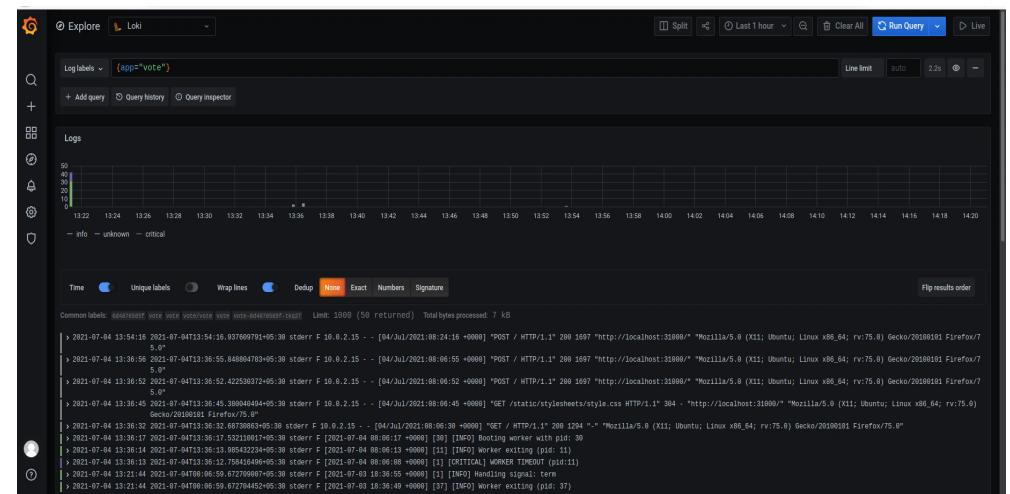
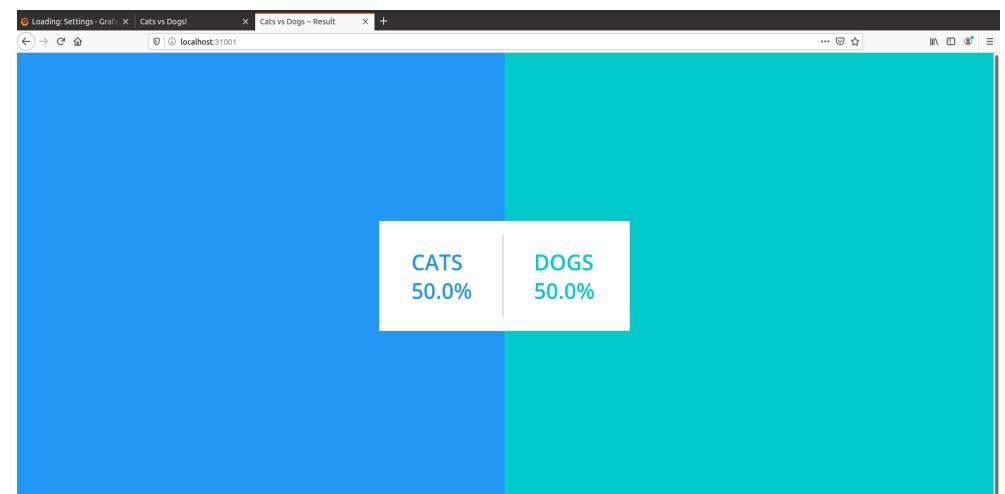
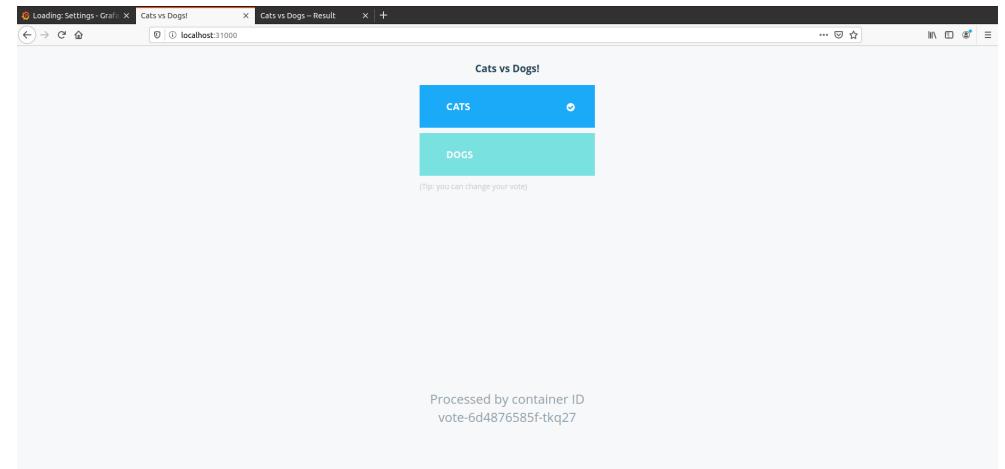
Logging



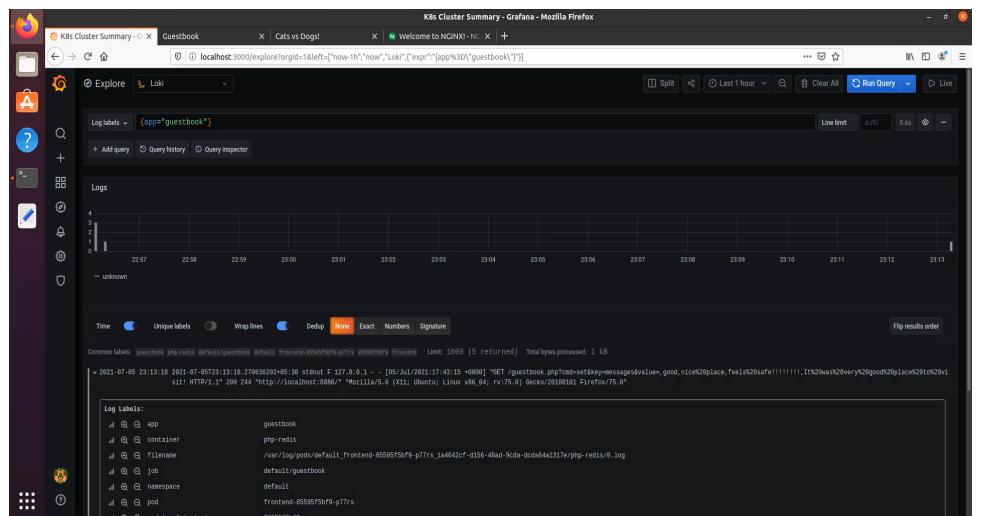
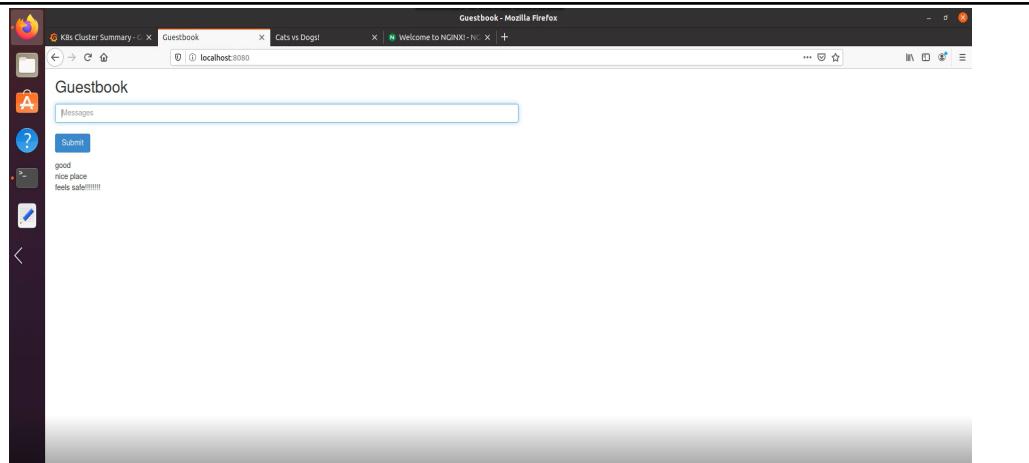
- We have deployed an app which randomly generates 30 names of dragons for every 30 seconds. The below figure shows the logs of random-generator app:



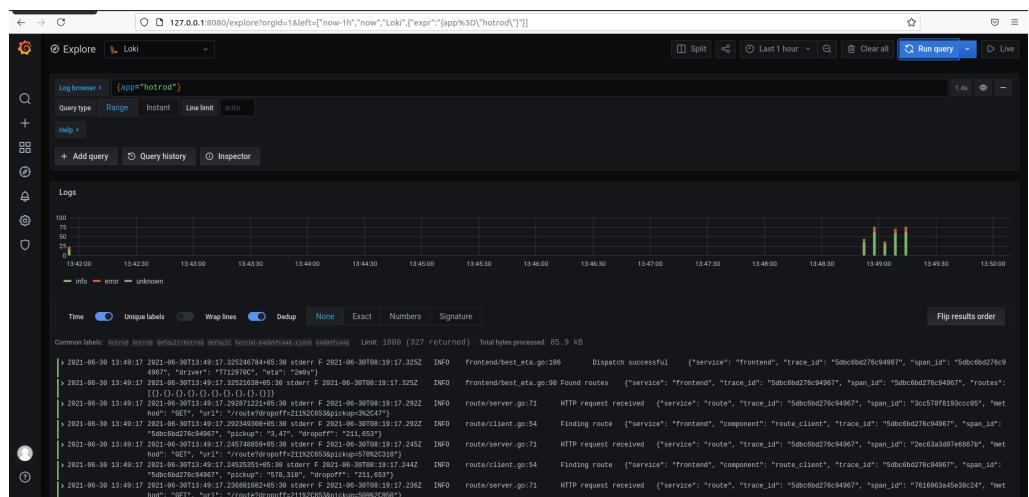
- We have deployed a voting app in which we can vote between cats and dogs; logs for the same are displayed below:



- 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:



	<pre> 2021-06-30 13:49:17 2021-06-30T13:49:17.325Z+05:30 steer F 2021-06-30T08:19:17.325Z INFO frontend/best_eta.go:106 Dispatch successful {"service": "frontend", "trace_id": "5dbc0bd276c94967", "span_id": "5dbc0bd276c94967", "routes": "49677", "driver": "7712979BC", "eta": "2m0s"} {"D, O, D, O, O, O, O, O, O, O, O, O} 2021-06-30 13:49:17 2021-06-30T13:49:17.295Z+05:30 steer F 2021-06-30T08:19:17.292Z INFO route/server.go:73 HTTP request received {"service": "route", "trace_id": "5dbc0bd276c94967", "span_id": "5dbc0bd276c94967", "routes": "49677", "driver": "7712979BC", "eta": "2m0s"} {"D, O, D, O, O, O, O, O, O, O, O, O} 2021-06-30 13:49:17 2021-06-30T13:49:17.292Z+05:30 steer F 2021-06-30T08:19:17.245Z INFO route/server.go:73 HTTP request received {"service": "route", "trace_id": "5dbc0bd276c94967", "span_id": "5dbc0bd276c94967", "routes": "49677", "driver": "7712979BC", "eta": "2m0s"} {"D, O, D, O, O, O, O, O, O, O, O, O} 2021-06-30 13:49:17 2021-06-30T13:49:17.245Z+05:30 steer F 2021-06-30T08:19:17.236Z INFO route/server.go:73 HTTP request received {"service": "route", "trace_id": "5dbc0bd276c94967", "span_id": "5dbc0bd276c94967", "routes": "49677", "driver": "7712979BC", "eta": "2m0s"} {"D, O, D, O, O, O, O, O, O, O, O, O} 2021-06-30 13:49:17 2021-06-30T13:49:17.236Z+05:30 steer F 2021-06-30T08:19:17.232Z INFO route/client.go:54 Finding route {"service": "frontend", "component": "route_client", "trace_id": "5dbc0bd276c94967", "span_id": "5dbc0bd276c94967", "routes": "49677", "driver": "7712979BC", "eta": "2m0s"} {"D, O, D, O, O, O, O, O, O, O, O, O} 2021-06-30 13:49:17 2021-06-30T13:49:17.232Z+05:30 steer F 2021-06-30T08:19:17.226Z INFO route/server.go:73 HTTP request received {"service": "route", "trace_id": "5dbc0bd276c94967", "span_id": "5dbc0bd276c94967", "routes": "49677", "driver": "7712979BC", "eta": "2m0s"} {"D, O, D, O, O, O, O, O, O, O, O, O} 2021-06-30 13:49:17 2021-06-30T13:49:17.226Z+05:30 steer F 2021-06-30T08:19:17.189Z INFO route/client.go:54 Finding route {"service": "frontend", "component": "route_client", "trace_id": "5dbc0bd276c94967", "span_id": "5dbc0bd276c94967", "routes": "49677", "driver": "7712979BC", "eta": "2m0s"} {"D, O, D, O, O, O, O, O, O, O, O, O} 2021-06-30 13:49:17 2021-06-30T13:49:17.189Z+05:30 steer F 2021-06-30T08:19:17.188Z INFO route/client.go:54 Finding route {"service": "frontend", "component": "route_client", "trace_id": "5dbc0bd276c94967", "span_id": "5dbc0bd276c94967", "routes": "49677", "driver": "7712979BC", "eta": "2m0s"} {"D, O, D, O, O, O, O, O, O, O, O, O} 2021-06-30 13:49:17 2021-06-30T13:49:17.188Z+05:30 steer F 2021-06-30T08:19:17.182Z INFO route/client.go:54 Finding route {"service": "frontend", "component": "route_client", "trace_id": "5dbc0bd276c94967", "span_id": "5dbc0bd276c94967", "routes": "49677", "driver": "7712979BC", "eta": "2m0s"} {"D, O, D, O, O, O, O, O, O, O, O, O} 2021-06-30 13:49:17 2021-06-30T13:49:17.178Z+05:30 steer F 2021-06-30T08:19:17.173Z INFO route/server.go:73 HTTP request received {"service": "route", "trace_id": "5dbc0bd276c94967", "span_id": "5dbc0bd276c94967", "routes": "49677", "driver": "7712979BC", "eta": "2m0s"} {"D, O, D, O, O, O, O, O, O, O, O, O} 2021-06-30 13:49:17 2021-06-30T13:49:17.173Z+05:30 steer F 2021-06-30T08:19:17.172Z INFO route/server.go:73 HTTP request received {"service": "route", "trace_id": "5dbc0bd276c94967", "span_id": "5dbc0bd276c94967", "routes": "49677", "driver": "7712979BC", "eta": "2m0s"} {"D, O, D, O, O, O, O, O, O, O, O, O} </pre>
--	--

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:

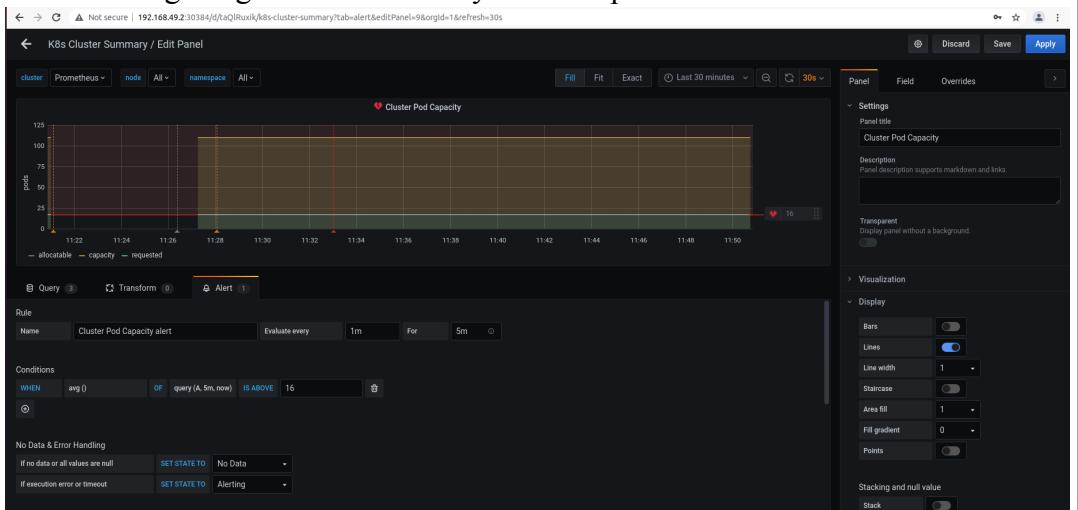
- Create a slack webhook URL : It will help us create an endpoint for sending messages to a specific channel of our Slack server.

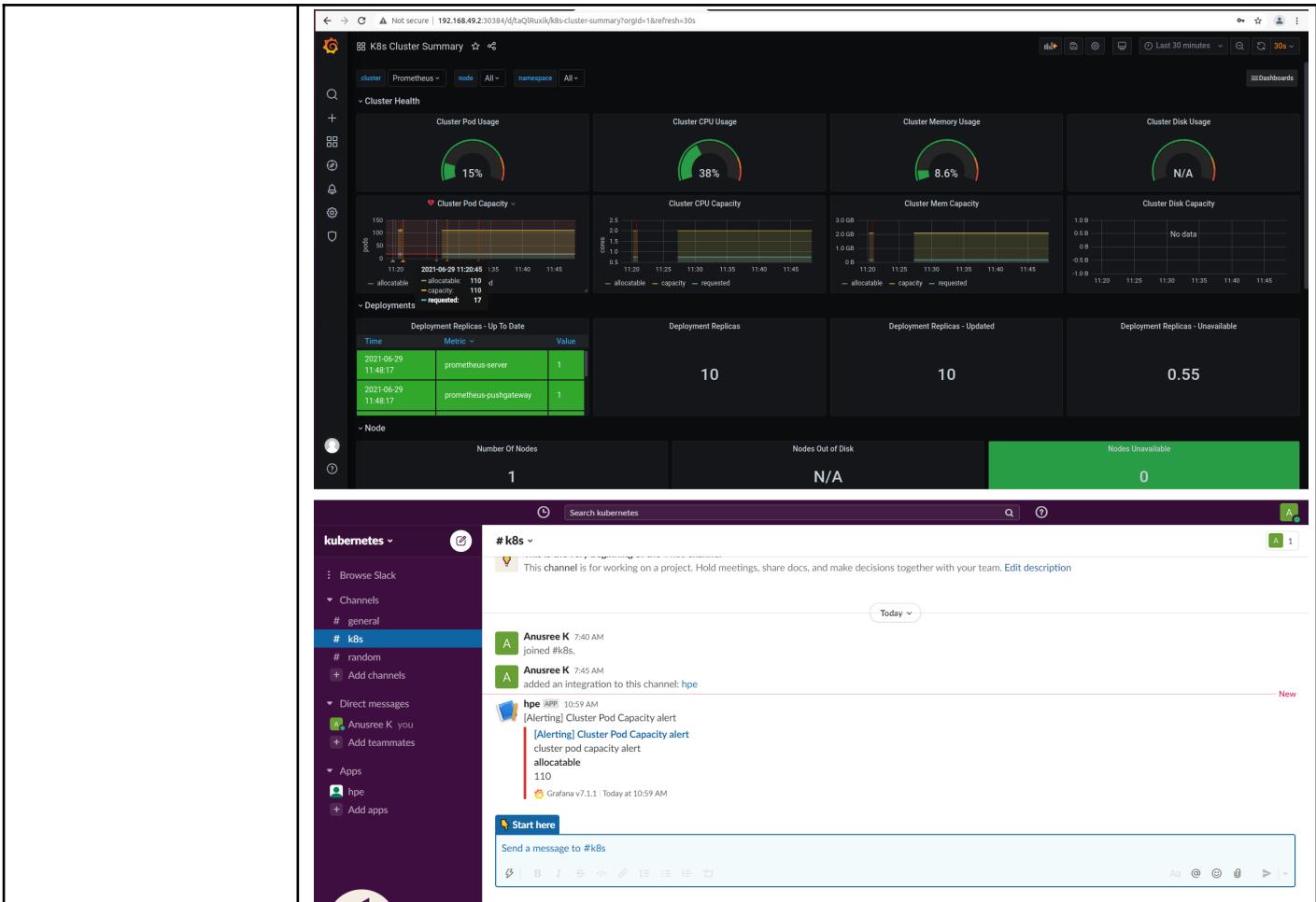
Webhook URL	Channel	Added By
https://hooks.slack.com/services/T026L... <button type="button">Copy</button>	#k8s	Anusree K Jun 28, 2021

Alerting

Add New Webhook to Workspace

- 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
```

The screenshot shows the Prometheus UI with multiple tabs open. The 'Alerts' tab is active, displaying a single alert named 'InstanceDown' with status 'Firing (0)'. The alert configuration is as follows:

```
alert: InstanceDown
expr: up == 0
for: 1m
labels:
  severity: page
  anomalies:
  description: `{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 1 minute.`
  summary: Instance {{ $labels.instance }} down
```

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

Alerting: Notification channel | **Prometheus Time Series** | **Alertmanager** | Not secure | 192.168.49.2:32750/alerts

Prometheus Alerts Graph Status Help

Alerts

Inactive (0) Pending (1) Firing (0)

Show annotations

/etc/config/alerts > Instances

InstanceDown (1 active)

```
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
```

Labels	State	Active Since	Value
alarmname="InstanceDown" instance="prometheus-pushgateway-monitoring.svc:9091" job="prometheus-pushgateway" severity="page"	PENDING	2021-07-04 09:32:53.862877577 +0000 UTC	0

Alerting: Notification channel | **Prometheus Time Series** | **Alertmanager** | Not secure | 192.168.49.2:32750/alerts

Prometheus Alerts Graph Status Help

Alerts

Inactive (0) Pending (0) Firing (1)

Show annotations

/etc/config/alerts > Instances

InstanceDown (1 active)

```
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
```

Labels	State	Active Since	Value
alarmname="InstanceDown" instance="prometheus-pushgateway-monitoring.svc:9091" job="prometheus-pushgateway" severity="page"	FIRING	2021-07-04 09:32:53.862877577 +0000 UTC	0

Alerting: Notification channel | **Prometheus Time Series** | **Alertmanager** | localhost:9093/#/alerts

Alertmanager Alerts Silences Status Help New Silence

Filter Group Receiver: All Silenced Inhibited

Custom matcher, e.g. env=production

Expand all groups

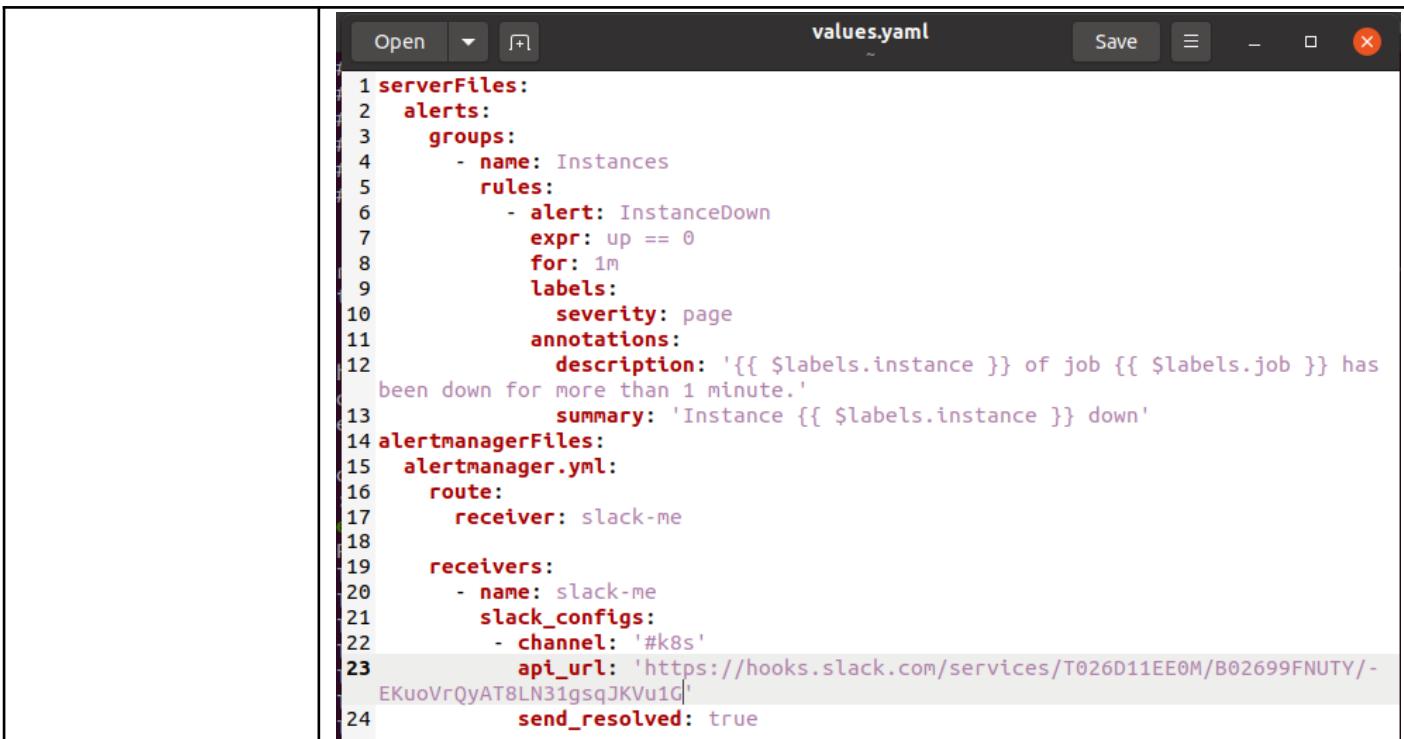
Not grouped 1 alert

09:33:53, 2021-07-04 (UTC) + Info ↗ Source ✘ Silence

alarmname="InstanceDown" instance="prometheus-pushgateway-monitoring.svc:9091" job="prometheus-pushgateway" severity="page"

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.

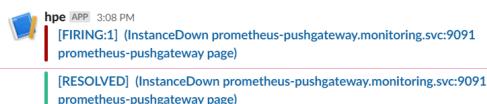


```

1 serverFiles:
2   alerts:
3     groups:
4       - name: Instances
5         rules:
6           - alert: InstanceDown
7             expr: up == 0
8             for: 1m
9             labels:
10               severity: page
11             annotations:
12               description: '{{ $labels.instance }} of job {{ $labels.job }} has
been down for more than 1 minute.'
13             summary: 'Instance {{ $labels.instance }} down'
14 alertmanagerFiles:
15   alertmanager.yml:
16     route:
17       receiver: slack-me
18
19     receivers:
20       - name: slack-me
21       slack_configs:
22         - channel: '#k8s'
23         api_url: 'https://hooks.slack.com/services/T026D11EE0M/B02699FNUTY/-EkuoVrQyAT8LN31gsqJKVu1G'
24         send_resolved: true

```

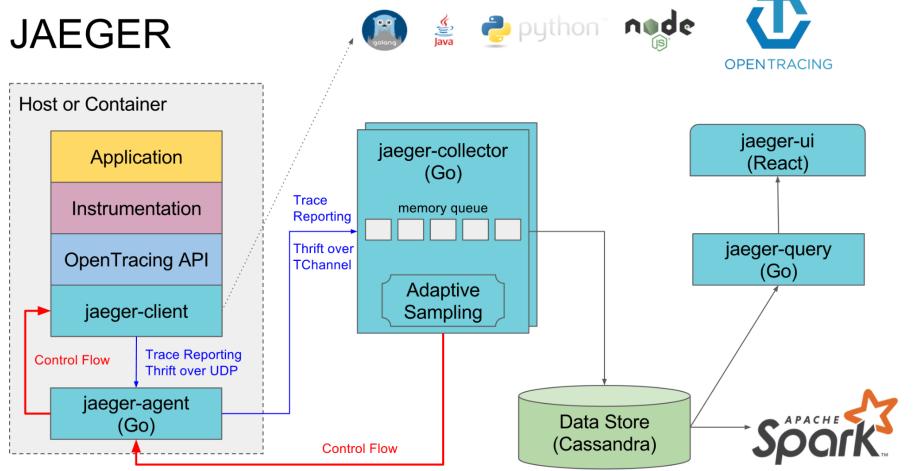
Now we upgrade the helm chart to get the alerts on slack



Tracing

- 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:

JAEGER



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 the docker-compose manifests to kubernetes using kompose.

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	10.42.0.46	kube1
alertmanager-prometheus-kube-prometheus-alertmanager-0	2/2	Running	6	5d18h	10.42.0.45	kube1
prometheus-prometheus-kube-prometheus-prometheus-0	2/2	Running	7	5d18h	10.42.0.52	kube1
prometheus-kube-prometheus-operator-98d5bcd6-2rwjl	1/1	Running	3	5d19h	10.42.0.56	kube1
grafana-6f67f447ff-59qnb	1/1	Running	3	5d18h	10.42.0.58	kube1
lokl-prmtail-qnvqz	1/1	Running	0	17m	10.42.0.59	kube1
lokl-0	1/1	Running	0	17m	10.42.0.60	kube1
hotrod-64d95fc446-5vjvv	1/1	Running	0	6m49s	10.42.0.60	kube1
jaeger-7dd885bd4b-rvxxv9	1/1	Running	0	4m21s	10.42.0.61	kube1

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

A screenshot of the 'Hot R.O.D.' (Rides On Demand) application interface. At the top, there's a header bar with navigation icons and the URL '127.0.0.1:8000'. Below the header, a message says 'Your web client's id: 5917'. The main content area features a title 'Hot R.O.D.' and a subtitle 'Rides On Demand'. There are four buttons labeled 'Rachel's Floral Designs', 'Trom Chocolatier', 'Japanese Desserts', and 'Amazing Coffee Roasters'. A note below the buttons says 'Click on customer name above to order a car.' Below this, a list of requests is displayed:

```

HotROD T712970C arriving in 2min [req: 5917-8, latency: 680ms] [find trace]
HotROD T770282C arriving in 2min [req: 5917-9, latency: 671ms] [find trace]
HotROD T750168C arriving in 2min [req: 5917-6, latency: 790ms] [find trace]
HotROD T754674C arriving in 2min [req: 5917-5, latency: 729ms] [find trace]
HotROD T777974C arriving in 2min [req: 5917-4, latency: 778ms] [find trace]
HotROD T732168C arriving in 2min [req: 5917-3, latency: 772ms] [find trace]
HotROD T797855C arriving in 2min [req: 5917-2, latency: 732ms] [find trace]
HotROD T7732325C arriving in 2min [req: 5917-1, latency: 1146ms] [find trace]

```

- 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.

```

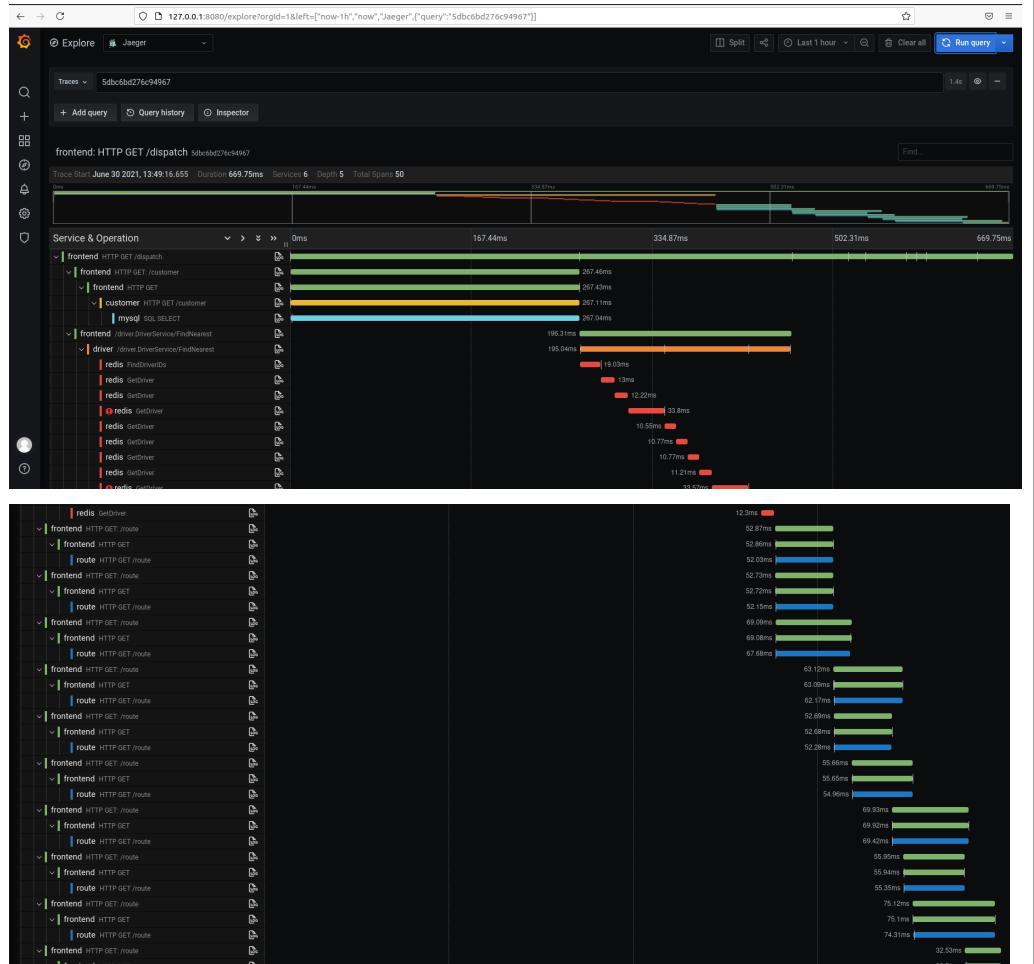
2021-06-30 14:01:00 2021-06-30T14:01:00+00:00 26125859d49530 51err F 2021-06-30T08:31:00.264Z INFO  Frontend/best_eta.go:186 Dispatch successful {"service": "Frontend", "trace_id": "5e1ab51a15e061a1", "span_id": "5e1ab51a15e061a15", "driver": "TT40937C", "eta": "200ms"}

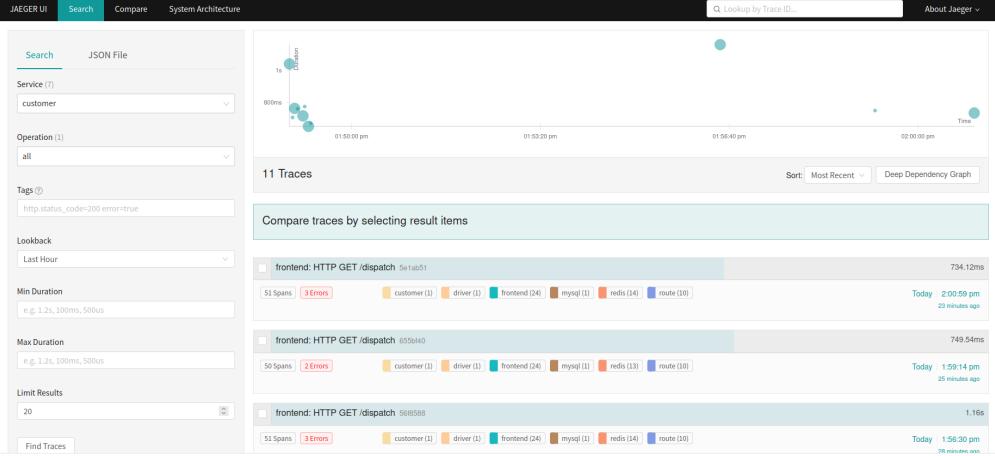
Log labels
  ↳ @ app          hotrod
    ↳ @ container   hotrod
    ↳ @ filename    /var/log/pods/default_hotrod-64d95fc446-xjds9_b7fba529-51c5-4be8-b8e0-164d65b69693/hotrod/0.log
    ↳ @ job          default/hotrod
    ↳ @ namespace   default
    ↳ @ pod          hotrod-64d95fc446-xjds9
    ↳ @ pod_template_hash 64d95fc446

Detected fields ⚡
  ↳ ⚡ TraceID      5e1ab51a15e061a1 [🔗 Jaeger]
  ↳ ⚡ ts           2021-06-30T08:31:00.498Z
  ↳ ⚡ tsNs         3625041868496915793

```

- 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.



	<p>This shows the Jaeger UI which enables us to filter traces based on trace ID, services and operations and view them.</p> 
Multi-Cluster	<p>What is Multi-cluster?</p> <p>Multi-cluster is a strategy for deploying an application on or across multiple Kubernetes clusters with the goal of improving availability, isolation, and</p>

	<p>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.</p>
	<p>Why Multi-cluster?</p> <ol style="list-style-type: none"> 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. <p>Multi-Cluster Application Architecture</p> <ol style="list-style-type: none"> 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

	<p>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.</p> <p>Installation of Multicluster:</p> <p>Step 1: Install the multicluster control plane</p> <p>On each cluster, run:</p> <pre>linkerd multicluster install \ kubectl apply -f -</pre> <p>To verify that everything has started up successfully, run:</p> <pre>linkerd multicluster check</pre> <p>Step 2: Link the clusters</p> <p>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:</p> <pre>linkerd --context=west multicluster link --cluster-name east \ kubectl --context=west apply -f -</pre> <p>To verify that the credentials were created successfully and the clusters are able to reach each other, run:</p> <pre>linkerd --context=west multicluster check</pre> <p>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:</p> <pre>linkerd --context=west multicluster gateways</pre> <p>Step 3: Export services</p> <p>Services are not automatically mirrored in linked clusters. By default, only services with the <code>mirror.linkerd.io/exported</code> label will be mirrored. For each service you would like mirrored to linked clusters, run:</p> <pre>kubectl label svc foobar mirror.linkerd.io/exported=true</pre>
--	---

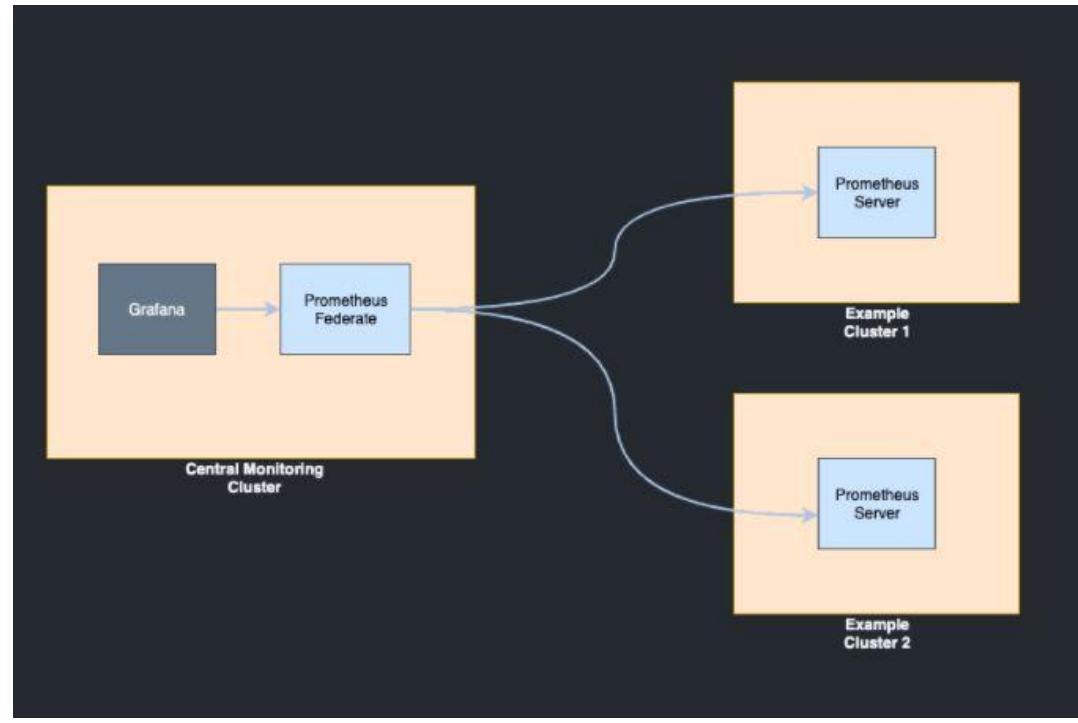
Monitoring Multi-cluster

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	<ul style="list-style-type: none"> ● 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	<ul style="list-style-type: none"> ● 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. ● Monitoring and observability using Open Telemetry. ● Setting up a High availability cluster using K3d. ● Deploying kubernetes cluster on cloud platforms like Azure, AWS, etc.
References	<p> 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/ </p>