Step wise approach for setting up alerts and notification using Grafana Loki to slack channel

1. Firstly before starting we need to check whether we are connected to the AWS account or not.
   Command- aws configure

```
PS C:\Users\Jaya shree> aws configure
AWS Access Key ID [****************UCZK]:
AWS Secret Access Key [****************jY0N]:
Default region name [us-east-1]:
Default output format [none]:
```

2. We will make a kubernetes cluster with a instance type that will satisfy the need.
   Command- eksctl create cluster --name=loki --nodes-min=2 --nodes-max=3 --node-type=t2.medium --region=us-east-1 --version=1.21

```
PS C:\Users\Jaya shree> eksctl create cluster --name=loki --nodes-min=2 --nodes-max=3  --node-type=t2.medium --region=us-east-1 --version=1.21
2023-01-31 10:56:00 [ℹ]  eksctl version 0.123.0
2023-01-31 10:56:00 [ℹ]  using region us-east-1
2023-01-31 10:56:02 [ℹ]  setting availability zones to [us-east-1c us-east-1d]
2023-01-31 10:56:02 [ℹ]  subnets for us-east-1c - public:192.168.0.0/19 private:192.168.64.0/19
2023-01-31 10:56:02 [ℹ]  subnets for us-east-1d - public:192.168.32.0/19 private:192.168.96.0/19
```

3. For deployment of prometheus we need the metric server. We would deploy that using the following command.
   Command-kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml

```
PS C:\Users\Jaya shree> kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
serviceaccount/metrics-server created
clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
clusterrole.rbac.authorization.k8s.io/system:metrics-server created
rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created
clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
service/metrics-server created
deployment.apps/metrics-server created
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
```

4. Check whether metric server is up.
   Command-kubectl get deployment metrics-server -n kube-system

```
PS C:\Users\Jaya shree> kubectl get deployment metrics-server -n kube-system
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
metrics-server   1/1     1            1           40s
```

5. Install helm repo of prometheus.
   Command- helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

```
PS C:\Users\Jaya shree> helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
"prometheus-community" already exists with the same configuration, skipping
```

6. Now, we will deploy the prometheus in the default namespace
   Command- helm upgrade -i prometheus prometheus-community/prometheus --set alertmanager.persistentVolume.storageClass="gp2",server.persistentVolume.storageClass="gp2"

```
PS C:\Users\Jaya shree> helm upgrade -i prometheus prometheus-community/prometheus --set alertmanager.persistentVolume.storageClass="gp2",server.persistentVolume.storageClass="gp2"
Release "prometheus" does not exist. Installing it now.
```

7. Install Grafana Loki stack with promtail using helm
   Command- helm repo add loki https://grafana.github.io/loki/charts

```
PS C:\Users\Jaya shree> helm repo add loki https://grafana.github.io/loki/charts
"loki" already exists with the same configuration, skipping
```

8. Now we will deploy the grafana loki stack.
   Command- helm upgrade --install loki loki/loki-stack

```
PS C:\Users\Jaya shree> helm upgrade --install loki loki/loki-stack
```

9. We check the list of repo in the cluster
   Command- helm repo list

```
PS C:\Users\Jaya shree> helm repo list
NAME                    URL
prometheus-community    https://prometheus-community.github.io/helm-charts
loki                    https://grafana.github.io/loki/charts
```

10. Install Grafana using the yaml file that we have configured
    Command- kubectl apply -f grafana.yaml

```
PS C:\Users\Jaya shree> kubectl apply -f grafana.yaml
persistentvolume/pv-volume created
persistentvolumeclaim/grafana-pvc created
deployment.apps/grafana created
service/grafana created
```

Code(grafana.yaml)-

```
kind: PersistentVolume
apiVersion: v1
metadata:
```

```
  name: pv-volume
spec:
  storageClassName: gp2
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/pv2"
  claimRef:
     namespace: grafana
     name: grafana-pvc
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: grafana-pvc
spec:
  storageClassName: gp2
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: grafana
  name: grafana
spec:
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      labels:
        app: grafana
    spec:
      securityContext:
        fsGroup: 472
        supplementalGroups:
          - 0
      containers:
        - name: grafana
          image: grafana/grafana:7.5.2
          imagePullPolicy: IfNotPresent
```

```
        ports:
          - containerPort: 3000
            name: http-grafana
            protocol: TCP
        readinessProbe:
          failureThreshold: 3
          httpGet:
            path: /robots.txt
            port: 3000
            scheme: HTTP
          initialDelaySeconds: 10
          periodSeconds: 30
          successThreshold: 1
          timeoutSeconds: 2
        livenessProbe:
          failureThreshold: 3
          initialDelaySeconds: 30
          periodSeconds: 10
          successThreshold: 1
          tcpSocket:
            port: 3000
          timeoutSeconds: 1
        resources:
          requests:
            cpu: 250m
            memory: 750Mi
        volumeMounts:
          - mountPath: /var/lib/grafana
            name: grafana-pv
      volumes:
        - name: grafana-pv
          persistentVolumeClaim:
            claimName: grafana-pvc
---
apiVersion: v1
kind: Service
metadata:
  name: grafana
spec:
  ports:
    - port: 3000
      protocol: TCP
      targetPort: http-grafana
  selector:
    app: grafana
  sessionAffinity: None
  type: LoadBalancer
```

11. Deployment of wordpress application.

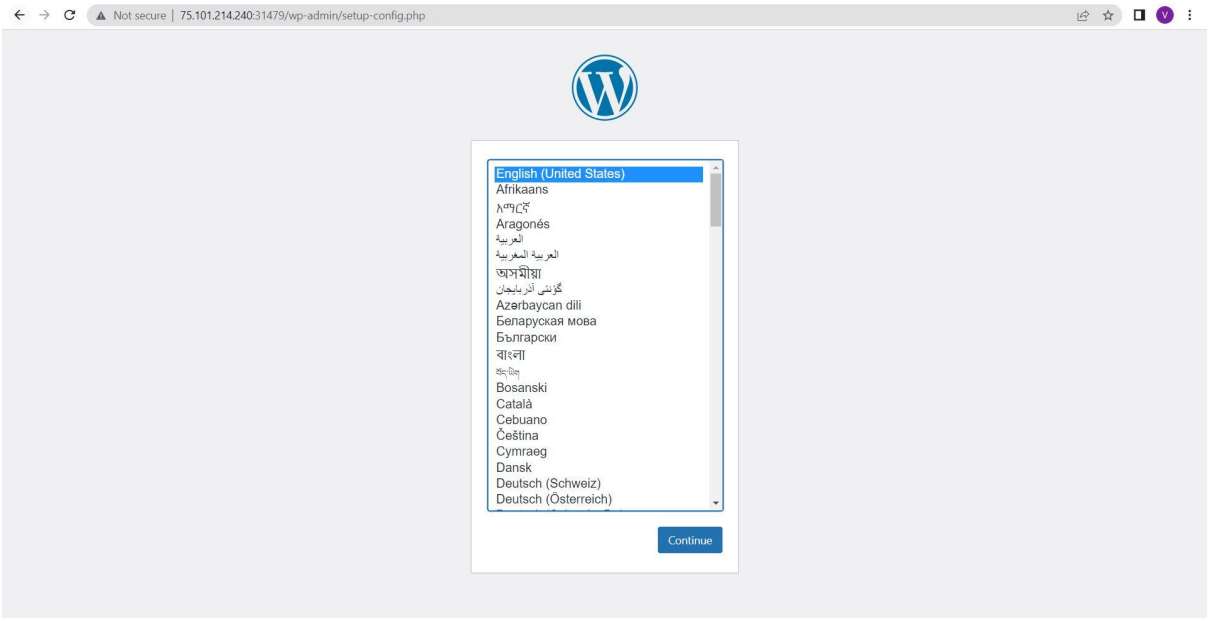
Command- kubectl apply -f deployment.yml

```
PS C:\Users\Jaya shree> kubectl apply -f deployment.yml
deployment.apps/server-demo created
service/backend-service created
```

deployment.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: server-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: back-end
          image: 655682474236.dkr.ecr.us-east-1.amazonaws.com/wordpress:latest
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  type: NodePort
  selector:
    app: web
  ports:
    - nodePort: 31479
      port: 8080
      targetPort: 80
```

12. We will open the nodeport to connect with the external port. Follow the steps as what we did earlier.



13. We will check for whether the pods of prometheus,grafana loki, grafana and deployment are up or not.

Command-kubectl get pods

```
PS C:\Users\Jaya shree> kubectl get pods
NAME                                             READY   STATUS    RESTARTS   AGE
grafana-756fb84d84-r9cgn                         0/1     Running   0          35s
loki-0                                           0/1     Running   0          65s
loki-promtail-b5txl                              1/1     Running   0          66s
loki-promtail-gxlcq                              1/1     Running   0          66s
prometheus-alertmanager-0                        1/1     Running   0          102s
prometheus-kube-state-metrics-7cdcf7cc98-76qpf   1/1     Running   0          103s
prometheus-prometheus-node-exporter-mczs5        1/1     Running   0          103s
prometheus-prometheus-node-exporter-qstgf        1/1     Running   0          103s
prometheus-prometheus-pushgateway-9d598d466-bldrs   1/1     Running   0          103s
prometheus-server-6479f8ff6-krhsx                2/2     Running   0          103s
server-demo-6bbc759f64-tb8wv                     1/1     Running   0          19s
```

14. We will check for the services that are running.

Command- kubectl get svc

```
PS C:\Users\Jaya shree> kubectl get svc
NAME                            TYPE           CLUSTER-IP       EXTERNAL-IP                                                              PORT(S)          AGE
backend-service                 NodePort       10.100.66.86     <none>                                                                   8080:31479/TCP   2m22s
grafana                         LoadBalancer   10.100.21.247    a7baabaa2aec1493e9a8179c592c40fd-34878862.us-east-1.elb.amazonaws.com     3000:32704/TCP   2m38s
kubernetes                      ClusterIP      10.100.0.1       <none>                                                                   443/TCP          22m
loki                            ClusterIP      10.100.237.38    <none>                                                                   3100/TCP         3m9s
loki-headless                   ClusterIP      None             <none>                                                                   3100/TCP         3m9s
prometheus-alertmanager         ClusterIP      10.100.21.240    <none>                                                                   9093/TCP         3m47s
prometheus-alertmanager-headless ClusterIP     None             <none>                                                                   9093/TCP         3m47s
prometheus-kube-state-metrics   ClusterIP      10.100.47.184    <none>                                                                   8080/TCP         3m46s
prometheus-prometheus-node-exporter ClusterIP  10.100.115.194   <none>                                                                   9100/TCP         3m46s
prometheus-pushgateway          ClusterIP      10.100.56.254    <none>                                                                   9091/TCP         3m46s
prometheus-server               ClusterIP      10.100.13.135    <none>                                                                   80/TCP           3m46s
```

15. We will get details about nodes

Command- kubectl get nodes -o wide

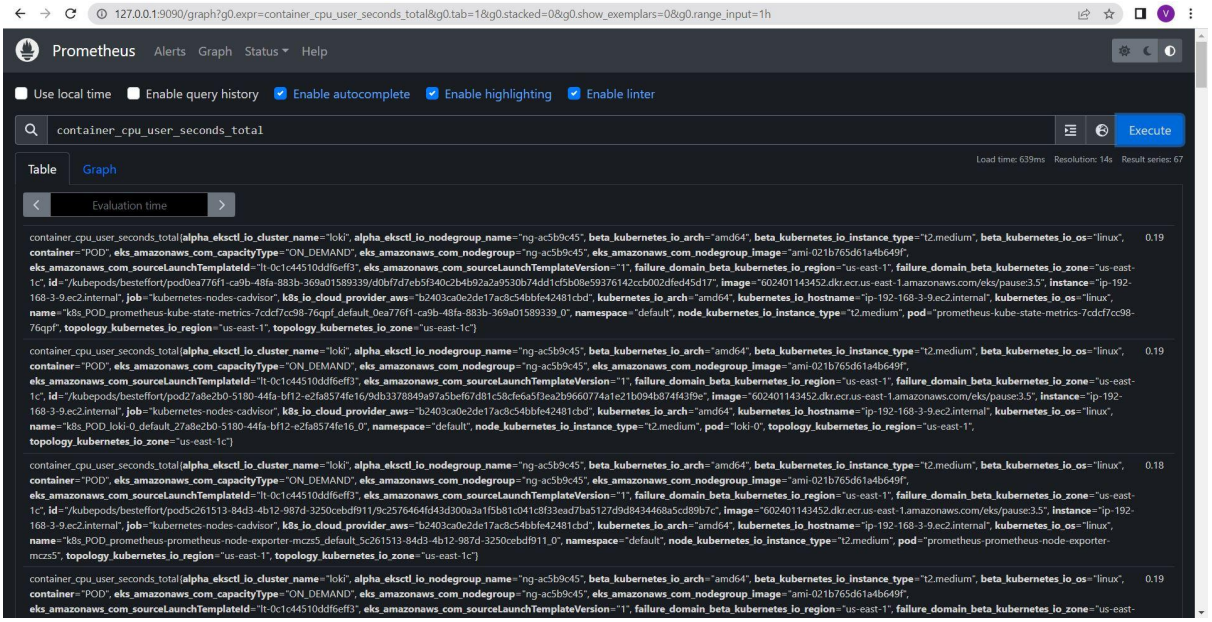
```
PS C:\Users\Jaya shree> kubectl get nodes -o wide
NAME                            STATUS   ROLES    AGE   VERSION               INTERNAL-IP      EXTERNAL-IP     OS-IMAGE         KERNEL-VERSION                  CONTAINER-RUNTIME
ip-192-168-3-9.ec2.internal     Ready    <none>   20m   v1.21.14-eks-fb459a0  192.168.3.9      54.234.142.140  Amazon Linux 2   5.4.226-129.415.amzn2.x86_64    docker://20.10.17
ip-192-168-62-189.ec2.internal  Ready    <none>   21m   v1.21.14-eks-fb459a0  192.168.62.189   75.101.214.240  Amazon Linux 2   5.4.226-129.415.amzn2.x86_64    docker://20.10.17
```

16. Now, we will portforward the prometheus just to connect with external ip

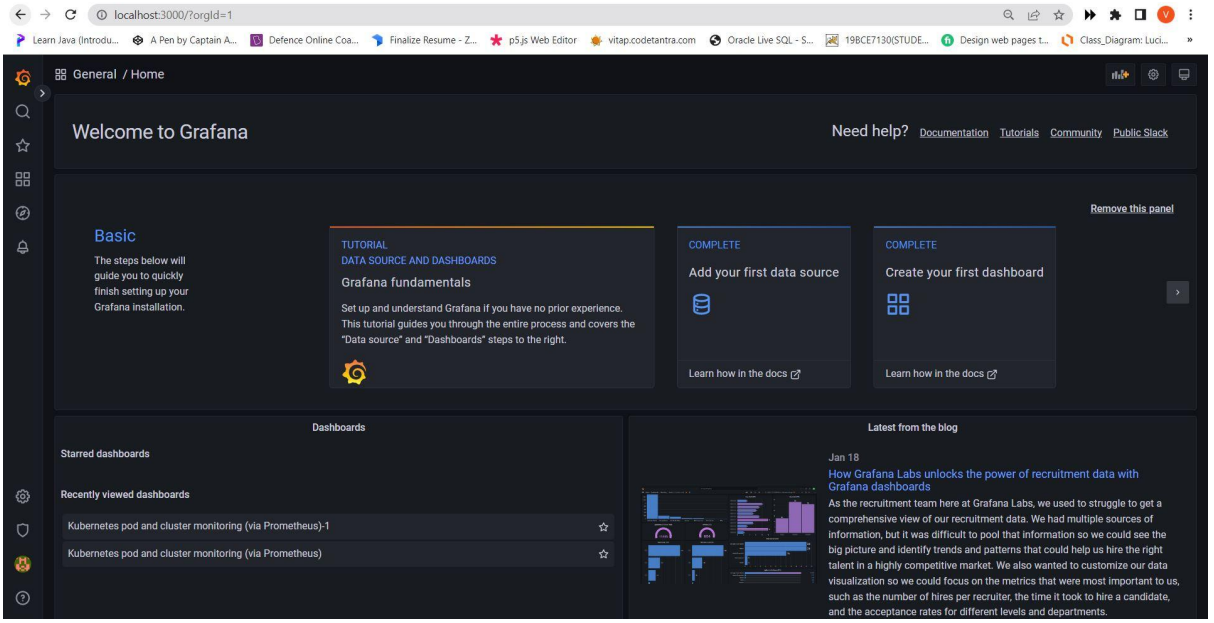
Command- kubectl port-forward deploy/prometheus-server 9090:9090

```
PS C:\Users\Jaya shree> kubectl port-forward deploy/prometheus-server 9090:9090
Forwarding from 127.0.0.1:9090 -> 9090
Forwarding from [::1]:9090 -> 9090
```

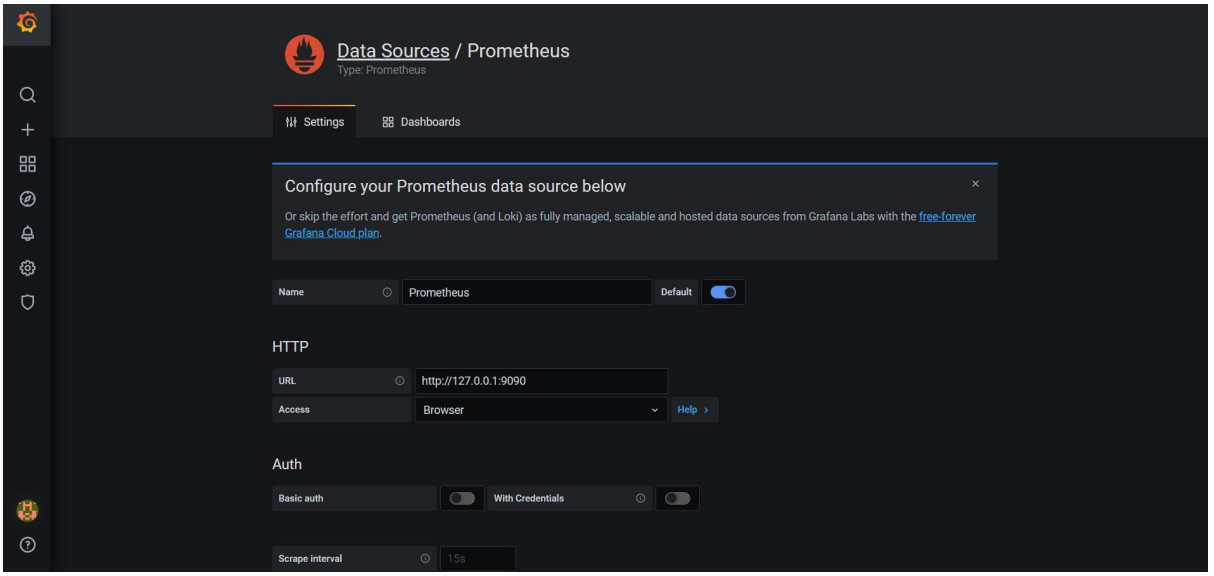
17. We will check whether the prometheus server is deployed or not.

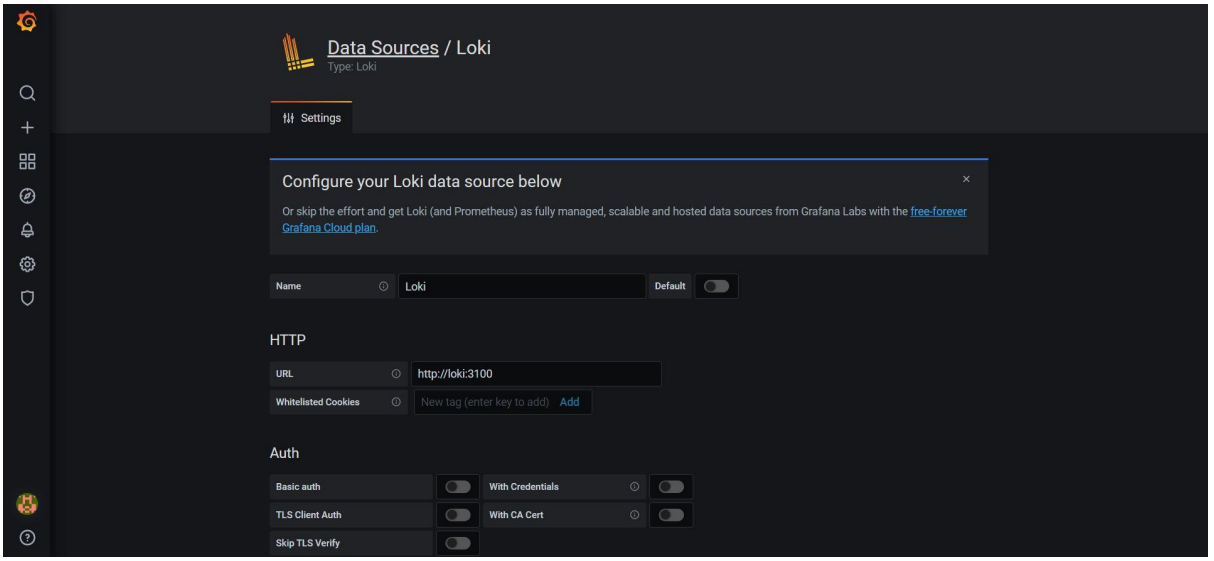


18. Now, we will portforward the grafana and connect through browser
<u>Command</u>-kubectl port-forward deploy/grafana 3000:3000

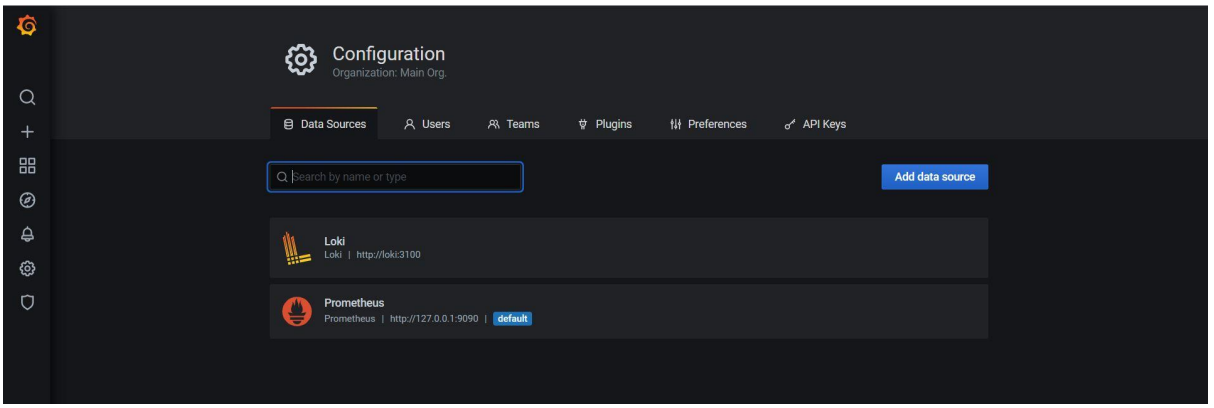


19. We will Configure the datasource as we did earlier but now we will also add one more data source that grafana loki.

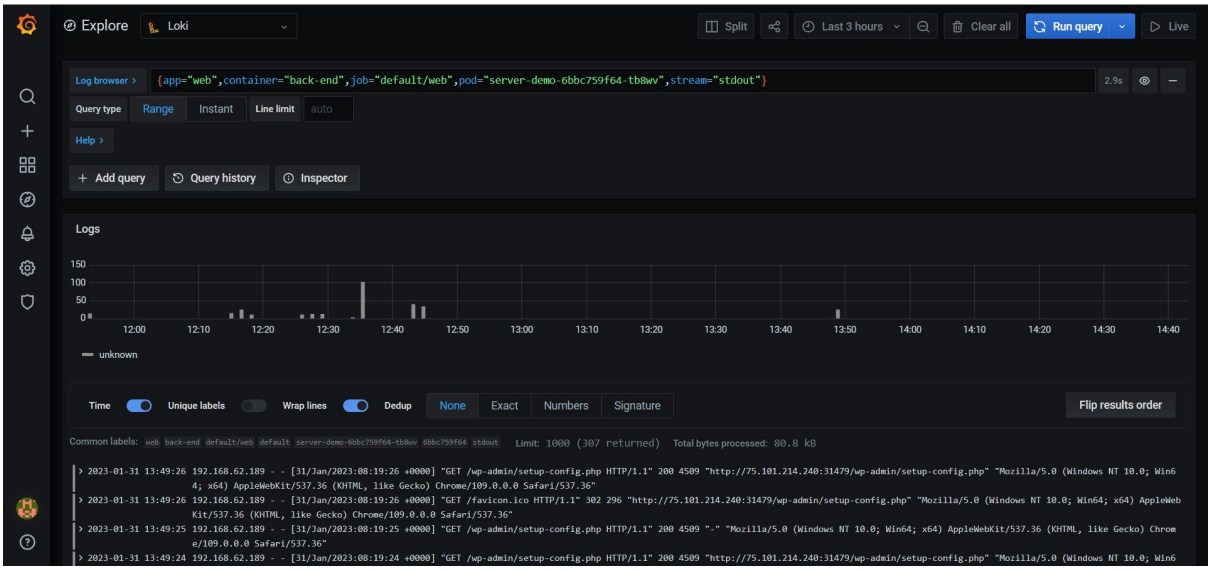




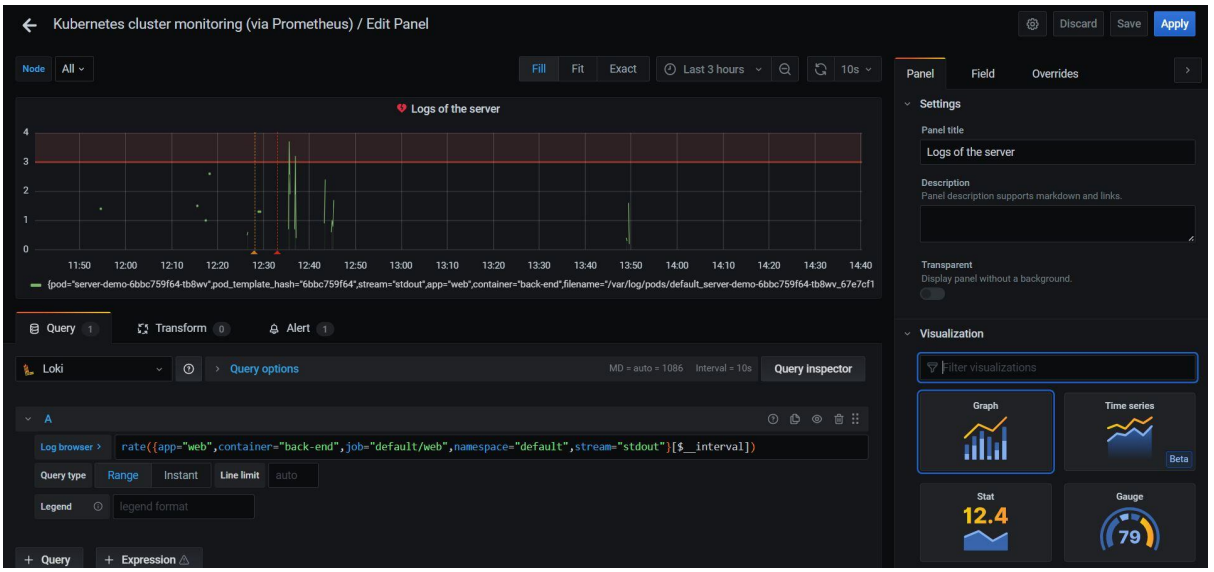
20. Later in the data sources, we should be able to see two data sources.



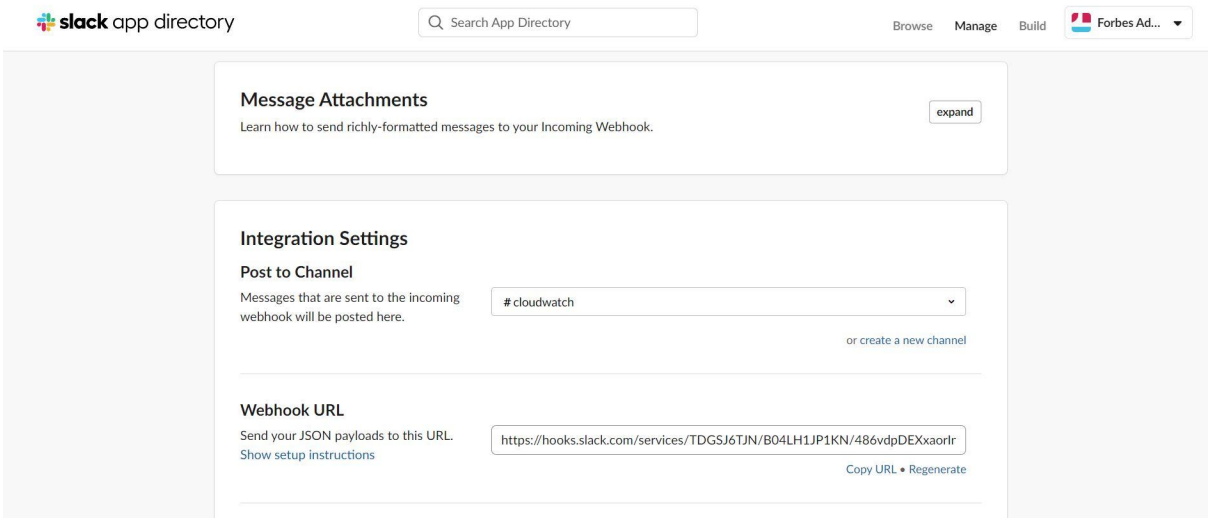
21. We will check in ‘Explore’ tab and selecting grafana loki whether we are getting the logs when apply the following query which represents the response at a given time.



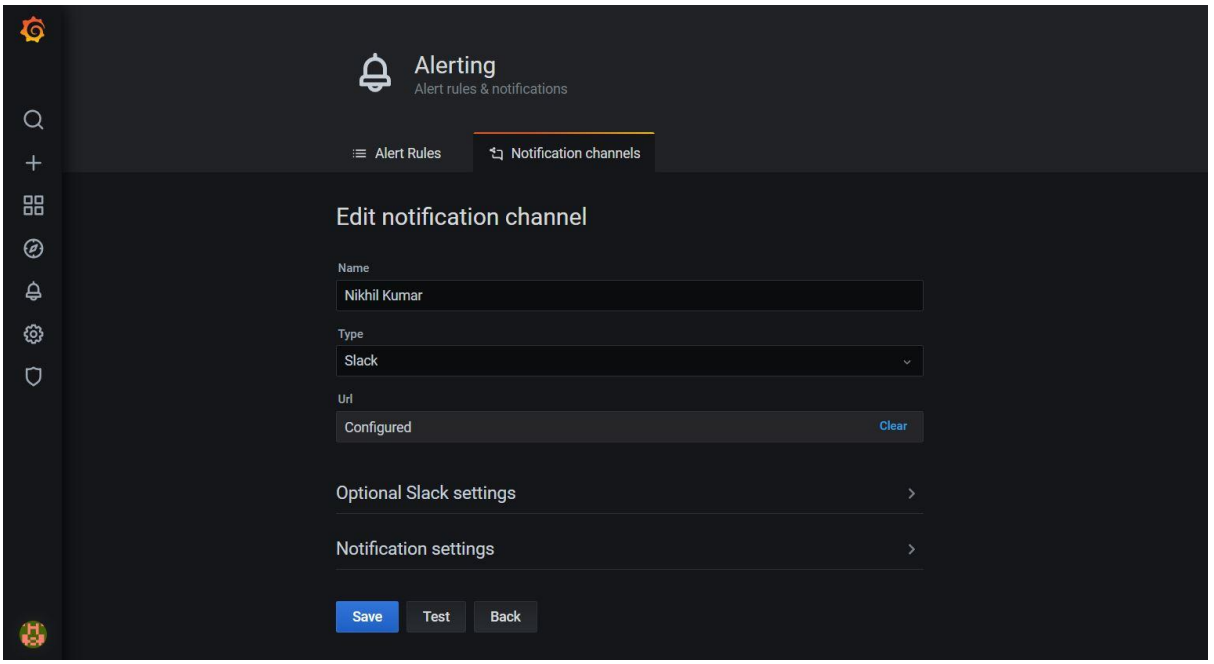
22. After that in we will setup the dashboard as we did earlier. Here, we will add the Logs panel that would take data from grafana loki, by adding the same query that we used in the ‘explore’ tab.



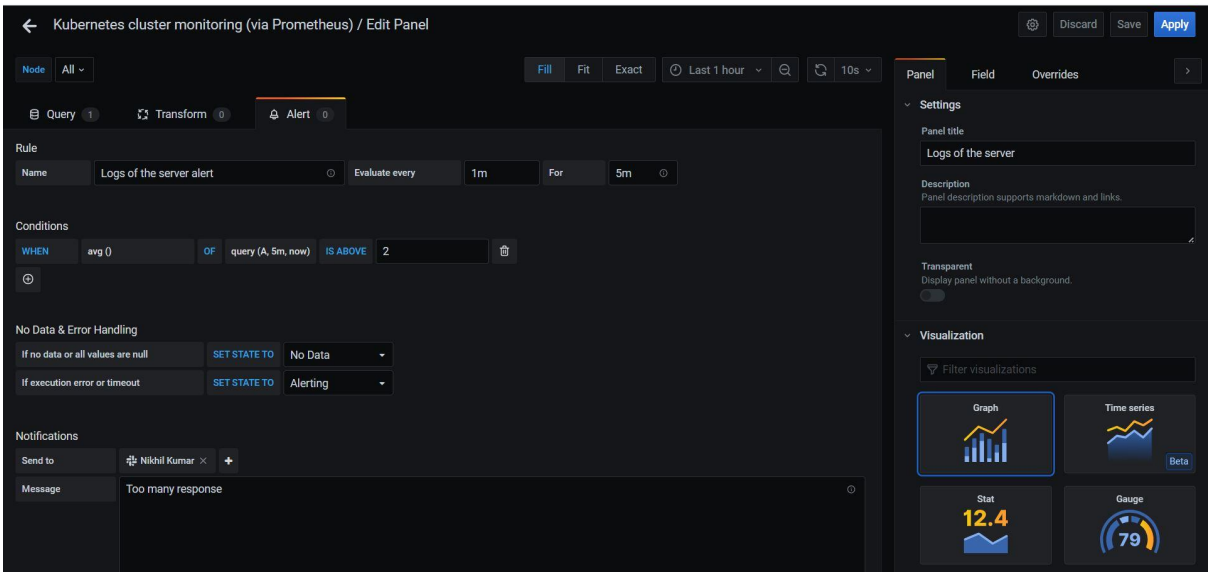
23. Before setting up the alert we need to setup the notification channel where the notification would be sent if the alarm is triggered.Here we setting up slack channel using web hook.
For getting the web hook url we can get it from the channel



24. Now that we have the url we can setup the notification Channel

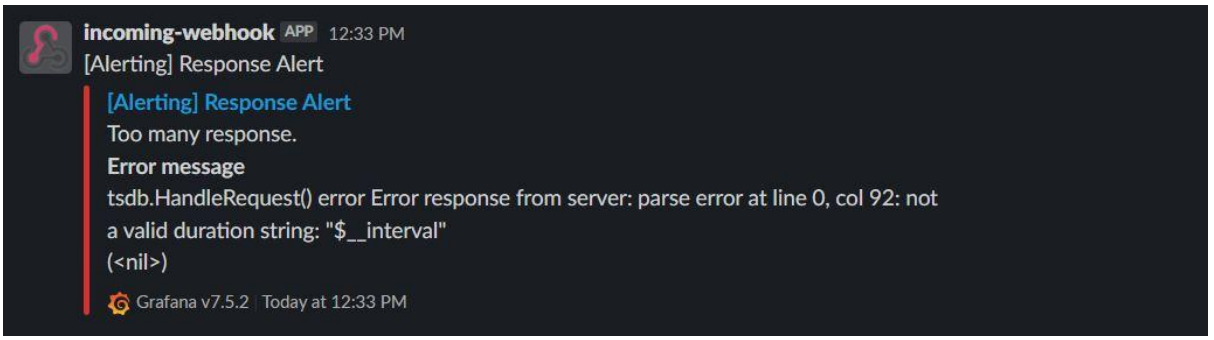


25. Now we setup the alert, for this instance we taking no of response at a given time should not exceed a given limit. If it does the alert will be triggered and notification would be sent to the respective channel



26. After clicking on apply we should get the panel ready on our dashboard.

27. At the End whenever the alert is triggered we would get a notification of the same on our slack channel.



28. After the implementation, now we can delete the cluster and all the resources that are associated with the cluster

```
PS C:\Users\Jaya shree> eksctl delete cluster --region=us-east-1 --name=loki
```