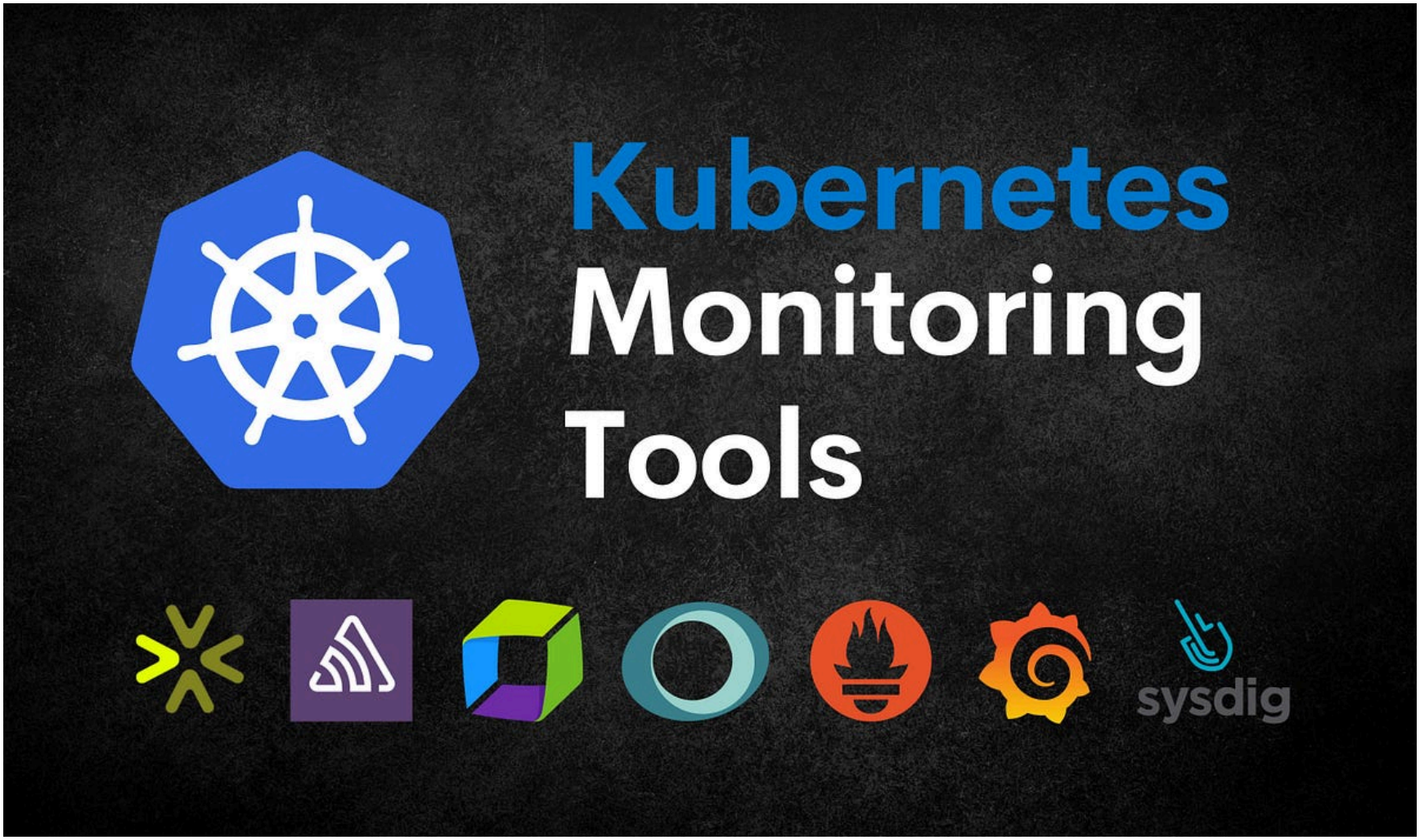
Table of Contents

Introduction: Monitoring and Logging in Kubernetes Tools and Techniques



0.1: Importance of Monitoring and Logging in Kubernetes

1. **Dynamic Environment:** *Challenge:* Kubernetes orchestrates containerized applications across a dynamic environment with constant changes in pod instances and deployments. *Importance:* Monitoring provides real-time insights into the state of the cluster, ensuring prompt detection and response to issues.
2. **Containerized Microservices:** *Challenge:* Microservices architectures in Kubernetes lead to increased complexity and interdependencies among services. *Importance:* Logging facilitates traceability and diagnostics, enabling efficient troubleshooting in distributed systems.
3. **Scalability and Resource Management:** *Challenge:* Kubernetes' scalability features require meticulous resource management to prevent bottlenecks. *Importance:* Monitoring assists in resource utilization tracking, aiding in capacity planning and optimizing performance.
4. **Service Discovery and Networking:** *Challenge:* Managing service discovery and networking complexities in a containerized environment. *Importance:* Observability tools help in tracking network traffic, identifying latency issues, and ensuring seamless communication between services.

0.2: Monitoring Tools in Kubernetes

1. **Prometheus:** *Description:* An open-source monitoring and alerting toolkit designed for reliability and scalability in dynamic environments. *Features:* Time-series data collection, querying language (PromQL), and seamless integration with Kubernetes through the Prometheus Operator.
2. **Grafana:** *Description:* An open-source platform for monitoring and observability, offering customizable dashboards and visualization of data from various sources. *Features:* Grafana integrates seamlessly with Prometheus and other data sources, providing a unified view of metrics.
3. **Kube-state-metrics:** *Description:* A service that listens to the Kubernetes API server and generates metrics about the state of objects. *Features:* Exposes metrics about nodes, pods, deployments, and other Kubernetes resources, enhancing visibility into the cluster state.
4. **Container Runtime Metrics (cAdvisor):** *Description:* Container Advisor, an open-source container resource usage collector, integrated into the kubelet. *Features:* Provides detailed information about container resource usage, performance metrics, and runtime statistics.

5. **Kubernetes Dashboard:** *Description:* A web-based user interface for monitoring and managing Kubernetes clusters. *Features:* Offers a graphical representation of cluster metrics, resource usage, and application status for a quick overview.

0.3: Logging Solutions in Kubernetes

1. **Fluentd:** *Description:* An open-source data collector that unifies data collection and consumption for better understanding and analysis. *Features:* Fluentd supports Kubernetes logging by collecting, filtering, and forwarding logs from various sources to centralized storage.
2. **Elasticsearch:** *Description:* A distributed, RESTful search, and analytics engine that can be used for log storage and retrieval. *Features:* Elasticsearch, when coupled with Logstash and Kibana (ELK stack), forms a powerful logging solution for Kubernetes environments.
3. **Fluent Bit:** *Description:* A lightweight and fast log processor and forwarder that works seamlessly with Fluentd. *Features:* Tailored for Kubernetes environments, Fluent Bit efficiently collects and forwards logs to centralized log storage. Monitoring and Logging in Kubernetes Tools and Techniques.
4. **Loki:** *Description:* A horizontally scalable, highly available, and multi-tenant log aggregation system inspired by Prometheus. *Features:* Loki is designed to handle high volumes of logs, offering efficient querying and filtering capabilities.
5. **Splunk:** *Description:* A platform for searching, monitoring, and analyzing machine-generated data. *Features:* Splunk provides a comprehensive solution for log management, allowing users to gain insights from logs generated across the Kubernetes cluster.

0.4: Techniques for Effective Monitoring and Logging

1. **Use of Labels and Annotations:** *Technique:* Attach labels and annotations to Kubernetes objects to add metadata for better categorization and organization. *Benefits:* Enables more granular monitoring and targeted logging for specific components or applications.
2. **Centralized Logging:** *Technique:* Implement a centralized logging approach to aggregate logs from all containers and pods in the cluster. *Benefits:* Simplifies log analysis, troubleshooting, and compliance monitoring, providing a unified view of cluster-wide logs.

3. **Custom Metrics and Alerts:** *Technique:* Define custom metrics and alerts based on specific application or business logic requirements. *Benefits:* Allows proactive monitoring and alerting tailored to the unique needs of the applications running in the Kubernetes environment.
4. **Tracing and Distributed Context Propagation:** *Technique:* Implement distributed tracing to track the flow of requests across microservices. *Benefits:* Facilitates identification of performance bottlenecks and latency issues, enhancing overall application observability.
5. **Security Monitoring:** *Technique:* Integrate security monitoring tools to detect and respond to potential security threats. *Benefits:* Enhances the security posture of the Kubernetes cluster by identifying unauthorized access, potential vulnerabilities, or suspicious activities.

0.5: Challenges in Monitoring and Logging

1. **Overhead and Resource Consumption:** *Challenge:* Implementing monitoring and logging solutions can introduce additional resource overhead. *Mitigation:* Optimize configurations and utilize efficient tools to minimize the impact on cluster performance.
2. **Data Volume and Retention:** *Challenge:* Handling large volumes of log and metric data, and determining appropriate retention policies. *Mitigation:* Employ log rotation strategies, implement data pruning, and leverage storage solutions capable of handling large datasets. Monitoring and Logging in Kubernetes Tools and Techniques
3. **Integration Complexity:** *Challenge:* Integrating multiple monitoring and logging tools seamlessly within the Kubernetes ecosystem. *Mitigation:* Choose tools with good compatibility, utilize standardized interfaces, and follow best practices for integration.
4. **Real-time Visibility:** *Challenge:* Achieving real-time visibility into the dynamic and distributed nature of Kubernetes clusters. *Mitigation:* Utilize tools with real-time monitoring capabilities and establish efficient alerting mechanisms for prompt issue detection.
5. **Ensuring Data Privacy and Compliance:** *Challenge:* Safeguarding sensitive information contained in logs and ensuring compliance with data privacy regulations. *Mitigation:* Implement encryption for log data in transit and at rest, and regularly audit logging configurations for compliance.

What is my goal here (Monitoring & Alerting) :

- To collect node-level metrics
- To collect pod-level metrics
- To collect pod logs
- To collect node-level logs (dmesg, /var/log/messages, syslog, etc.)

Notification method I am going to use: Alerts via **email** (using Prometheus Alertmanager, of course)

What I am going to install:

- Node Exporters
- Promtail-Loki
- Prometheus
- Alertmanager

Some other notes:

- I am going to use **hostPath-based storage for observability (**Persistent Volume, you could say). It's better to change it according to your use case.
- I am going to use **Loki (single binary/monolithic).** Other options available as well, like Loki-scalable, which I am not covering.
- **Kubernetes distribution I am going to use:** MicroK8s
- I have already decided what my **external URLs for Prometheus UI, Grafana, AlertManager, and Promtail** will be (Not Covering Ingress and that Stuff).
    - I am going to use NodePort for all for externally public access.
    - In My Case it would be (Public_ip_of_kmaster:NodePort)

1.0 : Installation / Configuration of full Observability Stack in one go (Grafana / Loki + Promtail / Prometheus / Node Exporter)

Grafana : Visualization
Loki + Promtail : For logs
Prometheus: For kubernetes pod/container level metrics
Node Exporter: Node Level Metrics.

Observability Source Code : https://github.com/vishalk17/observability-stack-kubernetes

1.1 : Clone Observability Source Code :

```
git clone https://github.com/vishalk17/observability-stack-kubernetes observability
```

1.2 : Modified / Verify Few things in Source Code :

- 1.2.1 : Open observability-install.sh file and Replace your smtp pass then save and exit

```
#!/bin/bash

# script by github.com/vishalk17
# contact t.me/vishalk17

# Error handling
set -e

# Set environment variables
kubectl="kubectl"
kubectl="microk8s kubectl"  # if not exit then pick this
helm="helm"
helm="microk8s helm3"   # if not exit then pick this
smtp_pass="your-pass"
```

==========

- 1.2.2 : Open grafana-values.yaml and Check your smtp credentials + pvc claim name for grafana

```yaml
grafana:
  persistence:
    enabled: true
    existingClaim: grafana-pvc
  grafana.ini:
    angular_support_enabled: true    # angular js support is likely going to Deprecate , legacy flag may work for some broken dashboards.
    smtp:  # Configuration for Grafana email notifications
      enabled: true
      from_name: grafana-k8s-notify
      host: smtp.vishalk17.com
      user: vishalkapadi95@gmail.com  # Password should be provided in a secret
```

-

1.2.3 : Open grafana-values.yaml and check external alertmanager url + retention period to be set + pvc Details

- Replace alertmanager External url (Must be Publically accessible)

```yaml
alertmanager:
  configmapReload:
    enabled: true
  # Configuration for Alertmanager
  alertmanagerSpec:
    externalUrl: http://kube-prom-stack-kube-prome-alertmanager.observability.svc.cluster.local:9093/  # replace with acutal external alertmanager url
    retention: 720h # 720h = 30 days , Time duration Alertmanager shall retain data
    alertmanagerConfigSelector:
      matchLabels:
        alertmanagerConfig: custom
    alertmanagerConfigNamespaceSelector:
      matchLabels:
        alertmanagerconfig: enabled
  #  alertmanagerConfiguration:
  #    alertmanagerConfig: custom
    forceEnableClusterMode: true
    storage:
      volumeClaimTemplate:
        spec:
          storageClassName: microk8s-hostpath
          accessModes: ["ReadWriteOnce"]
          resources:
            requests:
              storage: 30Gi
          selector:
            matchLabels:
              type: alertmanager-storage
```

- 1.2.3 : Open grafana-values.yaml and check external Prometheus url + retention period to be set + pvc Details + Scrape Intervals
    - Replace Prometheus External url (Must be Publically accessible)

```
prometheus:
  prometheusSpec:
    externalUrl: http://kube-prom-stack-kube-prome-prometheus.observability.svc.cluster.local:9090/ # replace with acutal external prometheus url
    scrapeTimeout: 30s          # Default to 30s
    scrapeInterval: 30s         # Default to 30s # Below 30s , It wont work
    evaluationInterval: 15s     # Default to 30s
    ## How long to retain metrics
    retention: 90d
    # Rules yaml has to match with this label in order to take those into account
    ruleSelector:
      matchLabels:
        resource: prom-custom

    storageSpec:
      volumeClaimTemplate:
        spec:
          storageClassName: microk8s-hostpath
          accessModes: ["ReadWriteOnce"]
          resources:
            requests:
              storage: 500Gi
          selector:
            matchLabels:
              type: prometheus-storage
```

If all details has been modified in grafana-values.yaml then save / Exit

1.2.4 : **Open** loki-obser/loki/loki-values.yaml

- Check log retention period
- Replace External url of loki / alertmanager url (Must be Publically accessible) needs to be updated

```
      admin_api_directory: /var/loki/admin

  compactor: # which compacts index shards for performance.
    working_directory: /var/loki/compactor/retention # Directory where files can be downloaded for compaction.
    compaction_interval: 10m
    retention_enabled: true
    retention_delete_delay: 2h
    retention_delete_worker_count: 150
    delete_request_store: filesystem

  limits_config:
    max_query_lookback: 2160h # 90 days
    retention_period: 2160h # 90 days

  storage_config:
```

```
rulerConfig:
# Base URL of the Grafana instance.
  external_url: "http://kube-prom-stack-grafana.observability.svc.cluster.local:80"
# Comma-separated list of Alertmanager URLs to send notifications to. Each
# Alertmanager URL is treated as a separate group in the configuration. Multiple
  alertmanager_url: "http://kube-prom-stack-kube-prome-alertmanager.observability.svc.cluster.local:9093"
# Datasource UID for the dashboard.
  datasource_uid: "loki"
```

1.2.5 : Check all persistent volume definitions

- Check hostpath volume mount point
- Check permission to be be given that hostPath mount point

Files location as follows :

- grafana-pv.yaml (It Includes for Prometheus / Alertmanager / Grafana)
- loki-pv-pvc.yaml (For Loki)

1.2.6: Update loki-obser/loki/loki-rules-configmap.yaml (If Required)

For more info. Config << Click Here

1.2.7: Update Prometheus Metrics rules.(If Required)

For more info. Config << Click Here

1.2.8: Update/Check Alertmanager.yaml (If Required)

For more info. 5.2: Alertmanager Configuration << Click Here

1.3: Create observability NameSpace & Add / Sync Helm repos :

```
kubectl create ns observability
helm repo add kube-prom-stack https://prometheus-community.github.io/helm-charts
helm repo add grafana https://grafana.github.io/helm-charts
helm repo update
```

1.4: Installing Promtail :

```
ubuntu@kmaster:~/vishal/observability$ kubectl apply -f loki-obser/promtail/promtail-service.yaml
service/promtail-service created
ubuntu@kmaster:~/vishal/observability$ helm upgrade --install --values
loki-obser/promtail/promtail-values.yaml promtail grafana/promtail -n observability

Release "promtail" does not exist. Installing it now.
NAME: promtail
LAST DEPLOYED: Mon Jun 17 20:48:55 2024
NAMESPACE: observability
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
**********************************************************************
 Welcome to Grafana Promtail
 Chart version: 6.16.0
 Promtail version: 3.0.0
**********************************************************************

Verify the application is working by running these commands:
* kubectl --namespace observability port-forward daemonset/promtail 3101
```

```
ubuntu@kmaster:~/vishal/observability$ kubectl get pods -n observability
NAME             READY   STATUS    RESTARTS   AGE
promtail-mhnwf   1/1     Running   0          2m44s
promtail-bn9x7   1/1     Running   0          2m44s
promtail-k4xqv   1/1     Running   0          2m44s
```

1.5: Installing Loki (Monolithic - Single Binary) :

Install pv-pvc for loki:

```
ubuntu@kmaster:~/vishal/observability$ kubectl apply -f loki-obser/loki/loki-pv-pvc.yaml
persistentvolumeclaim/storage-loki-0 created
persistentvolume/loki-pv created
```

Install Loki Alerting rules:

```
ubuntu@kmaster:~/vishal/observability$ kubectl apply -f loki-obser/loki/loki-rules-configmap.yaml
configmap/loki-alerting-rules created
```

Install loki :

```
ubuntu@kmaster:~/vishal/observability$ helm upgrade --install --values loki-obser/loki/loki-values.yaml
loki grafana/loki -n observability

Release "loki" does not exist. Installing it now.
NAME: loki
LAST DEPLOYED: Mon Jun 17 20:57:47 2024
```

==========

```
NAMESPACE: observability
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
**********************************************************************
 Welcome to Grafana Loki
 Chart version: 6.6.3
 Chart Name: loki
 Loki version: 3.0.0
**********************************************************************

** Please be patient while the chart is being deployed **

Tip:

  Watch the deployment status using the command: kubectl get pods -w --namespace observability

If pods are taking too long to schedule make sure pod affinity can be fulfilled in the current cluster.

**********************************************************************
Installed components:
**********************************************************************
* loki

Loki has been deployed as a single binary.
This means a single pod is handling reads and writes. You can scale that pod vertically by adding more
CPU and memory resources.
```

```
**********************************************************************
Sending logs to Loki
**********************************************************************

Loki has been configured with a gateway (nginx) to support reads and writes from a single component.

You can send logs from inside the cluster using the cluster DNS:

http://loki-gateway.observability.svc.cluster.local/loki/api/v1/push

You can test to send data from outside the cluster by port-forwarding the gateway to your local machine:

  kubectl port-forward --namespace observability svc/loki-gateway 3100:80 &

And then using http://127.0.0.1:3100/loki/api/v1/push URL as shown below:

```
curl -H "Content-Type: application/json" -XPOST -s "http://127.0.0.1:3100/loki/api/v1/push"  \
--data-raw "{\"streams\": [{\"stream\": {\"job\": \"test\"}, \"values\": [[\"$(date +%s)000000000\",
\"fizzbuzz\"]]}]}"
```

Then verify that Loki did received the data using the following command:

```
curl "http://127.0.0.1:3100/loki/api/v1/query_range" --data-urlencode 'query={job="test"}' | jq
.data.result
```

****
```

```
Connecting Grafana to Loki
**********************************************************************

If Grafana operates within the cluster, you'll set up a new Loki datasource by utilizing the following
URL:

http://loki-gateway.observability.svc.cluster.local/
```

```
ubuntu@kmaster:~/vishal/observability$ kubectl get pods -n observability
NAME                            READY   STATUS    RESTARTS   AGE
loki-0                          1/1     Running   0          57s
loki-gateway-64656c774d-q264v   1/1     Running   0          44s
promtail-n8xrz                  1/1     Running   0          21s
promtail-vvr2p                  1/1     Running   0          21s
promtail-w7zbh                  1/1     Running   0          21s
```

1.6: Installing kube-prometheus-stack:

- By default it will install , node-exporter, kube-state-metrics, Prometheus , Prometheus Operator , Grafana, Alertmanager

1.6.1: Install pv-pvc for Prometheus, Grafana, Alertmanager

```
ubuntu@kmaster:~/vishal/observability$ ls
alertmanager.yaml  grafana-alert-provisioning.yaml  grafana-pv.yaml  grafana-values.yaml  loki-obser
observability-install.sh  rules

ubuntu@kmaster:~/vishal/observability$ kubectl apply -f grafana-pv.yaml
persistentvolumeclaim/grafana-pvc created
persistentvolume/grafana-pv created
persistentvolume/alertmanager-pv created
```

```
persistentvolume/prometheus-pv created
```

1.6.2: Execute observability-install.sh Script

What is Inside of this Script :

- When execute it will install kube-prometheus-stack
- Fixup grafana Smtp
- Add loki , prometheus data sources to Grafana

```
ubuntu@kmaster:~/vishal/observability$ bash observability-install.sh
"kube-prom-stack" already exists with the same configuration, skipping
"grafana" already exists with the same configuration, skipping
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "grafana" chart repository
...Successfully got an update from the "loki" chart repository
...Successfully got an update from the "kube-prom-stack" chart repository
Update Complete. ⎈Happy Helming!⎈
Release "kube-prom-stack" does not exist. Installing it now.
NAME: kube-prom-stack
LAST DEPLOYED: Fri Jul 12 19:14:22 2024
NAMESPACE: observability
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace observability get pods -l "release=kube-prom-stack"

Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create &
configure Alertmanager and Prometheus instances using the Operator.

fixing smtp for grafana
```

```
applying patch for pass of smtp. make sure you have edited this script to include

Warning: resource configmaps/kube-prom-stack-grafana is missing the
kubectl.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl
apply should only be used on resources created declaratively by either kubectl create --save-config or
kubectl apply. The missing annotation will be patched automatically.
configmap/kube-prom-stack-grafana configured

Restart the deployment of grafana

deployment.apps/kube-prom-stack-grafana restarted

fixed smtp
====================================
 grafana username: admin
 grafana pass: prom-operator
====================================
```

```
ubuntu@kmaster:~$ kubectl get pv,pvc -n observability
NAME                                 CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
persistentvolume/alertmanager-pv     30Gi       RWO            Retain           Bound    observability/alertmanager-kube-prom-stack-kube-prome-alertmanager-db-alertmanager-kube-prom-stack-kube-prome-ale
persistentvolume/grafana-pv          30Gi       RWO            Retain           Bound    observability/grafana-pvc
persistentvolume/loki-pv             500Gi      RWO            Retain           Bound    observability/storage-loki-0
persistentvolume/prometheus-pv       500Gi      RWO            Retain           Bound    observability/prometheus-kube-prom-stack-kube-prome-prometheus-db-prometheus-kube-prom-stack-kube-prome-prometheus

NAME                                                                                                                                                STATUS   VOLUME           CAPACITY   ACCESS MODES   STORAGECLASS
persistentvolumeclaim/alertmanager-kube-prom-stack-kube-prome-alertmanager-db-alertmanager-kube-prom-stack-kube-prome-alertmanager-0   Bound    alertmanager-pv   30Gi       RWO            microk8s-host
persistentvolumeclaim/grafana-pvc                                                                                                                   Bound    grafana-pv        30Gi       RWO            microk8s-host
persistentvolumeclaim/prometheus-kube-prom-stack-kube-prome-prometheus-db-prometheus-kube-prom-stack-kube-prome-prometheus-0                       Bound    prometheus-pv     500Gi      RWO            microk8s-host
persistentvolumeclaim/storage-loki-0                                                                                                                Bound    loki-pv           500Gi      RWO            microk8s-host
```

```
ubuntu@kmaster:~/vishal/observability/loki-obser/promtail$ kubectl get pods -n observability
NAME                                                     READY   STATUS    RESTARTS      AGE
alertmanager-kube-prom-stack-kube-prome-alertmanager-0   1/2     Running   2 (23s ago)   3m54s
kube-prom-stack-grafana-695457cd8d-v5sc7                 3/3     Running   0             13m
kube-prom-stack-kube-prome-operator-7d7d8fd89f-8xbhr     1/1     Running   0             13m
kube-prom-stack-kube-state-metrics-5cf7d4cf6b-hfdvl      1/1     Running   0             13m
kube-prom-stack-prometheus-node-exporter-cpgfn           1/1     Running   0             13m
kube-prom-stack-prometheus-node-exporter-rwgk8           1/1     Running   0             13m
kube-prom-stack-prometheus-node-exporter-xktf6           1/1     Running   0             13m
loki-0                                                   1/1     Running   3 (7m4s ago)  7m30s
loki-gateway-7f79d5cd74-z5c5h                            1/1     Running   0             7m10s
prometheus-kube-prom-stack-kube-prome-prometheus-0       2/2     Running   0             3m54s
promtail-2szr7                                           1/1     Running   0             93s
promtail-b9ds2                                           1/1     Running   0             93s
promtail-k4xqv                                           1/1     Running   0             93s
ubuntu@kmaster:~/vishal/observability/loki-obser/promtail$
```

All pods are running and up

1.6.3: Add/ update/ configured Alerting rules for prometheus :

```
ubuntu@kmaster:~/vishal/observability$ kubectl apply -f rules/recording-rules-only/./
prometheusrule.monitoring.coreos.com/kube-prometheus-rules created
prometheusrule.monitoring.coreos.com/kubernetes-monitoring-rules created
prometheusrule.monitoring.coreos.com/node-exporter-recording-rules created

ubuntu@kmaster:~/vishal/observability$ kubectl apply -f rules/./
prometheusrule.monitoring.coreos.com/resource-monitor-node-rules unchanged
prometheusrule.monitoring.coreos.com/resource-monitor-pod-rules unchanged
prometheusrule.monitoring.coreos.com/probes-rules unchanged
```

Ref for more info.
6.3.2: Installation of Recording rules ,
6.4 : Add / Remove / Update Prometheus Alerting Rules (based off metrics) :

==========

1.7.0: Update AlertManager Secret

Encode alertmanager.yaml in base64

```
ubuntu@kmaster:~/vishal/observability$ ls
alertmanager.yaml  grafana-alert-provisioning.yaml  grafana-pv.yaml  grafana-values.yaml  loki-obser
observability-install.sh  rules

ubuntu@kmaster:~/vishal/observability$ cat "alertmanager.yaml" | base64 -w0
```


```
IyBNb2RpZmllZCBieSBnaXRodWIuY29tL3Zpc2hhbGsxNwojIEdsb2JhbCBjb25maWd1cmF0aW9uCmdsb2JhbDoKICByZXNvbHZlX3Rpb
WVvdXQ6IDFtCgojIFJvdXRpbmcgY29uZmlndXJhdGlvbgpyb3V0ZToKICAjIEdyb3VwaW5nIGNyaXRlcmlhIGZvciBhbGVydHMKICBncm
91cF9ieTogWycuLi4nXSAgICMgJy4uLicgYnkgYWRkaW5nIHRoaXMgd2UgY2FuIGRpc2FibGUgZ3JvdXBpbmcgb2YgYWxlcnRzIGluIHN
pbmdsZSBub3RpZmljYXRpb24KICAgICAgICAgICAgICAgICAgICAgICAgICMgIHRoaXMgd2lsbCBhdm9pZCBtZXNzdXAgb2Ygbm90aWZpY2F0
aW9ucyBvZiBhbGVydHMgd2l0aGluIHNpbmdsZSBub3RpZmljYXRpb24uCiAgIyBXYWl0IHRpbWUgYmVmb3JlIHNlbmRpbmcgaW5pdGlhb
CBub3RpZmljYXRpb24gZm9yIGEgZ3JvdXAgb2YgYWxlcnRzCiAgZ3JvdXBfd2FpdDogMXMKICAjIEludGVydmFsIGJldHdlZW4gc2VuZG
luZyBub3RpZmljYXRpb25zIGZvciB0aGUgc2FtZSBncm91cCBvZiBhbGVydHMKICBncm91cF9pbnRlcnZhbDogMXMKICAjIEludGVydmF
sIGJldHdlZW4gcmVzZW5kaW5nIGFsZXJ0cyB0aGF0IGhhdmUgYmVlbiBzdWNjZXNzZnVsbHkgc2VudAogIHJlcGVhdF9pbnRlcnZhbDog
MWgKICAjIFJlY2VpdmVyIHRvIHNlbmQgbm90aWZpY2F0aW9ucyB0bwogIHJlY2VpdmVyOiAnZW1haWwtYWxlcnQnCiAgIyBSb3V0ZXMgZ
m9yIG1hdGNoaW5nIGFsZXJ0cwogIHJvdXRlczoKICAgIC0gbWF0Y2g6CiAgICAgICAgc2V2ZXJpdHk6ICdjcml0aWNhbHx3YXJuaW5nJw
oKIyBSZWNlaXZlciBjb25maWd1cmF0aW9uCnJlY2VpdmVyczoKICAjIEdvb2dsZSBDaGF0IHJlY2VpdmVyCiAgLSBuYW1lOiAnR29vZ2x
```

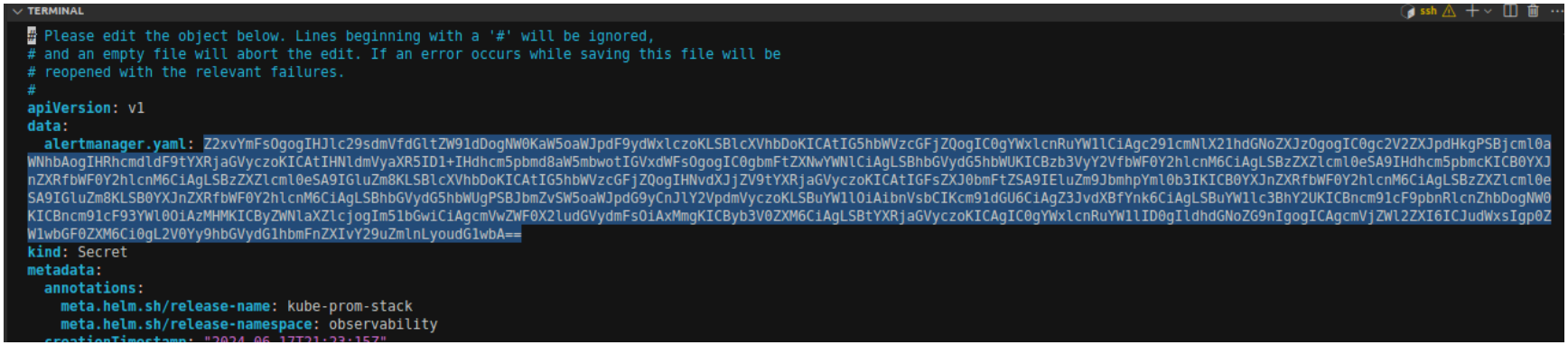

Save this encoded text somewhere so later we can use it.

```
ubuntu@kmaster:~/vishal/observability$ kubectl get secret -n observability | grep alertmanager
alertmanager-kube-prom-stack-kube-prome-alertmanager                Opaque    1     20m
alertmanager-kube-prom-stack-kube-prome-alertmanager-generated      Opaque    1     20m
alertmanager-kube-prom-stack-kube-prome-alertmanager-tls-assets-0   Opaque    0     20m
alertmanager-kube-prom-stack-kube-prome-alertmanager-web-config     Opaque    1     20m
```

```
ubuntu@kmaster:~/vishal/observability$ kubectl edit secret alertmanager-kube-prom-stack-kube-prome-alertmanager -n observability
```

This will open `alertmanager-kube-prom-stack-kube-prome-alertmanager` in Text Editor

Replace this Encoded text with encoded text Created in previous Step.



- Save and Exit.

```
ubuntu@kmaster:~/vishal/observability$ kubectl edit secret alertmanager-kube-prom-stack-kube-prome-alertmanager -n observability
secret/alertmanager-kube-prom-stack-kube-prome-alertmanager edited
```

1.8.0: Expose Grafana, Prometheus UI , AlertManager UI to internet

Using a NodePort service will allow us to access Grafana, Prometheus, and Alertmanager UI over the internet. This setup will expose these services on a specific port, enabling access via Public_ip:NodePort

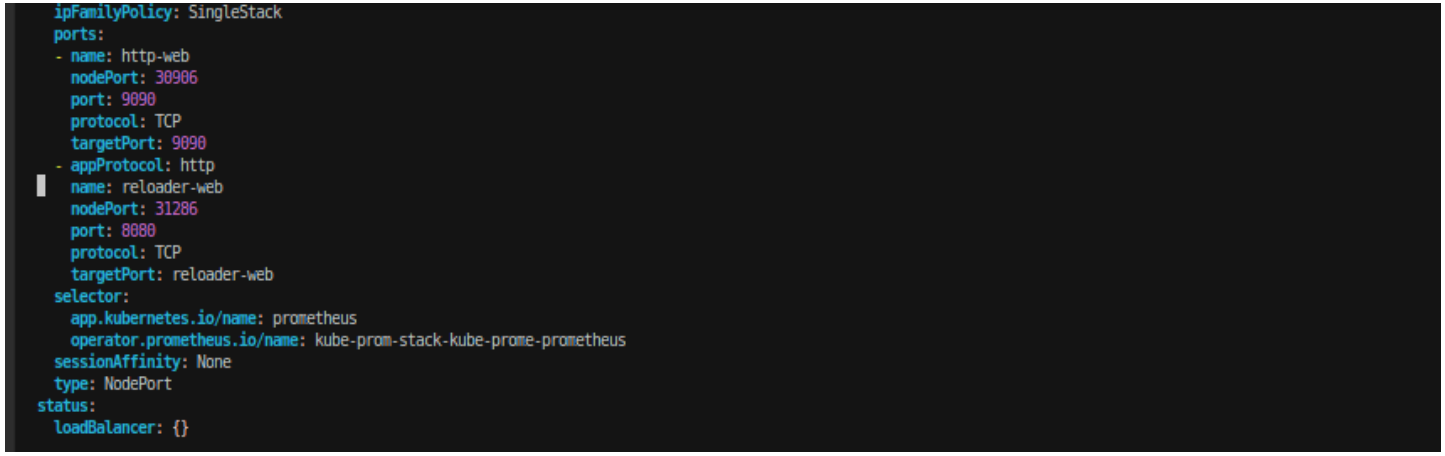
We will going to svc one by one and then we will change service type to NodePort and then we will add NodePort

List all the services,

```
ubuntu@kmaster:~/vishal/observability$ kubectl get svc -n observability
NAME                                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
promtail-service                          NodePort    10.152.183.74    <none>        3101:32560/TCP               17h
loki-headless                             ClusterIP   None             <none>        3100/TCP                     17h
loki-memberlist                           ClusterIP   None             <none>        7946/TCP                     17h
loki                                      ClusterIP   10.152.183.145   <none>        3100/TCP,9095/TCP            17h
loki-gateway                              ClusterIP   10.152.183.100   <none>        80/TCP                       17h
kube-prom-stack-kube-prome-prometheus     ClusterIP   10.152.183.35    <none>        9090/TCP,8080/TCP            16h
kube-prom-stack-grafana                   ClusterIP   10.152.183.230   <none>        80/TCP                       16h
kube-prom-stack-prometheus-node-exporter  ClusterIP   10.152.183.144   <none>        9100/TCP                     16h
kube-prom-stack-kube-state-metrics        ClusterIP   10.152.183.49    <none>        8080/TCP                     16h
kube-prom-stack-kube-prome-operator       ClusterIP   10.152.183.25    <none>        443/TCP                      16h
kube-prom-stack-kube-prome-alertmanager   ClusterIP   10.152.183.142   <none>        9093/TCP,8080/TCP            16h
alertmanager-operated                     ClusterIP   None             <none>        9093/TCP,9094/TCP,9094/UDP   16h
prometheus-operated                       ClusterIP   None             <none>        9090/TCP                     16h
```

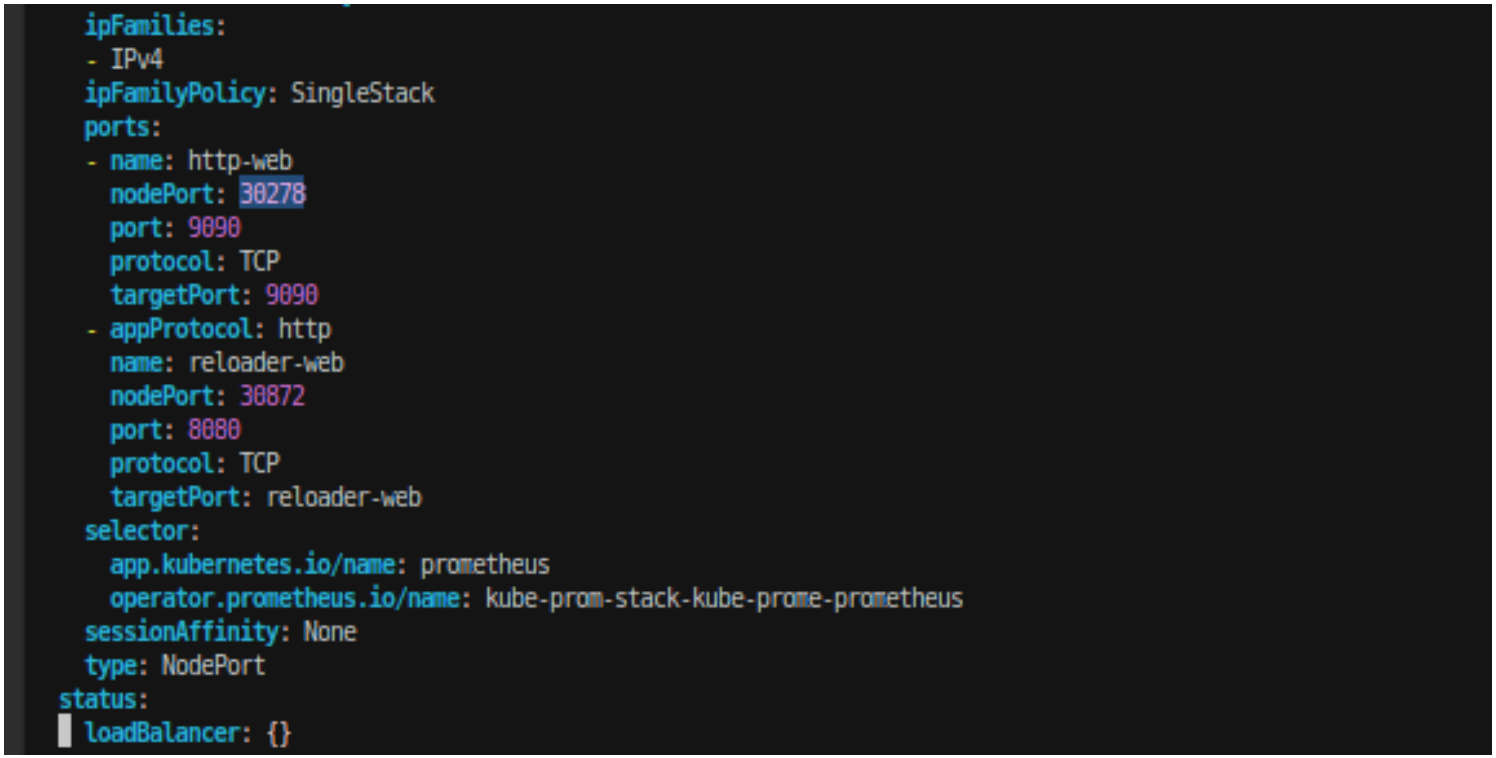
1.8.1: Prometheus: (Change type: NodePort , and add nodePort : 30278)

Before:



```
  ipFamilyPolicy: SingleStack
  ports:
  - name: http-web
    nodePort: 30906
    port: 9090
    protocol: TCP
    targetPort: 9090
  - appProtocol: http
    name: reloader-web
    nodePort: 31286
    port: 8080
    protocol: TCP
    targetPort: reloader-web
  selector:
    app.kubernetes.io/name: prometheus
    operator.prometheus.io/name: kube-prom-stack-kube-prome-prometheus
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

After:



```
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http-web
    nodePort: 30278
    port: 9090
    protocol: TCP
    targetPort: 9090
  - appProtocol: http
    name: reloader-web
    nodePort: 30872
    port: 8080
    protocol: TCP
    targetPort: reloader-web
  selector:
    app.kubernetes.io/name: prometheus
    operator.prometheus.io/name: kube-prom-stack-kube-prome-prometheus
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

- Save & Exit

1.8.2: Follow same thing for others

AlertManager : `kube-prom-stack-kube-prome-alertmanager` (Change type: NodePort , and add nodePort : 32057)
Grafana: `kube-prom-stack-grafana` (Change type: NodePort , and add nodePort : 32218)

Final result :

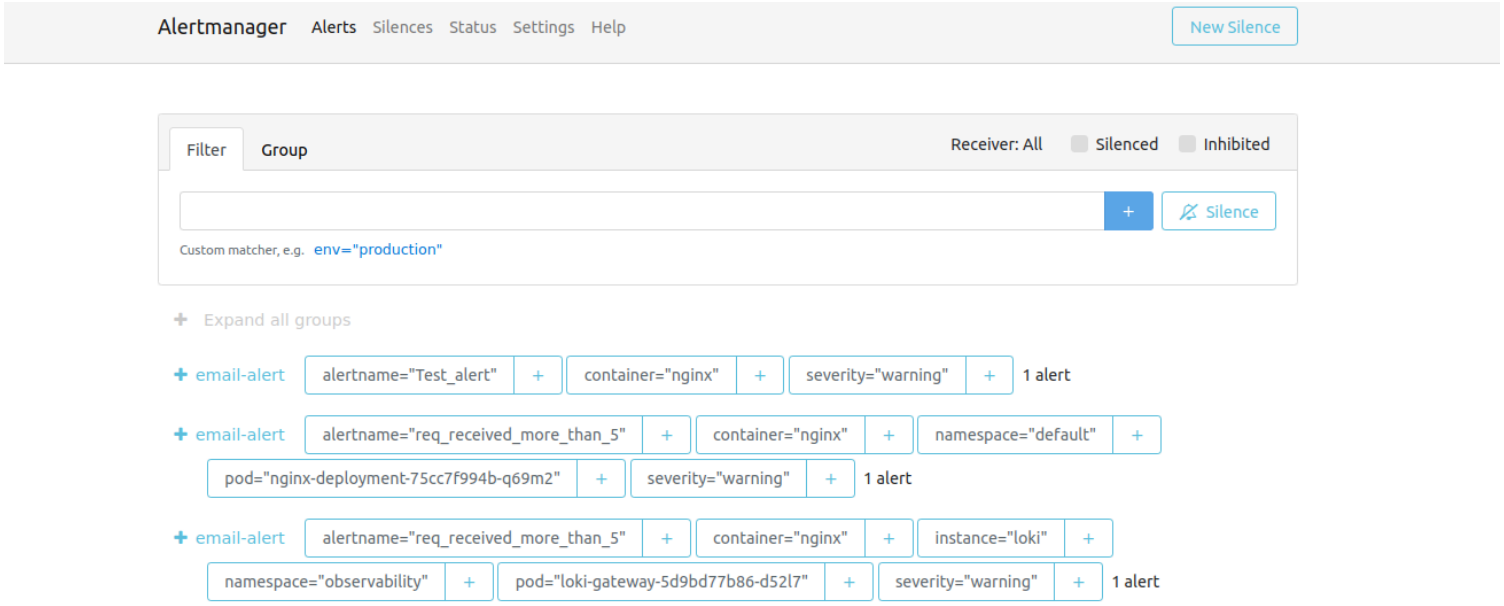
```
ubuntu@kmaster:~/vishal/observability$ kubectl get svc -n observability
NAME                                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
promtail-service                          NodePort    10.152.183.74    <none>        3101:32560/TCP               19h
loki-headless                             ClusterIP   None             <none>        3100/TCP                     18h
loki-memberlist                           ClusterIP   None             <none>        7946/TCP                     18h
loki                                      ClusterIP   10.152.183.145   <none>        3100/TCP,9095/TCP            18h
loki-gateway                              ClusterIP   10.152.183.100   <none>        80/TCP                       18h
kube-prom-stack-prometheus-node-exporter  ClusterIP   10.152.183.144   <none>        9100/TCP                     18h
kube-prom-stack-kube-state-metrics        ClusterIP   10.152.183.49    <none>        8080/TCP                     18h
kube-prom-stack-kube-prome-operator       ClusterIP   10.152.183.25    <none>        443/TCP                      18h
alertmanager-operated                     ClusterIP   None             <none>        9093/TCP,9094/TCP,9094/UDP   18h
prometheus-operated                       ClusterIP   None             <none>        9090/TCP                     18h
kube-prom-stack-kube-prome-prometheus     NodePort    10.152.183.35    <none>        9090:30278/TCP,8080:31286/TCP   18h
kube-prom-stack-grafana                   NodePort    10.152.183.230   <none>        80:32218/TCP                 18h
kube-prom-stack-kube-prome-alertmanager   NodePort    10.152.183.142   <none>        9093:32057/TCP,8080:31057/TCP   18h
```

1.9.0 : Import / Create Grafana Dashboards :

Access grafana ui then Import our customized dashboards or You can also Create them

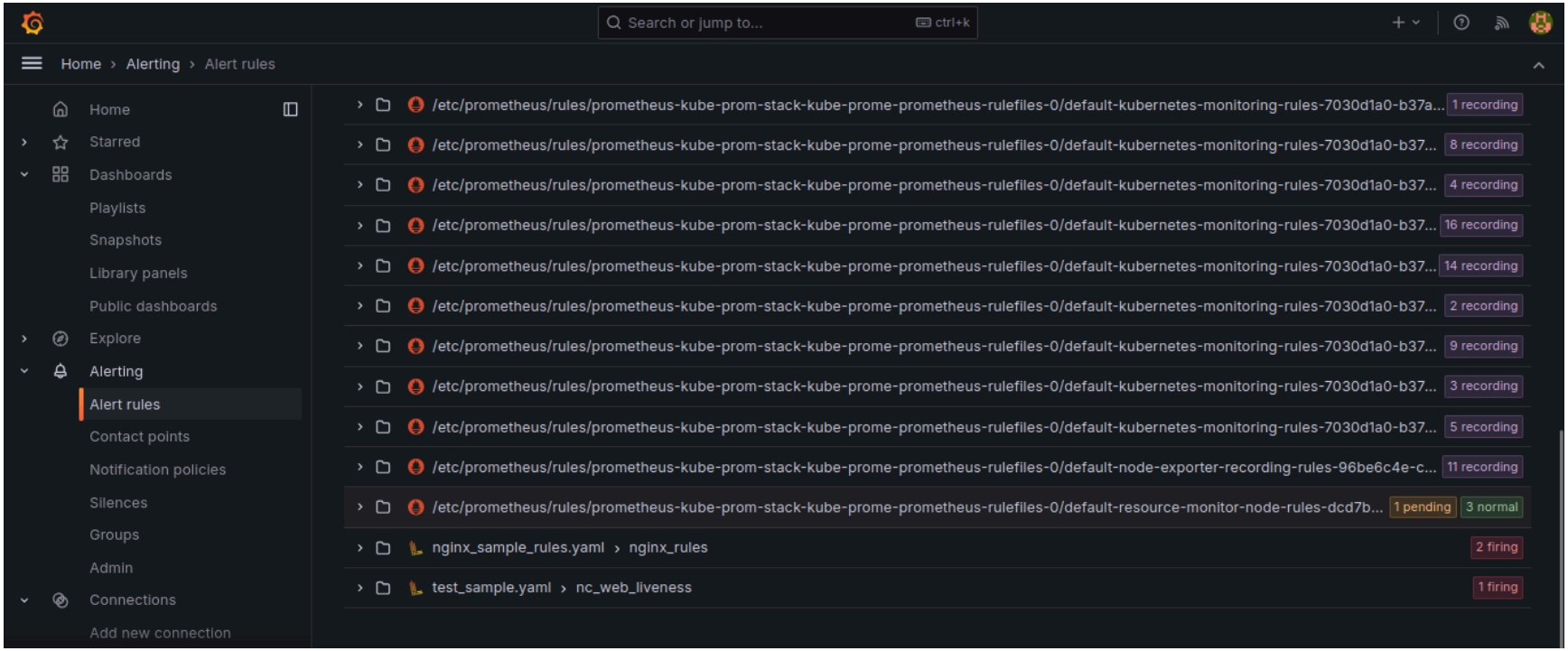
For more info. 7.3.0 : Grafana DashBoards : << click here

1.10.0 : Screenshot of Alerts :

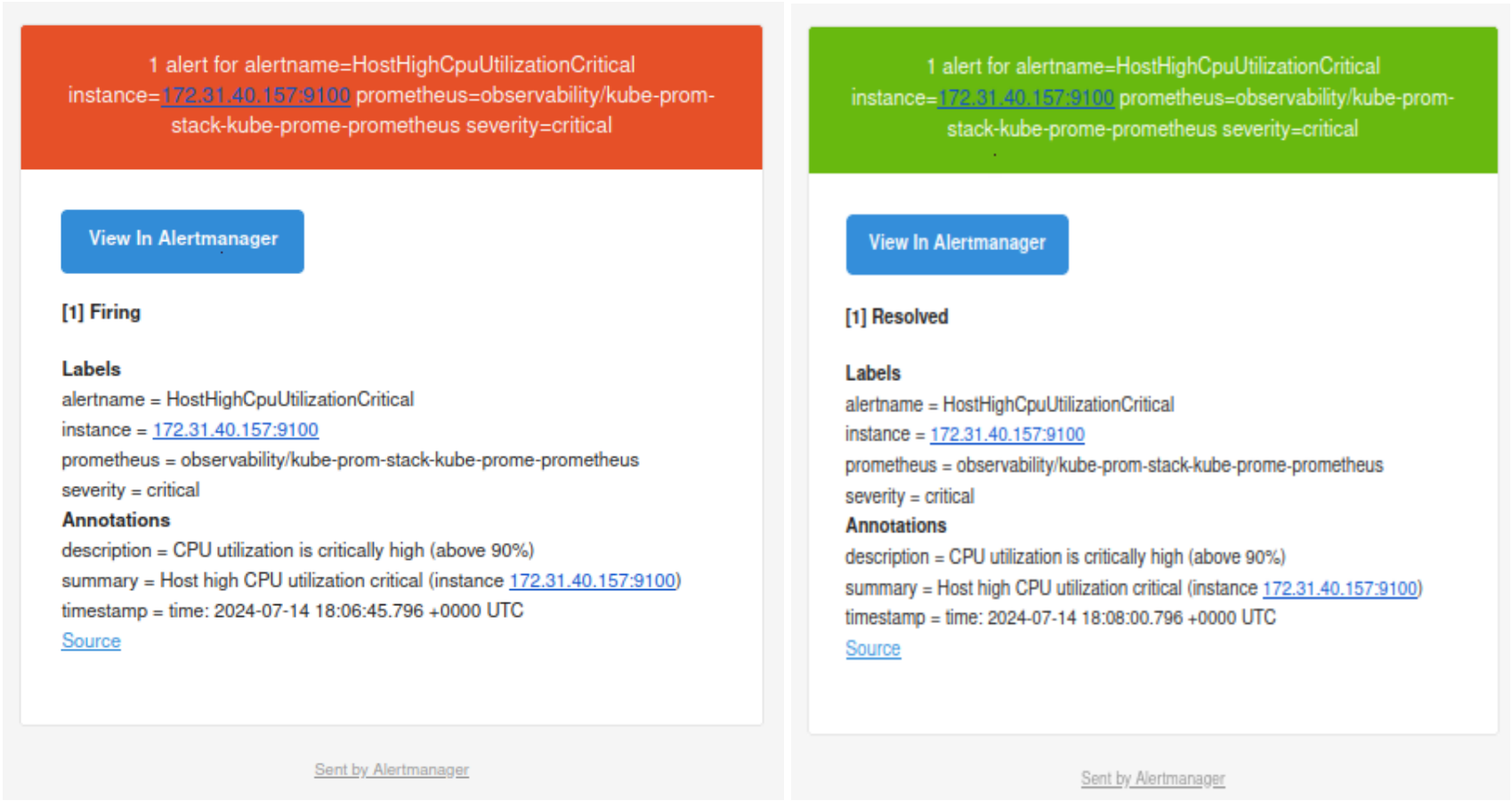


==========

Additionally Grafana will only show which one is fired , pending and Normal (We Haven't configured rules from grafana)



==========



1 alert for alertname=HostHighCpuUtilizationCritical instance=172.31.40.157:9100 prometheus=observability/kube-prom-stack-kube-prome-prometheus severity=critical

View In Alertmanager

[1] Firing

Labels
alertname = HostHighCpuUtilizationCritical
instance = 172.31.40.157:9100
prometheus = observability/kube-prom-stack-kube-prome-prometheus
severity = critical
Annotations
description = CPU utilization is critically high (above 90%)
summary = Host high CPU utilization critical (instance 172.31.40.157:9100)
timestamp = time: 2024-07-14 18:06:45.796 +0000 UTC
Source

Sent by Alertmanager

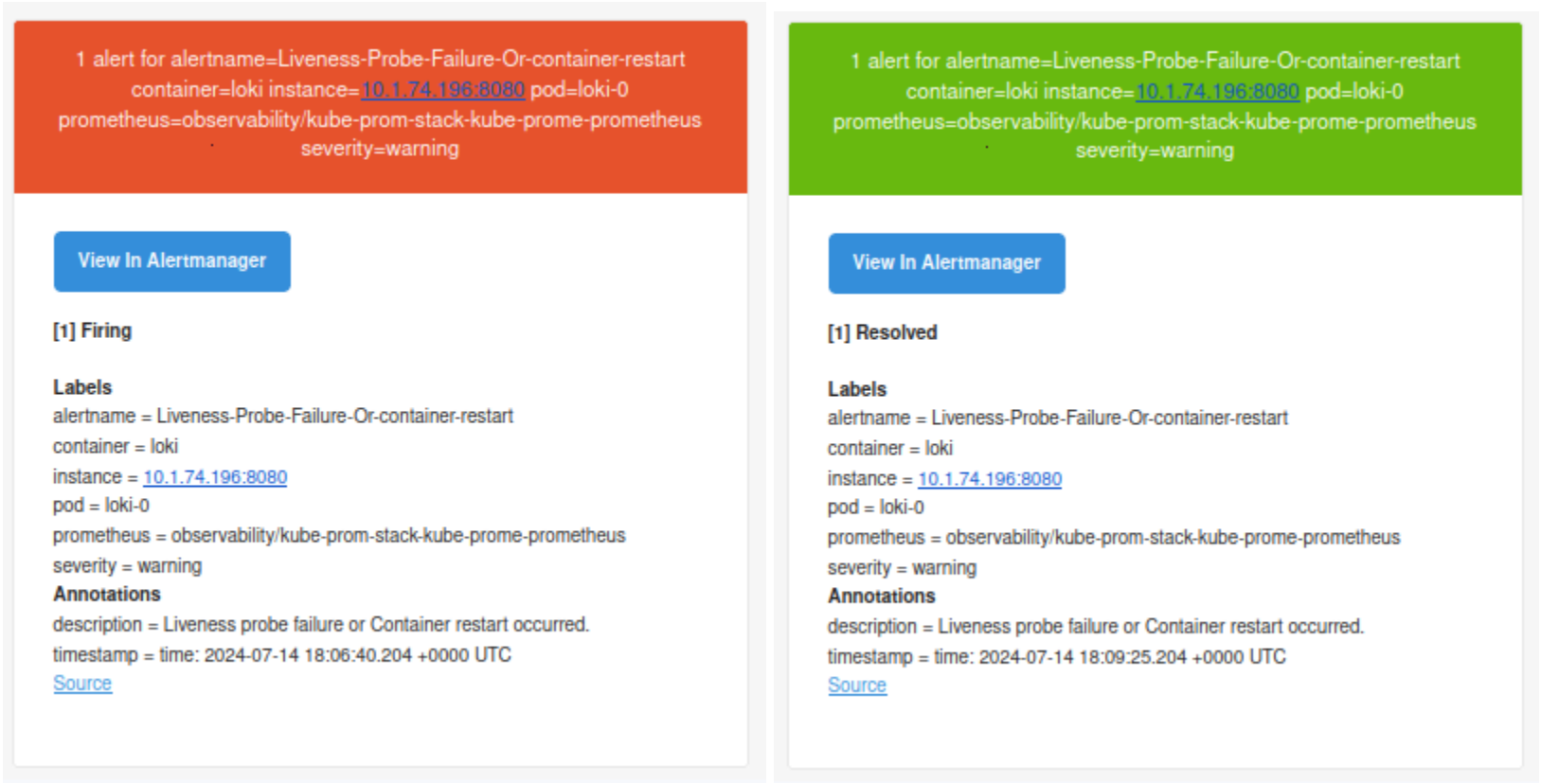
1 alert for alertname=HostHighCpuUtilizationCritical instance=172.31.40.157:9100 prometheus=observability/kube-prom-stack-kube-prome-prometheus severity=critical

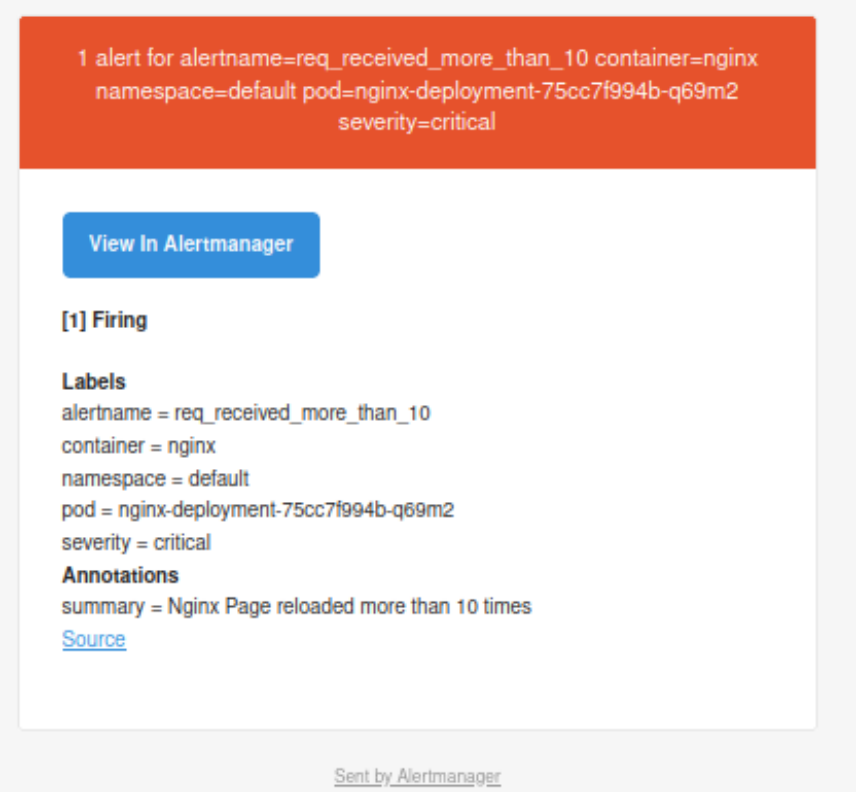
View In Alertmanager

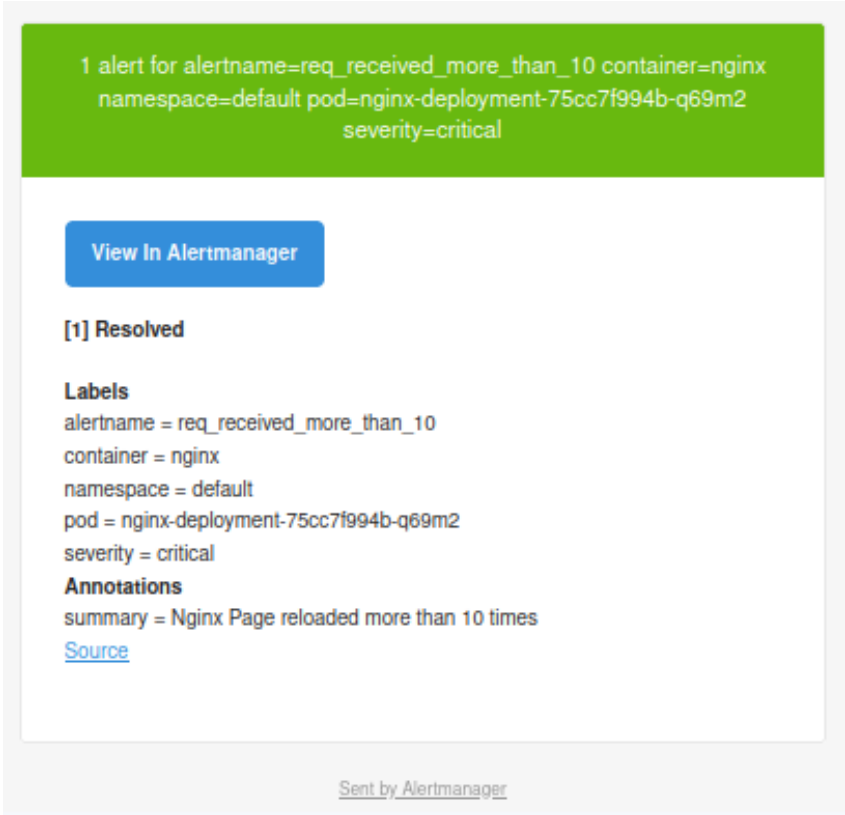
[1] Resolved

Labels
alertname = HostHighCpuUtilizationCritical
instance = 172.31.40.157:9100
prometheus = observability/kube-prom-stack-kube-prome-prometheus
severity = critical
Annotations
description = CPU utilization is critically high (above 90%)
summary = Host high CPU utilization critical (instance 172.31.40.157:9100)
timestamp = time: 2024-07-14 18:08:00.796 +0000 UTC
Source

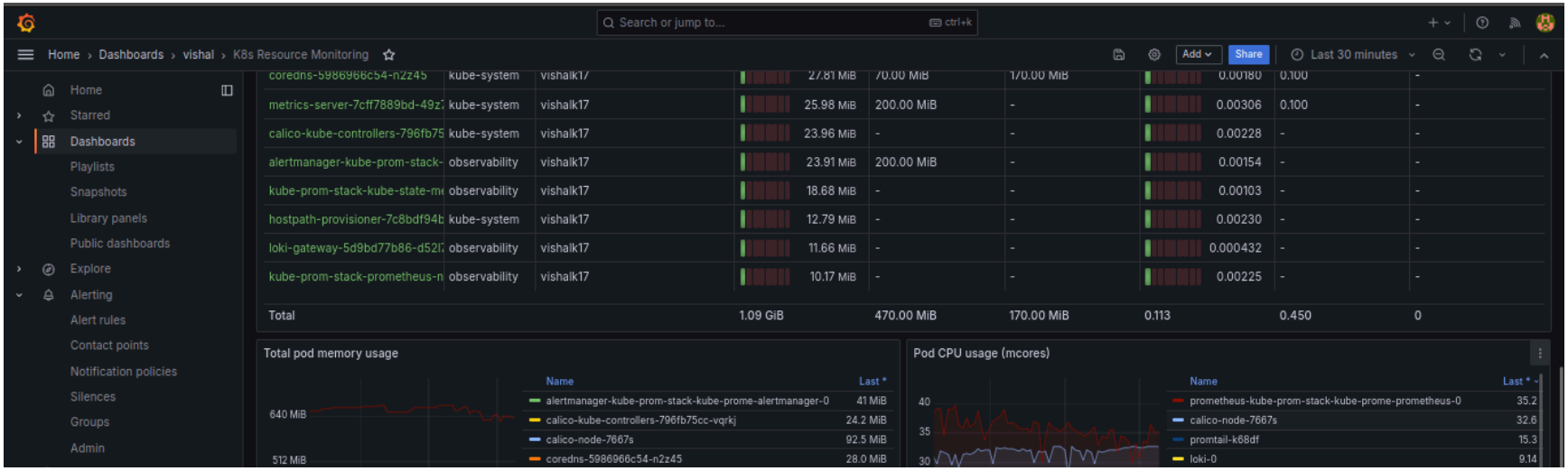
Sent by Alertmanager

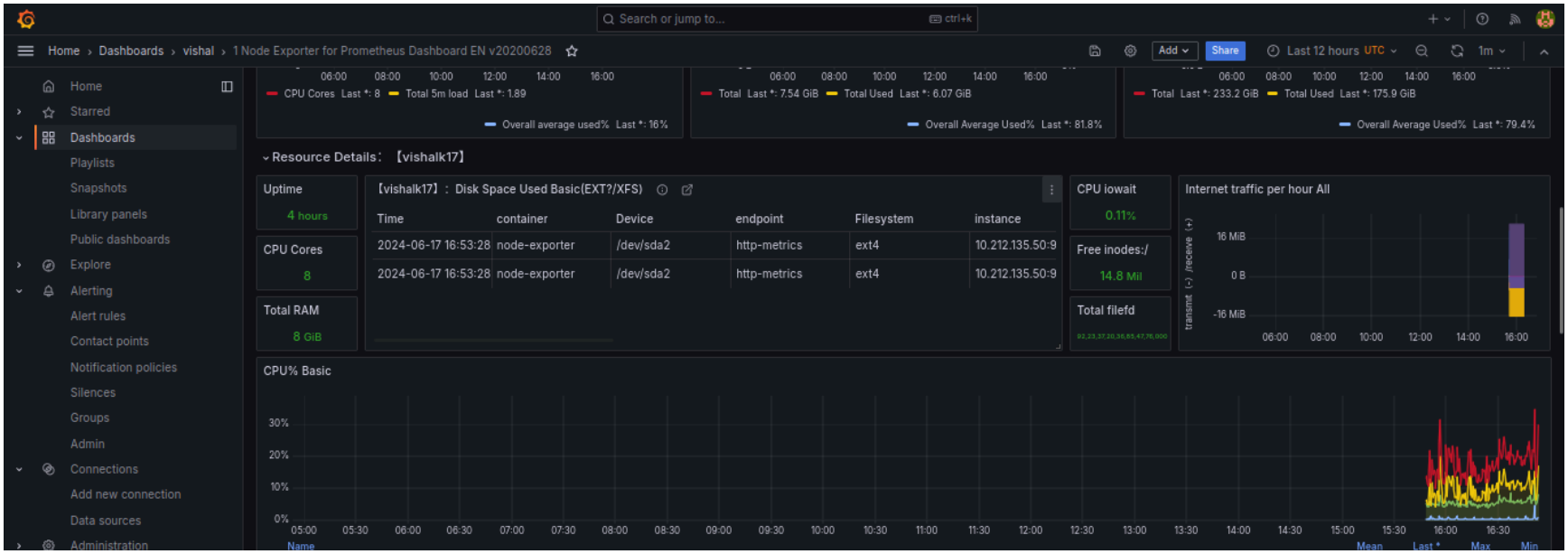

1 alert for alertname=Liveness-Probe-Failure-Or-container-restart container=loki instance=10.1.74.196:8080 pod=loki-0 prometheus=observability/kube-prom-stack-kube-prome-prometheus severity=warning
View In Alertmanager
[1] Firing
Labels
alertname = Liveness-Probe-Failure-Or-container-restart
container = loki
instance = 10.1.74.196:8080
pod = loki-0
prometheus = observability/kube-prom-stack-kube-prome-prometheus
severity = warning
Annotations
description = Liveness probe failure or Container restart occurred.
timestamp = time: 2024-07-14 18:06:40.204 +0000 UTC
Source
1 alert for alertname=Liveness-Probe-Failure-Or-container-restart container=loki instance=10.1.74.196:8080 pod=loki-0 prometheus=observability/kube-prom-stack-kube-prome-prometheus severity=warning
View In Alertmanager
[1] Resolved
Labels
alertname = Liveness-Probe-Failure-Or-container-restart
container = loki
instance = 10.1.74.196:8080
pod = loki-0
prometheus = observability/kube-prom-stack-kube-prome-prometheus
severity = warning
Annotations
description = Liveness probe failure or Container restart occurred.
timestamp = time: 2024-07-14 18:09:25.204 +0000 UTC
Source

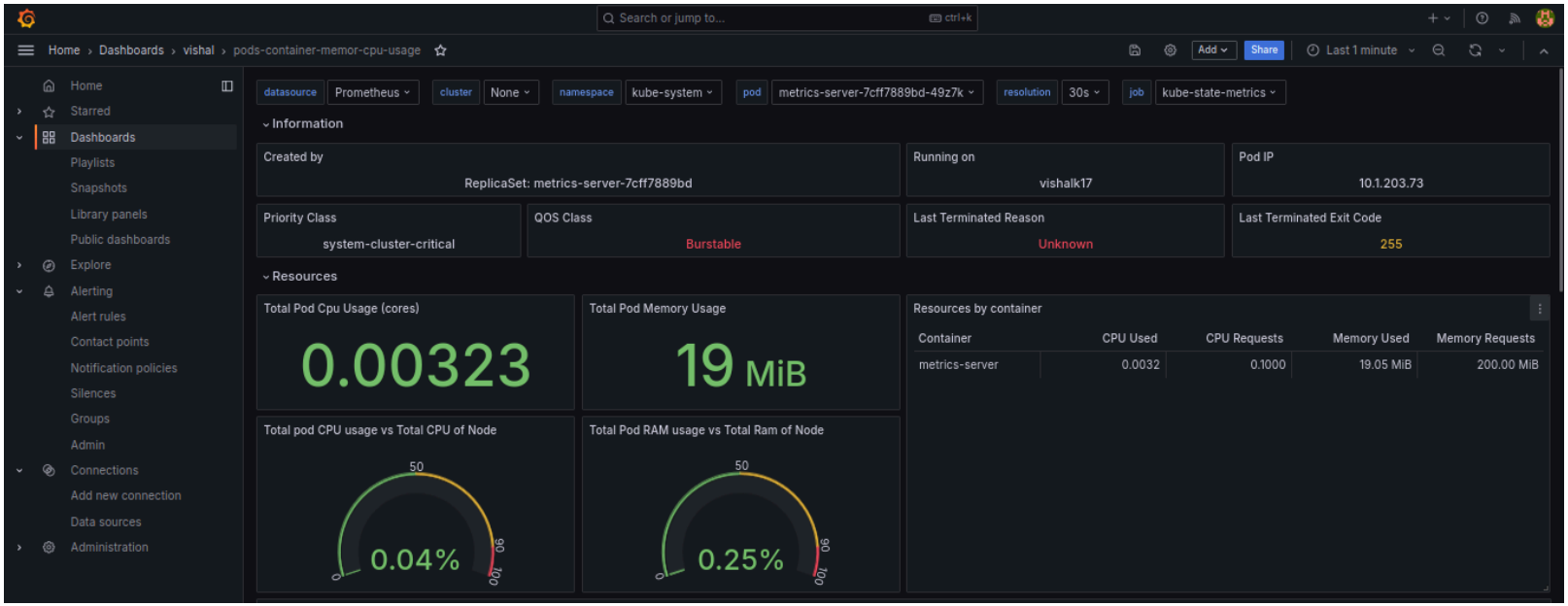

1 alert for alertname=req_received_more_than_10 container=nginx namespace=default pod=nginx-deployment-75cc7f994b-q69m2 severity=critical
View In Alertmanager
[1] Firing
Labels
alertname = req_received_more_than_10
container = nginx
namespace = default
pod = nginx-deployment-75cc7f994b-q69m2
severity = critical
Annotations
summary = Nginx Page reloaded more than 10 times
Source
Sent by Alertmanager

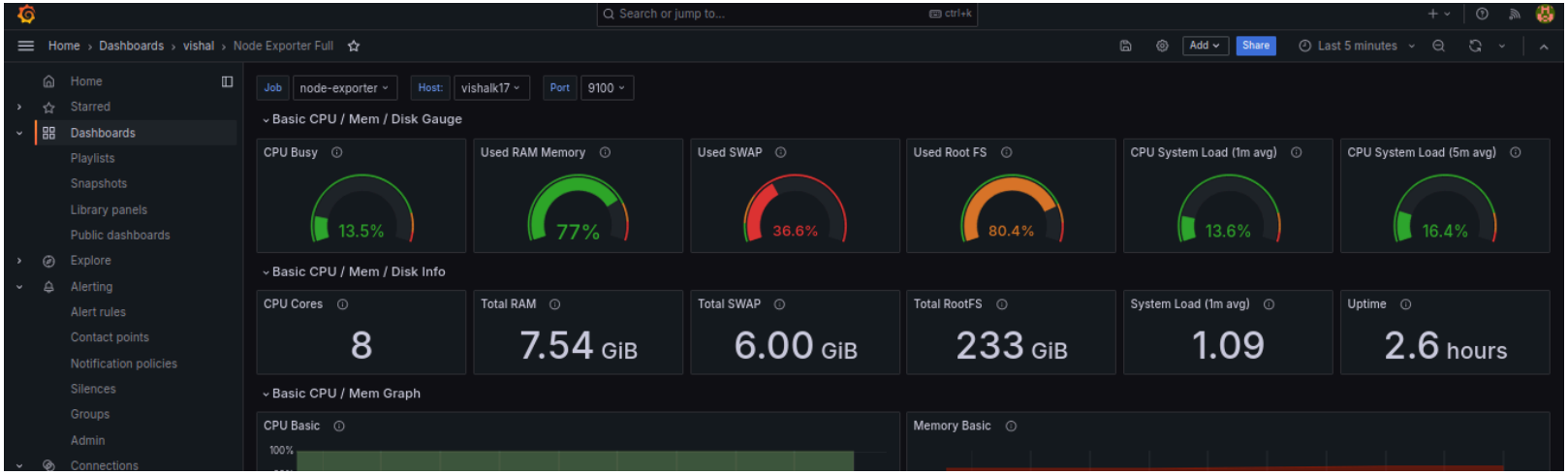

1 alert for alertname=req_received_more_than_10 container=nginx namespace=default pod=nginx-deployment-75cc7f994b-q69m2 severity=critical
View In Alertmanager
[1] Resolved
Labels
alertname = req_received_more_than_10
container = nginx
namespace = default
pod = nginx-deployment-75cc7f994b-q69m2
severity = critical
Annotations
summary = Nginx Page reloaded more than 10 times
Source
Sent by Alertmanager

1.11.0: Screenshot of Grafana Dashboards:

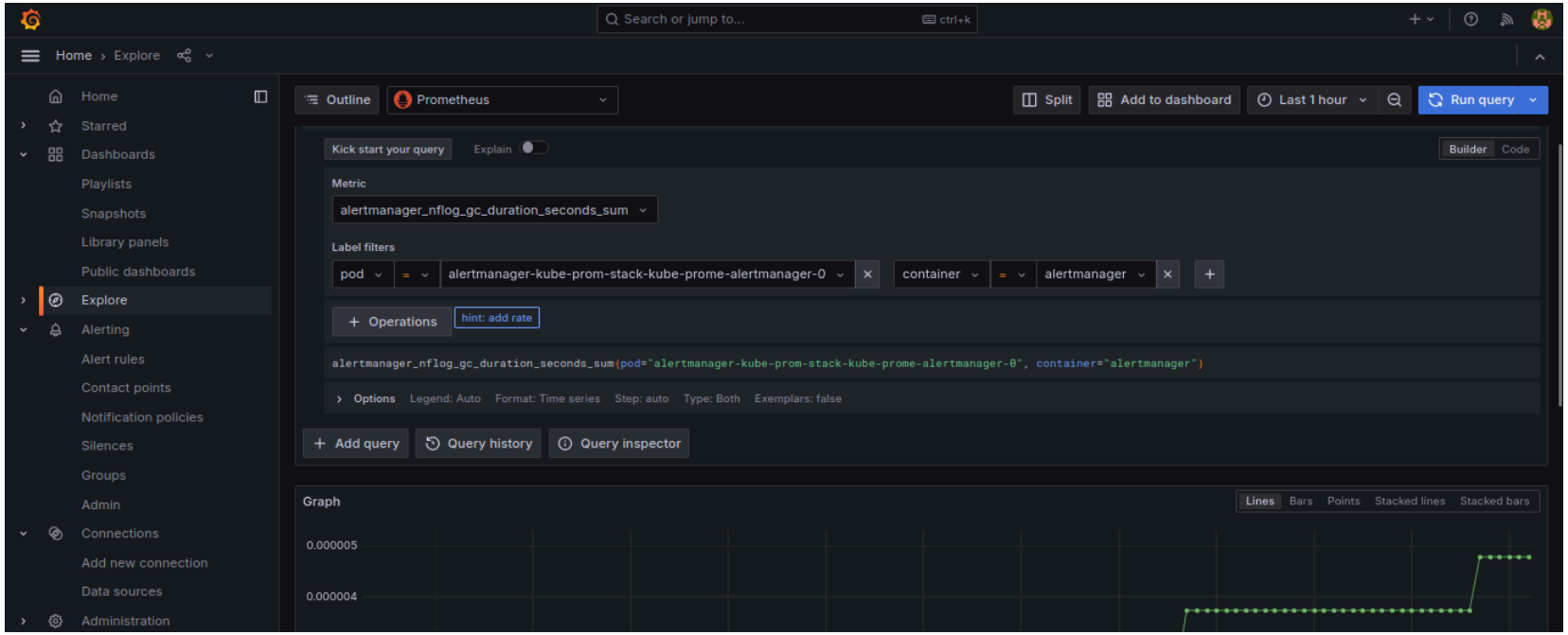




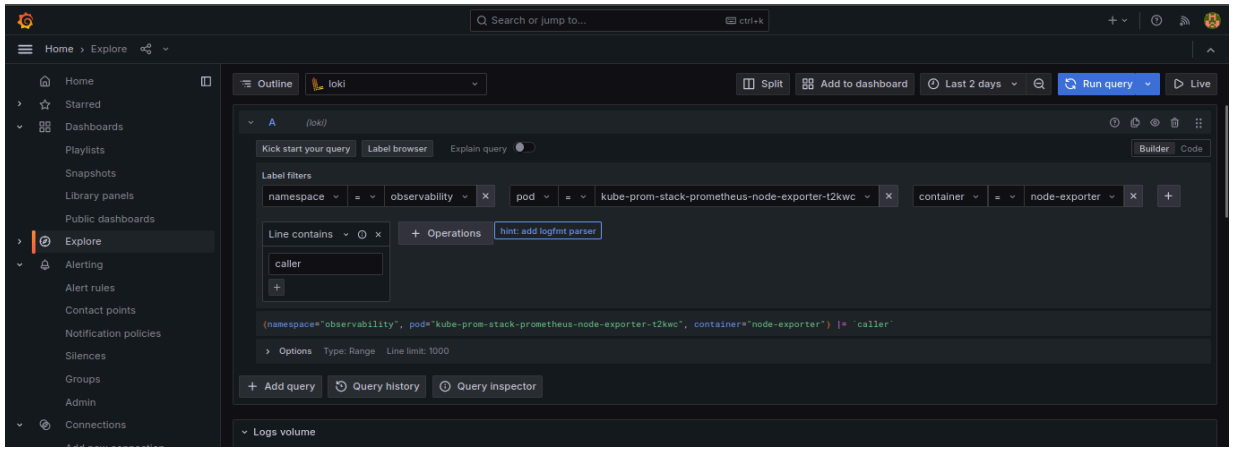

Home › Dashboards › vishal › pods-container-memor-cpu-usage
Add
Share
Last 1 minute
datasource Prometheus
cluster None
namespace kube-system
pod metrics-server-7cff7889bd-49z7k
resolution 30s
job kube-state-metrics
Information
Created by
ReplicaSet: metrics-server-7cff7889bd
Running on
vishalk17
Pod IP
10.1.203.73
Priority Class
system-cluster-critical
QOS Class
Burstable
Last Terminated Reason
Unknown
Last Terminated Exit Code
255
Resources
Total Pod Cpu Usage (cores)
0.00323
Total Pod Memory Usage
19 MiB
Resources by container
Container
CPU Used
CPU Requests
Memory Used
Memory Requests
metrics-server
0.0032
0.1000
19.05 MiB
200.00 MiB
Total pod CPU usage vs Total CPU of Node
0.04%
Total Pod RAM usage vs Total Ram of Node
0.25%

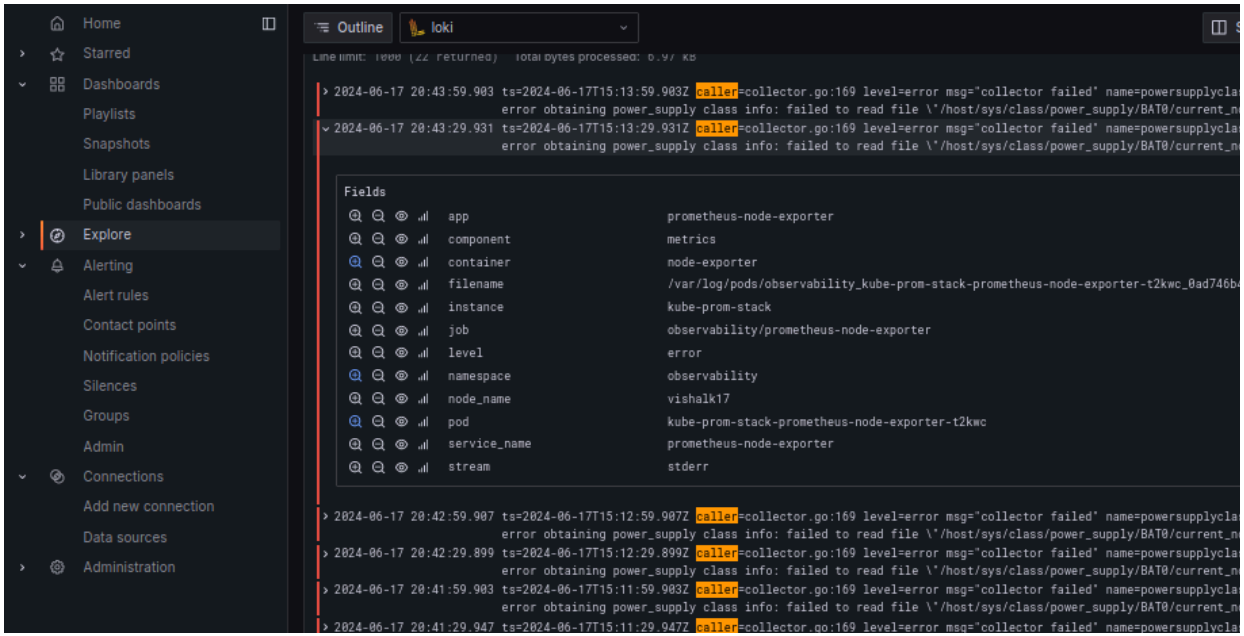

Home › Dashboards › vishal › Node Exporter Full
Add
Share
Last 5 minutes
Job node-exporter
Host: vishalk17
Port 9100
Basic CPU / Mem / Disk Gauge
CPU Busy
13.5%
Used RAM Memory
77%
Used SWAP
36.6%
Used Root FS
80.4%
CPU System Load (1m avg)
13.6%
CPU System Load (5m avg)
16.4%
Basic CPU / Mem / Disk Info
CPU Cores
8
Total RAM
7.54 GiB
Total SWAP
6.00 GiB
Total RootFS
233 GiB
System Load (1m avg)
1.09
Uptime
2.6 hours
Basic CPU / Mem Graph
CPU Basic
Memory Basic

1.12.0 : Exploring Prometheus Metrics :



1.13.0 : Exploring Logs using Loki:





Installation part has been finished.

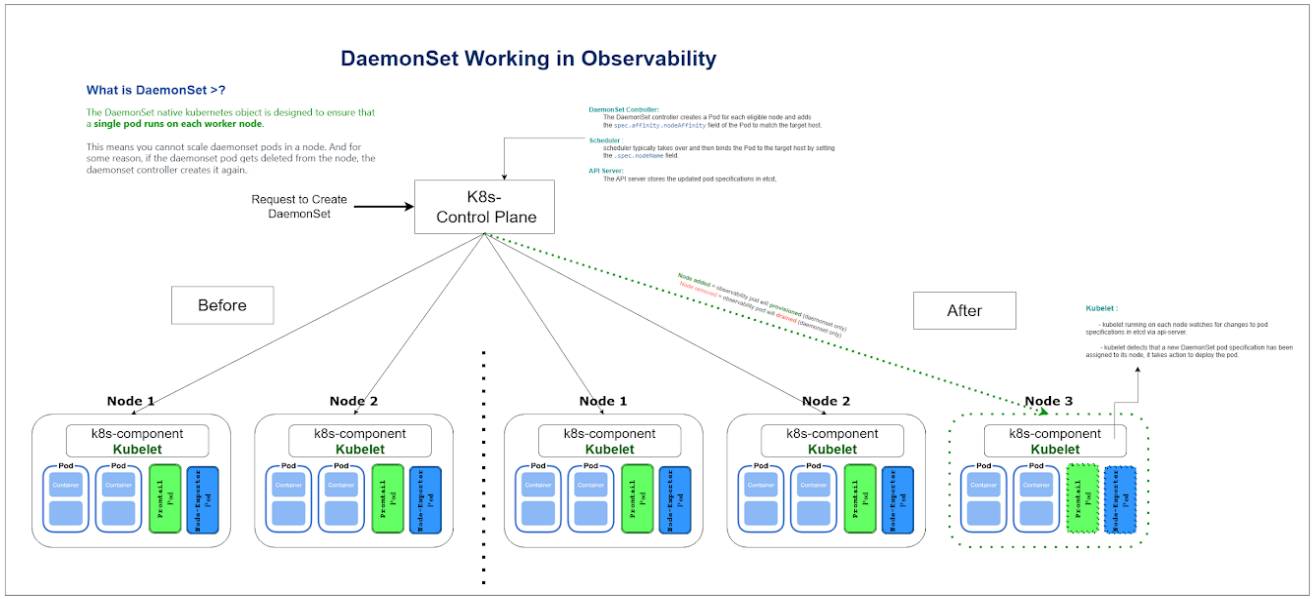
Further, is a detailed explanation.

2.0: Promtail (collects logs and sends them to Loki)

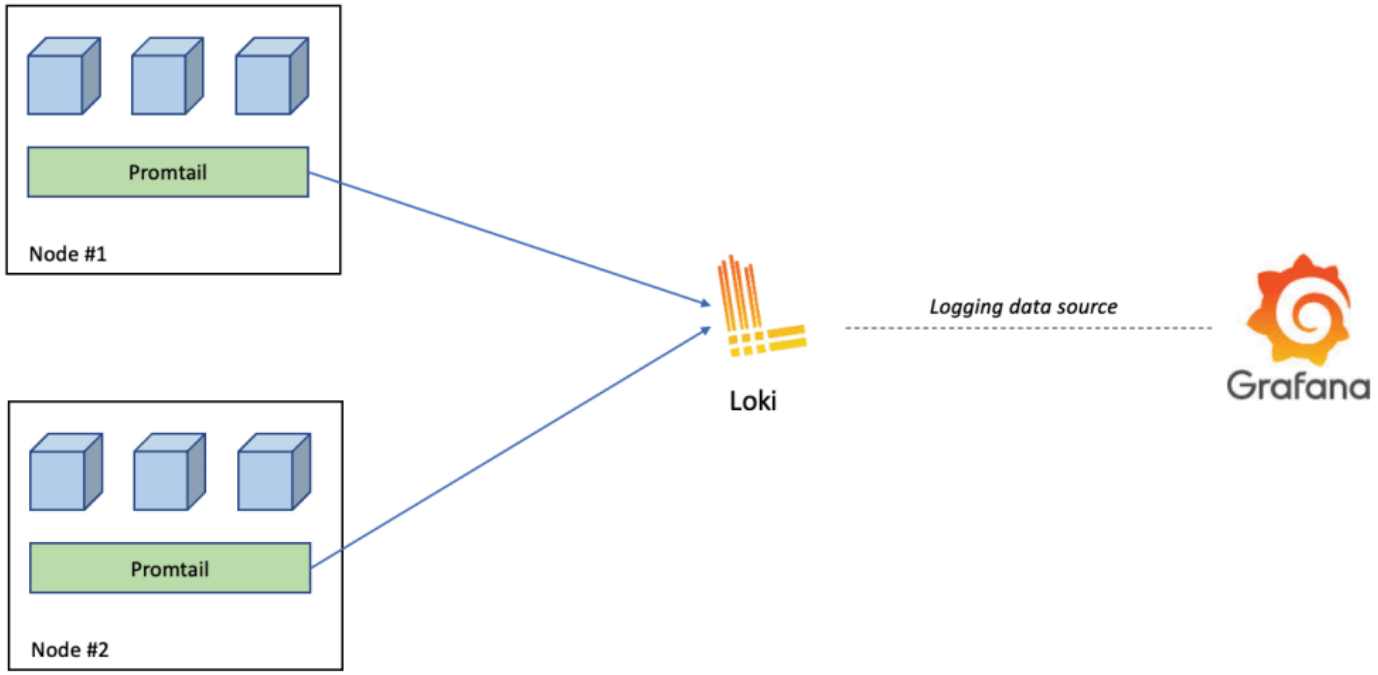
Link: https://github.com/vishalk17/observability-stack-kubernetes/blob/main/loki-obser/promtail/promtail-values.yaml
Note: Always refer latest data from the link given

In Kubernetes, a DaemonSet ensures that a copy of a pod runs on all (or some) nodes in the cluster. When it comes to deploying Promtail as a DaemonSet, this setup ensures that Promtail is running on each node,

Ref. Diagram (to know more about DaemonSet) :



2.1 : What is Promtail For?

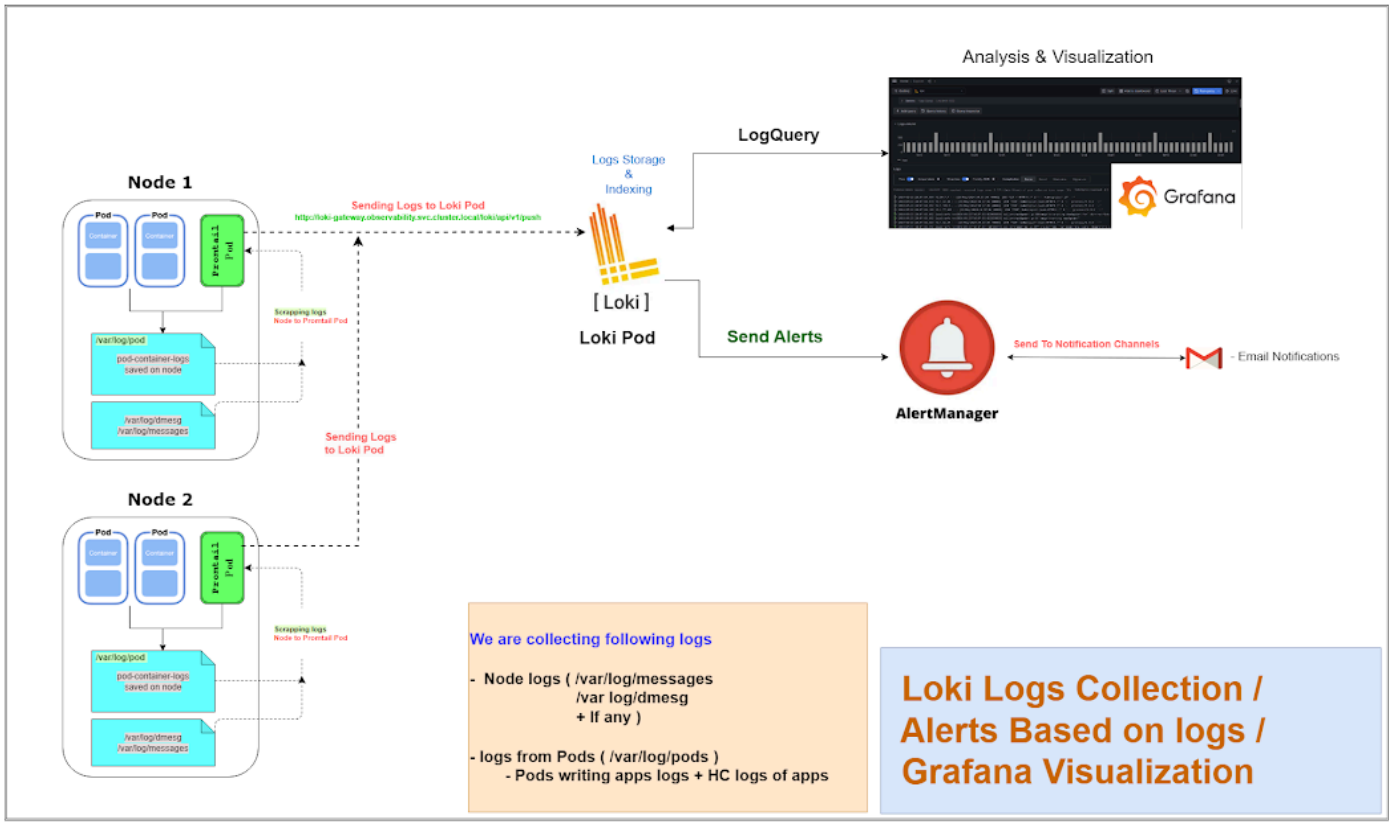


Purpose: Promtail is the log collection agent. Its job is to find and tail log files from various sources (applications, servers, containers).

How it Works:

- Discovers: Promtail automatically identifies log files based on configurations you set.
- Tails: It continuously reads the end of these log files for new entries.
- Labels: Promtail attaches labels (key-value pairs) to each log line. These labels are essential for filtering and querying later. (scrape interval : 1s)
- Ships: It sends labeled log data to the Loki server.

==========



2.2: Configure Promail to Send Host Logs:

In our setup, by Default Promtail sending all pods logs to loki. We have an additional requirement to collect the logs from /var/log/messages and /var/log/dmesg of all nodes in the Kubernetes cluster. Since Promtail is deployed using a DaemonSet, Promtail will be available on all nodes, enabling us to collect host logs as well.

Here's how we can configure Promtail to collect logs from /var/log/messages and /var/log/dmesg on each node:

1. **Update the Promtail configuration file** to include the paths for /var/log/messages and /var/log/dmesg.

Since we have installed promtail using helm chart we will override default helm values to include our changes.:

https://github.com/vishalk17/observability-stack-kubernetes/blob/main/loki-obser/promtail/promtail-values.yaml

Add below lines to above custom promtail values file. promtail-values.yaml

```
# Scrape config to read syslog file from node
config:
 snippets:
  extraScrapeConfigs: |
   # Add an additional scrape config for syslog
   - job_name: node-syslog
     static_configs:
     - targets:
       - localhost
       labels:
         log: syslog
         __path__: /var/log/host/syslog
         node_name: '${HOSTNAME}'

   - job_name: node-messages
     static_configs:
     - targets:
       - localhost
       labels:
         log: messages
         __path__: /var/log/host/messages
```

```
        node_name: '${HOSTNAME}'

    - job_name: node-dmesg
      static_configs:
      - targets:
        - localhost
        labels:
          log: dmesg
          __path__: /var/log/host/dmesg
          node_name: '${HOSTNAME}'

extraVolumes:
 - name: node-logs
   hostPath:
     path: /var/log

extraVolumeMounts:
 - name: node-logs
   mountPath: /var/log/host
   readOnly: true
```

2.3 : Install / Upgrade Promtail with Custom Value:

kubectl apply -f promtail-service.yaml
helm upgrade --install --values promtail-values.yaml promtail grafana/promtail -n observability

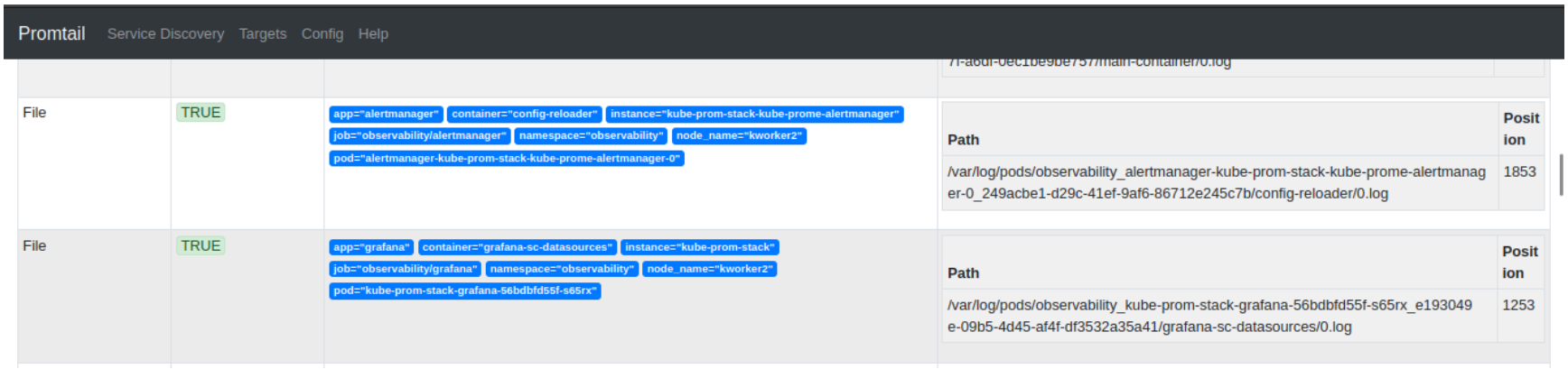
==========

```
ubuntu@kmaster:~/vishal/observability/loki-obser/promtail$ ls
promtail-service.yaml  promtail-values.yaml  Readme.md

ubuntu@kmaster:~/vishal/observability/loki-obser/promtail$ kubectl apply -f promtail-service.yaml
service/promtail-service configured

ubuntu@kmaster:~/vishal/observability/loki-obser/promtail$ helm upgrade --install --values promtail-values.yaml
promhelm upgrade --install --values promtail-values.yaml promtail grafana/promtail -n observability

Release "promtail" does not exist. Installing it now.
NAME: promtail
LAST DEPLOYED: Thu Jun 13 20:12:17 2024
NAMESPACE: observability
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
**********************************************************************
 Welcome to Grafana Promtail
 Chart version: 6.16.0
 Promtail version: 3.0.0
**********************************************************************

Verify the application is working by running these commands:
* kubectl --namespace observability port-forward daemonset/promtail 3101
* curl http://127.0.0.1:3101/metrics

ubuntu@kmaster:~/vishal/observability/loki-obser/promtail$ kubectl get pods -n observability
```

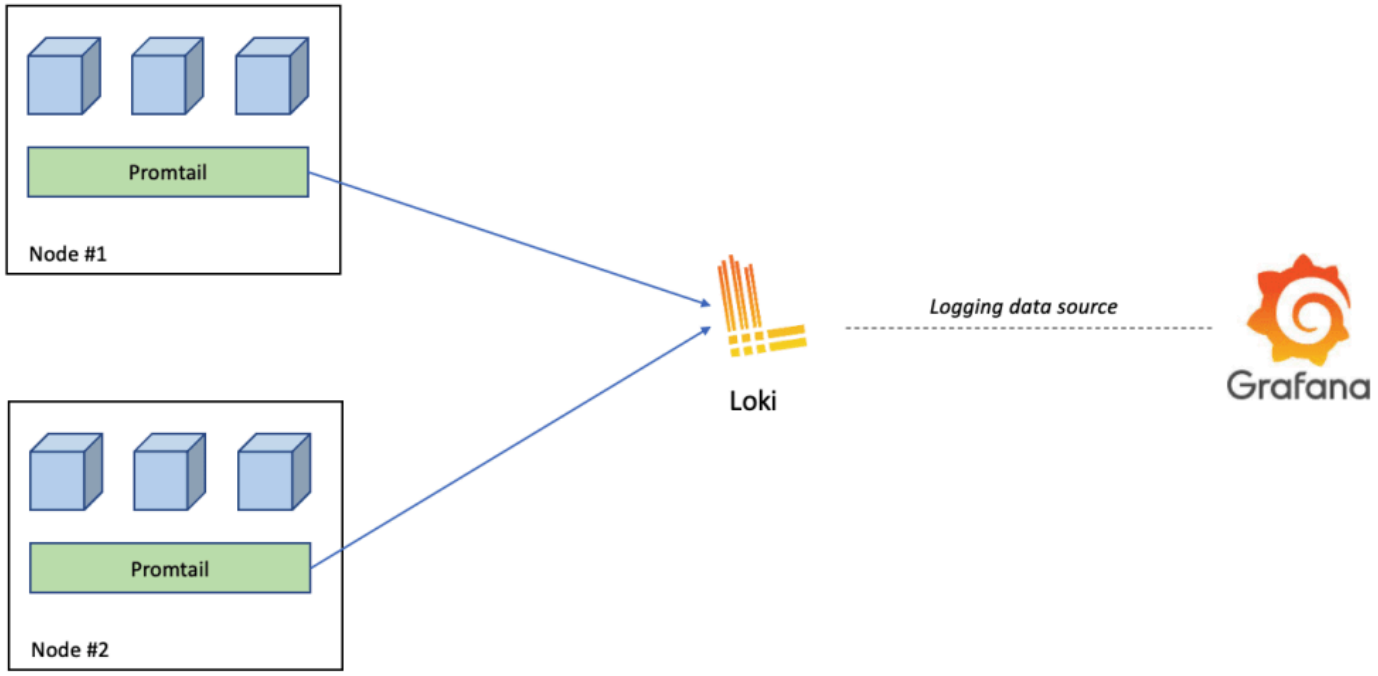
```
promtail-c66kg   1/1   Running   0   3m38s
promtail-bjqps   1/1   Running   0   3m38s
promtail-qcb4w   1/1   Running   0   3m38s
```

Promtail Web portal :

Accessible using **ip:nodeport**



3.0: Loki (Stores and Queries logs)

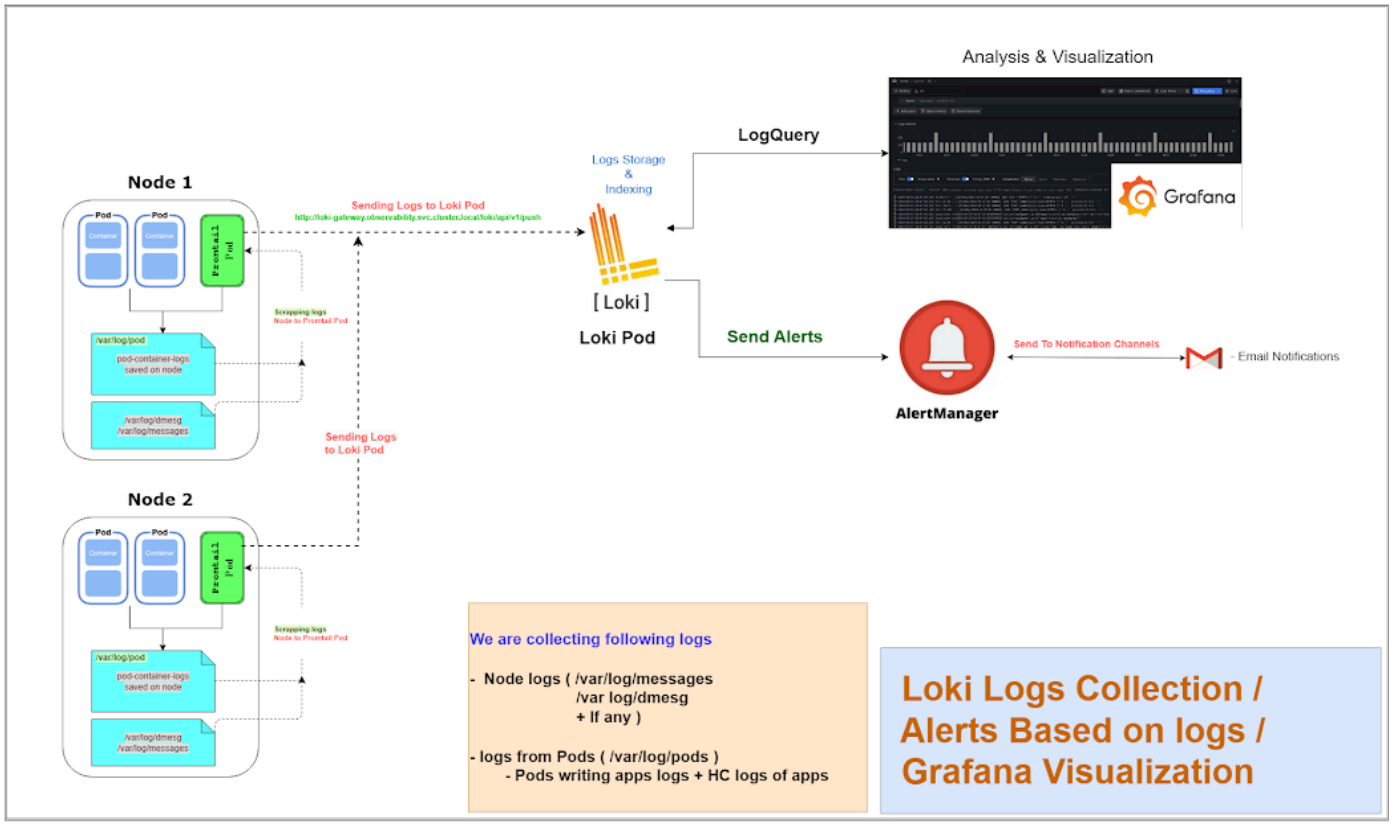


- **Purpose:** Loki is a log aggregation system. It's optimized for storing and querying log data.
- **How it Works:**
  - Receives: Loki accepts log streams from Promtail (and potentially other sources).
  - Indexes logs
  - Stores: Log data is compressed and stored efficiently.
  - Queries: Loki's query language (LogQL) allows you to filter and search through logs based on labels and time ranges.

Link: https://github.com/vishalk17/observability-stack-kubernetes/blob/main/loki-obser/loki/loki-values.yaml

Note: Always refer latest data from the link given

==========



3.1: Configs in Loki-values.yaml :

I have configured below things with ref. to official doc. And default values of this helm chart,

Urls:

```
rulerConfig:
```

```
# Base URL of the Grafana instance.

  external_url: "http://kube-prom-stack-grafana.observability.svc.cluster.local:80"  # Replace it with public url of grafana

# Comma-separated list of Alertmanager URLs to send notifications to. Each

# Alertmanager URL is treated as a separate group in the configuration. Multiple

  alertmanager_url: "http://kube-prom-stack-kube-prome-alertmanager.observability.svc.cluster.local:9093"  # Replace it with public url
```

Persistent Volume :

```
persistence:
  enableStatefulSetAutoDeletePVC: false
  enabled: true
  storageClass: microk8s-hostpath
  selector:
    matchLabels:
      type: loki-storage
```

Loki Rules labels:

```
rules:
  namespace: observability
  labels:
    loki_rule: custom
```

==========

Loki rules mounted in below dir. of container: (We will deploy them using configmap **loki-alerting-rules**)

```
extraVolumeMounts:
  - name: rules
    mountPath: /var/loki/rules/fake
extraVolumes:
  - name: rules
    configMap:
      name: loki-alerting-rules
```

How longer loki will keep data : (Currently set to 90d)

```
tableManager:

 retention_deletes_enabled: true

 retention_period: 2160h # 90 days


compactor: # which compacts index shards for performance.
  working_directory: /var/loki/compactor/retention # Directory where files can be downloaded for compaction.
  compaction_interval: 10m
  retention_enabled: true
  retention_delete_delay: 2h
  retention_delete_worker_count: 150
  delete_request_store: filesystem
limits_config:
  max_query_lookback: 2160h # 90 days
  retention_period: 2160h # 90 days
```

==========

3.2 : Configured / Update loki rules (based on logs)

If you want to add/remove rules based on logs then here you can configured loki-rules-configmap.yaml

One Example :

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: loki-alerting-rules
 namespace: observability
 labels:
   loki_rule: custom
data:
 rules.yaml: |
   groups:
     - name: test_alert
       rules:
         - alert: TestAlert
           expr: sum(rate({app="loki"} | logfmt | level="info"[1m])) by (container) > 0
           for: 1m
           labels:
             severity: warning
           annotations:
             summary: Loki2 info warning per minute rate > 0
```

After Configuring rules in a file you can update them in k8s,

```
vishal@vishalk17:~/vishal/observability-k8s-temp/loki-obser/loki$ ls
loki-pv-pvc.yaml  loki-rules-configmap.yaml  loki-values.yaml  Readme.md

vishal@vishalk17:~/vishal/observability-k8s-temp/loki-obser/loki$ kubectl apply -f loki-rules-configmap.yaml
configmap/loki-alerting-rules created
```

3.3: Installation of Loki (Monolithic - Single Binary) :

Install pv-pvc for loki first (I m using hostpath based storage)

```
kubectl apply -f loki-pv-pvc.yaml
```

Install configmap for alerting rules (this is needed before deploying loki using helm otherwise container will not create)

```
kubectl apply -f loki-rules-configmap.yaml
```

Install loki

```
helm upgrade --install --values loki-values.yaml loki grafana/loki -n observability
```

```
ubuntu@kmaster:~/vishal/observability/loki-obser/loki$ kubectl get pods -n observability
NAME                          READY   STATUS    RESTARTS      AGE
loki-gateway-854d9df4c-4g652  1/1     Running   0             23h
promtail-c66kg                1/1     Running   0             23h
promtail-bjqps                1/1     Running   0             23h
promtail-qcb4w                1/1     Running   0             23h
loki-0                        1/1     Running   0             23h
```

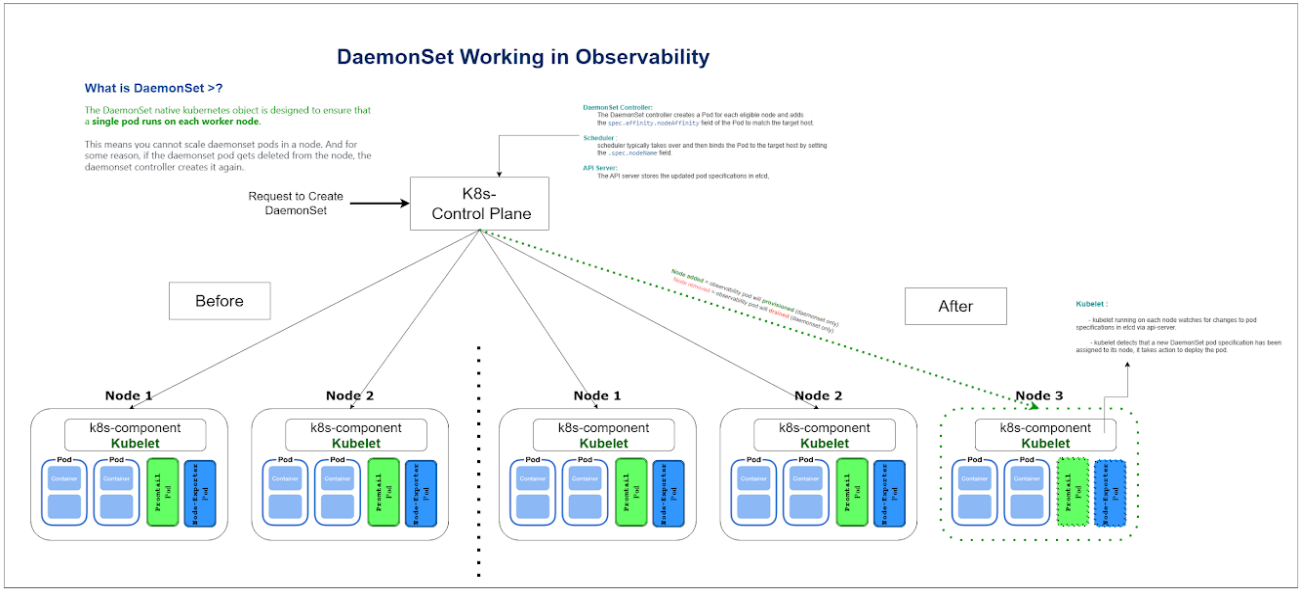
4.0 : Prometheus Node Exporter (For Node Level Metrics):

4.1 : Deployment as DaemonSet

By default, the kube-prometheus-stack Helm chart deploys the Node Exporter as a DaemonSet. This means:

- **One Pod Per Node:** A Node Exporter pod is automatically scheduled and runs on each node in your Kubernetes cluster.
- **Automatic Scaling:** If new nodes are added or removed from the cluster, the DaemonSet controller ensures that a Node Exporter pod is always present on each node.
- **High Availability:** This architecture ensures that node-level metrics are collected from every node, even if nodes fail or are replaced.

Ref. Diagram (to know more about DaemonSet) :

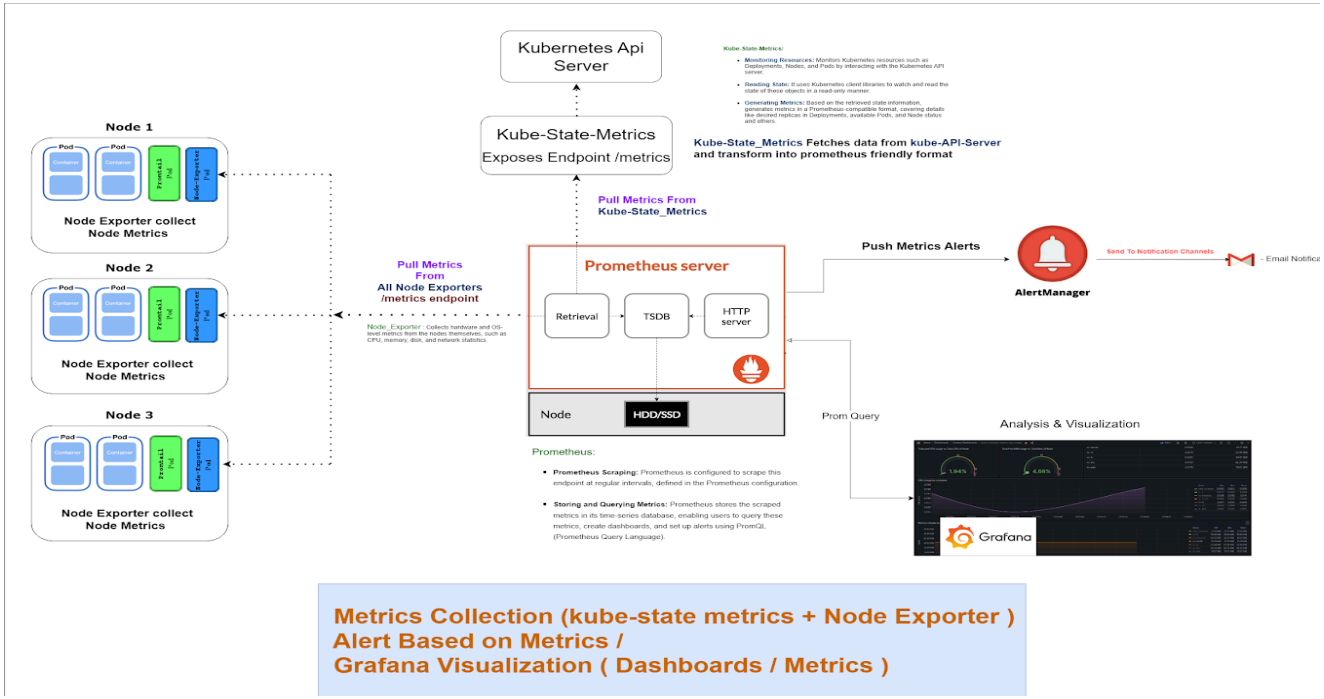


4.2 : What is the Prometheus Node Exporter?

The Node Exporter is a Prometheus component that collects hardware and operating system level metrics from the nodes within your Kubernetes cluster. These metrics include CPU usage, memory usage, disk I/O, network statistics, and more.

4.3: Why Use Node Exporter in Your Kubernetes Cluster?

- **Comprehensive Monitoring:** It provides insights into the health and performance of the underlying nodes, allowing you to identify bottlenecks and resource constraints.
- **Alerting:** Combined with Prometheus's alerting capabilities, you can set up alerts to be notified when node-level metrics reach critical thresholds.
- **Performance Optimization:** Understanding node-level resource utilization helps you make informed decisions about scaling and resource allocation.



4.4 : Installation with kube-prometheus-stack

The kube-prometheus-stack Helm chart simplifies the deployment of a complete Prometheus monitoring stack, including the Node Exporter, in your Kubernetes cluster.

5.0: AlertManager (Alerts over Email , Webhooks, Others):

Alertmanager is a component of the Prometheus ecosystem designed to handle alerts generated by Prometheus (or other monitoring systems like Loki). It provides features like:

Key Features

1. **Grouping**: Grouping of alerts of similar nature to reduce noise.
2. **Inhibition**: Suppresses notifications for alerts that are superseded by other alerts.
3. **Silencing**: Manually silencing alerts for a specified period.
4. **Routing**: Routes alerts to the correct receiver based on their labels.
5. **Receivers**: Configurable alert receivers such as email, Slack, and custom webhooks.

5.1: Installation with kube-prometheus-stack

The `kube-prometheus-stack` Helm chart includes the Alertmanager and automatically integrates it with Prometheus.

5.2: Alertmanager Configuration

Ref. latest config file from here : (Ref. alertmanager.yaml)

In our case, I have configured AlertManager only for email alerts. This configuration typically involves setting up SMTP details and specifying email recipients.

Note: AlertManager currently doesn't support Google Chat webhook integration.

This is an example of an alertmanager configuration file,

```
# Modified by github.com/vishalk17
# Global configuration
global:
```

==========

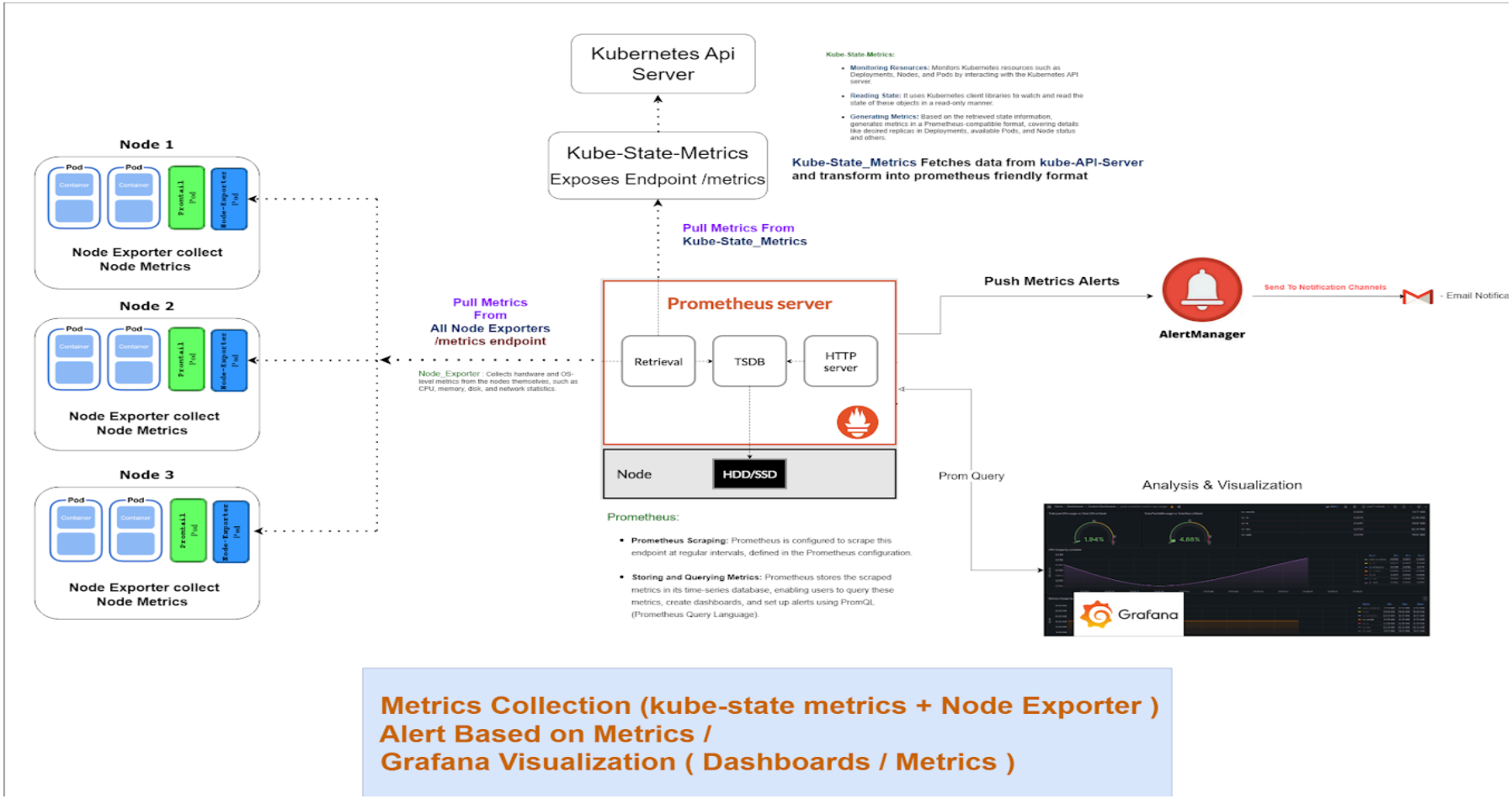
```
  resolve_timeout: 1m

# Routing configuration
route:
  # Grouping criteria for alerts
  group_by: ['...']   # '...' by adding this we can disable grouping of alerts in single notification
                      #  this will avoid messup of notifications of alerts within single notification.
  # Wait time before sending initial notification for a group of alerts
  group_wait: 1s
  # Interval between sending notifications for the same group of alerts
  group_interval: 1s
  # Interval between resending alerts that have been successfully sent
  repeat_interval: 1h
  # Receiver to send notifications to
  receiver: 'email-alert'
  # Routes for matching alerts
  routes:
    - match:
        severity: 'critical|warning'

# Receiver configuration
receivers:
  # Google Chat receiver
  - name: 'Google Chat'
    # Webhook notification configuration
    webhook_configs:
      - url: 'https://chat.googleapis.com/v1/spaces/AAAAVrkKI3M/messages?key=vishalk17'
        send_resolved: true

  # Email receiver
  - name: 'email-alert'
    # Email notification configuration
    email_configs:
```

```
      # To specify multiple email addresses, separate them with commas
      # Example: 'to: 'email1@example.com, email2@example.com''
      - to: 'vishal.kapadi@vishalk17.com, hello_world@vishalk17.com'
        from: 'k8s-notifications <vishalk17@vishalk17.com>'
        smarthost: smtp.vishalk17.net:587
        auth_username: k8s-notify@vishalk17.com
        auth_identity: Alertmanger-k8s
        # SMTP password (replace with your actual password)
        auth_password: **********
        # Send resolved alerts as well
        send_resolved: true
        require_tls: true # false if your SMTP does not support tls
        headers:
          subject:  '[Status: {{ .Status | toUpper }}] - [{{ .Status | toUpper }}{{ if eq .Status "firing" }}:{{
.Alerts.Firing | len }}{{ end }}] - [{{ .CommonLabels.severity }}] - AlertName: {{ .CommonLabels.alertname }} -
Pod: {{ .CommonLabels.pod }} - Container: {{ .CommonLabels.container }}'
```

In the above config file, we are configuring various things like alerting labels, the endpoints where alerts should go; in our case, it's email alerts. For email alerts, we have configured SMTP credentials.

6.0 Prometheus (Pod Level Metrics):

Prometheus is an open-source systems monitoring and alerting toolkit originally developed at SoundCloud. Since its inception in 2012, it has become a widely adopted monitoring tool, particularly in the cloud-native ecosystem. Prometheus is known for its powerful querying language (PromQL), multidimensional data model, and robust alerting capabilities.



Metrics Collection (kube-state metrics + Node Exporter)
Alert Based on Metrics /
Grafana Visualization (Dashboards / Metrics)

Key Features

1. **Time Series Data**: Prometheus stores all data as time series, meaning that it records metrics over time, enabling efficient analysis and visualization of trends.
2. **Flexible Query Language (PromQL)**: PromQL allows users to select and aggregate time series data in real-time.
3. **Dimensional Data Model**: Data is identified by metric names and key-value pairs, providing a rich context for monitoring.
4. **Efficient Storage**: Prometheus is designed for high performance and storage efficiency, capable of handling large volumes of metrics.
5. **Alerting**: Prometheus can generate alerts based on query results, which are then sent to AlertManager for handling.
6. **Service Discovery**: Automatically discovers targets to monitor through various service discovery mechanisms such as Kubernetes, Consul, and more.
7. **Visualization**: Can be integrated with Grafana for powerful visualizations of the collected metrics.

6.1: Installation Using kube-prom-stack

The `kube-prom-stack` is a Helm chart - that deploys Prometheus along with other related monitoring tools like AlertManager, Grafana, and node-exporters in a Kubernetes cluster. This stack simplifies the deployment and management of a complete monitoring solution.

6.2: Custom values.yaml for installing using helm chart.

Ref. latest file from here : grafana-values.yaml

It Includes,

- ➢ Scrap interval
- ➢ We are setting External URL for prometheus
- ➢ How long to retain metrics (90d in our case)
- ➢ Persistent Storage for prometheus

```
prometheus:
 prometheusSpec:   # replace with acutal external prometheus url
   externalUrl: http://kube-prom-stack-kube-prome-alertmanager.observability.svc.cluster.local:9093/
   scrapeTimeout: 30s          # Default to 30s
   scrapeInterval: 30s         # Default to 30s # Below 30s , It wont work
   evaluationInterval: 15s     # Default to 30s
   ## How long to retain metrics
   retention: 90d
   # Rules yaml has to match with this label in order to take those into account
   ruleSelector:
     matchLabels:
       resource: prom-custom

   storageSpec:
     volumeClaimTemplate:
       spec:
         storageClassName: microk8s-hostpath
         accessModes: ["ReadWriteOnce"]
         resources:
           requests:
             storage: 500Gi
         selector:
           matchLabels:
             type: prometheus-storage
```

6.3 : Installation of recording rules:

6.3.1: Overview

In Prometheus, recording rules are a way to pre-calculate and store the results of frequently used or computationally expensive PromQL (Prometheus Query Language) expressions as new time series. These pre-calculated results can then be queried much faster than evaluating the original expressions repeatedly.

Key Benefits:

- **Improved Query Performance:** Recording rules avoid redundant calculations, leading to significant performance improvements for complex or frequently executed queries.
- **Reduced Resource Usage:** By offloading the computation to recording rules, you can reduce the load on the Prometheus server during query time.
- **Simplified Queries:** Recording rules can act as a layer of abstraction, allowing you to create simpler queries that reference the pre-calculated results.

How Recording Rules Work:

1. **Definition:** You define recording rules in YAML files, specifying the new metric name to store the results under and the PromQL expression to evaluate.
2. **Evaluation:** Prometheus evaluates the expressions in the recording rules at regular intervals (usually the same as the scrape interval).
3. **Storage:** The results of the evaluations are stored as new time series in Prometheus.
4. **Querying:** You can then query the pre-calculated results using the new metric names.

6.3.2: Installation of Recording rules

You can find recording-rules here. Recording-rules-only << Click Here

Note that we are going to deploy them using the prometheus operator. As you can see a simple example:

observability-k8s-temp / rules / recording-rules-only / **kubePrometheus-prometheusRule.yaml**

vishalk17 recording rules: watch for 1m

Code | Blame | 29 lines (28 loc) · 1.3 KB | Code 55% faster with GitHub Copilot

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  labels:
    resource: prom-custom
  name: kube-prometheus-rules
spec:
  groups:
  - name: kube-node-recording.rules
    rules:
    - expr: sum(rate(node_cpu_seconds_total{mode!="idle",mode!="iowait",mode!="steal"}[3m])) BY (instance)
      record: instance:node_cpu:rate:sum
    - expr: sum(rate(node_network_receive_bytes_total[3m])) BY (instance)
      record: instance:node_network_receive_bytes:rate:sum
    - expr: sum(rate(node_network_transmit_bytes_total[3m])) BY (instance)
      record: instance:node_network_transmit_bytes:rate:sum
    - expr: sum(rate(node_cpu_seconds_total{mode!="idle",mode!="iowait",mode!="steal"}[1m])) WITHOUT (cpu,
      record: instance:node_cpu:ratio
    - expr: sum(rate(node_cpu_seconds_total{mode!="idle",mode!="iowait",mode!="steal"}[1m]))
      record: cluster:node_cpu:sum_rate1m
    - expr: cluster:node_cpu:sum_rate1m / count(sum(node_cpu_seconds_total) BY (instance, cpu))
      record: cluster:node_cpu:ratio
```

You can simply install them as a regular manifest file in kubernetes.

```
vishal@vishalk17:~/vishal/observability-k8s-temp/rules/recording-rules-only$ ls -la
total 52
drwxrwxr-x 2 vishal vishal  4096 Jun 14 02:04 .
drwxrwxr-x 3 vishal vishal  4096 Jun 14 02:04 ..
-rw-rw-r-- 1 vishal vishal  1331 Jun 14 02:04 kubePrometheus-prometheusRule.yaml
-rw-rw-r-- 1 vishal vishal 34739 Jun 14 02:04 kubernetesControlPlane-prometheusRule.yaml
```

```
-rw-rw-r-- 1 vishal vishal  2806 Jun 14 02:04 nodeExporter-prometheusRule.yaml
vishal@vishalk17:~/vishal/observability-k8s-temp/rules/recording-rules-only$ kubectl apply -f ./
prometheusrule.monitoring.coreos.com/kube-prometheus-rules created
prometheusrule.monitoring.coreos.com/kubernetes-monitoring-rules created
prometheusrule.monitoring.coreos.com/node-exporter-recording-rules created
```

6.4 : Add / Remove / Update Prometheus Alerting Rules (based off metrics) :

Alerting rules in Prometheus are a way to define conditions that trigger alerts based on the values of metrics. These rules are written using PromQL (Prometheus Query Language) expressions and specify a threshold for when an alert should be fired.

6.4.1 : Overview :

Alerting rules in Prometheus are a way to define conditions that trigger alerts based on the values of metrics. These rules are written using PromQL (Prometheus Query Language) expressions and specify a threshold for when an alert should be fired.

Key Components:

- **Alert Name:** A unique identifier for the alert.
- **Expression:** A PromQL expression that evaluates to a set of time series and their values.
- **Threshold:** A condition that, when met by the expression's results, triggers the alert.
- **For Duration:** How long a condition must hold true before an alert is considered firing.
- **Labels:** Key-value pairs that can be attached to the alert for filtering and organization (e.g., severity, team).
- **Annotations:** Additional information that can be included in the alert notification (e.g., summary, description).

How Alerting Rules Work:

1. **Evaluation:** Prometheus periodically evaluates the expressions in alerting rules against the stored time series data.
2. **Triggering:** If an expression result meets the threshold condition for the specified duration, an alert is triggered.
3. **Notification:** The triggered alert is sent to the Alertmanager, which handles grouping, routing, and silencing of alerts.
4. **Resolution:** When the condition that triggered the alert no longer holds true, the alert is resolved.

6.4.2 : Add / update Prometheus Alerting rules:

You can find alering rules here **rules** << click here

You can edit those files and update existing ones. Note that we are deploying prometheus rules using prometheus operator and an example Structure as follows. For new rules to be added or update existing ones.

observability-k8s-temp / rules / **node-rule-crd.yaml**

vishalk17 update node rules

Code | Blame | 57 lines (54 loc) · 2.35 KB | Code 55% faster with GitHub Copilot

```
1   apiVersion: monitoring.coreos.com/v1
2   kind: PrometheusRule
3   metadata:
4     name: resource-monitor-node-rules
5     labels:
6       resource: prom-custom
7   spec:
8     groups:
9     - name: resource-monitor-node
10      rules:
11      - alert: HostHighCpuUtilizationWarning
12        expr: (100 - (avg by(instance) (rate(node_cpu_seconds_total{mode="idle"}[2m])) * 100)) > 80
13        for: 3m
14        labels:
15          severity: warning
16        annotations:
17          timestamp: >
18            time: {{ with query "time()" }}{{ . | first | value | humanizeTimestamp }}{{ end }}
19          summary: Host high CPU utilization warning (instance {{ $labels.instance }})
20          description: "CPU utilization is above 80%\n"
21        # Triggered when CPU utilization is above 80%.
22
```

Apply rules in k8s cluster :

```
vishal@vishalk17:~/vishal/observability-k8s-temp/rules$ ls -la
total 32
drwxrwxr-x 3 vishal vishal 4096 Jun 14 02:04 .
drwxrwxr-x 5 vishal vishal 4096 Jun 14 02:04 ..
-rw-rw-r-- 1 vishal vishal 2409 Jun 14 02:04 node-rule-crd.yaml
-rw-rw-r-- 1 vishal vishal 7942 Jun 14 02:04 pod-rule-crd.yaml
-rw-rw-r-- 1 vishal vishal 1280 Jun 14 02:04 probes-rules.yaml
-rw-rw-r-- 1 vishal vishal   94 Jun 14 02:04 README.md
drwxrwxr-x 2 vishal vishal 4096 Jun 14 02:04 recording-rules-only
vishal@vishalk17:~/vishal/observability-k8s-temp/rules$ kubectl apply -f node-rule-crd.yaml
prometheusrule.monitoring.coreos.com/resource-monitor-node-rules created

Apply others as well
```

7.0 : Grafana (Visualization) :

Grafana is a powerful open-source platform for monitoring and observability, allowing users to query, visualize, and alert on metrics and logs from a variety of data sources. In the context of the kube-prom-stack, Grafana serves as the visualization layer for metrics collected by Prometheus and logs collected by Loki.

7.1 : Grafana Installation and Configuration :

The `kube-prometheus-stack` Helm chart includes Grafana by default, simplifying the installation process. By installing this helm chart it will install grafana with it.

7.2 : How to fix smtp in Grafana

Note 1 : Since we are using Alert Manager which is simplifying our alerting system, we will likely not need to use this grafana alerting. However, I have figured out how to resolve it, so I am putting down the solution in case future demands require us to deal with it.

Note 2 : Other values (SMTP) we are passing in grafana-values.yaml while installing this chart.

Step A :
Below command add password line just below user line:

```
kubectl get cm kube-prom-stack-grafana -n observability -o yaml | \
  sed -e '/^ *password *=/d' \
      -e '/^ *user *= */a \ \ \ \ password = your-pass | \
  kubectl apply -f -
```

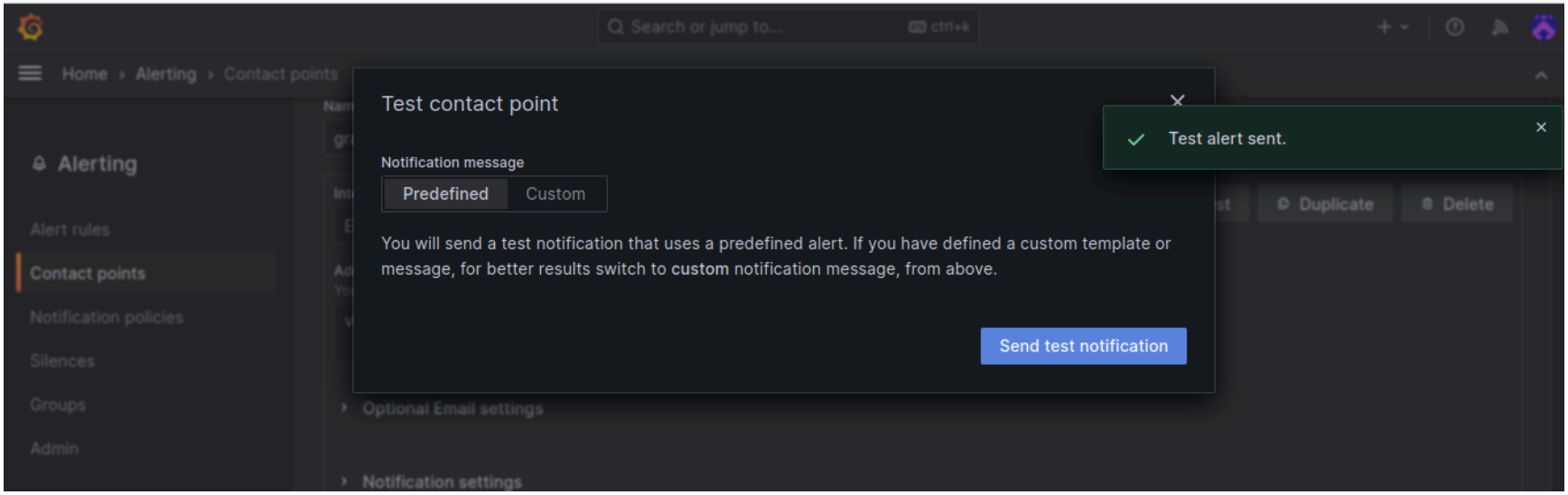
This sed command does the following:

1. /^ *password *=/d: It searches for lines starting with "password =", and if found, deletes the entire line.
2. /^ *user *= */a \ \ \ \ password = your_pass: After deleting the existing password line (if it exists), it searches for lines starting with "user =", and appends the password line just below it.

This command ensures that if the password line already exists, it will be deleted first and then added below the user line. If it doesn't exist, it will simply be added below the user line.

Step B : Restart deployment of grafana to include changes done to cm

```
kubectl rollout restart deployment  kube-prom-stack-grafana -n observability
```



7.3.0 : Grafana DashBoards :

Grafana itself provides various dashboards for Kubernetes. We just need to use them for our purposes or create new ones suitable for our case.

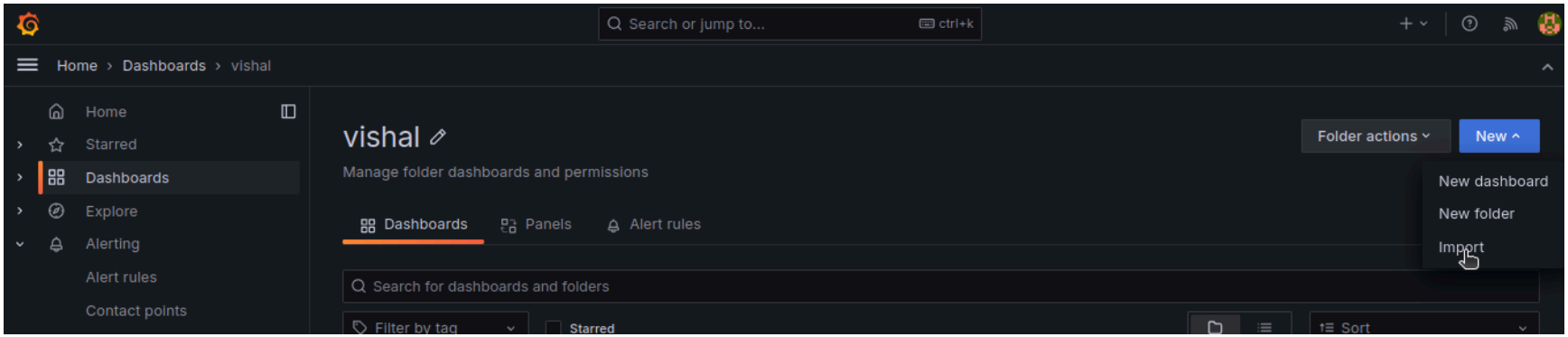
7.3.1 : How to Import Grafana DashBoards :

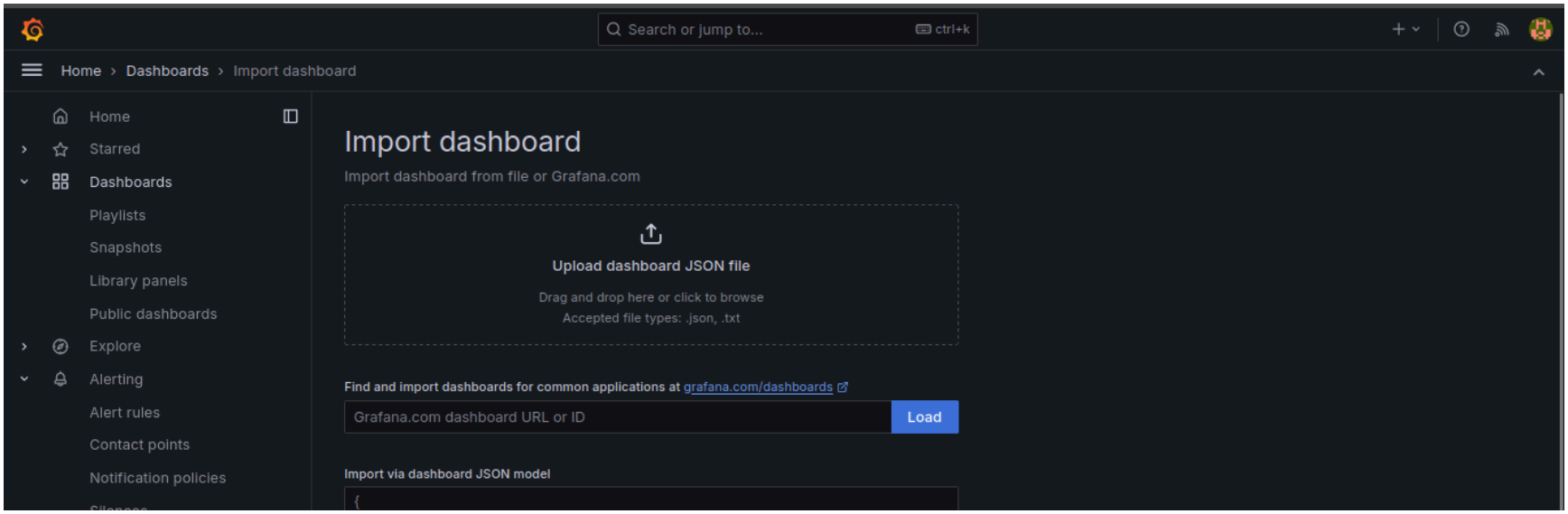
1. Importing from Grafana.com:

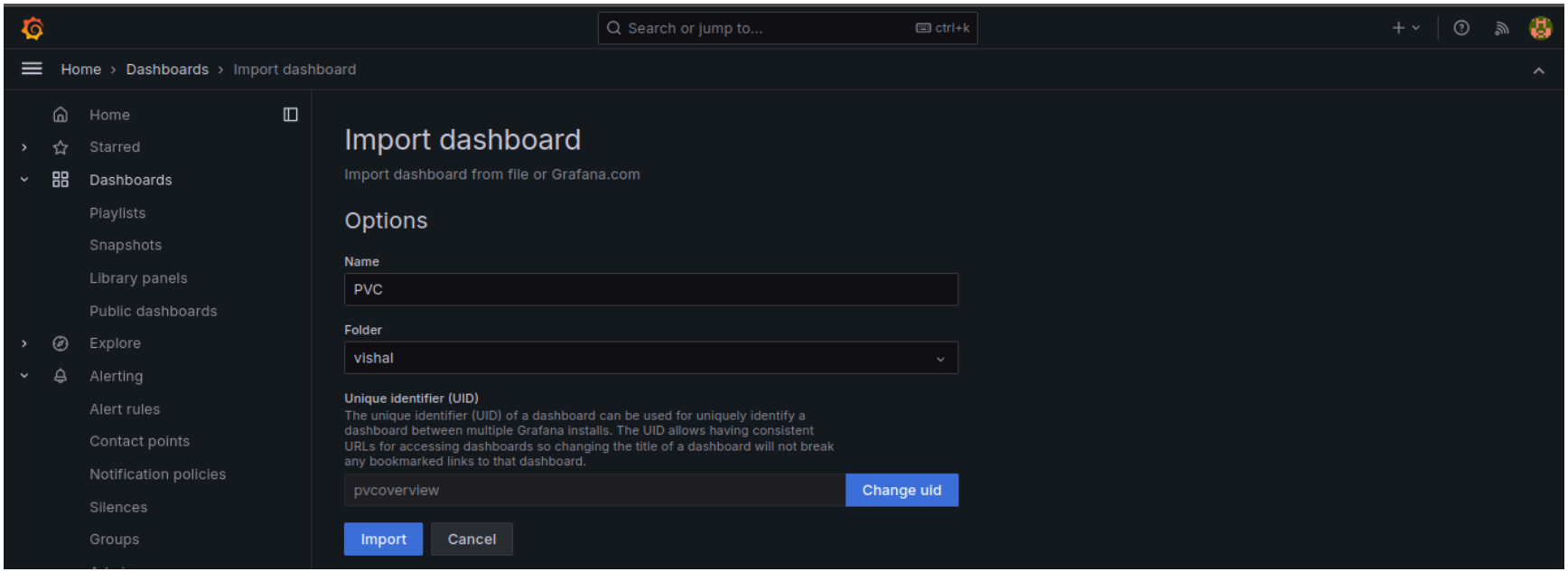
- **Dashboard ID:**
    1. Go to the Grafana dashboard page you want to import.
    2. Find the dashboard's unique ID in the URL (e.g., `https://grafana.com/grafana/dashboards/12345`).
    3. In your Grafana instance, go to "Dashboards" -> "New" -> "Import".
    4. Paste the dashboard ID and click "Load."
    5. Select the appropriate data source and click "Import."

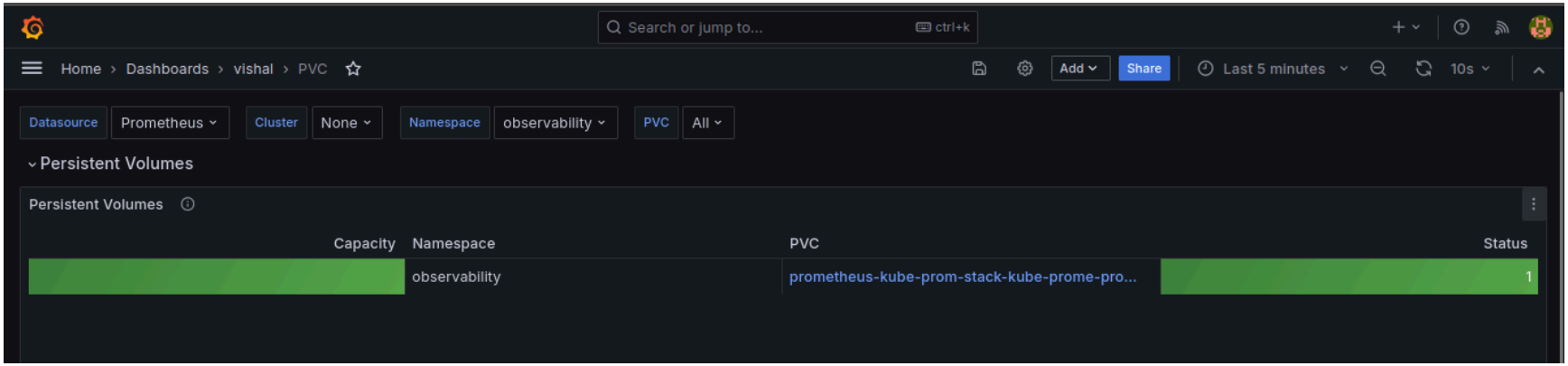
2. Importing from a JSON File:

- **Download JSON:**
    1. Download the dashboard JSON file from a source like Grafana Labs, community repositories, or your own exports.
- **Import in Grafana:**
    1. In your Grafana instance, go to "Dashboards" -> "New" -> "Import."
    2. Click on the "Upload JSON file" button and select the downloaded file.
    3. Configure options like dashboard name and folder.
    4. Select the appropriate data source and click "Import."


Search or jump to... ctrl+k
Home › Dashboards › vishal
Home
Starred
Dashboards
Explore
Alerting
Alert rules
Contact points
vishal
Manage folder dashboards and permissions
Folder actions
New
New dashboard
New folder
Import
Dashboards
Panels
Alert rules
Search for dashboards and folders
Filter by tag
Starred
Sort


Search or jump to... ctrl+k
Home › Dashboards › Import dashboard
Home
Starred
Dashboards
Playlists
Snapshots
Library panels
Public dashboards
Explore
Alerting
Alert rules
Contact points
Notification policies
Import dashboard
Import dashboard from file or Grafana.com
Upload dashboard JSON file
Drag and drop here or click to browse
Accepted file types: .json, .txt
Find and import dashboards for common applications at grafana.com/dashboards
Grafana.com dashboard URL or ID
Load
Import via dashboard JSON model

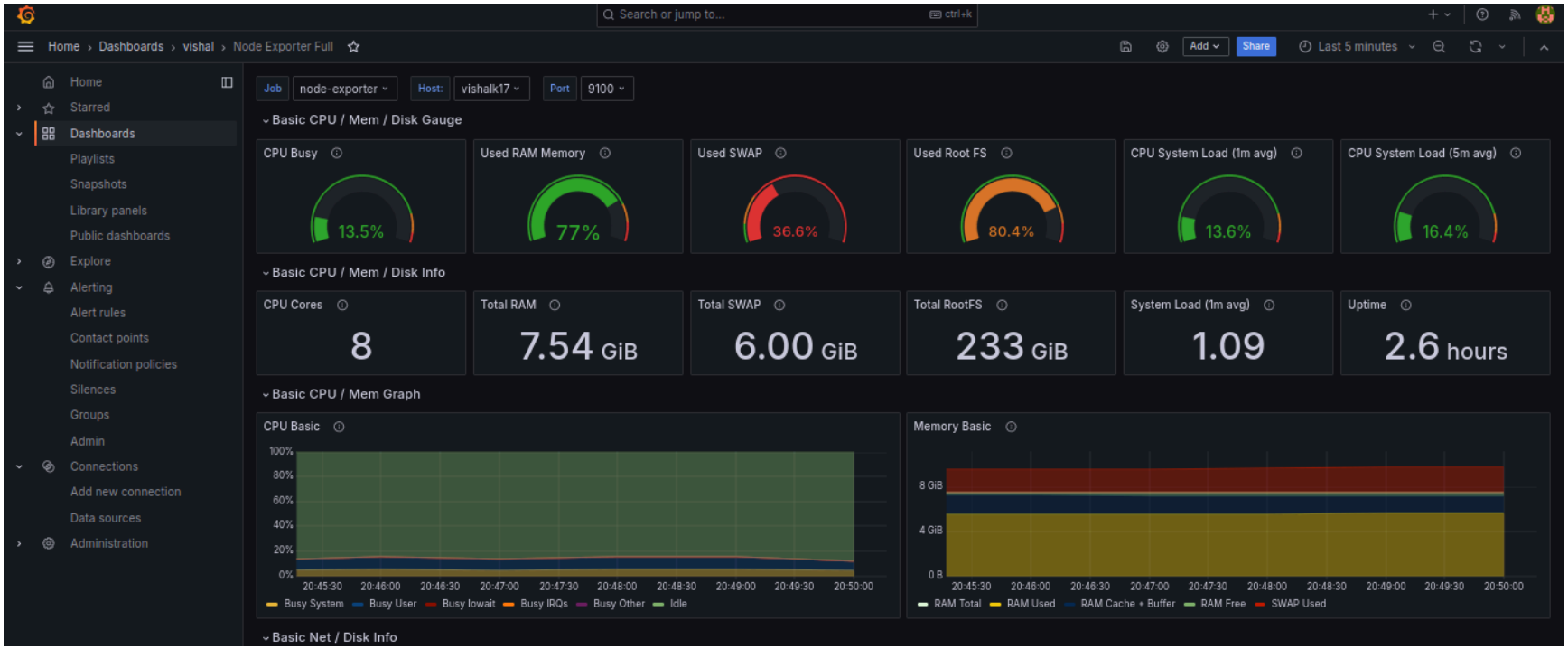

Search or jump to... ctrl+k
Home › Dashboards › Import dashboard
Home
Starred
Dashboards
Playlists
Snapshots
Library panels
Public dashboards
Explore
Alerting
Alert rules
Contact points
Notification policies
Silences
Groups
Import dashboard
Import dashboard from file or Grafana.com
Options
Name
PVC
Folder
vishal
Unique identifier (UID)
The unique identifier (UID) of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs. The UID allows having consistent URLs for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.
pvcoverview
Change uid
Import
Cancel

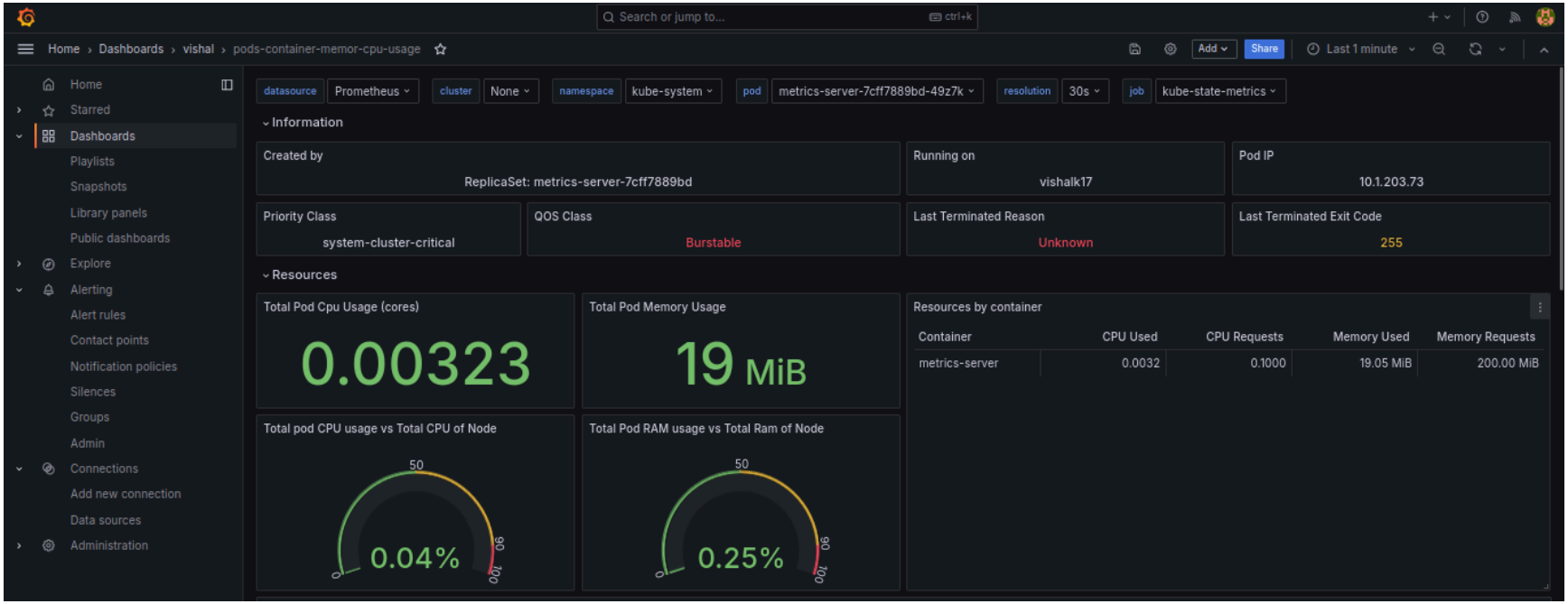

Search or jump to... ctrl+k
Home › Dashboards › vishal › PVC
Add
Share
Last 5 minutes
10s
Datasource
Prometheus
Cluster
None
Namespace
observability
PVC
All
Persistent Volumes
Persistent Volumes
Capacity
Namespace
PVC
Status
observability
prometheus-kube-prom-stack-kube-prome-pro...
1

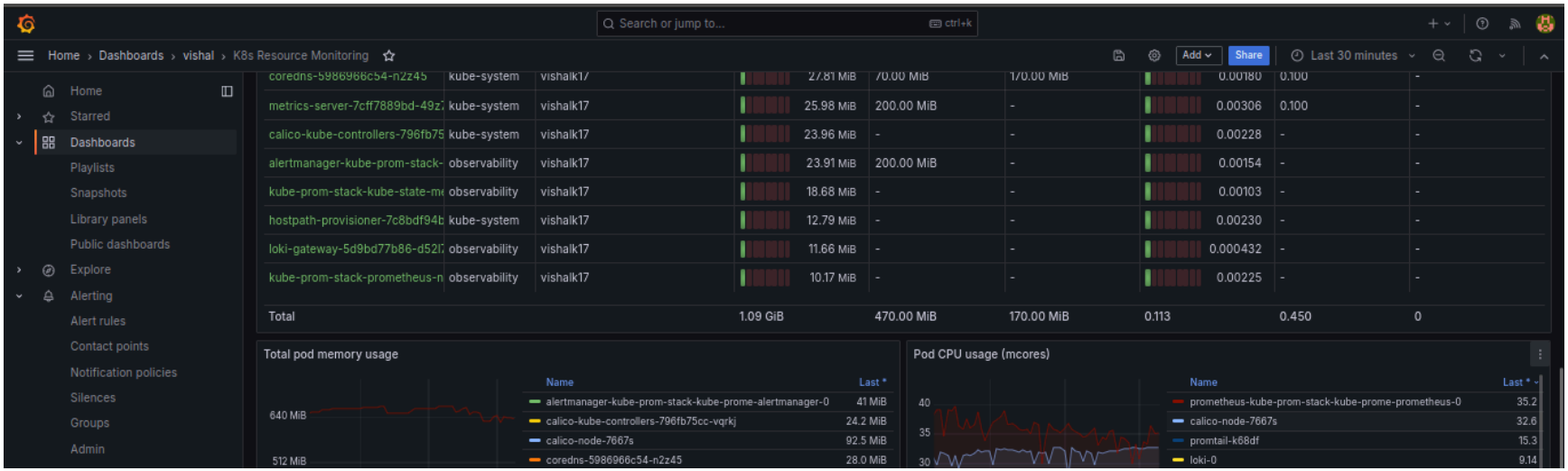
3. Importing via Direct JSON:

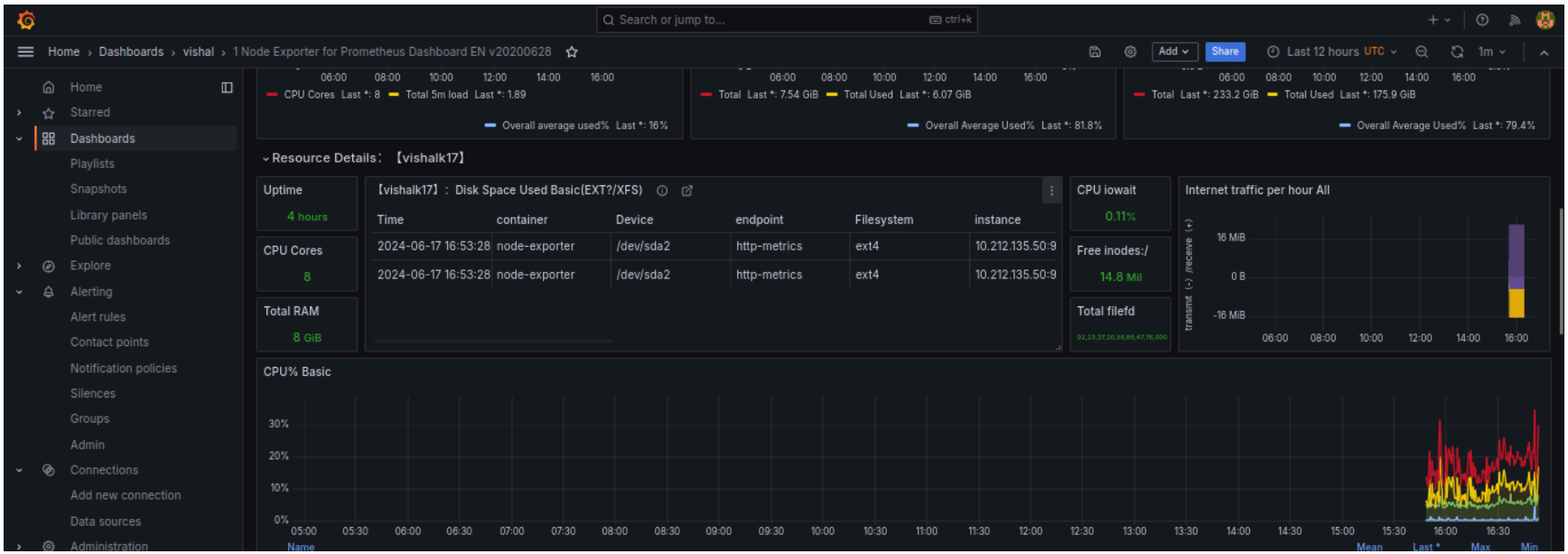
- **Copy JSON Text:**
  1. Copy the full JSON text of the dashboard you want to import.
- **Paste in Grafana:**
  1. In your Grafana instance, go to "Dashboards" -> "New" -> "Import."
  2. Select the "Import via panel json" option.
  3. Paste the JSON text and configure options as needed.
  4. Select the data source and click "Import."

7.3.2 : Various other Grafana DashBoards :




Search or jump to... ctrl+k
Home › Dashboards › vishal › pods-container-memor-cpu-usage
Add
Share
Last 1 minute
Home
Starred
Dashboards
Playlists
Snapshots
Library panels
Public dashboards
Explore
Alerting
Alert rules
Contact points
Notification policies
Silences
Groups
Admin
Connections
Add new connection
Data sources
Administration
datasource Prometheus
cluster None
namespace kube-system
pod metrics-server-7cff7889bd-49z7k
resolution 30s
job kube-state-metrics
Information
Created by
ReplicaSet: metrics-server-7cff7889bd
Running on
vishalk17
Pod IP
10.1.203.73
Priority Class
system-cluster-critical
QOS Class
Burstable
Last Terminated Reason
Unknown
Last Terminated Exit Code
255
Resources
Total Pod Cpu Usage (cores)
0.00323
Total Pod Memory Usage
19 MiB
Resources by container
Container
CPU Used
CPU Requests
Memory Used
Memory Requests
metrics-server
0.0032
0.1000
19.05 MiB
200.00 MiB
Total pod CPU usage vs Total CPU of Node
0.04%
Total Pod RAM usage vs Total Ram of Node
0.25%


Search or jump to... ctrl+k
Home › Dashboards › vishal › K8s Resource Monitoring
Add Share Last 30 minutes
Home
Starred
Dashboards
Playlists
Snapshots
Library panels
Public dashboards
Explore
Alerting
Alert rules
Contact points
Notification policies
Silences
Groups
Admin
coredns-5986966c54-n2z45 kube-system vishalk17 27.81 MiB 70.00 MiB 170.00 MiB 0.00180 0.100 -
metrics-server-7cff7889bd-49z7 kube-system vishalk17 25.98 MiB 200.00 MiB - 0.00306 0.100 -
calico-kube-controllers-796fb75 kube-system vishalk17 23.96 MiB - - 0.00228 - -
alertmanager-kube-prom-stack- observability vishalk17 23.91 MiB 200.00 MiB - 0.00154 - -
kube-prom-stack-kube-state-me observability vishalk17 18.68 MiB - - 0.00103 - -
hostpath-provisioner-7c8bdf94b kube-system vishalk17 12.79 MiB - - 0.00230 - -
loki-gateway-5d9bd77b86-d52l observability vishalk17 11.66 MiB - - 0.000432 - -
kube-prom-stack-prometheus-n observability vishalk17 10.17 MiB - - 0.00225 - -
Total 1.09 GiB 470.00 MiB 170.00 MiB 0.113 0.450 0
Total pod memory usage
640 MiB
512 MiB
Name Last *
alertmanager-kube-prom-stack-kube-prome-alertmanager-0 41 MiB
calico-kube-controllers-796fb75cc-vqrkj 24.2 MiB
calico-node-7667s 92.5 MiB
coredns-5986966c54-n2z45 28.0 MiB
Pod CPU usage (mcores)
40
35
30
Name Last *
prometheus-kube-prom-stack-kube-prome-prometheus-0 35.2
calico-node-7667s 32.6
promtail-k68df 15.3
loki-0 9.14


Search or jump to... ctrl+k
Home › Dashboards › vishal › 1 Node Exporter for Prometheus Dashboard EN v20200628
Add Share Last 12 hours UTC 1m
Home
Starred
Dashboards
Playlists
Snapshots
Library panels
Public dashboards
Explore
Alerting
Alert rules
Contact points
Notification policies
Silences
Groups
Admin
Connections
Add new connection
Data sources
Administration
CPU Cores Last *: 8 Total 5m load Last *: 1.89
Overall average used% Last *: 16%
Total Last *: 7.54 GiB Total Used Last *: 6.07 GiB
Overall Average Used% Last *: 81.8%
Total Last *: 233.2 GiB Total Used Last *: 175.9 GiB
Overall Average Used% Last *: 79.4%
Resource Details: 【vishalk17】
Uptime 4 hours
CPU Cores 8
Total RAM 8 GiB
【vishalk17】: Disk Space Used Basic(EXT?/XFS)
Time container Device endpoint Filesystem instance
2024-06-17 16:53:28 node-exporter /dev/sda2 http-metrics ext4 10.212.135.50:9
2024-06-17 16:53:28 node-exporter /dev/sda2 http-metrics ext4 10.212.135.50:9
CPU iowait 0.11%
Free inodes:/ 14.8 Mil
Total filefd
Internet traffic per hour All
16 MiB
0 B
-16 MiB
transmit (-) /receive (+)
CPU% Basic
30%
20%
10%
0%
Name Mean Last * Max Min

7.4.0: Explore Metrics/ Logs in Grafana.

Grafana's Explore feature provides a powerful and flexible way to interact with your time-series data, whether it's from Prometheus, Loki, or other supported data sources. It enables you to:

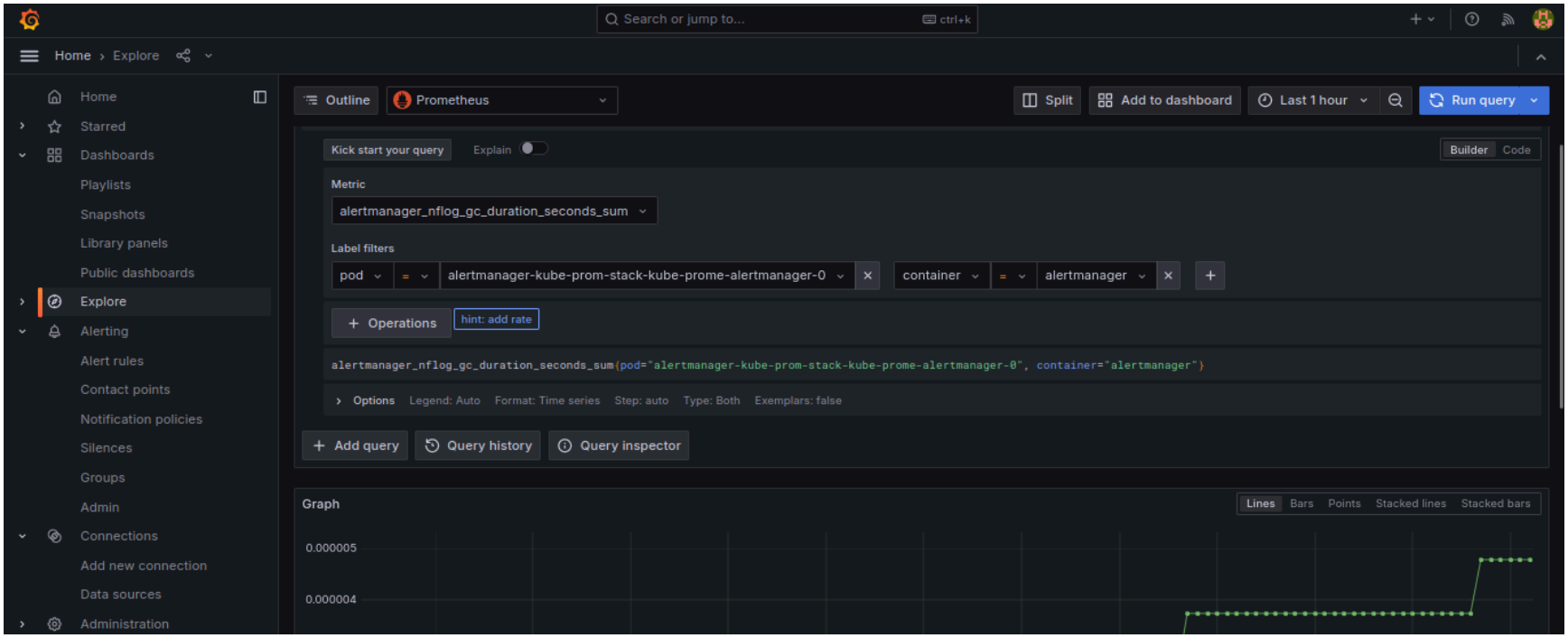
- **Query and Visualize:** Run PromQL queries (for Prometheus) or LogQL queries (for Loki) and instantly see the results as graphs, tables, or other visualizations.
- **Investigate Anomalies:** Dive deep into specific time ranges, zoom in on details, and split graphs to compare different time periods or dimensions.
- **Troubleshoot Issues:** Quickly identify patterns, correlations, and anomalies in your data to troubleshoot and resolve problems.

Accessing Explore:

1. **Click "Explore"** in the left-hand sidebar of Grafana.
2. **Select Data Source:** Choose the data source you want to explore (e.g., Prometheus, Loki).

7.4.1: Exploring Metrics (Prometheus):

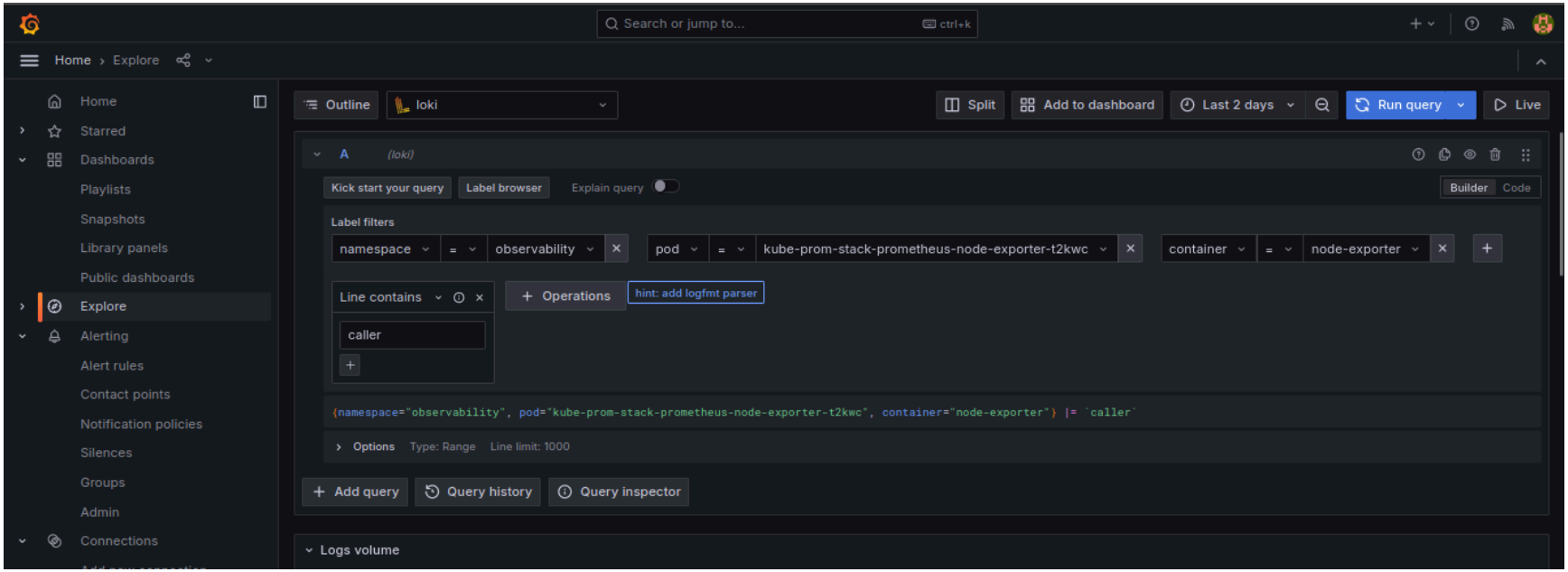
1. **Query Field:**
   - Type a PromQL query to filter and aggregate metrics.
   - Use the autocomplete suggestions to discover available metrics and labels.
   - Grafana 11 and later offers an easy-to-use metrics browser that doesn't require knowledge of PromQL.
2. **Visualization Options:**
   - Choose the type of visualization you want (e.g., line graph, heatmap, table).
   - Adjust the time range, resolution, and other display settings.
3. **Explore Features:**
   - Split View: Compare multiple queries or time ranges side-by-side.
   - Zoom and Pan: Navigate through the time series data.
   - Inspect: Get detailed information about individual data points.
   - Add to Dashboard: Save your query and visualization to a dashboard.
   - Share: Share your exploration with others.

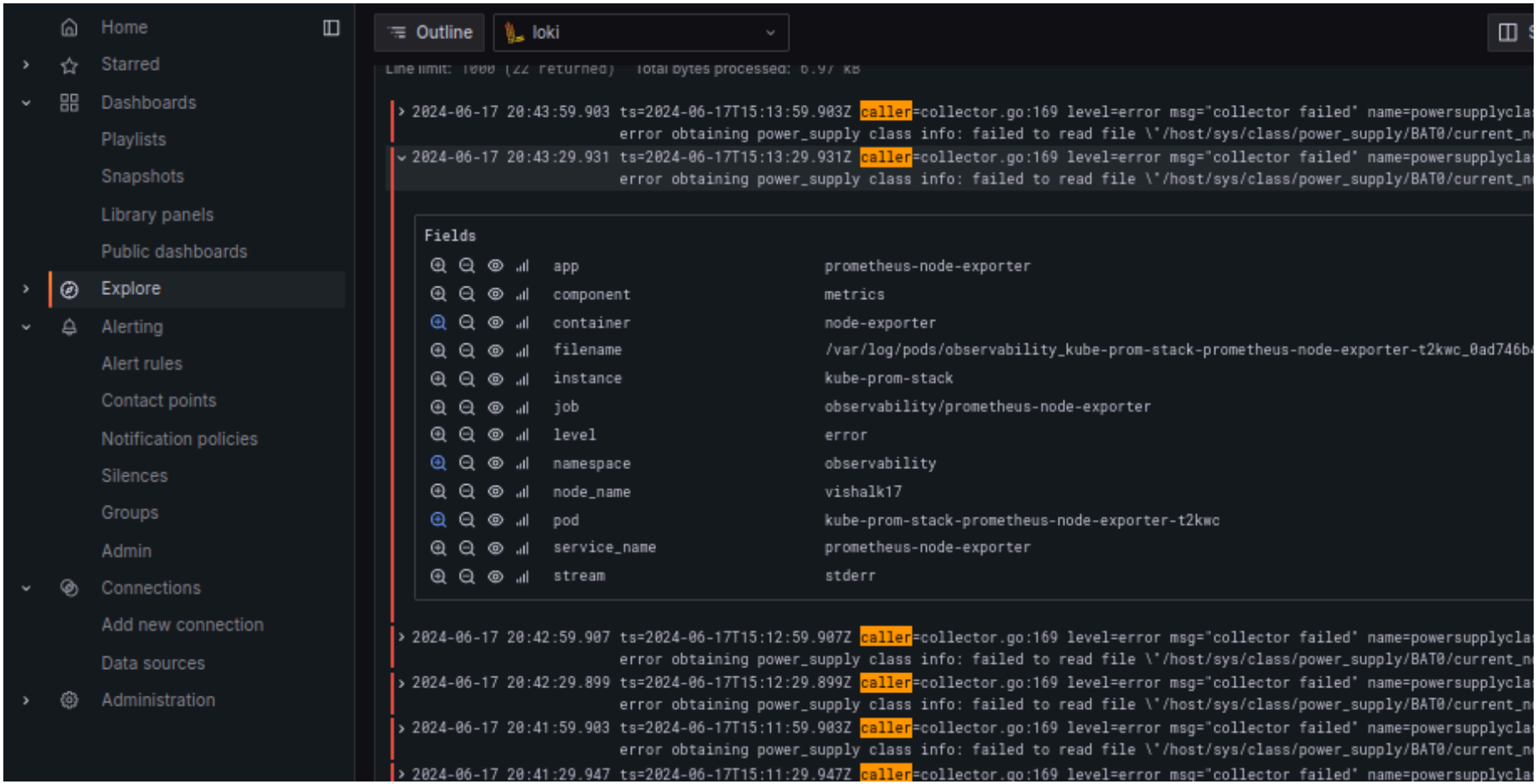

Search or jump to... ctrl+k
Home › Explore
Home
Starred
Dashboards
Playlists
Snapshots
Library panels
Public dashboards
Explore
Alerting
Alert rules
Contact points
Notification policies
Silences
Groups
Admin
Connections
Add new connection
Data sources
Administration
Outline
Prometheus
Split
Add to dashboard
Last 1 hour
Run query
Kick start your query
Explain
Builder
Code
Metric
alertmanager_nflog_gc_duration_seconds_sum
Label filters
pod = alertmanager-kube-prom-stack-kube-prome-alertmanager-0
container = alertmanager
Operations
hint: add rate
alertmanager_nflog_gc_duration_seconds_sum{pod="alertmanager-kube-prom-stack-kube-prome-alertmanager-0", container="alertmanager"}
Options Legend: Auto Format: Time series Step: auto Type: Both Exemplars: false
Add query
Query history
Query inspector
Graph
Lines Bars Points Stacked lines Stacked bars
0.000005
0.000004


Library panels
Public dashboards
Explore
Alerting
Alert rules
Contact points
Notification policies
Silences
Groups
Admin
Connections
Add new connection
Data sources
Administration
0.000003
0.000002
0.000001
0
19:45
19:50
19:55
20:00
20:05
20:10
20:15
20:20
20:25
20:30
20:35
20:40
{__name__="alertmanager_nflog_gc_duration_seconds_sum", container="alertmanager", endpoint="http-web", instance="[redacted]", job="kube-prom-stack-kube-prome-alertmanager", namespace="observability", pod="a
Raw
Table Raw
Expand results
Result series: 1
alertmanager_nflog_gc_duration_seconds_sum{container="alertmanager", endpoint="http-web", instance="[redacted]", job="kube-prom-stack-kube-prome-alertmanager",
0.0000047

7.4.2: Exploring Logs (Loki):

1. **Query Field:**
   - Type a LogQL query to filter and search your log data.
   - Use labels and filters to narrow down your results.
2. **Visualization Options:**
   - View logs as a list or table.
   - Create time-series graphs of log events based on labels or extracted fields.
3. **Explore Features:**
   - Filter by Labels: Quickly filter logs based on label values.
   - Search within Logs: Find specific text within log messages.
   - Parse Logs: Extract fields from structured log lines.




Home
Starred
Dashboards
Playlists
Snapshots
Library panels
Public dashboards
Explore
Alerting
Alert rules
Contact points
Notification policies
Silences
Groups
Admin
Connections
Add new connection
Data sources
Administration
Outline
loki
Fields
app prometheus-node-exporter
component metrics
container node-exporter
filename /var/log/pods/observability_kube-prom-stack-prometheus-node-exporter-t2kwc_0ad746b
instance kube-prom-stack
job observability/prometheus-node-exporter
level error
namespace observability
node_name vishalk17
pod kube-prom-stack-prometheus-node-exporter-t2kwc
service_name prometheus-node-exporter
stream stderr