2. Hands-On Monitoramento

Pré Requisitos

Instalação do Helm Chart cert-manager

- helm repo add jetstack https://charts.jetstack.io --force-update
- helm install cert-manager jetstack/cert-manager --namespace cert-manager --create-namespace --version v1.16.2 --set crds.enabled=true

Instalação do Helm Chart `kube-prometheus-stack`

- helm repo add prometheus-community https://prometheus-community.github.io/helm-chart
- helm repo update
- Add nodeport customizado para o prometheus e o grafana
- helm upgrade --install kube-prometheus-stack prometheus-community/kube-prometheus-stack -f custom-values.yaml -n monitoring

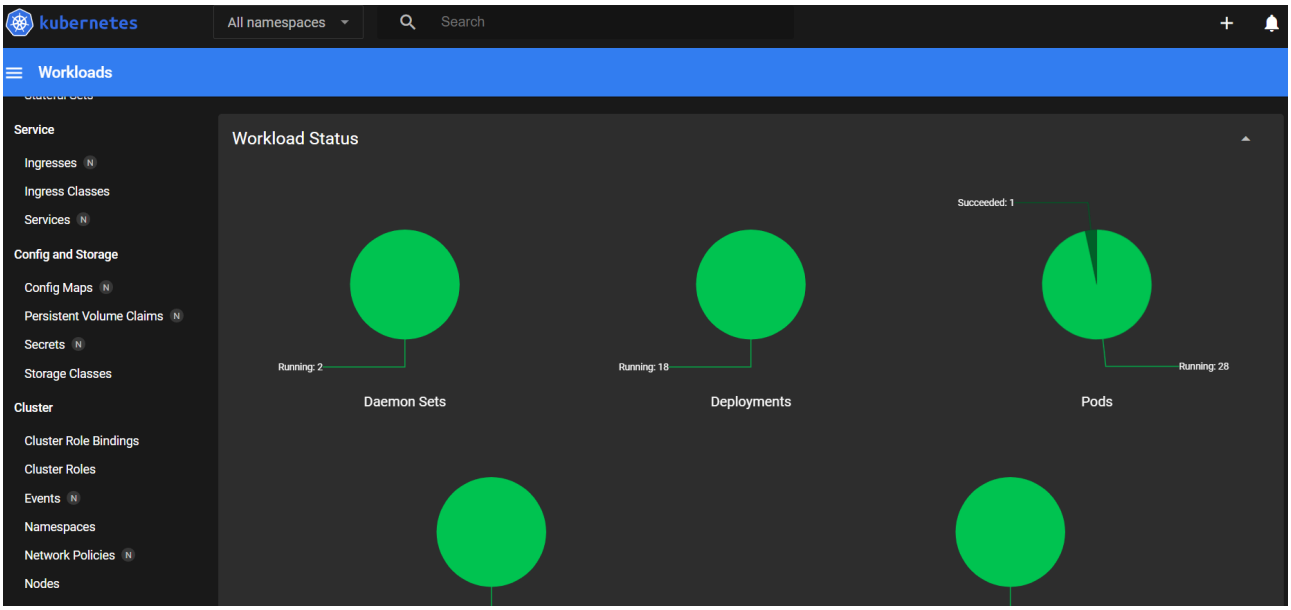
Monitoramento do Cluster

Atualmente temos 2 tipos de acompanhamento de métricas para o cluster Kubernetes: Via Prometheus+Grafana ou via Kubernetes Dashboard.

Kubernetes Dashboard

Para habilitar o monitoramento do ambiente de maneira prática, executamos os seguintes passos:

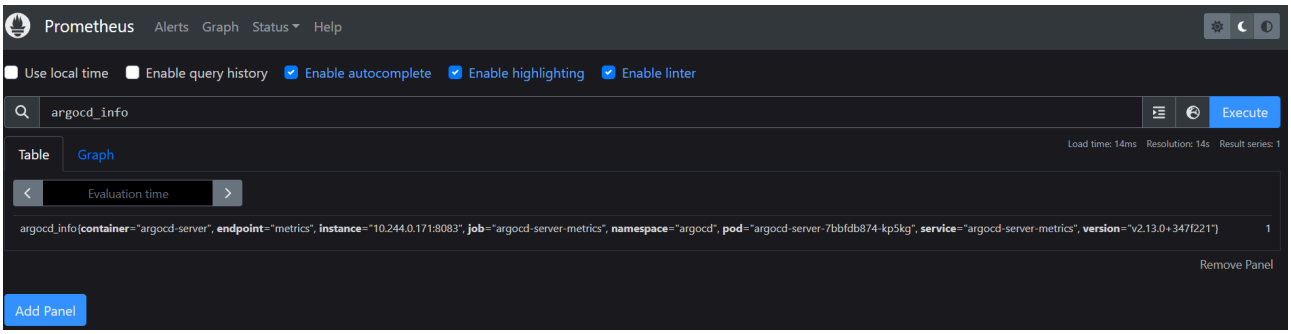
- minikube -p nomecluster addons enable metrics-server
- minikube dashboard -p nomecluster

- 

- Podemos realizar:
  - Acompanhar construção de workloads: pods, services, pvcs, secrets e configmaps
  - Acompanhar CRDS e seus estados desejados
  - Acompanhar eventos do cluster
  - Acompanhar Services
  - Gerenciar NameSpaces

Prometheus Operator

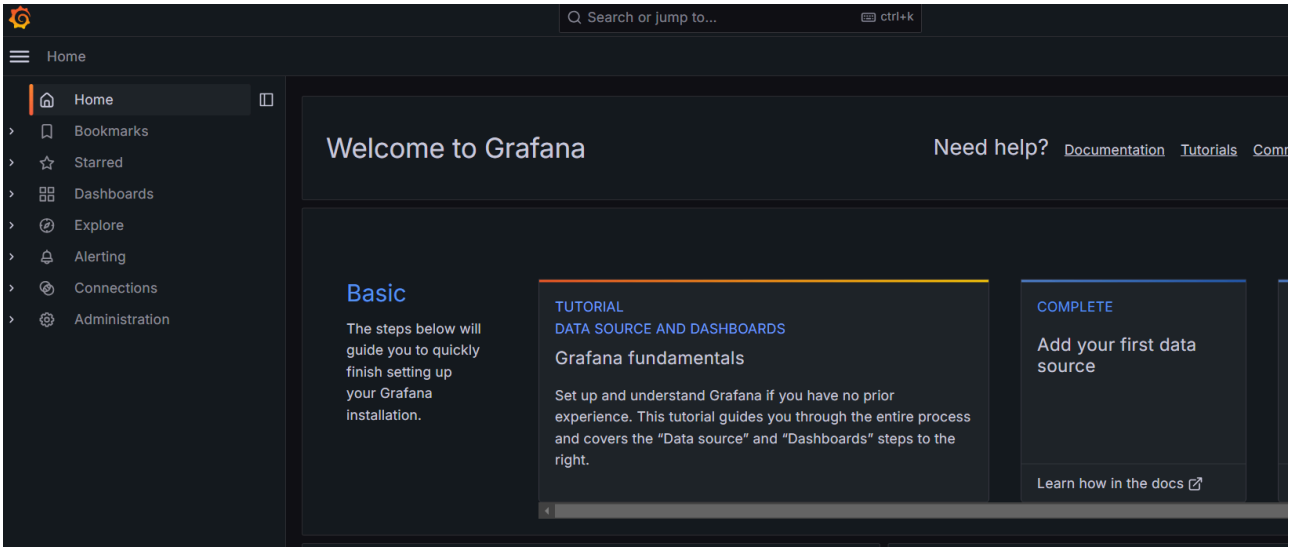
Um scraper de métricas construído no cluster como Operador.
Para acessar o painel do Prometheus UI, utilizar o comando abaixo:

- minikube service kube-prometheus-stack-prometheus -p testeJupyter -n monitoring
- 

- Para pesquisar quais tipos de métricas o Prometheus está rastreando, é só pesquisar pelas tags
  - É possível ver o resultado em formato tabular ou em grafico
- Na aba de alertas podemos acompanhar quais os status de alguns acompanhamentos do Cluster

Grafana

Um recurso de Dashboard construído para consumir as métricas raspadas pelo Prometheus Operator.
Para acessar o painel do Grafana UI, utilizar o comando abaixo:

- minikube service kube-prometheus-stack-grafana -p testeJupyter -n monitoring
- 

- Navegar até a aba Dashboards
- A lista de dashboards já cadastrados vem do helm chart que instalamos anteriormente
- É possível add novos Dashboards através do Grafana Labs
  - https://grafana.com/grafana/dashboards/

Monitoramento do ArgoCD

Para vincular as métricas do ArgoCD com o Prometheus, é necessário averiguar se a lista de services abaixo foram implantados:

- argocd-server
- argocd-server-metrics
- argocd-dex-server
- argocd-notification-metrics

Com isso feito, criamos 4 arquivos yaml, 1 pra cada server do argocd, conforme yaml abaixo:

- ServiceMonitor são aplicações que são criadas dentro do Operador do Prometheus, apontando qual service e qual porta o operador deve buscar as métricas necessárias.
- https://argo-cd.readthedocs.io/en/stable/operator-manual/metrics/#application-controller-metrics

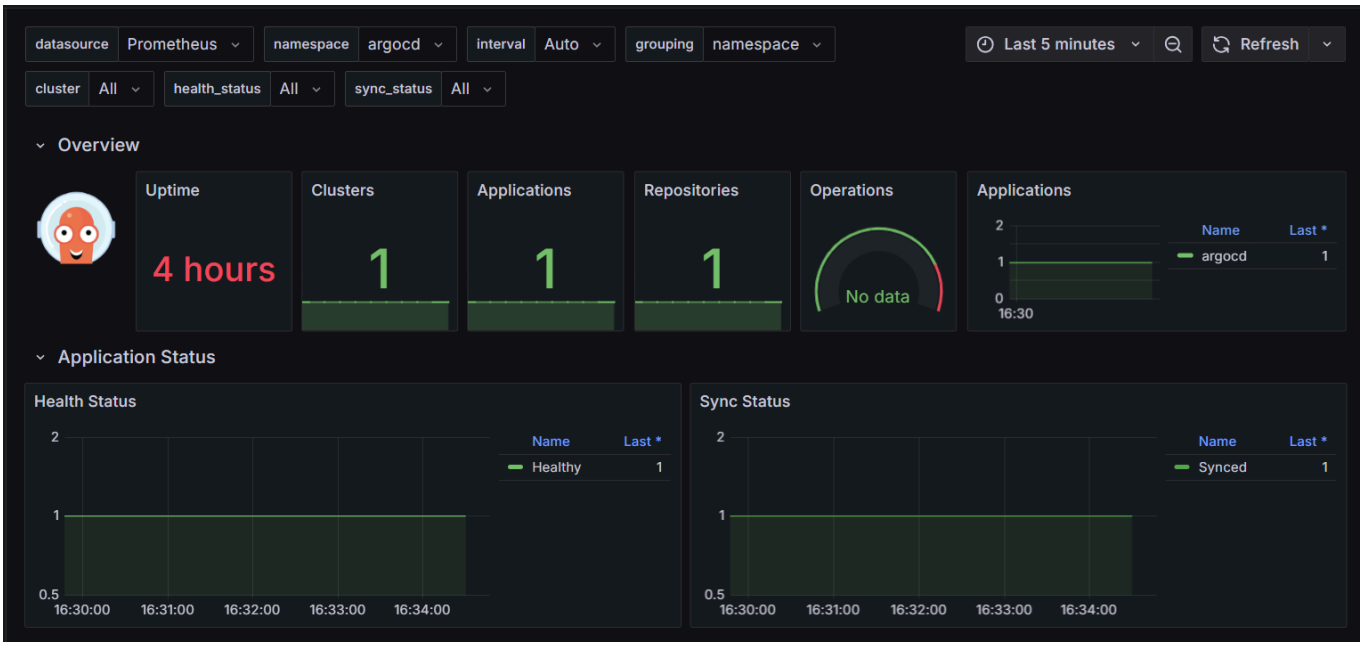
```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
```

```yaml
  name: argocd-dex-server
  namespace: monitoring
  labels:
    release: kube-prometheus-stack
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: argocd-dex-server
  namespaceSelector:
    matchNames:
      - argocd
  endpoints:
  - port: metrics
    interval: 30s
    scrapeTimeout: 10s
# ==================================================
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: argocd-metrics
  namespace: monitoring
  labels:
    release: kube-prometheus-stack
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: argocd-metrics
  namespaceSelector:
    matchNames:
      - argocd
  endpoints:
  - port: metrics
    interval: 30s
    scrapeTimeout: 10s
# ================================================
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: argocd-notifications-controller
  namespace: monitoring
  labels:
    release: kube-prometheus-stack
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: argocd-notifications-controller-metrics
```

```yaml
  namespaceSelector:
    matchNames:
      - argocd
  endpoints:
  - port: metrics
    interval: 30s
    scrapeTimeout: 10s
# ====================================================
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: argocd-server-metrics
  namespace: monitoring
  labels:
    release: kube-prometheus-stack
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: argocd-server-metrics
  namespaceSelector:
    matchNames:
      - argocd
  endpoints:
  - port: metrics
    interval: 30s
    scrapeTimeout: 10s
```

Para visualizar os dados no Grafana, utilizamos o dashboard do link: https://grafana.com/grafana/dashboards/14584-argocd/

Ao importar o dashboard via ID para dentro do Grafana e com os ServiceMonitor aplicados temos o resultado abaixo:



Monitoramento do Spark Operator

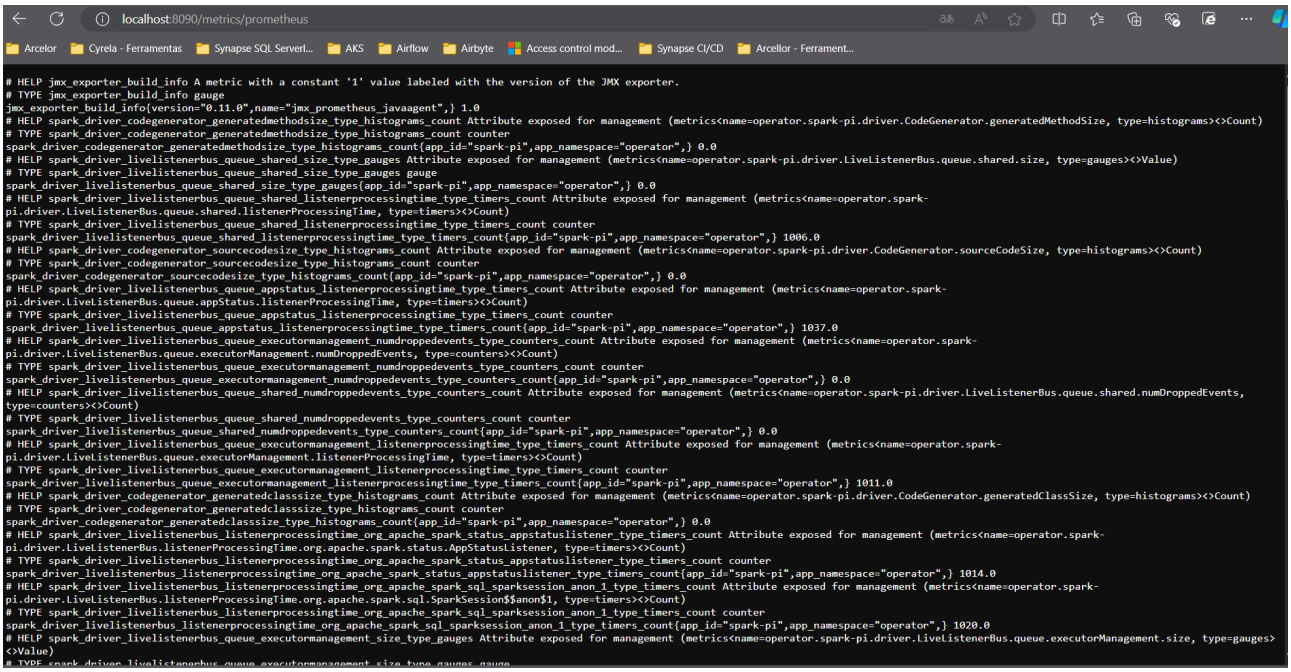
Como vincular Prometheus Operator com as Métricas do Operator para acompanhar métricas de Driver e Executor.

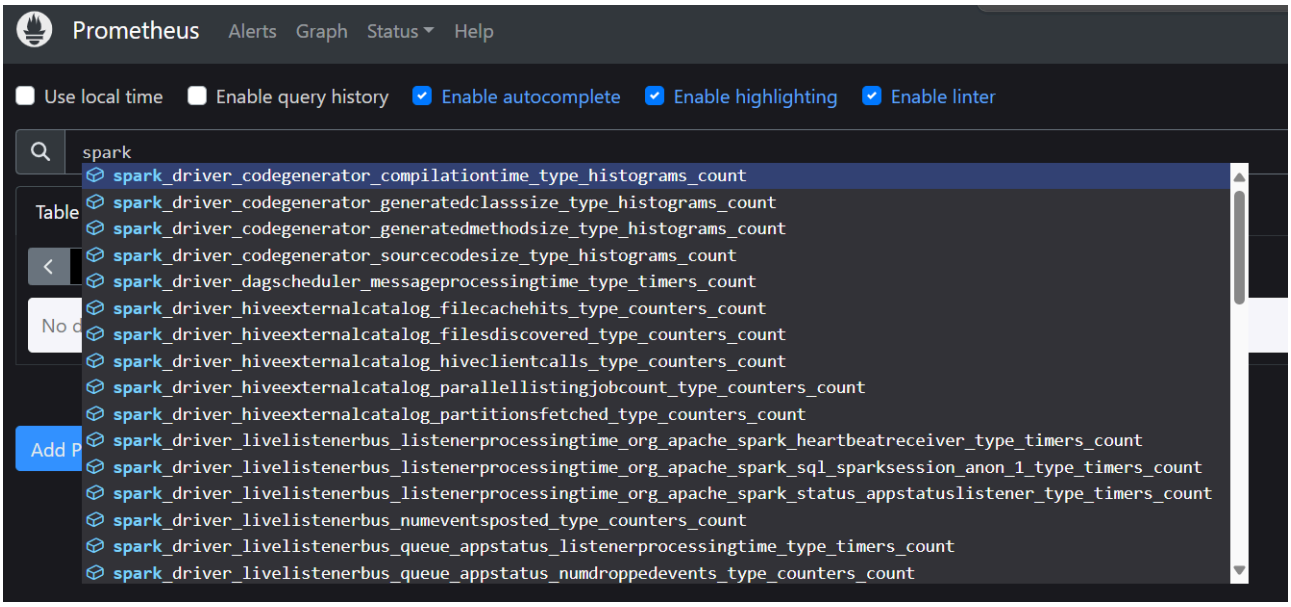
Issue aberta no github do Kubeflow Spark Operator:

- Mesmo configurando os arquivos do JMX-Exporter, o Operador não encontra o arquivo de configuração
- https://github.com/kubeflow/spark-operator/issues/1921

Dica -> O driver do Spark deixa as métricas expostas na porta 8090 para o prometheus buscas:

- ```
  ubectl port-forward pods/spark-pi-driver 8090:8090 -n operator
  ```
  ```

- 

- 

Como alcançar as métricas relacionadas acima (seguir os passos):

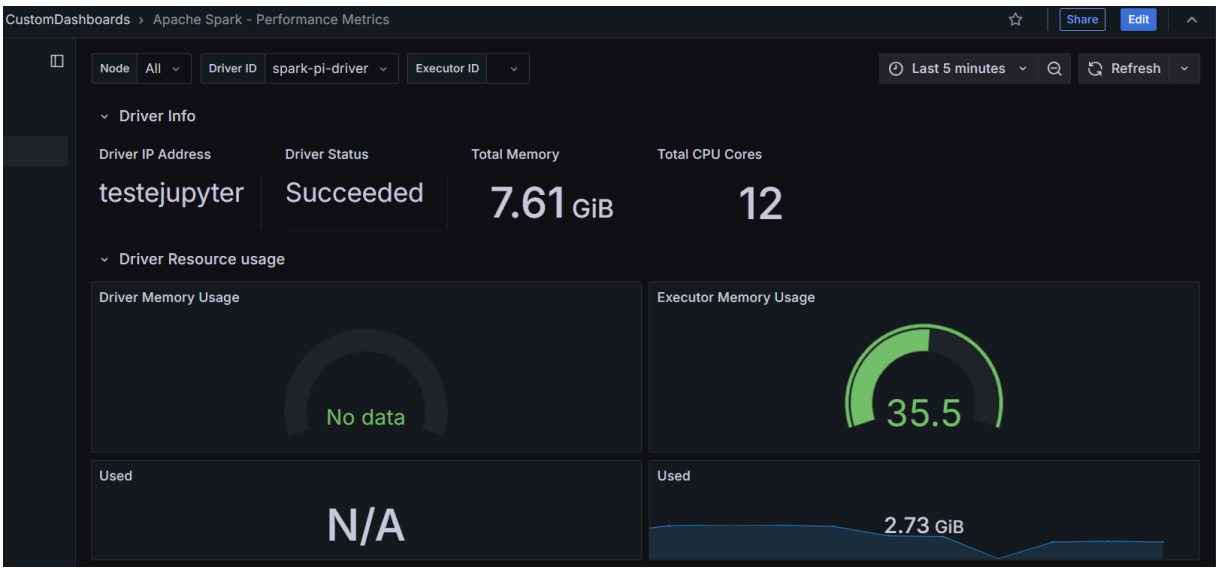
- Colocar os arquivos de configuração na imagem customizada
  - metrics.properties
  - jmx_javaagent.jar
  - prometheus.yaml
- Criar imagem e deixar disponível
- Criar manifesto spark app com os dados abaixo (o restante não muda):

```
    monitoring:
  exposeDriverMetrics: true
  exposeExecutorMetrics: true
  prometheus:
```

```
      jmxExporterJar: /prometheus/jmx_prometheus_javaagent-0.11.0.jar
      port: 8090
      portName: prometheus
```

- Criar um manifesto do tipo podMonitor para acompanhar o pod diretamente ao inves do driver

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: spark-pod-application-metrics
  namespace: monitoring
  labels:
    release: kube-prometheus-stack
spec:
  selector:
    matchLabels:
      sparkoperator.k8s.io/app-name: spark-pi
      spark-role: driver
      sparkoperator.k8s.io/launched-by-spark-operator: "true"
  namespaceSelector:
    matchNames:
      - operator  # Nome do namespace onde o Spark está rodando
  podMetricsEndpoints:
  - path: /metrics
    port: prometheus
    interval: 15s
    scrapeTimeout: 10s
```

- Ao fazer os apply's de ambos os arquivos, só acompanhar a busca das métricas
- Logar no Grafana: (Login: admin & Senha: prom-operator)
- 

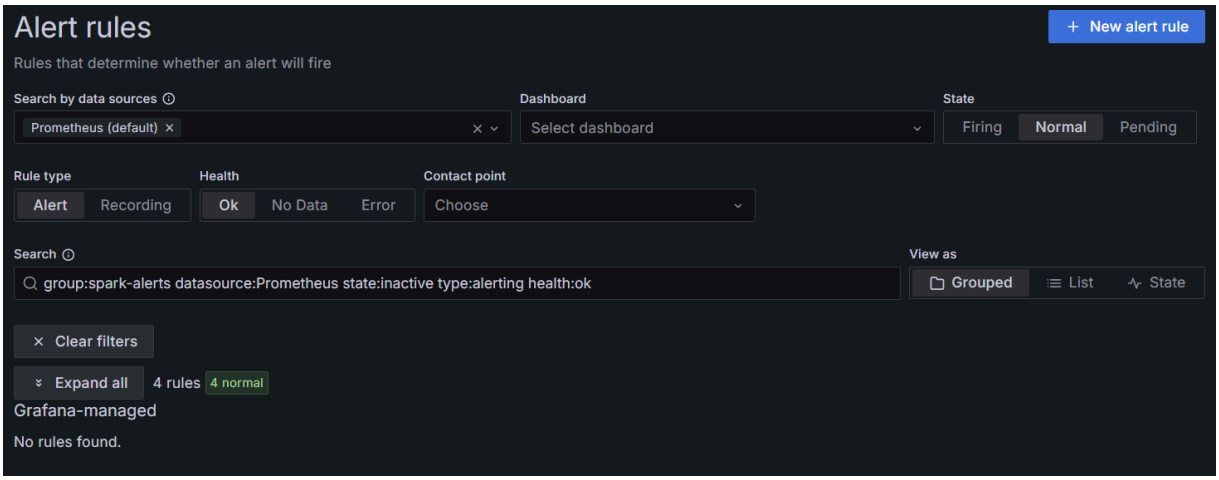
- Regras de Alerta Criadas para acompanhar os recursos Spark

State	Name	Health	Summary	Actions
Normal	HighDriverMemoryUsage	ok	High memory usage in Spark Driver	More
Normal	ExecutorFailed	ok	Spark Executor Failure	More
Normal	DriverFailed	ok	Spark Driver Failure	More
Normal	HighExecutorMemoryUsage	ok	High memory usage in Spark Executor	More

- Manifesto YAML para criar alerta de regras

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: spark-alerts
  namespace: monitoring
  labels:
    release: kube-prometheus-stack
spec:
  groups:
    - name: spark-alerts
      rules:
        - alert: HighDriverMemoryUsage
          expr: spark_driver_jvm_heap_used_bytes /
spark_driver_jvm_heap_max_bytes > 0.8
          for: 2m
          labels:
            severity: warning
          annotations:
            summary: "High memory usage in Spark Driver"
            description: "Driver memory usage is above 80% for more than 2
minutes."
        - alert: ExecutorFailed
          expr: spark_executor_failed_tasks > 0
          for: 1m
          labels:
            severity: critical
          annotations:
            summary: "Spark Executor Failure"
            description: "One or more executors have failed tasks."
        - alert: DriverFailed
          expr: spark_driver_failed_tasks > 0
          for: 1m
          labels:
            severity: critical
          annotations:
```

```
            summary: "Spark Driver Failure"
            description: "The Spark Driver has failed tasks."
        - alert: HighExecutorMemoryUsage
          expr: spark_executor_memory_used / spark_executor_memory_max > 0.9
          for: 2m
          labels:
            severity: warning
          annotations:
            summary: "High memory usage in Spark Executor"
            description: "Executor memory usage is above 90% for more than 2
minutes."
```

- Menu de Pesquisa de Alertas
  - 

- Exploração de Métricas Gerais
  - 

Monitoramento do MinIO

Configuração de exportação das métricas do MinIO:

- MinIO Operator (values.yaml) -> Add variáveis de ambiente de autenticação e mapeamento do Prometheus

```
env:
  - name: MINIO_PROMETHEUS_AUTH_TYPE
    value: "public"
  - name: PROMETHEUS_NAMESPACE
    value: "monitoring"
  - name: MINIO_PROMETHEUS_URL
    value: "http://kube-prometheus-stack-
prometheus.monitoring.svc.cluster.local:9090"
```

- MinIO Tenant (values.yaml) ->

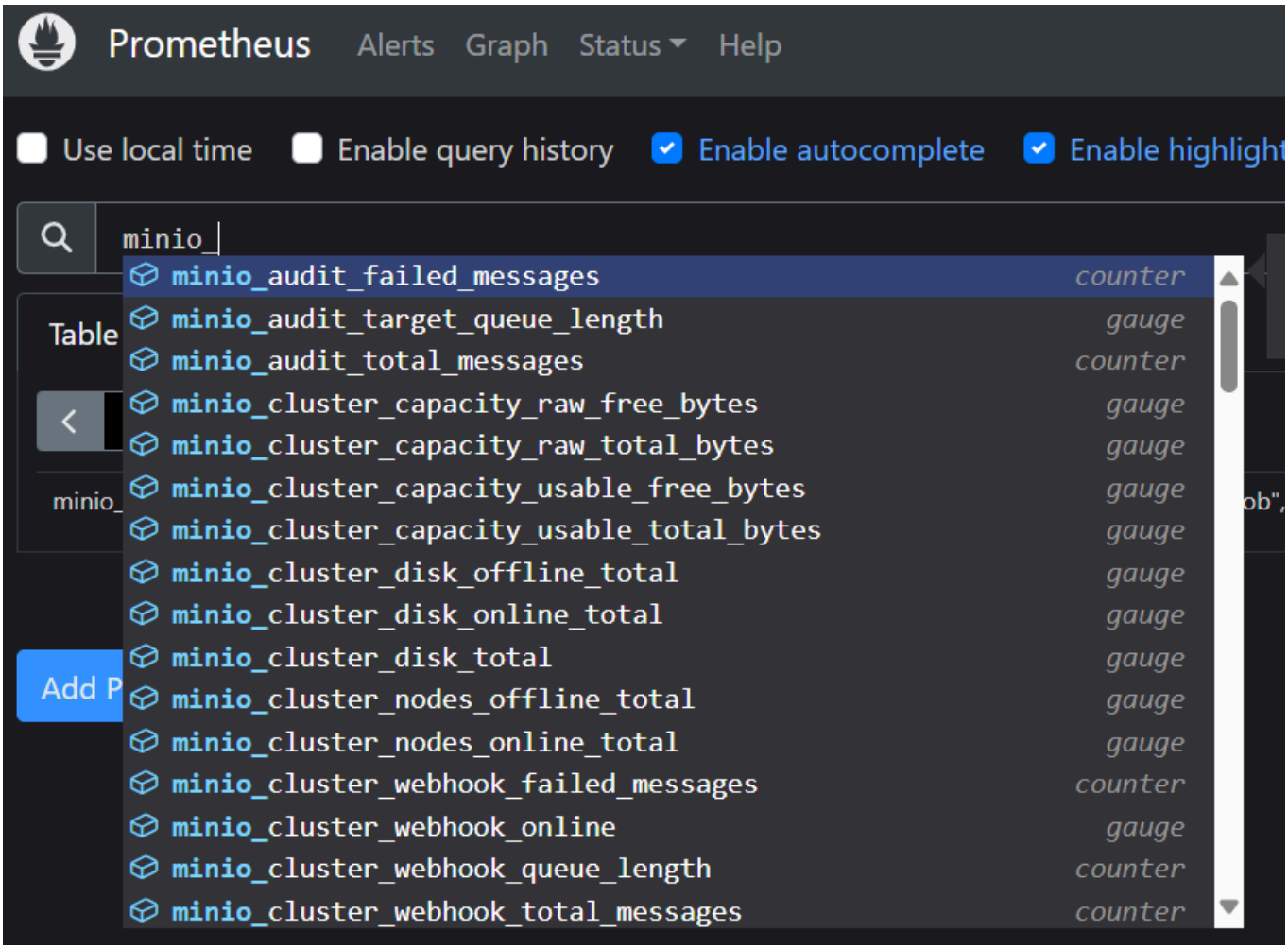
```
# pool metrics to be read by Prometheus
metrics:
  enabled: true
  port: 9000
  protocol: http
## exposeServices defines the exposure of the MinIO object storage and Console
services.
## service is exposed as a loadbalancer in k8s service.
exposeServices:
  minio: true
  console: true
# Tenant scrape configuration will be added to prometheus managed by the
prometheus-operator.
prometheusOperator: true
## Add environment variables to be set in MinIO container
(https://github.com/minio/minio/tree/master/docs/config)
env:
  - name: MINIO_PROMETHEUS_AUTH_TYPE
        value: "public"
  - name: PROMETHEUS_NAMESPACE
        value: "monitoring"
  - name: MINIO_PROMETHEUS_URL
        value: "http://kube-prometheus-stack-
prometheus.monitoring.svc.cluster.local:9090"
```

- Prometheus Service Monitor

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
```

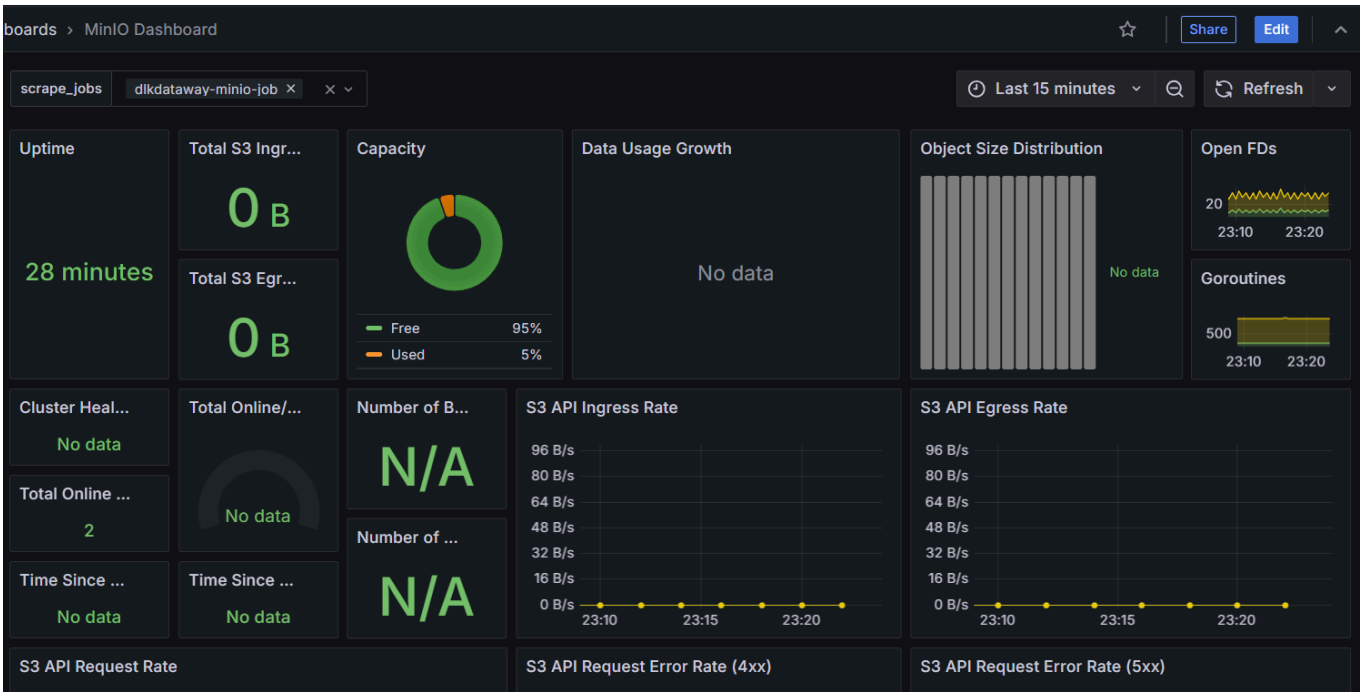
```
    name: minio-metrics
    namespace: monitoring
    labels:
      release: kube-prometheus-stack
  spec:
    selector:
      matchLabels:
        app.kubernetes.io/name: minio
        v1.min.io/tenant: dlkdataway
    namespaceSelector:
      matchNames:
        - storage
    endpoints:
    - port: http-minio
      path: /minio/v2/metrics/cluster
      interval: 15s
      scrapeTimeout: 10s
```

As métricas sendo raspadas pelo Prometheus:



Como o Prometheus UI enxerga o endpoint Target do MinIO:

Após importar o Dashboard do MinIO para o Grafana, selecione no campo da esquerda o minio-job para filtrar somente o recursodo MinIO.



Monitoramento do Airflow

O Airflow por padrão vem com um pod/service do statsd-exporter que tem como funcionalidade exportar as métricas da release e do scheduler para o formato Prometheus. Por padrão, ela já vem habilitado, logo só precisamos configurar o monitoramento do Prometheus ao Service.

Formato das métricas a serem exportadas, para visualizar:

- `kubectl port-forward svc/airflow-datawaybr-statsd 9091:9102 -n orquestrator`

localhost:9091/metrics

```
# HELP airflow_dag_processing_file_path_queue_size Metric autogenerated by statsd_exporter.
# TYPE airflow_dag_processing_file_path_queue_size gauge
airflow_dag_processing_file_path_queue_size 0
# HELP airflow_dag_processing_file_path_queue_update_count Metric autogenerated by statsd_exporter.
# TYPE airflow_dag_processing_file_path_queue_update_count counter
airflow_dag_processing_file_path_queue_update_count 11554
# HELP airflow_dag_processing_import_errors Metric autogenerated by statsd_exporter.
# TYPE airflow_dag_processing_import_errors gauge
airflow_dag_processing_import_errors 0
# HELP airflow_dag_processing_last_duration Metric autogenerated by statsd_exporter.
# TYPE airflow_dag_processing_last_duration summary
airflow_dag_processing_last_duration{quantile="0.5"} 0.112189
airflow_dag_processing_last_duration{quantile="0.9"} 0.17843
airflow_dag_processing_last_duration{quantile="0.99"} 0.557141
airflow_dag_processing_last_duration_sum 35.226628000000005
airflow_dag_processing_last_duration_count 192
# HELP airflow_dag_processing_last_duration_test_spark_operator Metric autogenerated by statsd_exporter.
# TYPE airflow_dag_processing_last_duration_test_spark_operator summary
airflow_dag_processing_last_duration_test_spark_operator{quantile="0.5"} 0.112189
airflow_dag_processing_last_duration_test_spark_operator{quantile="0.9"} 0.17843
airflow_dag_processing_last_duration_test_spark_operator{quantile="0.99"} 0.557141
airflow_dag_processing_last_duration_test_spark_operator_sum 35.226628000000005
airflow_dag_processing_last_duration_test_spark_operator_count 192
# HELP airflow_dag_processing_last_run_seconds_ago Metric autogenerated by statsd_exporter.
# TYPE airflow_dag_processing_last_run_seconds_ago gauge
airflow_dag_processing_last_run_seconds_ago{dag_file="test-spark-operator"} 29.018702
# HELP airflow_dag_processing_processes Metric autogenerated by statsd_exporter.
# TYPE airflow_dag_processing_processes counter
airflow_dag_processing_processes 192
# HELP airflow_dag_processing_total_parse_time Metric autogenerated by statsd_exporter.
# TYPE airflow_dag_processing_total_parse_time gauge
airflow_dag_processing_total_parse_time 0.04226058600033866
# HELP airflow_dagbag_size Metric autogenerated by statsd_exporter.
# TYPE airflow_dagbag_size gauge
airflow_dagbag_size 1
# HELP airflow_dataset_orphaned Metric autogenerated by statsd_exporter.
# TYPE airflow_dataset_orphaned gauge
airflow_dataset_orphaned 0
# HELP airflow_executor_open_slots Metric autogenerated by statsd_exporter.
# TYPE airflow_executor_open_slots gauge
airflow_executor_open_slots 32
# HELP airflow_executor_queued_tasks Metric autogenerated by statsd_exporter.
# TYPE airflow_executor_queued_tasks gauge
airflow_executor_queued_tasks 0
# HELP airflow_executor_running_tasks Metric autogenerated by statsd_exporter.
```

Por padrão, o Airflow já vem com o módulo statsd-exporter instalado e habilitado por padrão. Logo, fica somente o apply do manifesto ServiceMonitor:

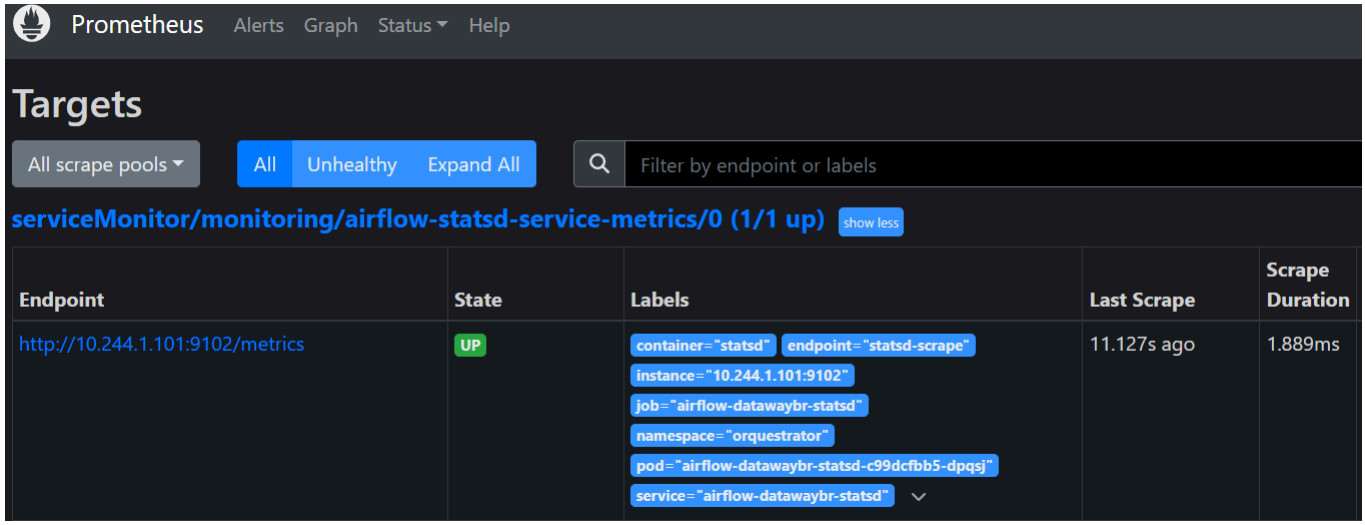
```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: airflow-statsd-service-metrics
  namespace: monitoring
  labels:
    release: kube-prometheus-stack
spec:
  selector:
    matchLabels:
      tier: airflow
```

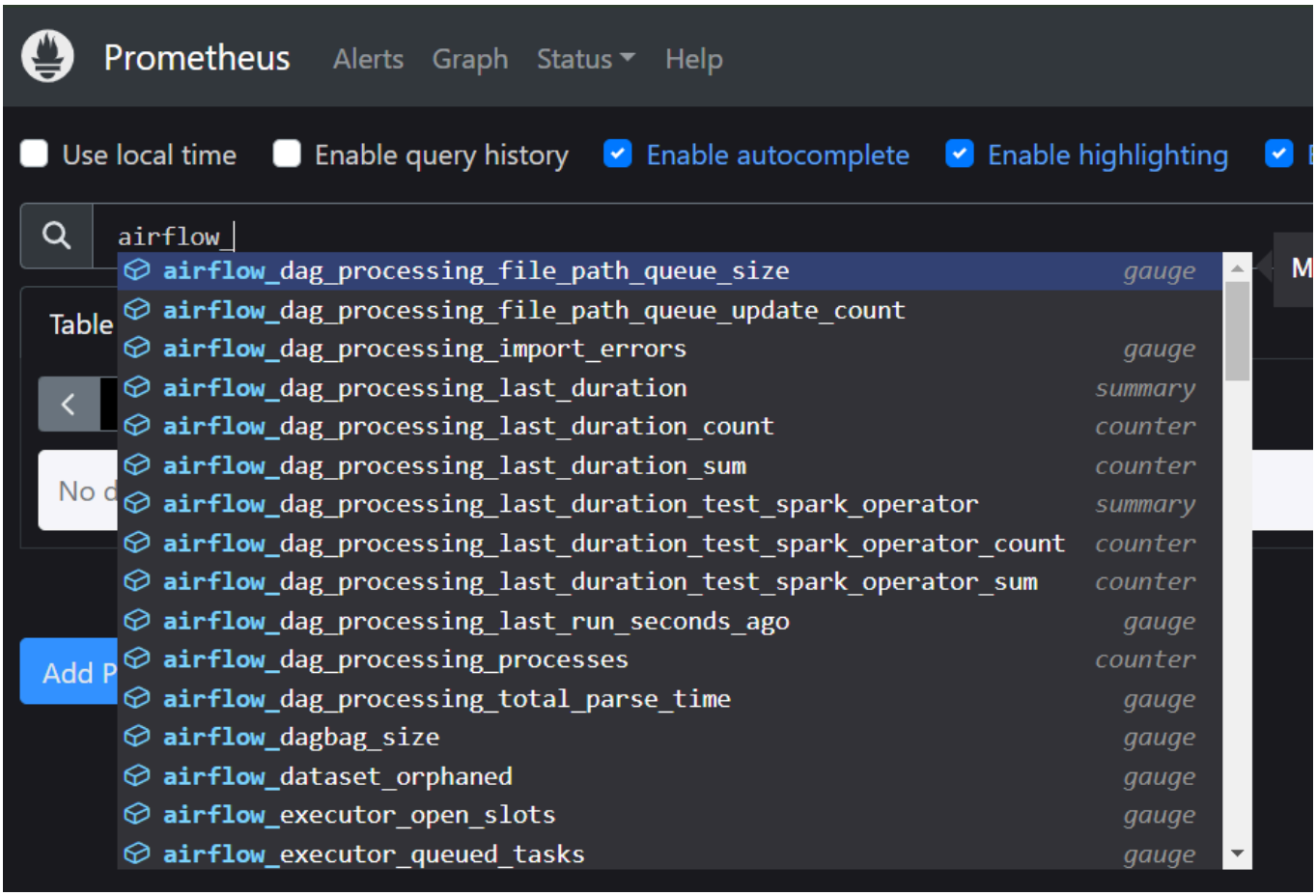
```
      component: statsd
      release: airflow-datawaybr
  namespaceSelector:
    matchNames:
      - orquestrator
  endpoints:
  - port: statsd-scrape
    interval: 15s
    scrapeTimeout: 10s
```

Na etapa de metrics do values.yaml, pode manter como default:

```
metrics:
    statsd_on: '{{ ternary "True" "False" .Values.statsd.enabled }}'
    statsd_port: 9125
    statsd_prefix: airflow
    statsd_host: '{{ printf "%s-statsd" (include "airflow.fullname" .) }}'
```

Com o ServiceMonitor aplicado, devemos visualizar os logs no Prometheus da seguinte forma:





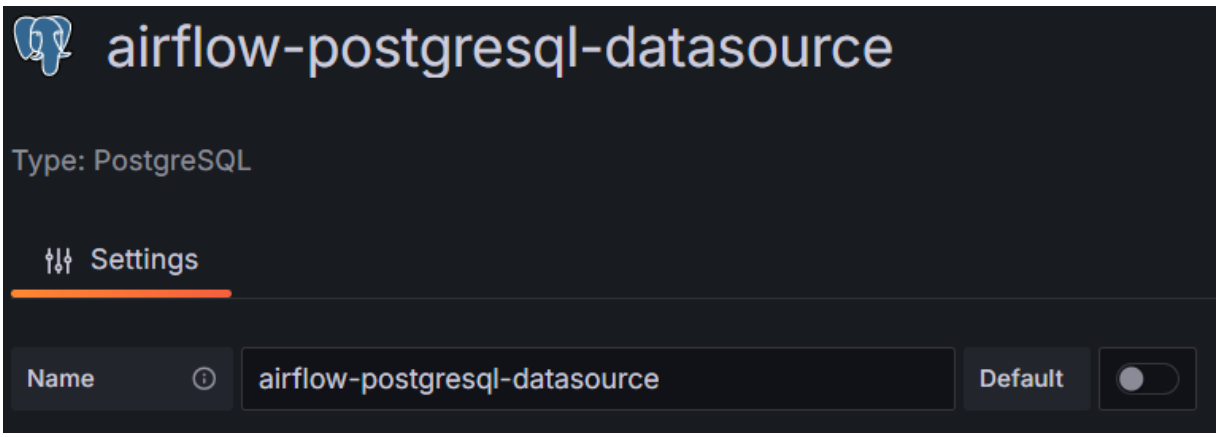
No momento não encontramos um Dashboard funcional para o Airflow no Grafana, logo vamos demonstrar mais as métricas em si (exploração).

Temos duas fontes de exploração de métricas no Airflow:

- Metadata Database (Postgres)
- Airflow Statsd (cluster)

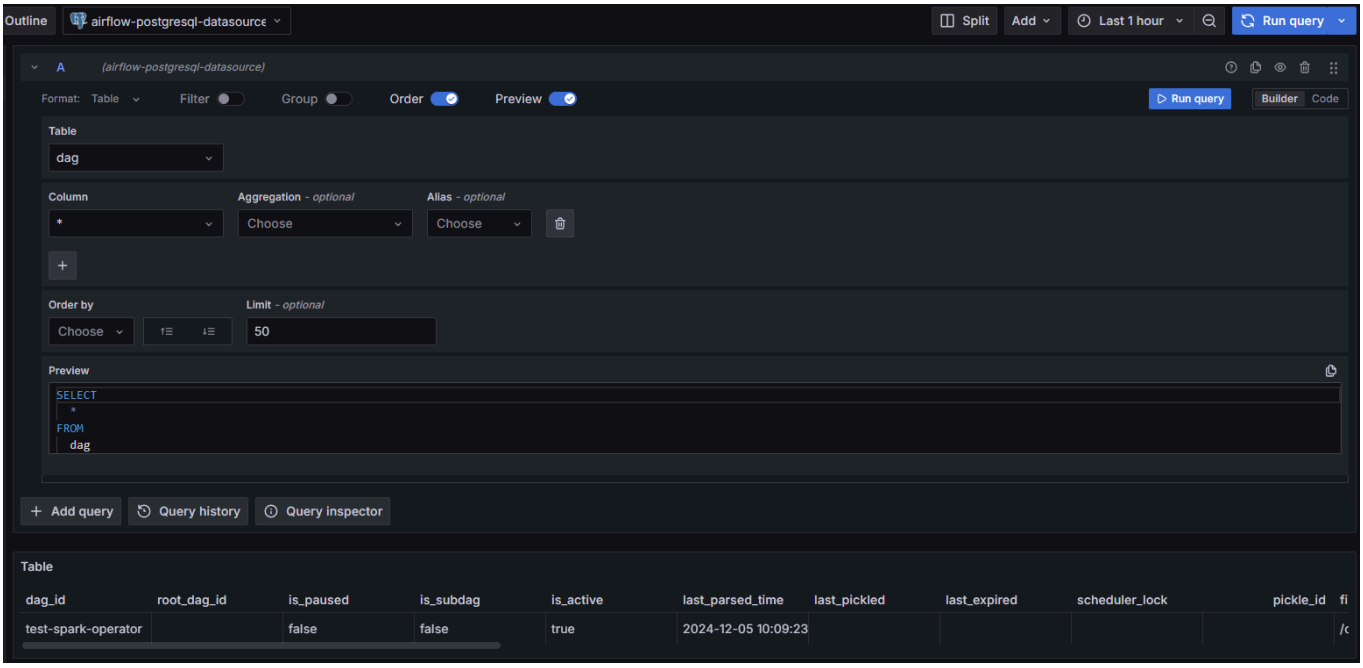
Explorando métricas do Airflow Statsd:

Explorando métricas do Metadata Database:

- Add datasource do PostgreSQL
- Renomear para
  - 

- Host e Database
  - `airflow-datawaybr-postgresql.orquestrator.svc.cluster.local:5432`
  - `postgres`
- Autenticação
  - Username: `postgres`
  - Password: `postgres`
- Versão:
  - PostgreSQL Version: 13

Realizando Consultas ao Banco de Dados:



Monitoramento dos Custos do Cluster

Já parou pra pensar que utilizando aplicações dentro do Cluster, ao final do mês só temos o valor gasto por CPU/Memória/Disco?
E se tivesse uma forma de segregar os custos que cada aplicação gasta em cima da CPU, da Memória e do Disco de forma dinâmica e automática?

- A resposta é KubeCost

Links do KubeCost

Guia Helm Chart: [kubecost/cost-analyzer-helm-chart: Kubecost helm chart]
Previsão de Custo: [kubecost/kubectl-cost: CLI for determining the cost of Kubernetes workloads]
Documentação: [Welcome to the Docs! | Kubecost Docs]

Configuração do KubeCost

DISCLAIMER:

- Não funciona tão bem quanto esperava no MiniKube
- Deve funcionar melhor na cloud

Vamos criar um cluster com mais memória pra esse teste:

- minikube start -p testeCost --cpus=4 --memory=5192 --driver=docker

Vamos fazer a instalação da ferramenta:

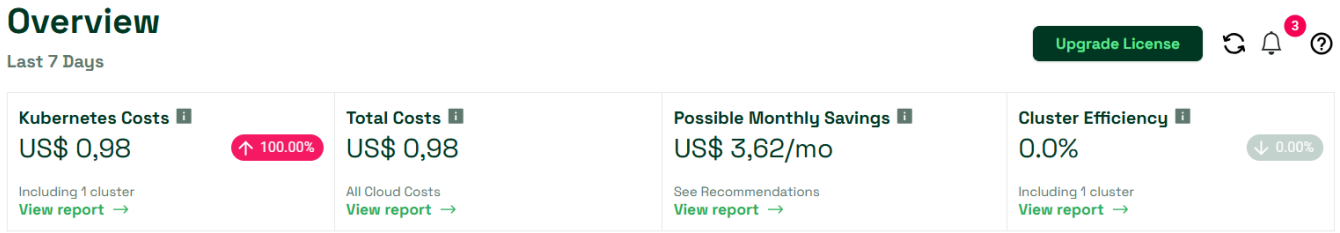
- helm repo add kubecost https://kubecost.github.io/cost-analyzer/
- helm install kubecost kubecost/cost-analyzer -n kubecost --set kubecostToken="aGVsbUBrdWJlY29zdC5jb20=xm343yadf98"

Ponto Importante após o Apply:

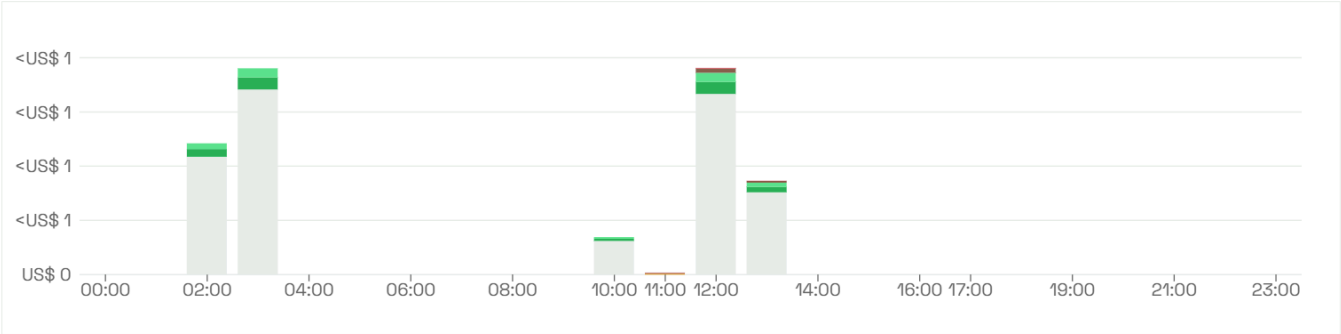
- kubectl port-forward -n kubecost deployment/kubecost-cost-analyzer 9090
- O Kubecost leva até 25 minutos para puxar todas as métricas para si. Uma barra de progresso aparece no topo da tela informando o status atual

UI do KubeCost

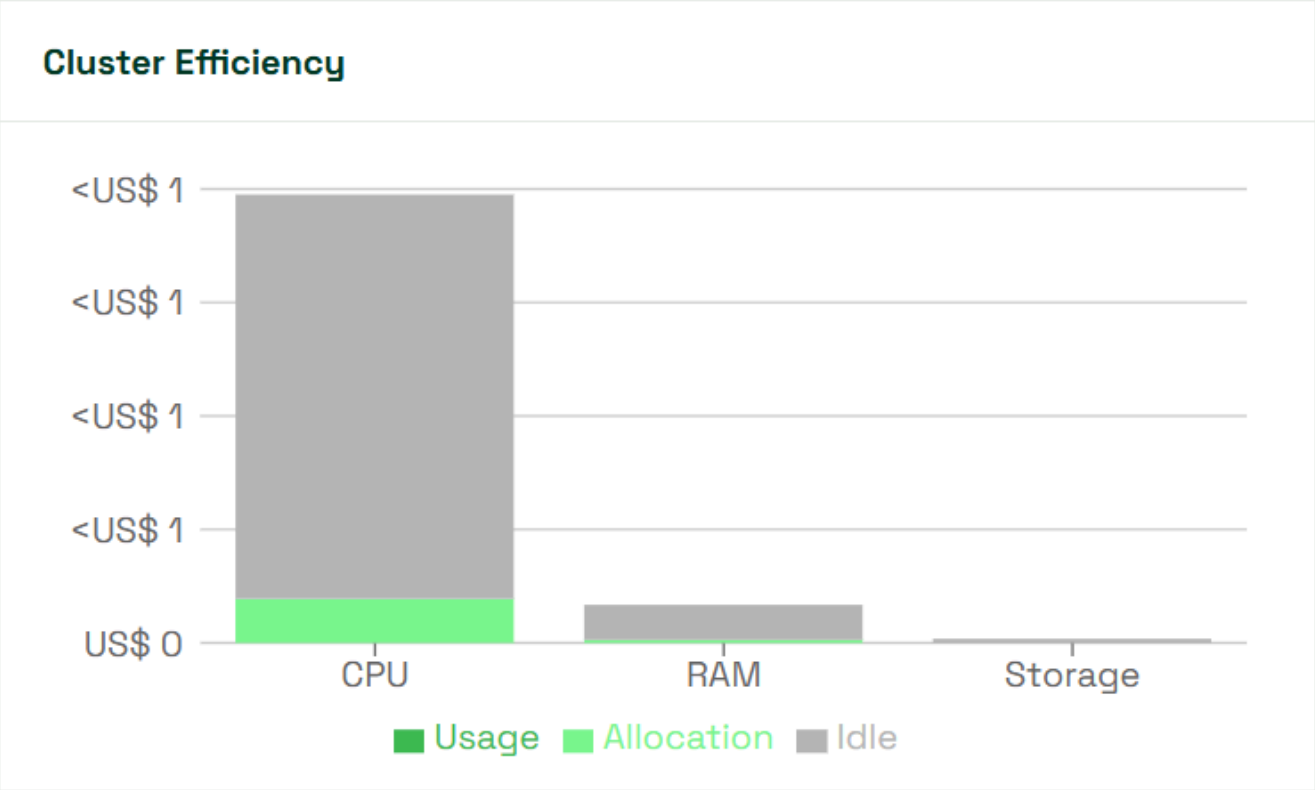
Overview dos custos:



Visão de Execução durante o dia dos custos por namespace:



Eficiência do Cluster em Termos de CPU/Memória e Storage



Configuração do Kubectl cost

Instalar o pré-requisito Krew (gerenciador de plugins do kubectl)
https://krew.sigs.k8s.io/docs/user-guide/setup/install/

- Seguir o link da documentação
- Baixar o binário
- Criar uma pasta no Disco C com o conteúdo baixado
- Abrir um cmd de administrador na pasta
- Rodar o comando do print

- Add o path do print no PATH do sistema

Windows

1. Make sure `git` is installed.
2. Download `krew.exe` from the Releases page to a directory.
3. Launch a command prompt (`cmd.exe`) with administrator privileges (since the installation requires use of symbolic links) and navigate to that directory.
4. Run the following command to install krew:

```
.\krew install krew
```

5. Add the `%USERPROFILE%\.krew\bin` directory to your `PATH` environment variable (how?)
6. Launch a new command-line window.
7. Run `kubectl krew` to check the installation.

Instalar o plugin cost do kubectl

- kubectl krew index remove default --force
- kubectl krew index add default https://github.com/kubernetes-sigs/krew-index.git
- kubectl krew install cost
-

```
C:\Windows\System32>kubectl krew install cost
Updated the local copy of plugin index.
Installing plugin: cost
Installed plugin: cost
\
 | Use this plugin:
 |      kubectl cost
 | Documentation:
 |      https://github.com/kubecost/kubectl-cost
 | Caveats:
 | \
 |  | Requires Kubecost (a cluster-side daemon) to be installed in your cluster.
 |  | See https://www.kubecost.com/install for installation instructions.
 | /
/
WARNING: You installed plugin "cost" from the krew-index plugin repository.
   These plugins are not audited for security by the Krew maintainers.
   Run them at your own risk.
```

kubectl cost predict -f 'spark-demo-manifest.yaml' --show-cost-per-resource-hr

Exemplos de Consulta

```
PS C:\Users\alex.fonseca_a3data\Documents\DataWayBR\k8s-argo-minio\src\spark-manifests> kubectl cost namespace
+-------------+-------------+--------------------+-----------------+
| CLUSTER     | NAMESPACE   | MONTHLY RATE (ALL) | COST EFFICIENCY |
+-------------+-------------+--------------------+-----------------+
| __idle__    | __idle__    |          68.104800 |        0.000000 |
| cluster-one | default     |          14.083200 |        0.000000 |
|             | storage     |           5.631490 |        0.000000 |
|             | kube-system |           4.526836 |        0.000000 |
|             | kubecost    |           3.368688 |        0.000000 |
|             | monitoring  |           0.597287 |        0.000000 |
|             | operator    |           0.000000 |        0.000000 |
|             | argocd      |           0.000000 |        0.000000 |
+-------------+-------------+--------------------+-----------------+
| SUMMED      |             |      USD 96.312302 |                 |
+-------------+-------------+--------------------+-----------------+
```

```
PS C:\Users\alex.fonseca_a3data\Documents\DataWayBR\k8s-argo-minio\src\spark-manifests> kubectl cost deployment --show-cpu
+-------------+-------------+-----------------------------------------+----------+----------+--------------------+-----------------+
| CLUSTER     | NAMESPACE   | DEPLOYMENT                              | CPU      | CPU EFF. | MONTHLY RATE (ALL) | COST EFFICIENCY |
+-------------+-------------+-----------------------------------------+----------+----------+--------------------+-----------------+
| __idle__    | __idle__    | __idle__                                | 62.806909 | 0.000000 |          68.104800 |        0.000000 |
| cluster-one | default     | __unallocated__                         | 11.404800 | 0.000000 |          14.083200 |        0.000000 |
|             | storage     | minio-operator                          | 4.551916 | 0.000000 |           5.315874 |        0.000000 |
|             | kube-system | __unallocated__                         | 3.810764 | 0.000000 |           3.887345 |        0.000000 |
|             | kubecost    | kubecost-cost-analyzer                  | 1.231200 | 0.000000 |           1.640817 |        0.000000 |
|             |             | kubecost-forecasting                    | 1.172291 | 0.000000 |           1.402691 |        0.000000 |
|             | kube-system | coredns                                 | 0.586473 | 0.000000 |           0.640145 |        0.000000 |
|             | monitoring  | __unallocated__                         | 0.000000 | 0.000000 |           0.597287 |        0.000000 |
|             | kubecost    | kubecost-prometheus-server              | 0.000000 | 0.000000 |           0.325181 |        0.000000 |
|             | storage     | __unallocated__                         | 0.000000 | 0.000000 |           0.315616 |        0.000000 |
|             | operator    | spark-operator-datawaybr-webhook        | 0.000000 | 0.000000 |           0.000000 |        0.000000 |
|             | argocd      | argocd-server                           | 0.000000 | 0.000000 |           0.000000 |        0.000000 |
|             |             | argocd-applicationset-controller        | 0.000000 | 0.000000 |           0.000000 |        0.000000 |
|             | kubecost    | kubecost-grafana                        | 0.000000 | 0.000000 |           0.000000 |        0.000000 |
|             | argocd      | argocd-dex-server                       | 0.000000 | 0.000000 |           0.000000 |        0.000000 |
|             | monitoring  | kube-prometheus-stack-grafana           | 0.000000 | 0.000000 |           0.000000 |        0.000000 |
|             | argocd      | argocd-repo-server                      | 0.000000 | 0.000000 |           0.000000 |        0.000000 |
|             |             | argocd-notifications-controller         | 0.000000 | 0.000000 |           0.000000 |        0.000000 |
|             | storage     | console                                 | 0.000000 | 0.000000 |           0.000000 |        0.000000 |
|             | operator    | spark-operator-controller               | 0.000000 | 0.000000 |           0.000000 |        0.000000 |
|             |             | spark-operator-datawaybr-controller     | 0.000000 | 0.000000 |           0.000000 |        0.000000 |
|             |             | spark-operator-webhook                  | 0.000000 | 0.000000 |           0.000000 |        0.000000 |
|             | argocd      | __unallocated__                         | 0.000000 | 0.000000 |           0.000000 |        0.000000 |
|             |             | argocd-redis                            | 0.000000 | 0.000000 |           0.000000 |        0.000000 |
|             | monitoring  | kube-prometheus-stack-operator          | 0.000000 | 0.000000 |           0.000000 |        0.000000 |
|             |             | kube-prometheus-stack-kube-state-metrics | 0.000000 | 0.000000 |           0.000000 |        0.000000 |
+-------------+-------------+-----------------------------------------+----------+----------+--------------------+-----------------+
| SUMMED      |             |                                         | 85.564352 |          |      USD 96.312956 |                 |
+-------------+-------------+-----------------------------------------+----------+----------+--------------------+-----------------+
```

Mais exemplos de consulta, verificar: https://github.com/kubecost/kubectl-cost