

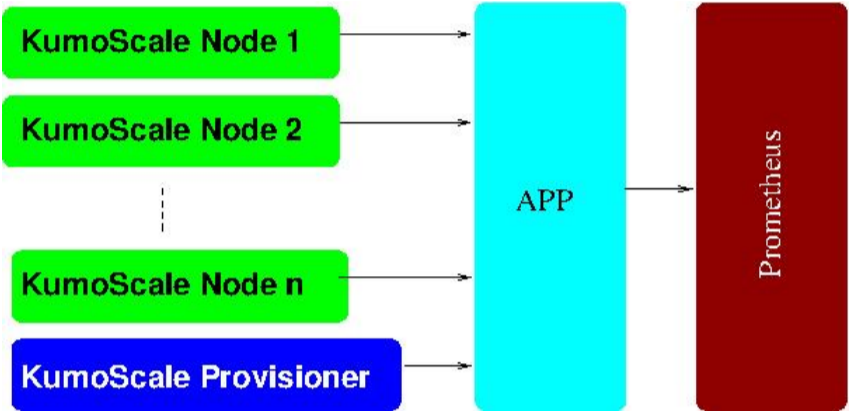
## Automating Metrics Collection

How to write an app to collect performance metrics from your KumoScale cluster and channel them your Prometheus server

KumoScale storage system provides a rich set of metrics that can be used to monitor the whole system. However, these metrics come from different parts of the KumoScale cluster. You also need proper credentials to access these metrics. This article will show you how to create a docker app that collects the metrics and deploy it using Kubernetes. We start with a Python script.

### Where do the metrics come from?

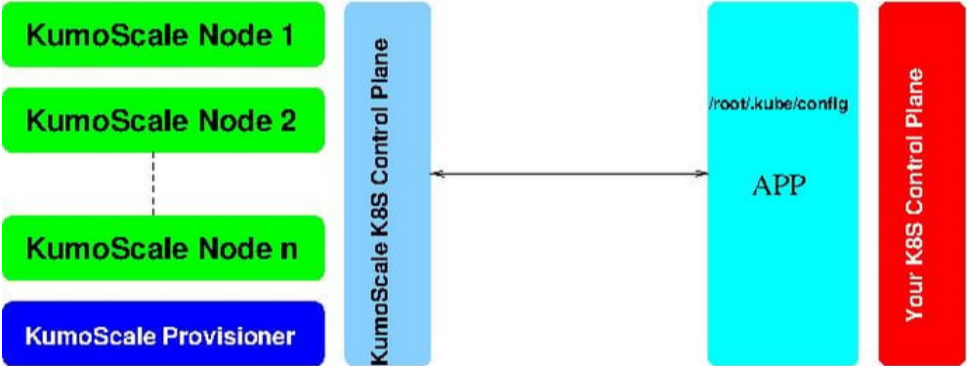
Every node in your KumoScale cluster provides metrics for its logical and physical components. In addition, the KumoScale Provisioner, which is the brain of the cluster, provides its own set of metrics. The figure below shows the top-level architecture of our approach.



To find out the addresses of the KumoScale Provisioner and nodes, our app needs to contact the KumoScale cluster. This is achieved by connecting to the KumoScale's private Kubernetes API.

### Loading the KumoScale configuration

The app needs to load KumoScale's internal Kubernetes cluster's configuration (not your Kubernetes cluster's configuration where the app will run) first. [This page](#) shows how to obtain this information in the section *Set up your Remote Host to Access the KumoScale Storage Cluster*. Save this configuration in a file called *config*. You will need to make this file available to the app in the folder */root/.kube*. We will see how to do that [later](#).



The following code will load this configuration in your app.

```
import kubernetes

kubernetes.config.load_kube_config()
configuration = kubernetes.client.Configuration() // k8s configuration
```

We use the configuration data structure to get the KumoScale Provisioner's IP address.

```
PROVISIONER = re.match(r'^http(s)?://(\d+\.\d+\.\d+\.\d+):\d+$',
    configuration.host).groups()[1]
```

We now connect to the KumoScale's internal Kubernetes's REST API.

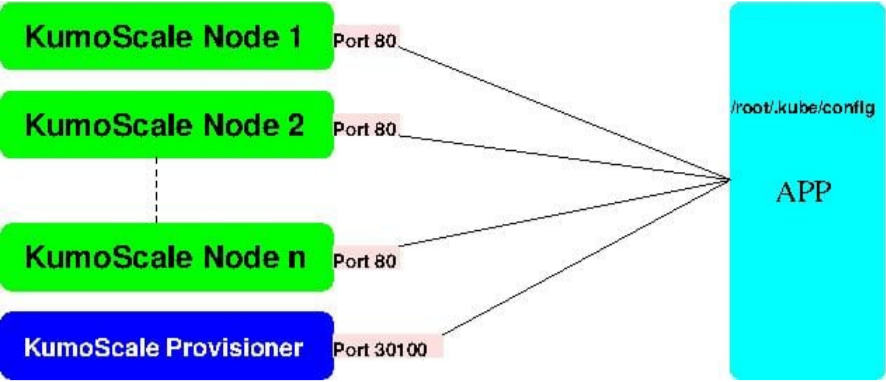
```
kapi = kubernetes.client.CustomObjectsApi() // k8s API handle
```

We use this handle to obtain information about the KumoScale nodes:

```
K8S_ORG    = 'kumoscale.kioxia.com'
K8S_VER    = 'v1'
STORAGENODES = [n['spec']['initMgmtIp'] for n in
    kapi.list_cluster_custom_object(K8S_ORG, K8S_VER, 'storagenodes')['items']]
```

### What is the access mechanism for the metric feeds?

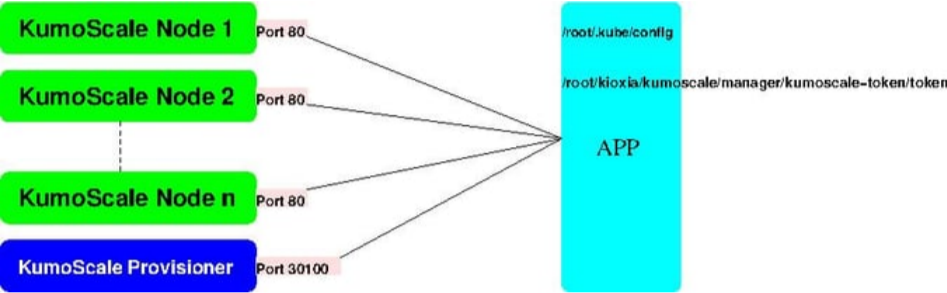
The metrics are made available via a TCP port using the HTTPS protocol.



The metric feeds are part of REST API provided by the KumoScale Provisioner and nodes. The REST API is secured by an access token. The HTTPS communication with the REST API also requires a certificate.

### Acquiring an access token for KumoScale's REST API

[This page](#) provides instructions on how to create an access token. This token will be made available to the app as a file, `/root/.kioxia/kumoscale/manager/kumoscale-token/token`(described [here](#)).

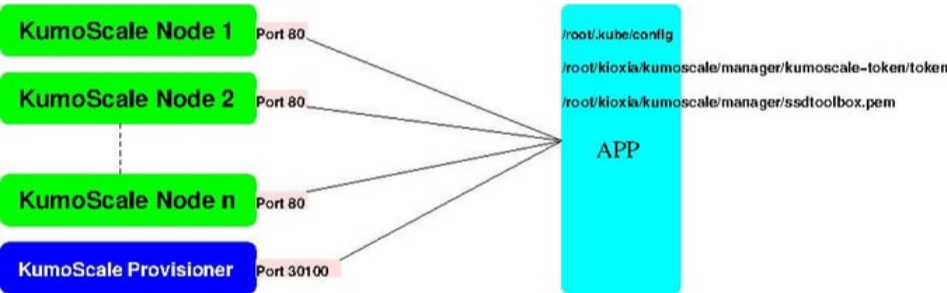


This token is used to create the HTTPS header for the requests.

```
with open("/root/.kioxia/kumoscale/manager/kumoscale-token/token", "r") as fp:
    token = fp.read()
HEADERS = {"Authorization": "Bearer {}".format(token),
           "Content-Type":"application/json"}
```

### Acquiring the TLS certificate

This certificate is required to communicate securely with KumoScale. The certificate file is named `ssdtoolbox.pem`. It is stored in the `/root` directory of a KumoScale node. Contact your KumoScale administrator to obtain a copy of this file. This file is made available to the app as the file `/root/.kioxia/kumoscale/manager/ssdtoolbox.pem` as described [here](#).



The variable `cert` is used to store the location of this file.

```
CERT = "/root/.kioxia/kumoscale/manager/ssdtoolbox.pem"
```

### Access the KumoScale Provisioner Metrics

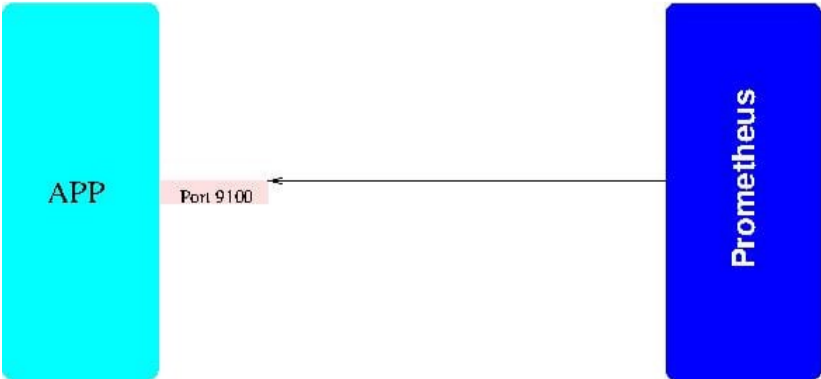
```
request = 'https://{0}:30100/metrics'.format(provisioner)
with requests.get(request, headers=HEADERS, cert=CERT, verify=False) as response:
    if response.ok:
        metrics = response.text
    else:
        # Error processing
```

### Access the KumoScale nodes' Metrics

```
for storagenode in STORAGENODES:
    request = 'https://{0}/SSDAgentServer/NVMEof/v1/metrics'.format(storagenode)
    with requests.get(request, headers=HEADERS, cert=CERT, verify=False) as response:
        if response.ok:
            metrics = response.text
        else:
            # Error processing
```

### Setting up HTTP server to serve the consolidated metric feed

The python module `http` implements a HTTP server. The server will listen on all available network interfaces at port 9100.



```

host_name = socket.gethostname()
server_port = 9100
with http.server.HTTPServer(('0.0.0.0', server_port), Server) as web_server:
    try:
        web_server.serve_forever()
    except ExitException:
        pass

```

This code requires a class called *Server* to handle the HTTP requests and an exception called *ExitException* that is raised when Kubernetes wants to terminate the app.

## HTTP Request Handler

The outline for the class that implements the HTTP request handler is shown below.

```

class Server(http.server.BaseHTTPRequestHandler):
    """
    ...
    Server class to handle http requests from prometheus
    ...
    def do_GET(self):
        """
        ...
        do_GET: method to handle http requests from prometheus
        ...

        # send HTTP protocol related information
        # send provisioner metric feed
        # send nodes' metric feed

```

The class is based on *http.server.BaseHTTPRequestHandler*. It implements a method called `do_GET` that is called when the server receives a HTTP request. The code is shown below.

```

class Server(http.server.BaseHTTPRequestHandler):
    """
    ...
    Server class to handle http requests from prometheus
    ...
    def do_GET(self):
        """
        ...
        do_GET: method to handle http requests from prometheus
        ...

        # send HTTP protocol related information
        self.send_response(200)
        self.send_header("Content-type", "text/plain")
        self.end_headers()

        global PROVISIONER
        global STORAGENODES
        global HEADERS

        # send provisioner metric feed
        request = 'https://{0}/metrics'.format(PROVISIONER)
        with requests.get(request, headers=HEADERS, cert=CERT, verify=False) as response:
            if response.ok:
                metrics = response.text
                self.wfile.write(bytes(metrics, "utf-8"))
            else:
                self.wfile.write(bytes(response.text, "utf-8"))

        # send nodes' metric feed
        for storagenode in STORAGENODES:
            server = storagenode['spec']['initMgmtIp']
            request = 'https://{0}/SSDAgentServer/NVMEof/v1/metrics'.format(server)
            with requests.get(request, headers=HEADERS, cert=CERT, verify=False) as response:
                if response.ok:
                    metrics = response.text
                    self.wfile.write(bytes(metrics, "utf-8"))
                else:
                    self.wfile.write(bytes(response.text, "utf-8"))

```

## Putting It All Together

The final code for the app is shown below.

```

"""
...
    ks-metrics
...

import sys
import os
import re
import socket
import http.server
import requests
import kubernetes
import signal

K8S_ORG    = 'kumoscale.kioxia.com'
K8S_VER    = 'v1'
HEADERS    = None
MYDIR      = '{HOME}/kioxia/kumoscale/manager'.format(HOME=os.getenv("HOME"))
CERT       = '{DIR}/ssdtoolbox.pem'.format(DIR=MYDIR)

PROVISIONER = None
STORAGENODES = []

class Server(http.server.BaseHTTPRequestHandler):
    """
    ...
    Server class to handle http requests from prometheus
    ...
    def do_GET(self):
        """
        ...
        do_GET: method to handle http requests from prometheus

```

```

    ...
    global PROVISIONER
    global STORAGENODES
    global HEADERS
    self.send_response(200)
    self.send_header("Content-type", "text/plain")
    self.end_headers()
    request = 'https://{0}/metrics'.format(PROVISIONER)
    with requests.get(request, headers=HEADERS, cert=CERT, verify=False) as response:
        if response.ok:
            metrics = response.text
            self.wfile.write(bytes(metrics, "utf-8"))
        else:
            self.wfile.write(bytes(response.text, "utf-8"))
    for storagenode in STORAGENODES:
        server = storagenode['spec']['initMgmtIp']
        request = 'https://{0}/SSDAgentServer/NVMEof/v1/metrics'.format(server)
        with requests.get(request, headers=HEADERS, cert=CERT, verify=False) as response:
            if response.ok:
                metrics = response.text
                self.wfile.write(bytes(metrics, "utf-8"))
            else:
                self.wfile.write(bytes(response.text, "utf-8"))

class ExitException(Exception):
    pass

def handler(signum, frame):
    print('Recieved SIGTERM', file=sys.stderr)
    raise ExitException()

def main():
    """
        main
    """

    signal.signal(signal.SIGTERM, handler)

    global PROVISIONER
    global STORAGENODES
    global HEADERS
    print("Initializing KumoScale Metrics Server", file=sys.stderr)
    kubernetes.config.load_kube_config()
    configuration = kubernetes.client.Configuration()
    PROVISIONER = re.match(r'^http(s)?://(\d+\.\d+\.\d+\.\d+):\d+$',
        configuration.host).groups()[1] + ':30100'
    kapi = kubernetes.client.CustomObjectsApi()
    STORAGENODES = kapi.list_cluster_custom_object(K8S_ORG, K8S_VER, 'storagenodes')['items']
    with open("/root/kioxia/kumoscale/manager/kumoscale-token/token", "r") as fp:
        token = fp.read()
    HEADERS = {"Authorization":
        "Bearer {0}".format(token),
        "Content-Type": "application/json"}

    host_name = socket.gethostname()
    server_port = 9100
    print("Starting KumoScale Metrics Server", file=sys.stderr)
    with http.server.HTTPServer(('0.0.0.0', server_port), Server) as web_server:
        print("Server started at http://{0}:{1}".format(host_name, server_port), file=sys.stderr)
        try:
            web_server.serve_forever()
        except ExitException:
            pass
    print("Server stopped at http://{0}:{1}".format(host_name, server_port), file=sys.stderr)

if __name__ == '__main__':
    main()

```

## Setting up the container

The container needs three pieces of information to run, [the KumoScale REST API token](#), [the TLS certificate for HTTPS connection to the KumoScale REST API](#) and [the configuration file for contacting the KumoScale's internal Kubernetes cluster](#) (where the KumoScale Provisioner resides).

Create an empty directory and store [the Python script for the app](#) in that directory. (Re)name the script *ks-metrics.py*. Execute the rest of the steps in that directory.

### REST API Token

This token is passed to our app via KumoScale Secret mechanism. Create a file called *kustomization.yaml* and put the following text in that file.

```

apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
secretGenerator:
- literals:
  - token=<YOUR TOKEN>
  name: kumoscale-token
  type: Opaque

```

Replace <YOUR TOKEN> with [the actual token string](#).

Next, apply the customization.

```
kubectl apply -k .
```

Finally, list the customization information,

```

$ kubectl kustomize .
apiVersion: v1
data:
  token: |
    ZXlKaGJHY2lPaUpJVXpJMU5pSjkuZXlKemRXSWlPaUpoWkcxcGJpSXNJbWp2YkdWek1qcG
    JJBEPQVEVWZlFVUk5TVTRpWFN3aWFXRjBJam94TmpNMU5ESTRPVGV5ZlEuT1FGTy1xS2Vm

```

```
WjBUTG1ZVjNhbX1DN3dPamtYc2J0d3B0cklKamxfcnBCcw==
kind: Secret
metadata:
  name: kumoscale-token-btt82h5ttc
type: Opaque
```

Note down the name of the secret (underlined in the command output). You will need it when you create the container's yaml file for the app.

### TLS Certificate

Copy [the certificate](#) named *ssdtoolbox.pem*.

### KumoScale K8S Configuration

Copy [it](#) in a file called *config*.

Your directory should now contain the following files.

```
config  ks-metrics.py  kustomization.yaml  ssdtoolbox.pem
```

### Build the Docker Image

Create a file called *Dockerfile* and put the following text in the file.

```
# syntax=docker/dockerfile:1
FROM fedora:34
USER root
WORKDIR /kumoscale-metrics
RUN dnf upgrade -y
RUN dnf install python3-pip python3-kubernetes -y
RUN pip3 install json2html
COPY ks-metrics.py ks-metrics.py
EXPOSE 9100
CMD [ "python3", "ks-metrics.py"]
```

Create the container image using the command:

```
docker build --tag kumoscale-metrics .
```

Verify that the the container image was created by running the command below:

```
docker image ls
```

Make sure that the image *kumoscale-metrics* appears in the output.

### Deploying the App

Finally we are ready to launch the app. Create a file called *ks-metrics.yaml* and add the following content to it:

```
---
apiVersion: v1
kind: Pod
metadata:
  name: kumoscale-metrics
spec:
  containers:
    - name: kumoscale-metrics
      image: kumoscale-metrics
      imagePullPolicy: Never
      ports:
        - containerPort: 9100
          name: "metrics-server"
      volumeMounts:
        - mountPath: "/root/kioxia/kumoscale/manager/ssdtoolbox.pem"
          name: ks-provisioner-tls
          readOnly: true
        - mountPath: "/root/.kube/config"
          name: ks-k8s-config
          readOnly: true
        - name: kumoscale-token
          mountPath: "/root/kioxia/kumoscale/manager/kumoscale-token"
          readOnly: true
  volumes:
    - name: ks-provisioner-tls
      hostPath:
        path: "/kumoscale-metrics/ssdtoolbox.pem"
        type: File
    - name: ks-k8s-config
      hostPath:
        path: "/kumoscale-metrics/config"
        type: File
    - name: kumoscale-token
      secret:
        secretName: <SECRET NAME>
  hostNetwork: true
  dnsPolicy: Default
```

Replace <SECRET NAME> with the name of the secret you obtained [previously](#).

Launch the app:

```
kubect1 create -f ks-metrics.yaml
```

Make sure that your app is running:

```
$ kubectl get pods kumoscale-metrics
NAME          READY   STATUS    RESTARTS   AGE
kumoscale-metrics  1/1     Running   0           6m17h
```

To see more details about your app:

```
$ minikube kubectl describe pods kumoscale-metrics
Name:          kumoscale-metrics
Namespace:     default
Priority:       0
Node:          minikube/192.168.39.31
Start Time:    Mon, 20 Dec 2021 18:17:26 -0500
Labels:        <none>
Annotations:   <none>
Status:        Running
IP:            192.168.39.31
IPs:
  IP: 192.168.39.31
Containers:
  kumoscale-metrics:
    Container ID:  docker://3e0daee65d379e4fa0fe415611f14a460b1ecf16b1cce1057b4112deea4806b2
    Image:         kumoscale-metrics
    Image ID:      docker://sha256:390edeb18c66b2684ac3fdd4c8528ccda70e1a3135d4412263f66a364919d853
    Port:          9100/TCP
    Host Port:     9100/TCP
    State:         Running
      Started:     Mon, 20 Dec 2021 18:17:27 -0500
    Ready:         True
    Restart Count: 0
    Environment:   <none>
    Mounts:
      /root/.kube/config from ks-k8s-config (ro)
      /root/kioxia/kumoscale/manager/kumoscale-token from kumoscale-token (ro)
      /root/kioxia/kumoscale/manager/ssdtoolbox.pem from ks-provisioner-tls (ro)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-8tvq4 (ro)
Conditions:
  Type            Status
  Initialized      True
  Ready           True
  ContainersReady True
  PodScheduled    True
Volumes:
  ks-provisioner-tls:
    Type:      HostPath (bare host directory volume)
    Path:      /kumoscale-metrics/ssdtoolbox.pem
    HostPathType: File
  ks-k8s-config:
    Type:      HostPath (bare host directory volume)
    Path:      /kumoscale-metrics/config
    HostPathType: File
  kumoscale-token:
    Type:      Secret (a volume populated by a Secret)
    SecretName: kumoscale-token-btt82h5ttc
    Optional:   false
  kube-api-access-8tvq4:
    Type:      Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds: 3607
    ConfigMapName: kube-root-ca.crt
    ConfigMapOptional: <nil>
    DownwardAPI: true
QoS Class:           BestEffort
Node-Selectors:      <none>
Tolerations:         node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                     node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type     Reason      Age   From          Message
  ----     -
  Normal   Scheduled   6m6s  default-scheduler  Successfully assigned default/kumoscale-metrics to minikube
  Normal   Pulled      6m6s  kubelet        Container image "kumoscale-metrics" already present on machine
  Normal   Created     6m6s  kubelet        Created container kumoscale-metrics
  Normal   Started     6m6s  kubelet        Started container kumoscale-metrics
```

Note the IP of the container (underlined in the above output).

## Connect to your app using a web browser

Enter the text `http://<IP ADDRESS>:9100` in your web browser. <IP ADDRESS> is [your app's IP address](#). The browser window will show the text of the metric values.

## Connecting to Prometheus

Open the Prometheus configuration file (usually `/etc/prometheus/prometheus.yml`) in an editor. Add the following text in the `scrape_configs` section.

```
- job_name: 'kumoscale-metrics'
  static_configs:
    - targets: ['<IP ADDRESS>:9100']
```

<IP ADDRESS> is [your app's IP address](#).

