SCGC Assignment

by

Alex Apostolescu
aapostolescu

O carte citită e un grătar pierdut.
Jador

1 Task 0 - Cluster setup

Task 0 is accomplished using `./deploy-cluster.sh` which uses `kind` to create a node. I also included dependencies (`helm` and `slowhttptest`) for other tasks in this script to ensure the work environment is suitable.

```
student@lab-kubernetes:~/scgc$ ./deploy-cluster.sh
Helm v3.12.0 is already latest
Creating cluster "kind" ...
 ✓ Ensuring node image (kindest/node:v1.23.4) 🖼
 ✓ Preparing nodes 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Have a nice day! 👋
node/kind-control-plane condition met
NAMESPACE            NAME                                         READY   STATUS    RESTARTS   AGE
kube-system          coredns-64897985d-2j9bh                      0/1     Pending   0          14s
kube-system          coredns-64897985d-86twc                      0/1     Pending   0          14s
kube-system          etcd-kind-control-plane                      1/1     Running   0          31s
kube-system          kindnet-4v5kw                                1/1     Running   0          14s
kube-system          kube-apiserver-kind-control-plane            1/1     Running   0          31s
kube-system          kube-controller-manager-kind-control-plane   1/1     Running   0          31s
kube-system          kube-proxy-vqgv8                             1/1     Running   0          14s
kube-system          kube-scheduler-kind-control-plane            1/1     Running   0          31s
local-path-storage   local-path-provisioner-5ddd94ff66-ftf7n      0/1     Pending   0          14s
student@lab-kubernetes:~/scgc$
```

After deploying the cluster, the script will wait up to 5 minutes for the nodes to be Ready.

2 Task 1 - Nginx

The base `nginx` is deployed using `./nginx/deploy-nginx.sh` which creates the configmaps for the main route and the metrics route, then applies the deployment, and starts a service that makes `nginx` available to the host.

```
student@lab-kubernetes:~/scgc/nginx$ ./deploy-nginx.sh
configmap/nginx-html-config created
configmap/nginx-metrics-config created
deployment.apps/nginx created
service/nginx created
pod/nginx-8ddbb6569-hndt9 condition met


Test:
<html><body>Not everybody can be bombardier!</body></html>
Active connections: 1
server accepts handled requests
 2 2 2
Reading: 0 Writing: 1 Waiting: 0
student@lab-kubernetes:~/scgc/nginx$
```

Afterward, the script waits for the pods to be up and runs tests for both routes by running `curl` on `http://172.18.0.2:30080` and `http://172.18.0.2:30088/metrics`, where `172.18.0.2` is the node IP.

3 Task 2 - Promexporter

Deploying `promexporer` follows the same path: run the deployment, expose it via a service, wait for it to be online, and test it.

```
student@lab-kubernetes:~/scgc/promexporter$ ./deploy-promexporter.sh
deployment.apps/promexporter created
service/promexporter created
pod/promexporter-654d546d5d-8b976 condition met


Test:
<!DOCTYPE html>
                        <title>NGINX Exporter</title>
                        <h1>NGINX Exporter</h1>
                        <p><a href="/metrics">Metrics</a></p>student@lab-kubernetes:~/scgc/promexporter$
```

Testing is realized via curl on `http://172.18.0.2:30113`.

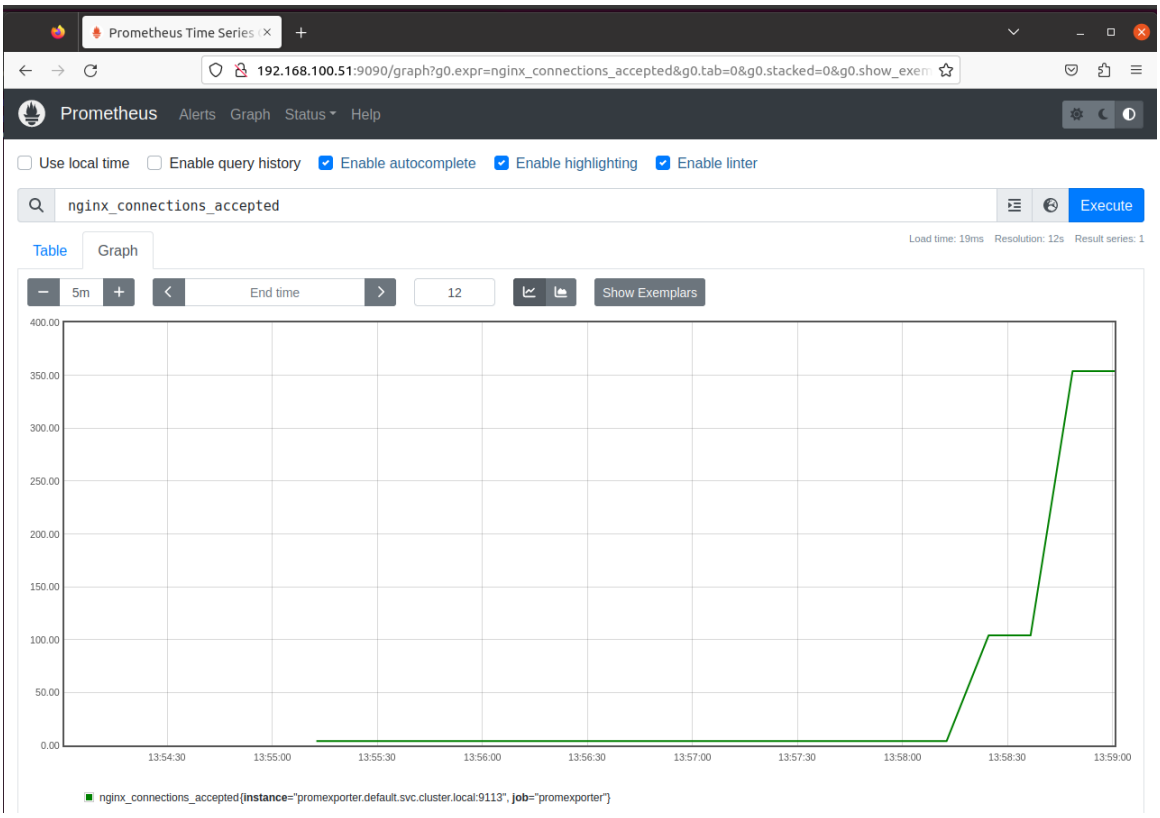
4 Task 3 - Prometheus

`Prometheus` is deployed in a namespace of its own, `monitoring` that will be created if it is missing.

After ensuring the namespace is present, `helm` is used to deploy `Prometheus` to this namespace and uses `values.yaml` to configure it. The only required configuration for our task are the scrape interval, the metrics path, and the target.

```
student@lab-kubernetes:~/scgc/prometheus$ ./deploy-prometheus.sh
registry.k8s.io/kube-state-metrics/kube-state-metrics:v2.8.0
namespace/monitoring created
"prometheus-community" already exists with the same configuration, skipping
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "bitnami" chart repository
Update Complete. ⎈Happy Helming!⎈
NAME: prometheus
LAST DEPLOYED: Fri May 12 13:53:32 2023
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The Prometheus server can be accessed via port 80 on the following DNS name from within your cluster:
prometheus-server.monitoring.svc.cluster.local


Get the Prometheus server URL by running these commands in the same shell:
  export POD_NAME=$(kubectl get pods --namespace monitoring -l "app=prometheus,component=server" -o jsonpath="{.items[0].metadata.name}")
  kubectl --namespace monitoring port-forward $POD_NAME 9090


The Prometheus alertmanager can be accessed via port  on the following DNS name from within your cluster:
prometheus-%!s(<nil>).monitoring.svc.cluster.local


Get the Alertmanager URL by running these commands in the same shell:
  export POD_NAME=$(kubectl get pods --namespace monitoring -l "app=prometheus,component=" -o jsonpath="{.items[0].metadata.name}")
  kubectl --namespace monitoring port-forward $POD_NAME 9093
#################################################################################
######   WARNING: Pod Security Policy has been disabled by default since    #####
######            it deprecated after k8s 1.25+. use                        #####
######            (index .Values "prometheus-node-exporter" "rbac"          #####
###### .          "pspEnabled") with (index .Values                         #####
######            "prometheus-node-exporter" "rbac" "pspAnnotations")       #####
######            in case you still need it.                                #####
#################################################################################


The Prometheus PushGateway can be accessed via port 9091 on the following DNS name from within your cluster:
prometheus-prometheus-pushgateway.monitoring.svc.cluster.local


Get the PushGateway URL by running these commands in the same shell:
  export POD_NAME=$(kubectl get pods --namespace monitoring -l "app=prometheus-pushgateway,component=pushgateway" -o jsonpath="{.items[0].metadata.name}")
  kubectl --namespace monitoring port-forward $POD_NAME 9091

For more information on running Prometheus, visit:
https://prometheus.io/
pod/prometheus-server-6cbbd55d6-tlpkh condition met
student@lab-kubernetes:~/scgc/prometheus$ Forwarding from 0.0.0.0:9090 -> 9090
```

Functionality is checked via the `Prometheus` UI, running queries on `nginx` statistics. This implies that for `Prometheus` to work correctly, the first 2 tasks should also be up and running.

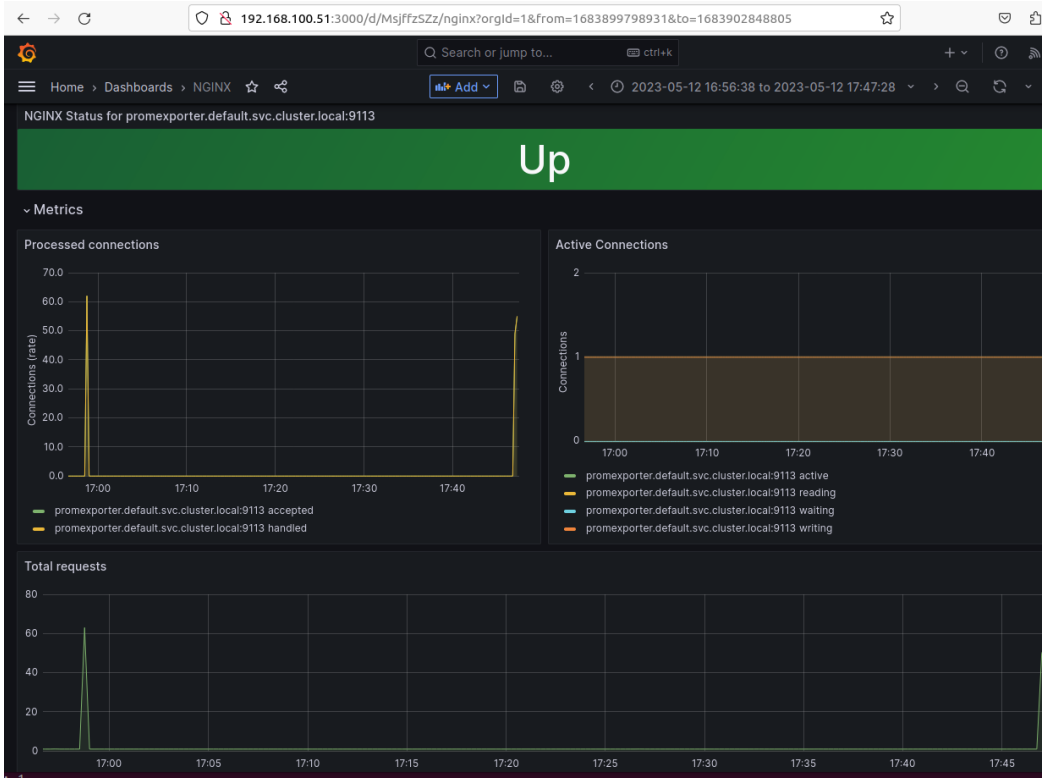

Prometheus Time Series
192.168.100.51:9090/graph?g0.expr=nginx_connections_accepted&g0.tab=0&g0.stacked=0&g0.show_exem
Prometheus
Alerts
Graph
Status
Help
Use local time
Enable query history
Enable autocomplete
Enable highlighting
Enable linter
nginx_connections_accepted
Execute
Load time: 19ms
Resolution: 12s
Result series: 1
Table
Graph
5m
End time
12
Show Exemplars
400.00
350.00
300.00
250.00
200.00
150.00
100.00
50.00
0.00
13:54:30
13:55:00
13:55:30
13:56:00
13:56:30
13:57:00
13:57:30
13:58:00
13:58:30
13:59:00
nginx_connections_accepted{instance="promexporter.default.svc.cluster.local:9113", job="promexporter"}

5 Task 4 - Grafana

`Grafana` is somewhat a similar story to `Prometheus` - lean back and let `helm` do the work. As before, the main configuration is in `values.yaml`, where I spent the better part of a day making the data source and dashboard load on startup, so this service will work without additional configuration.

```
student@lab-kubernetes:~/scgc/grafana$ ./deploy-grafana.sh
namespace/monitoring unchanged
"bitnami" already exists with the same configuration, skipping
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "bitnami" chart repository
Update Complete. ⎈Happy Helming!⎈
NAME: grafana
LAST DEPLOYED: Fri May 12 14:01:02 2023
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: grafana
CHART VERSION: 8.4.2
APP VERSION: 9.5.2

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

1. Get the application URL by running these commands:
    echo "Browse to http://127.0.0.1:8080"
    kubectl port-forward svc/grafana 8080:3000 &

2. Get the admin credentials:

    echo "User: admin"
    echo "Password: $(kubectl get secret grafana-admin --namespace monitoring -o jsonpath="{.data.GF_SECURITY_ADMIN_PASSWORD}" | base64 -d)"
pod/grafana-998c48454-pn5h4 condition met
Credentials: admin/WGth4rSEE8
student@lab-kubernetes:~/scgc/grafana$ Forwarding from 0.0.0.0:3000 -> 3000
```

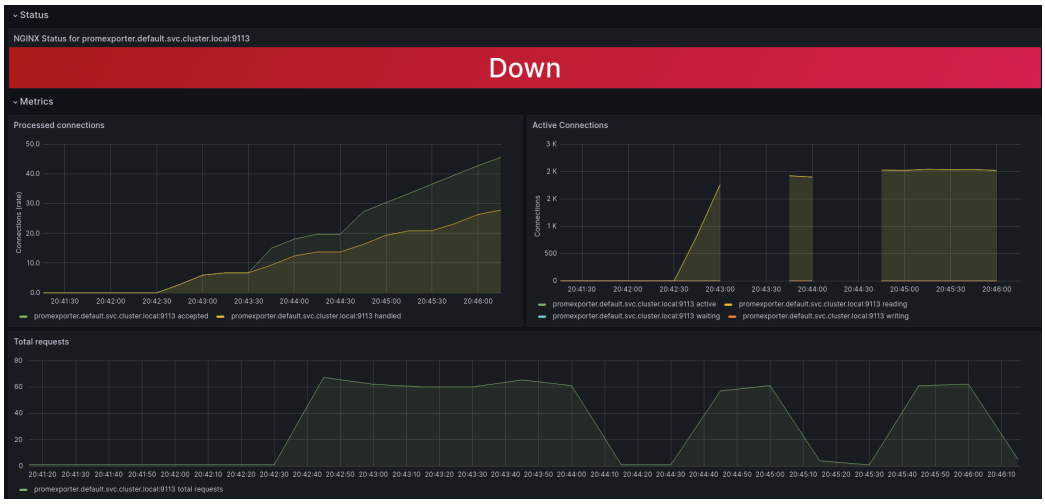
No better way to test an UI observability platform than by checking the UI after sending a bunch of requests to `nginx` like we did to test `task 1`.



6 Task 5 - Slowhttptest

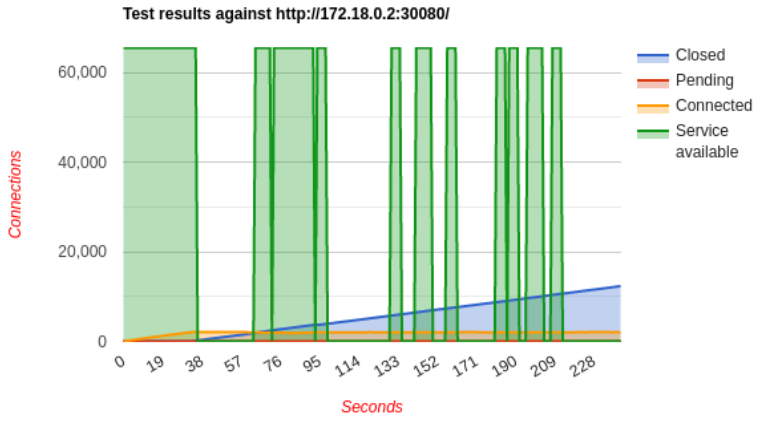
This task serves to demonstrate how our deployment does against a Denial of Service attack. Keeping the server under a heavy load of `300 connections` every second for `240 seconds` led to the server being unresponsive for a few intervals.

`Grafana` reported the server down and no statistics were available in those intervals.



The downtime is better highlighted in the output file of `slowhttptest`.

Test parameters	
Test type	SLOW HEADERS
Number of connections	65539
Verb	GET
Content-Length header value	4096
Extra data max length	52
Interval between follow up data	1 seconds
Connections per seconds	300
Timeout for probe connection	3
Target test duration	240 seconds
Using proxy	no proxy

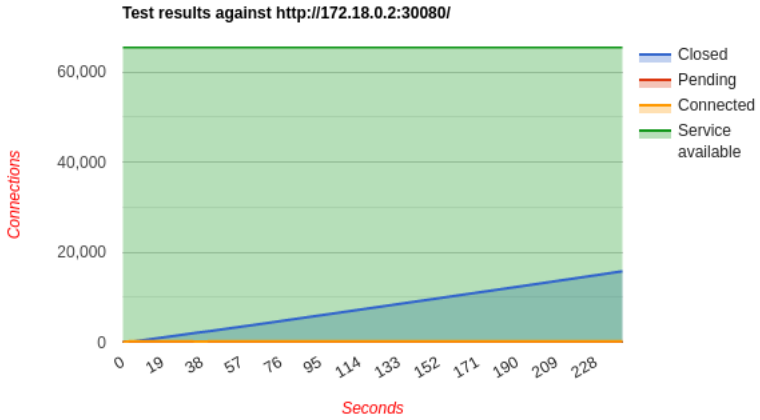


7 Task 6 - Secure Nginx

Knowing the attack model makes it easy to defend against it. Having built-in security features in both `nginx` [1] and `kubernetes` makes it even better.

The obvious ways to combat this were: caching `nginx` responses and scaling the service horizontally, both of which require extra resources. I opted instead for limiting the number of requests per client, limiting the number of connections per client, and closing slow connections, which do not bring additional costs and are well-suited for most applications.

Test parameters	
Test type	SLOW HEADERS
Number of connections	65539
Verb	GET
Content-Length header value	4096
Extra data max length	52
Interval between follow up data	1 seconds
Connections per seconds	300
Timeout for probe connection	3
Target test duration	240 seconds
Using proxy	no proxy



[1] `https://www.nginx.com/blog/mitigating-ddos-attacks-with-nginx-and-nginx-plus/`

8 Mentions

To combat deployment failures caused by slow internet speed, deployment scripts download the required docker image and load it to the cluster.

Due to low space on the machine, I commented the majority of these additions, and only kept those that posed a problem on numerous redeploys e.g. `registry.k8s.io/kube-state-metrics/kube-state-metrics:v2.8.0`.