Homework SCGC

Setup :

```
student@hw-brunner-titouan:~/work$ virsh net-dhcp-leases labvms
 Expiry Time          MAC address        Protocol   IP address          Hostname        Client ID or DUID
-------------------------------------------------------------------------------------------------------------------------------------------------
 2024-05-14 12:19:08   52:54:00:00:05:01   ipv4       192.168.100.51/24   lab-kubernetes   ff:b5:5e:67:ff:00:02:00:00:ab:11:5b:11:44:e7:c7:a1:7d:fb
```

Connect to the IP address and resize :

```
student@lab-kubernetes:~$ sudo growpart /dev/sda 2
CHANGED: partition=2 start=4096 old: size=16771072 end=16775168 new: size=33550303 end=33554399
student@lab-kubernetes:~$ sudo resize2fs /dev/sda2
resize2fs 1.45.5 (07-Jan-2020)
Filesystem at /dev/sda2 is mounted on /; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 2
The filesystem on /dev/sda2 is now 4193787 (4k) blocks long.

student@lab-kubernetes:~$ V
V: command not found
student@lab-kubernetes:~$ df -h | grep /dev/sda2
/dev/sda2        16G  1.4G   14G   9% /
```

Step 1 :

Deployment of a single node cluster with Kind command

```
student@lab-kubernetes:~$ kind create cluster
Creating cluster "kind" ...
 √ Ensuring node image (kindest/node:v1.29.2) 🖼
 √ 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

Not sure what to do next? 😅  Check out https://kind.sigs.k8s.io/docs/user/quick-start/
student@lab-kubernetes:~$ kubectl get nodes
NAME                 STATUS   ROLES           AGE   VERSION
kind-control-plane   Ready    control-plane   94s   v1.29.2
student@lab-kubernetes:~$ kubectl cluster-info
Kubernetes control plane is running at https://127.0.0.1:45463
CoreDNS is running at https://127.0.0.1:45463/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```

Ngninx deployment :

```
student@lab-kubernetes:~$ curl http://172.18.0.2:30080
<html><body>Love doing my SCGC Homework!</body></html>
student@lab-kubernetes:~$ curl http://172.18.0.2:30088/metrics
Active connections: 1
server accepts handled requests
 5 5 5
Reading: 0 Writing: 1 Waiting: 0
student@lab-kubernetes:~$
```

Explanation :

I reuse the lab about “Container orchestration with Kubernetes” and recreate the files from the lab. So I look at the lab and use it to configure nginx.yaml files with asked characteristics.

```
student@lab-kubernetes:~$ ls
nginx-config.yaml  nginx-deployment.yaml  nginx-html.yaml  nginx-service.yaml
```

```
student@lab-kubernetes:~$ cat nginx-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-conf
data:
  default.conf: |
    server {
      listen       80;
      server_name  localhost;


      location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
      }
    }

    server {
        listen      8080;
        server_name metrics;

        location /metrics {
           stub_status;
        }
    }
```

In nginx-config.yaml I add a server “metrics” to provide metrics about itself on location /metics and on port 8080 and use stub_status.

```
student@lab-kubernetes:~$ cat nginx-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: gitlab.cs.pub.ro:5050/scgc/cloud-courses/nginx:latest
        ports:
        - containerPort: 80
        - containerPort: 8080
        volumeMounts:
        - name: nginx-html-vol
          mountPath: "/usr/share/nginx/html/index.html"
          subPath: "index.html"
        - name: nginx-conf-vol
          mountPath: "/etc/nginx/conf.d/default.conf"
          subPath: "default.conf"
      volumes:
      - name: nginx-html-vol
        configMap:
          name: nginx-html
          items:
          - key: "index.html"
            path: "index.html"
      - name: nginx-conf-vol
        configMap:
          name: nginx-conf
          items:
          - key: "default.conf"
            path: "default.conf"
```

In the nginx-deployment.yaml I just add the containerPort 8080 to get the /metrics work well.

```
student@lab-kubernetes:~$ cat nginx-html.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-html
data:
 index.html: |
    <html><body>Love doing my SCGC Homework!</body></html>
```

Here I just change the text inside index.html

```
student@lab-kubernetes:~$ cat nginx-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - protocol: TCP
      name: index
      port: 80
      targetPort: 80
      nodePort: 30080
    - port : 8080
      name : metrics
      targetPort: 8080
      nodePort: 30088
```

I modify this file to deploy the nginx service that is exposed on port 80 inside the cluster and port 30080 outside the cluster. I also named the service nginx as asked.

I add a final modification in nginx-service.yaml to expose the metrics endpoint on the same nginx service on port 8080 inside the cluster and port 30088 outside the cluster.

Step 2 :

```
student@lab-kubernetes:~$ cat nginx-prometheus-exporter-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: promexporter
  labels:
    app.kubernetes.io/name: promexporter
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/name: promexporter
  template:
    metadata:
      labels:
        app.kubernetes.io/name: promexporter
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "9113"
    spec:
      containers:
      - name: nginx-exporter
        image: nginx/nginx-prometheus-exporter:latest
        args:
        - -nginx.scrape-uri=http://10.96.172.29:8080/metrics
        ports:
        - containerPort: 9113

student@lab-kubernetes:~$ cat nginx-prometheus-exporter-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: promexporter
spec:
  selector:
    app.kubernetes.io/name: promexporter
  ports:
    - protocol: TCP
      port: 9113
      targetPort: 9113

student@lab-kubernetes:~$
```

Here are files to configure prometheux exporter named here “promexporter”. It’s configured on port 9113 and I use the IP of nginx that I can find with “kubectl get services”kub

```
student@lab-kubernetes:~$ kubectl get deployments
NAME                   READY   UP-TO-DATE   AVAILABLE   AGE
nginx                  1/1     1            1           2d
nginx-prometheus-exporter   1/1     1            1           47h
student@lab-kubernetes:~$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
nginx-7ff7ffccf9-r4n9f                  1/1     Running   0          2d
nginx-prometheus-exporter-766d945c7f-fclrk   1/1     Running   0          47h
student@lab-kubernetes:~$ kubectl get srv
error: the server doesn't have a resource type "srv"
student@lab-kubernetes:~$ kubectl get service
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP                      2d
nginx         NodePort    10.96.172.29    <none>        80:30080/TCP,8080:30088/TCP  2d
promexporter  ClusterIP   10.96.255.152   <none>        9113/TCP                     47h
student@lab-kubernetes:~$ vim nginx-prometheus-exporter-deployment.yaml
student@lab-kubernetes:~$ kubectl apply -f nginx-prometheus-exporter-deployment.yaml
deployment.apps/nginx-prometheus-exporter configured
student@lab-kubernetes:~$ kubectl run curlpod --image=appropriate/curl --restart=Never -- sleep 3600
pod/curlpod created
student@lab-kubernetes:~$ kubectl exec -it curlpod -- curl -s http://promexporter.default.svc.cluster.local:9113/metrics
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0
go_gc_duration_seconds{quantile="1"} 0
go_gc_duration_seconds_sum 0
go_gc_duration_seconds_count 0
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 10
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.21.5"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
```

Then I run the http link on port 9113 and location metrics to see if it works in a pod that I create (and delete it after).

```
# TYPE process_virtual_memory_max_bytes gauge
process_virtual_memory_max_bytes 1.8446744073709552e+19
# HELP promhttp_metric_handler_requests_in_flight Current number of scrapes being served.
# TYPE promhttp_metric_handler_requests_in_flight gauge
promhttp_metric_handler_requests_in_flight 1
# HELP promhttp_metric_handler_requests_total Total number of scrapes by HTTP status code.
# TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 0
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
```

```
student@lab-kubernetes:~$ kubectl get pod
NAME                                    READY   STATUS    RESTARTS   AGE
nginx-7ff7ffccf9-r4n9f                  1/1     Running   0          5d3h
nginx-prometheus-exporter-65d8454d98-gc5wx   1/1     Running   0          3h53m
promexporter-757f898ff-ngqtt            1/1     Running   0          3h27m
student@lab-kubernetes:~$ kubectl get services
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
kubernetes     ClusterIP   10.96.0.1       <none>        443/TCP                      5d3h
nginx          NodePort    10.96.172.29    <none>        80:30080/TCP,8080:30088/TCP  5d3h
promexporter   ClusterIP   10.96.255.152   <none>        9113/TCP                     5d2h
student@lab-kubernetes:~$
```

We can see that Prometheus exporter is well deployed.

Step 3 :

First I need to install Helm.

To do that :

- Get the zip file : wget https://github.com/helm/helm/releases/download/v3.15.1/helm-v3.15.1-linux-386.tar.gz.asc
- Extract the file : tar -zxvf helm-v3.15.1-linux-386.tar.gz
- Execute the file : ./get-hem.sh

```
student@lab-kubernetes:~$ ./get_helm.sh
Helm v3.15.1 is available. Changing from version v3.15.0-rc.2.
Downloading https://get.helm.sh/helm-v3.15.1-linux-amd64.tar.g
Verifying checksum ... Done.
Preparing to install helm into /usr/local/bin
helm installed into /usr/local/bin/helm
```

Then I need to create the monitoring space.

```
student@lab-kubernetes:~$ kubectl create namespace monitoring
Error from server (AlreadyExists): namespaces "monitoring" already exists
```

(because already done)

Then I have to deploy Prometheus using Helm. To do that I add the Prometheus Helm chart repository :

```
student@lab-kubernetes:~$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
"prometheus-community" already exists with the same configuration, skipping
student@lab-kubernetes:~$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
Update Complete. ⎈Happy Helming!⎈
```

After that I install Prometheus in the monitoring namespace created before.

```
student@lab-kubernetes:~$ helm install prometheus prometheus-community/prometheus --namespace monitoring
Error: INSTALLATION FAILED: cannot re-use a name that is still in use
```

After that I can port-forward the Prometheus server :

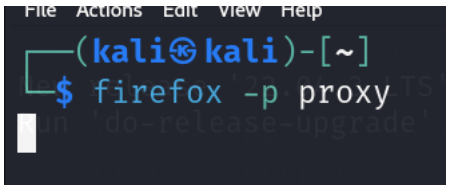
```
student@lab-kubernetes:~$ kubectl port-forward -n monitoring svc/prometheus-server 9090:80
Forwarding from 127.0.0.1:9090 → 9090
Forwarding from [::1]:9090 → 9090
```

For now it doesn't work because it still needs configurations.

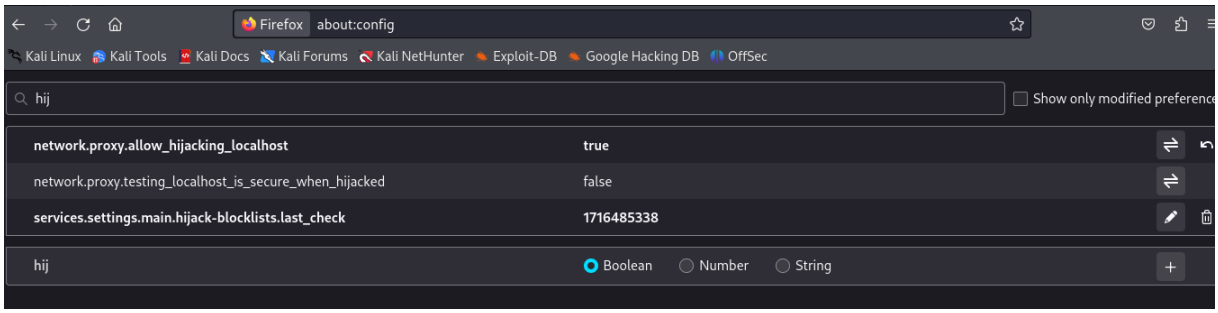
I need to create a ssh bridge to the VM that's open on the proxy's port, to do that I use : "ssh -fN -D 12345 student@192.168.100.51 -J openstack-10.9.4.21"

And finally, I need to configure the proxy of Firefox so I could access locolhost:9090

To do it, in a prompt :



That opens a firefox window in wich I can configure proxy manually by writing "about:config" and trust "network.proxy.allow_hijacking_localhost". Then I configure the proxy on port 12345 for localhost (to use ssh bridge).



I also modify the file values.yaml and add that to configure Prometheus :

To do that first I use this command :

```
student@lab-kubernetes:~$ helm show values prometheus-community/prometheus > values.yaml
```

Then I modify the values.yaml file

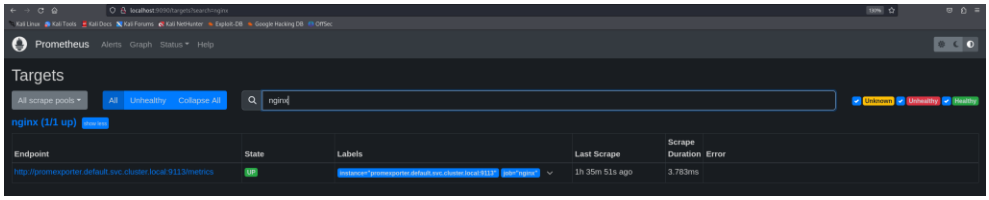
```
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
  - job_name: 'nginx'
    static_configs:
      - targets: ['promexporter.default.svc.cluster.local:9113']
```

After that I redeploy prometheus with the command “helm upgrade prometheus prometheus-community/prometheus --namespace monitoring -f values.yaml”

And now I can use Prometheus on “localhost:9090”.

```
student@lab-kubernetes:~$ kubectl port-forward -n monitoring svc/prometheus-server 9090:80
Forwarding from 127.0.0.1:9090 → 9090
Forwarding from [::1]:9090 → 9090
Handling connection for 9090
Handling connection for 9090
Handling connection for 9090
```

I verify that it works by verifying scrape target is good : nginx is on Prometheus.



Also nginx_connections_accepted :

And graph is updating :

```
student@lab-kubernetes:~$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
curlpod                                 1/1     Running   0          15m
curlpod2                                1/1     Running   0          9m15s
nginx-7ff7ffccf9-r4n9f                  1/1     Running   0          2d8h
nginx-prometheus-exporter-65d8454d98-g7t4r   1/1     Running   0          8h
```

I create 2 pods to test it.

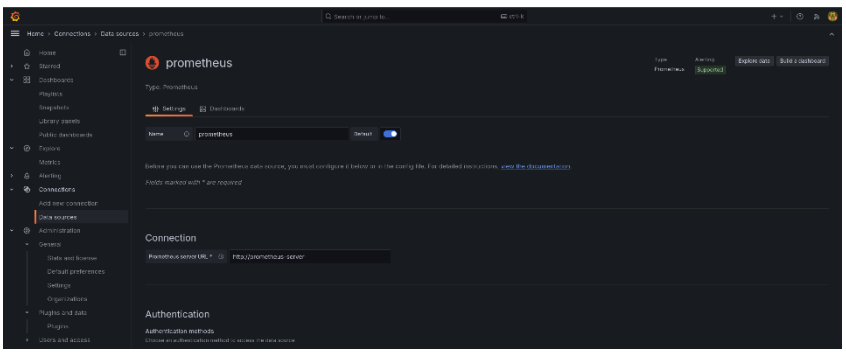
Step 4 :

I install Grafana with commands from helm as previously with Prometheus (but use bitnami instead of for “helm repo add” (don’t really know why I did that….) :

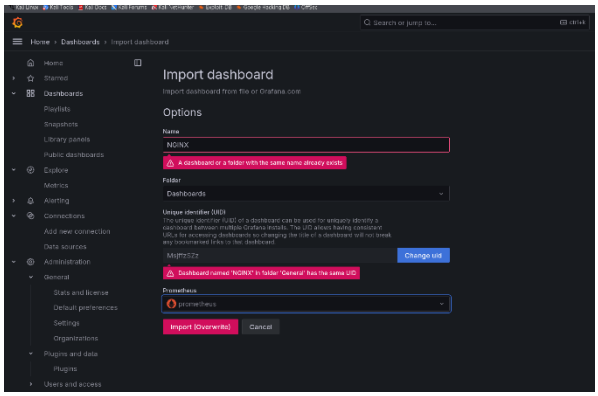
```
student@lab-kubernetes:~$ helm repo add bitnami https://charts.bitnami.com/bitnami
"bitnami" has been added to your repositories
student@lab-kubernetes:~$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "bitnami" chart repository
Update Complete. ⎈Happy Helming!⎈
student@lab-kubernetes:~$ helm install grafana bitnami/grafana --namespace monitoring
NAME: grafana
LAST DEPLOYED: Thu May 23 21:35:24 2024
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: grafana
CHART VERSION: 11.2.1
APP VERSION: 11.0.0

** 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)"
# Note: Do not include grafana.validateValues.database here. See https://github.com/bitnami/charts/issues/20629


WARNING: There are "resources" sections in the chart not set. Using "resourcesPreset" is not recommended for production. For production installations, please set the following values according to your work
eeds:
  - grafana.resources
+info https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
student@lab-kubernetes:~$ kubectl get svc -n monitoring
NAME                                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
grafana                              ClusterIP   10.96.33.184    <none>        3000/TCP   48s
prometheus-alertmanager              ClusterIP   10.96.179.3     <none>        9093/TCP   2d9h
prometheus-alertmanager-headless     ClusterIP   None            <none>        9093/TCP   2d9h
prometheus-kube-state-metrics        ClusterIP   10.96.235.43    <none>        8080/TCP   2d9h
prometheus-prometheus-node-exporter  ClusterIP   10.96.167.66    <none>        9100/TCP   2d9h
prometheus-prometheus-pushgateway    ClusterIP   10.96.224.240   <none>        9091/TCP   2d9h
prometheus-server                    ClusterIP   10.96.51.183    <none>        80/TCP     2d9h
```

Then I port-forward on port 3000 :

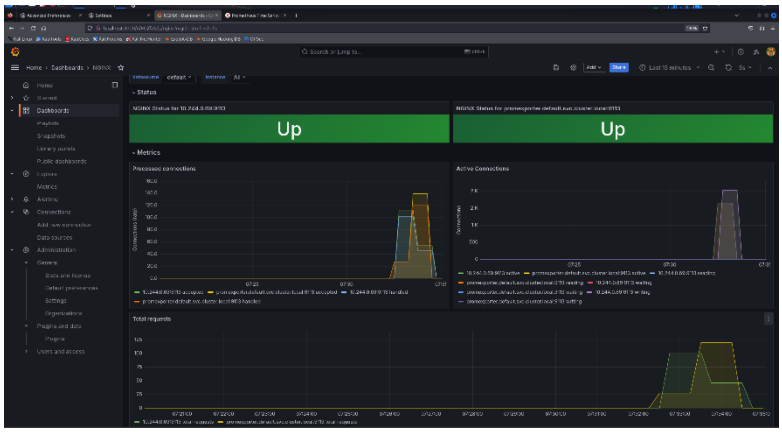
```
^Cstudent@lab-kubernetes:~$ kubectget secret grafana-admin --namespace monitoring -o jsonpath="{.data.GF_SECURITY_ADMIN_PASSWORD}" | base64 -d
WSwGfrvtlBstudent@lab-kubernetes:~$ ^C
student@lab-kubernetes:~$ kubectl port-forward -n monitoring svc/grafana 3000:3000
Forwarding from 127.0.0.1:3000 → 3000
Forwarding from [::1]:3000 → 3000
Handling connection for 3000
```



After that I add Prometheus data source with URL http://prometheus-server



Then I configure the dashboard using the json file from the Github (the content is in writeup_reproduce.txt) and add the Prometheus data source previously configured.

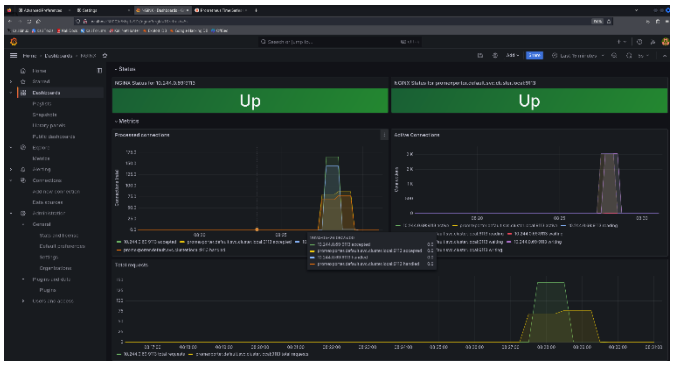


Try with command “slowhttptest -c 10000 -B -g -i 110 -r 1000 -s 8192 -u http://172.18.0.2:30080 -x 10 -p 3” to see if it’s updating.

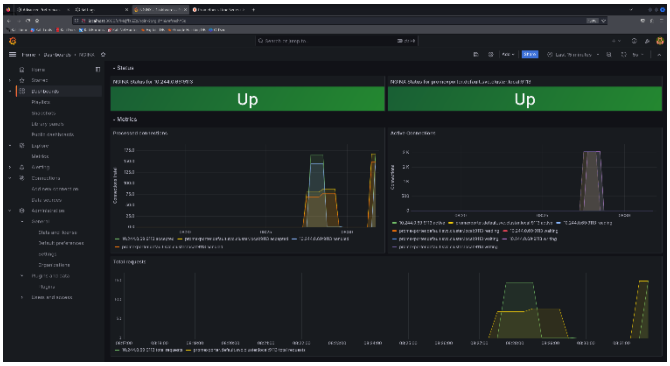
Step 5 :

I test different DoS attack with different parameters to see reaction of the server.

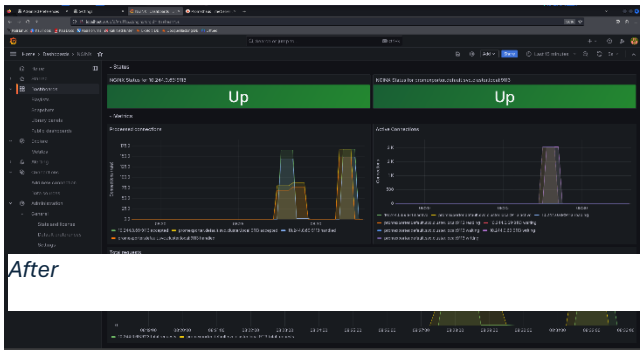
- slowhttptest -c 10000 -B -g -i 100 -r 1000 -s 8200 -u http://172.18.0.2:30080 -x 10 -p 3



Before

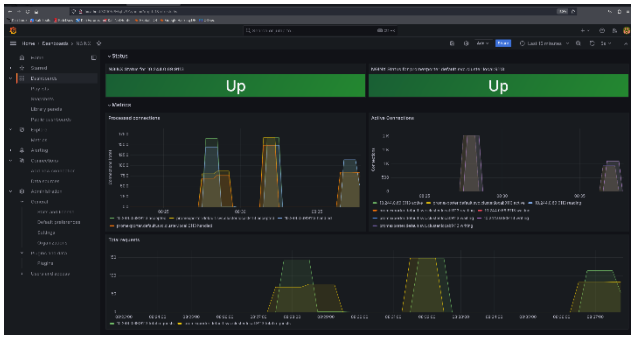


During

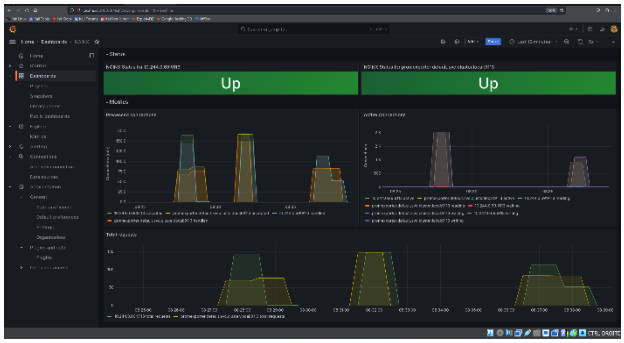


After

- slowhttptest -c 10000 -B -g -i 10 -r 200 -t GET -u http://172.18.0.2:30080 -x 24 -p 3

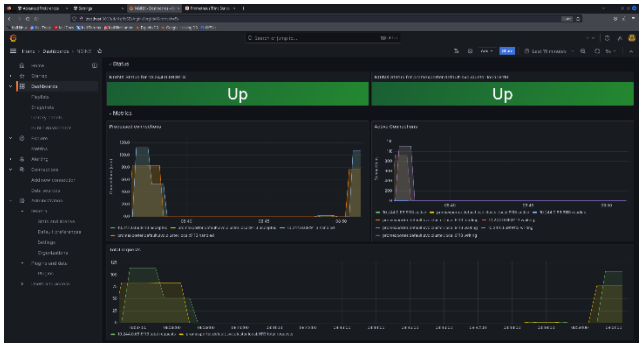


During

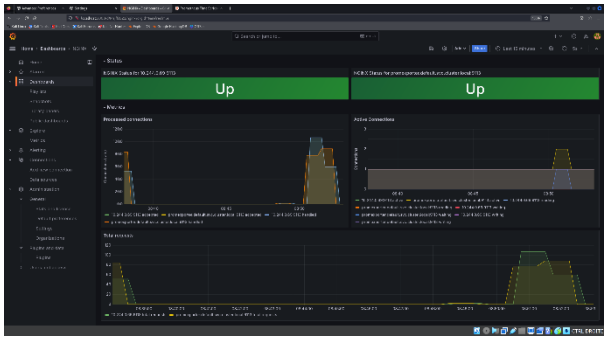


After

- Here I use “for i in {1..100}; do curl http://172.18.0.2:30080; done” (don’t see enough result) and then “for i in {1..10000}; do curl http://172.18.0.2:30080; done”



During



After

We can conclude thanks to the dashboard that it handles well requests of DoS attack and also accept Slow Read requests.

Step 6 :

Here is what could be done to secure infrastructure :

- First we could **update regularly Nginx, Prometheus and Grafana** to have the latest version to patch known vulnerabilities.
- Also we could make **regular backups and test restore procedures** in case of problems (to recover data for example)
- Then we could **monitor malicious activity** by deploying an Intrusion Detection System (IDS) for example and **use centralized logging** like Elasticsearch to analyse logs from Nginx, Prometheus and Grafana.
- Finally, we could secure each Nginx, Prometheus and Grafana.
    - Nginx :
        - Disable unnecessary modules to reduce attack surface.
        - Limit request size to prevent large payload attack (use “client_max_body_size”)
        - Implement HTTPS with a certificate authority (Let’s Encrypt) to get SSL/TLS certificates and then listen on port 443 instead of 80 with SSL.
    - Prometheus :
        - Use HTTPS to secure communications between services.
        - Implement Role-Based Access Control (RBAC) to control access.
        - Enable authentication.
    - Grafana
        - Use HTTPS to ensure data is encrypted.
        - Improve Password policies with two factor authentication for example-le.