

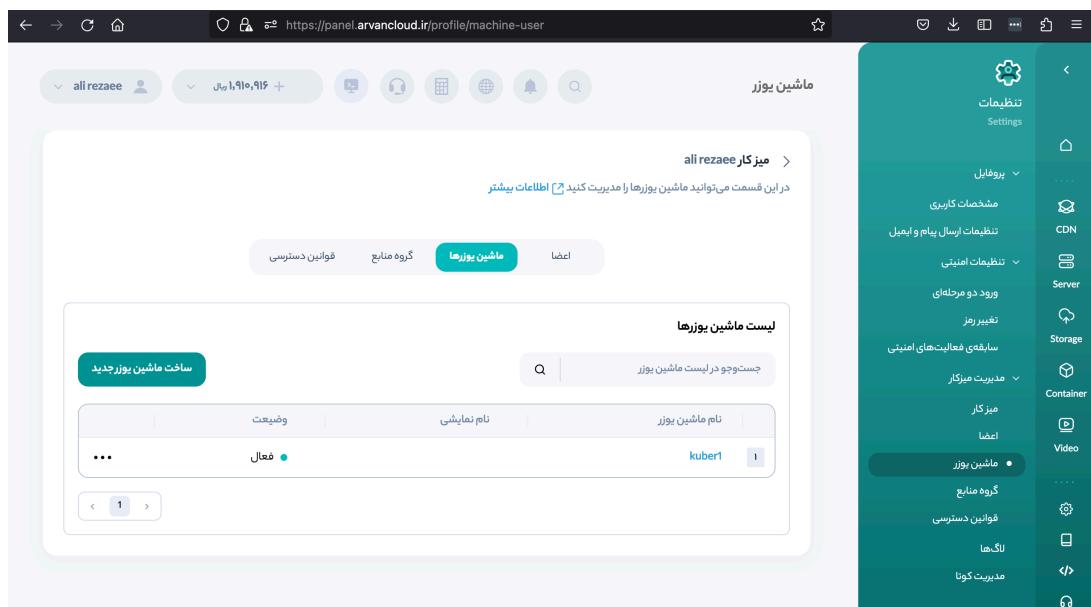
All commands in document were executed in Mac
You can access the code in private repository in git with this link:
Githubdsdfdsf

Phase 1(Terraform):

First install terraform:

```
brew tap hashicorp/tap
brew install hashicorp/tap/terraform
```

Then go to arvancloud site and create api key:



Then create folder terraform and make file called main.tf in it and write liked this in it and put your key in file like this:



```

main.tf
 terraform > main.tf
 1  terraform {
 2    required_providers {
 3      arvan = {
 4        source = "arvancloud/arvan"
 5        version = "0.6.4"
 6      }
 7    }
 8  }
 9  variable "ApiKey" {
10    type = string
11    default = "apikey 0027674a-a12b-5c83-adca-a4f8c7672711"
12    sensitive = true

```

Then write command :

```

● user@users-MacBook-Pro terraform-provider-arvan % terraform init
  Initializing the backend...
  Initializing provider plugins...
  - Reusing previous version of arvancloud/arvan from the dependency lock file
  - Using previously-installed arvancloud/arvan v0.6.4
  Terraform has been successfully initialized!

  You may now begin working with Terraform. Try running "terraform plan" to see
  any changes that are required for your infrastructure. All Terraform commands
  should now work.

  If you ever set or change modules or backend configuration for Terraform,
  rerun this command to reinitialize your working directory. If you forget, other
  commands will detect it and remind you to do so if necessary.

```

And see the result. It is important that before this your vpn is connected.

And then write this command :

```

⊗ user@users-MacBook-Pro terraform % terraform apply
Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
+ create
<= read (data resources)

Terraform will perform the following actions:

# data.arvan_iaas_abrak.get_abrak_id will be read during apply
# (depends on a resource or a module with changes pending)
<= data "arvan_iaas_abrak" "get_abrak_id" {
    + ha_enabled = (known after apply)
    + id         = (known after apply)
    + name       = "kuber1"
    + region     = "ir-thr-c1"
}

# arvan_iaas_abrak.kuber1 will be created
+ resource "arvan_iaas_abrak" "kuber1" {
    + addresses   = (known after apply)
    + disk_size   = 25
    + flavor      = "a1-4-8-0"
}

```

And write yes:

Enter a value: yes

You can see that it is processing :

```
Enter a value: yes  
arvan_iaas_abrak.kuber1: Creating...  
arvan_iaas_abrak.kuber1: Still creating... [10s elapsed]
```

But unfortunately, it gives me 403:

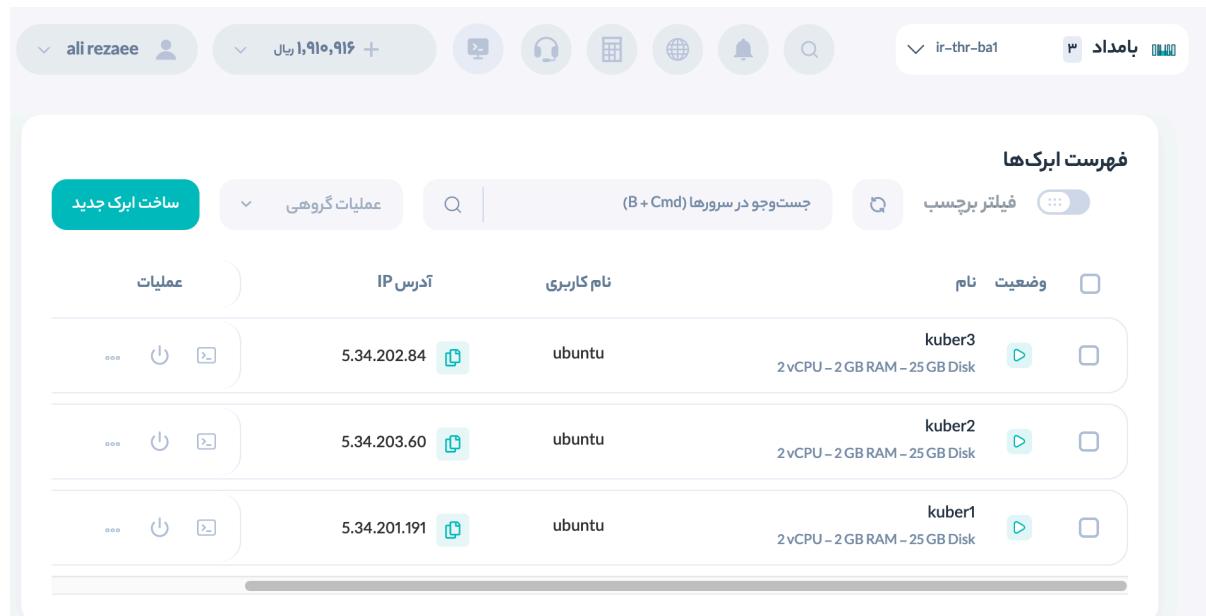
```
    "errors":[]}]  
  
with arvan_iaas_abrak.kuber1,  
on main.tf line 33, in resource "arvan_iaas_abrak" "kuber1":  
33: resource "arvan_iaas_abrak" "kuber1" {
```

And I change it's flavor or run this command:

- user@users-MacBook-Pro: ~ % TF_VAR_ApiKey="apikey 0027674a-a12b-5c83-adca-a4f8c7672711" terraform apply

But it didn't work, and I create VM manually

So you can see the result I create three vms :



Phase 2(K8s cluster):

Before this step you should set Shekan on all vms

Then I download ansible play book from this link:

<https://github.com/deveth0/kubernetes-cluster-ansible/tree/master>

to install kuber cluster on this three vms you should first configure the inventory file and take care that all user should be root:

```
kubernetes-cluster-ansible > ➜ inventory
1
2 [all]
3 kubernetes
4 kubernetes-node-1
5 kubernetes-node-1
6
7 [all:vars]
8 # Ubuntu uses python3 instead of python2, we need
9 ansible_python_interpreter=/usr/bin/python3
10
11 [kube-master]
12 root@5.34.202.84
13
14
15 [kube-node]
16 root@5.34.203.60
17 root@188.121.114.8
18
```

and then run kube-install-software.yml playbook by this command to install the requirements:

```
user@users-MacBook-Pro kubernetes-cluster-ansible-master % ansible-playbook -i inventory kube-install-software.yml -K
BECOME password:
[WARNING]: Invalid characters were found in group names but not replaced, use -vvvv to see details

PLAY [kube-master:kube-node] ****
TASK [Gathering Facts] ****
ok: [root@5.34.202.84]
ok: [root@5.34.203.60]
ok: [root@5.34.201.191]

TASK [Get Kubernetes apt-key] ****
ok: [root@5.34.203.60]
ok: [root@5.34.202.84]
ok: [root@5.34.201.191]

TASK [Add Kubernetes APT repository] ****
ok: [root@5.34.203.60]
ok: [root@5.34.202.84]
ok: [root@5.34.201.191]

TASK [Install required software] ****
ok: [root@5.34.202.84]
ok: [root@5.34.203.60]
ok: [root@5.34.201.191]

TASK [enable Docker service] ****
ok: [root@5.34.202.84]
ok: [root@5.34.203.60]
ok: [root@5.34.201.191]

PLAY RECAP ****
root@5.34.201.191 : ok=5    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
root@5.34.202.84 : ok=5    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
root@5.34.203.60 : ok=5    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

user@users-MacBook-Pro kubernetes-cluster-ansible-master % █
```

Then run k kube-setup-cluster.yml playbook by this command to setup master and join the node to it:

```
⑧ user@users-MacBook-Pro kubernetes-cluster-ansible-master % ansible-playbook -i inventory kube-setup-cluster.yml -K  
BECOME password:  
[WARNING]: Invalid characters were found in group names but not replaced, use -vvvv to see details  
PLAY [kube-master] *****  
TASK [Gathering Facts] *****  
ok: [root@5.34.202.84]  
TASK [Install kubectl on Master] *****  
ok: [root@5.34.202.84]  
TASK [Initialize Cluster] *****  
ok: [root@5.34.202.84]  
TASK [Create .kube directory] *****  
ok: [root@5.34.202.84]  
TASK [Copy admin.conf to the user's kube directory] *****  
ok: [root@5.34.202.84]  
TASK [Setup Flannel. Use log to prevent second installation] *****  
ok: [root@5.34.202.84]  
TASK [Create token to join cluster] *****  
changed: [root@5.34.202.84]  
TASK [Set join command as fact] *****  
ok: [root@5.34.202.84]  
PLAY [kube-node] *****  
TASK [Gathering Facts] *****  
ok: [root@5.34.203.60]  
ok: [root@5.34.201.191]  
TASK [Wait for master's port 6443] *****  
ok: [root@5.34.203.60]  
ok: [root@5.34.201.191]  
TASK [Join the cluster. Use log to prevent joining twice] *****  
fatal: [root@5.34.203.60]: FAILED! => {"msg": "The task includes an option with an undefined variable. The error was: \\\"hostvars['kube-master']\\\" is undefined. \\\"hostvars['kube-master']\\\" is undefined\nThe error appears to be in '/Users/user/Desktop/sreArvan/kubernetes-cluster-ansible/kube-setup-cluster.yml': line 52, column 7, but may\nbe elsewhere in the file depending on the exact syntax problem.\n\nThe offending line appears to be:\n      - name: Join the cluster. Use log to prevent joining twice\n        ^ here\n"}  
ok: [root@5.34.203.60]  
ok: [root@5.34.201.191]
```

But it gives me this at the final step when it wants to join the node to master and I changed the inventory file but it doesn't work so I join them manually by run this command on master to get token:

```
[root@kuber3:/home/ubuntu# sudo. unable to resolve host kuber3. Name or service not known  
[root@kuber3:/home/ubuntu# kubeadm token create --print-join-command  
kubeadm join 5.34.202.84:6443 --token bzmzyu.awlqzti3q8lggiw --disco
```

Then run this command to each node to connect to the master:

```
root@kuber1:~# kubeadm join 5.34.202.84:6443 --token bzmzyu.awlqzti3q8lggiw --discovery-  
[preflight] Running pre-flight checks  
    [WARNING Hostname]: hostname "kuber1" could not be reached  
    [WARNING Hostname]: hostname "kuber1": lookup kuber1 on 178.22.122.100:53: no suc  
[preflight] Reading configuration from the cluster...  
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kub  
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"  
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kub  
[kubelet-start] Starting the kubelet  
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...  
  
This node has joined the cluster:  
* Certificate signing request was sent to apiserver and a response was received.  
* The Kubelet was informed of the new secure connection details.  
  
Run 'kubectl get nodes' on the control-plane to see this node join the cluster.  
root@kuber1:~#
```

Now you should see this result when execute this command on master :

```
root@kuber3:/home/ubuntu# kubectl get nodes  
NAME      STATUS      ROLES      AGE      VERSION  
kuber1    NotReady   <none>     9s       v1.28.2  
kuber2    NotReady   <none>     40m      v1.28.2  
kuber3    Ready       control-plane   113m     v1.28.2  
root@kuber3:/home/ubuntu#
```

Phase 3(monitor k8s with Prometheus and Grafana):

I use this link to deploy Prometheus on Kubernetes:

<https://grafana.com/blog/2023/01/19/how-to-monitor-kubernetes-clusters-with-the-prometheus-operator/>

first I run this to deploy Prometheus:

```
[ubuntu@kuber3:~$ sudo su -
sudo: unable to resolve host kuber3: Name or service not known
[root@kuber3:~# kubectl apply -f https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/main/bundle.yaml
customresourcedefinition.apiextensions.k8s.io/alertmanagerconfigs.monitoring.coreos.com created
customresourcedefinition.apiextensions.k8s.io/alertmanagers.monitoring.coreos.com created
customresourcedefinition.apiextensions.k8s.io/podmonitors.monitoring.coreos.com created
customresourcedefinition.apiextensions.k8s.io/probes.monitoring.coreos.com created
customresourcedefinition.apiextensions.k8s.io/prometheusrules.monitoring.coreos.com created
customresourcedefinition.apiextensions.k8s.io/scrapeconfigs.monitoring.coreos.com created
customresourcedefinition.apiextensions.k8s.io/servicemonitors.monitoring.coreos.com created
customresourcedefinition.apiextensions.k8s.io/thanosrulers.monitoring.coreos.com created
clusterrolebinding.rbac.authorization.k8s.io/prometheus-operator created
clusterrole.rbac.authorization.k8s.io/prometheus-operator created
deployment.apps/prometheus-operator created
serviceaccount/prometheus-operator created
service/prometheus-operator created
Error from server (Invalid): error when creating "https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/main/ns.k8s.io "prometheusagents.monitoring.coreos.com" is invalid: metadata.annotations: Too long: must have at most 262144 bytes
Error from server (Invalid): error when creating "https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/main/ns.k8s.io "prometheuses.monitoring.coreos.com" is invalid: metadata.annotations: Too long: must have at most 262144 bytes

[root@kuber3:~# kubectl apply -f https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/main/bundle.yaml
customresourcedefinition.apiextensions.k8s.io/alertmanagerconfigs.monitoring.coreos.com serverside-applied
customresourcedefinition.apiextensions.k8s.io/alertmanagers.monitoring.coreos.com serverside-applied
customresourcedefinition.apiextensions.k8s.io/podmonitors.monitoring.coreos.com serverside-applied
customresourcedefinition.apiextensions.k8s.io/probes.monitoring.coreos.com serverside-applied
customresourcedefinition.apiextensions.k8s.io/prometheusagents.monitoring.coreos.com serverside-applied
customresourcedefinition.apiextensions.k8s.io/prometheusrules.monitoring.coreos.com serverside-applied
customresourcedefinition.apiextensions.k8s.io/scrapeconfigs.monitoring.coreos.com serverside-applied
customresourcedefinition.apiextensions.k8s.io/servicemonitors.monitoring.coreos.com serverside-applied
customresourcedefinition.apiextensions.k8s.io/thanosrulers.monitoring.coreos.com serverside-applied
clusterrolebinding.rbac.authorization.k8s.io/prometheus-operator serverside-applied
clusterrole.rbac.authorization.k8s.io/prometheus-operator serverside-applied
deployment.apps/prometheus-operator serverside-applied
serviceaccount/prometheus-operator serverside-applied
service/prometheus-operator serverside-applied]
```

You can see the result:

```
[root@kuber3:~# kubectl get deployments
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
prometheus-operator   1/1     1           1          79s
root@kuber3:~# ]
```

Then you should create rbac file for it like below:

```
root@kuber3:~# vim prometheus_rbac.yaml
root@kuber3:~#
```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/metrics
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources:
  - configmaps
  verbs: ["get"]
```

And apply it :

```
root@kuber3:~# kubectl apply -f prometheus_rbac.yaml
serviceaccount/prometheus created
clusterrole.rbac.authorization.k8s.io/prometheus created
clusterrolebinding.rbac.authorization.k8s.io/prometheus created
root@kuber3:~#
```

Then you should create instance file for it like below:

```
[root@kuber3:~# vim prometheus_instance.yaml  
root@kuber3:~# ]
```

```
apiVersion: monitoring.coreos.com/v1  
kind: Prometheus  
metadata:  
  name: prometheus  
spec:  
  serviceAccountName: prometheus  
  serviceMonitorSelector: {}  
  resources:  
    requests:  
      memory: 300Mi
```

And apply it:

```
[root@kuber3:~# kubectl apply -f prometheus_instance.yaml  
prometheus.monitoring.coreos.com/prometheus created  
root@kuber3:~# ]
```

And you can see the result:

```
[root@kuber3:~# kubectl get pods  
NAME                      READY   STATUS    RESTARTS   AGE  
prometheus-operator-6c9b57bcb8-9kg5g   1/1     Running   0          14m  
prometheus-prometheus-0            2/2     Running   0          9m13s  
root@kuber3:~# ]
```

Then you should create service file for it like below:

```
[root@kuber3:~# vim service_monitor.yaml  
root@kuber3:~# ]
```

yaml

```
apiVersion: monitoring.coreos.com/v1  
kind: ServiceMonitor  
metadata:  
  labels:  
    name: prometheus  
  name: prometheus  
spec:  
  endpoints:  
  - interval: 30s  
    targetPort: 9090  
    path: /metrics  
  namespaceSelector:  
    any: true  
  selector:  
    matchLabels:  
      operated-prometheus: "true"
```

And apply it:

```
[root@kuber3:~# vim service_monitor.yaml  
[root@kuber3:~# kubectl apply -f service_monitor.yaml  
servicemonitor.monitoring.coreos.com/prometheus created  
root@kuber3:~# ]
```

Then you should create expose file for it like below:

```
[^Croot@kuber3:~# vim expose_prometheus.yaml  
root@kuber3:~# ]
```

```
apiVersion: v1  
kind: Service  
metadata:  
  name: prometheus  
spec:  
  type: NodePort  
  ports:  
    - name: web  
      nodePort: 30900  
      port: 9090  
      protocol: TCP  
      targetPort: web  
  selector:  
    prometheus: prometheus
```

And apply it:

```
[root@kuber3:~# kubectl apply -f expose_prometheus.yaml  
service/prometheus created  
root@kuber3:~# ]
```

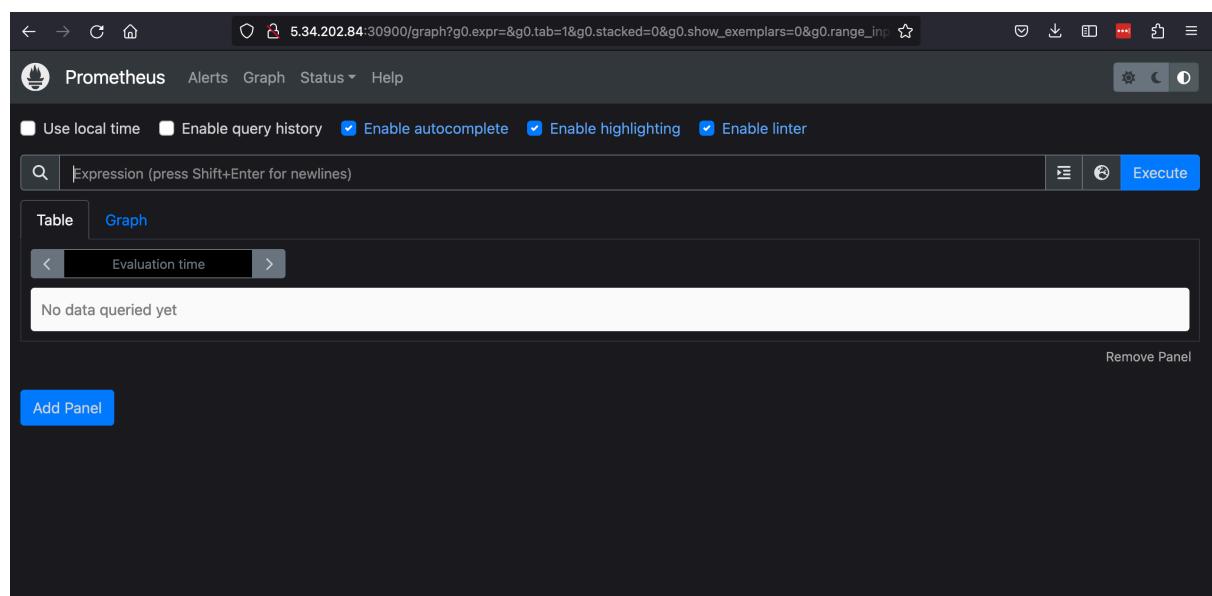
Now you can see the Prometheus panel in this link:

| http://<node_ip>:30900.

If you want to know your Ip you can run this :

```
[root@kuber3:~# kubectl get nodes -o wide
NAME     STATUS    ROLES      AGE   VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE
kuber1   Ready     <none>    15h   v1.28.2   188.121.114.8   <none>       Ubuntu 20.04.6 LTS
kuber2   NotReady  <none>    16h   v1.28.2   5.34.203.60    <none>       Ubuntu 20.04.6 LTS
kuber3   Ready     control-plane   17h   v1.28.2   5.34.202.84    <none>       Ubuntu 20.04.6 LTS
root@kuber3:~# ]
```

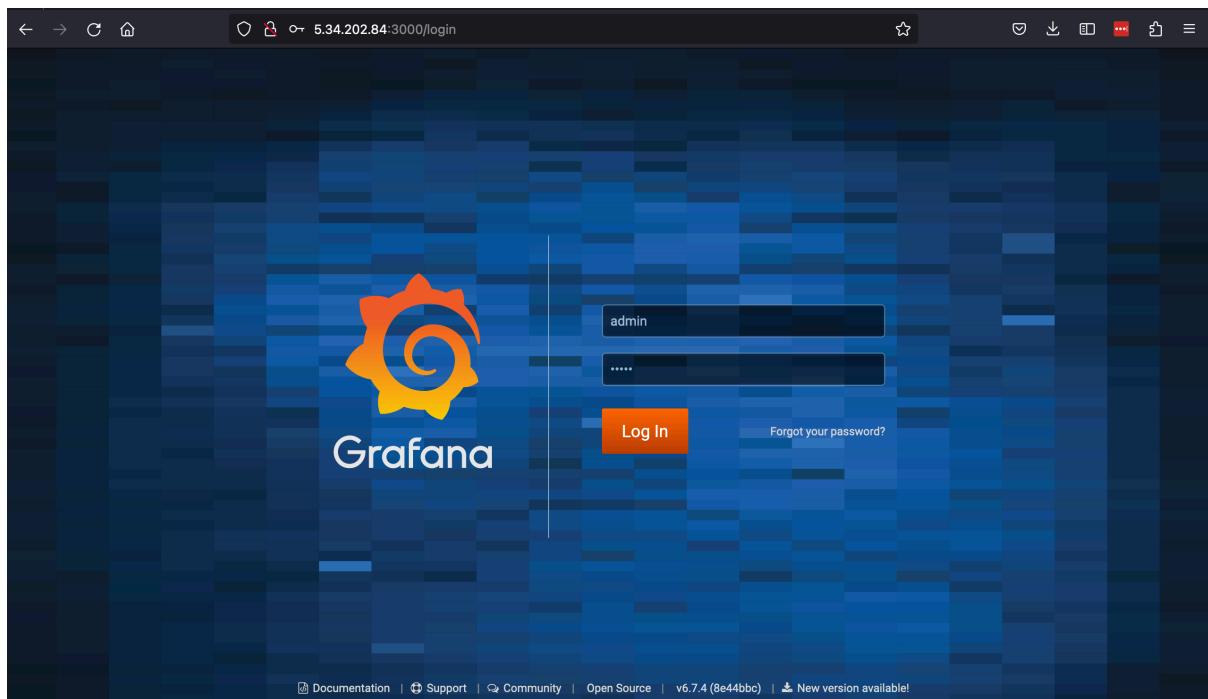
Here is Prometheus panel:



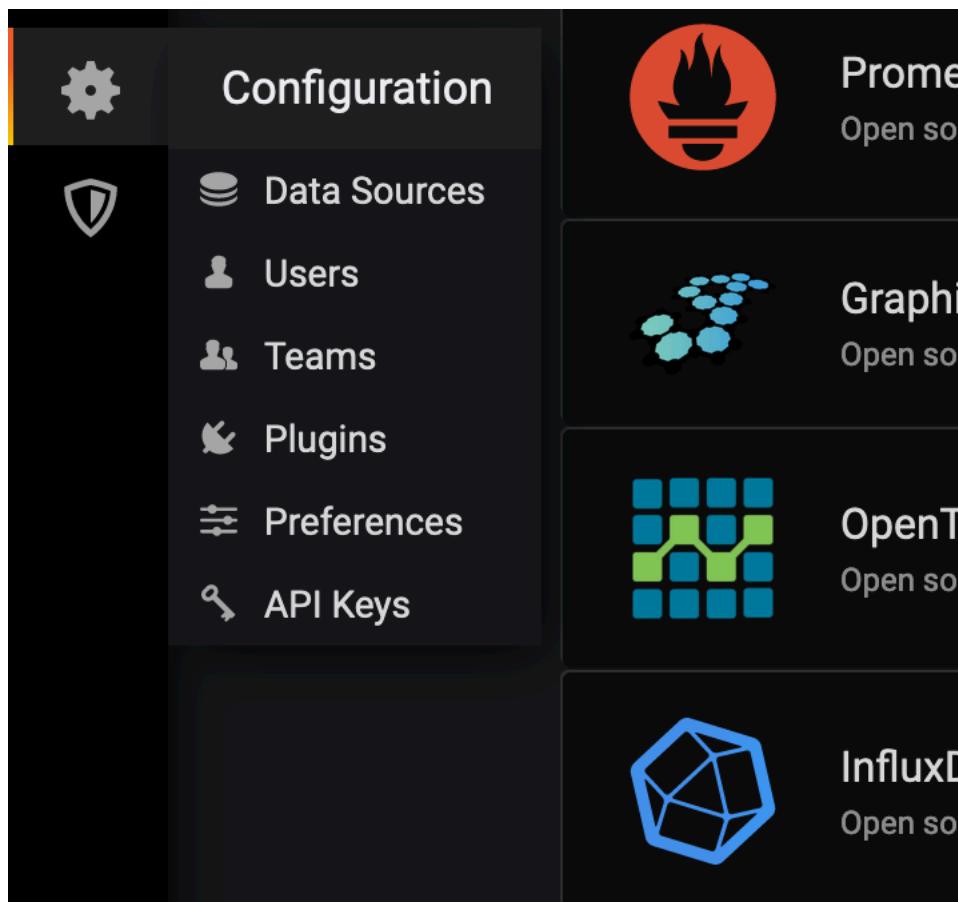
Then I install Grafana by this :

```
[root@kuber3:~# sudo snap install grafana
sudo: unable to resolve host kuber3: Name or service not known
grafana 6.7.4 from Canonical✓ installed
```

And you can see the panel below:



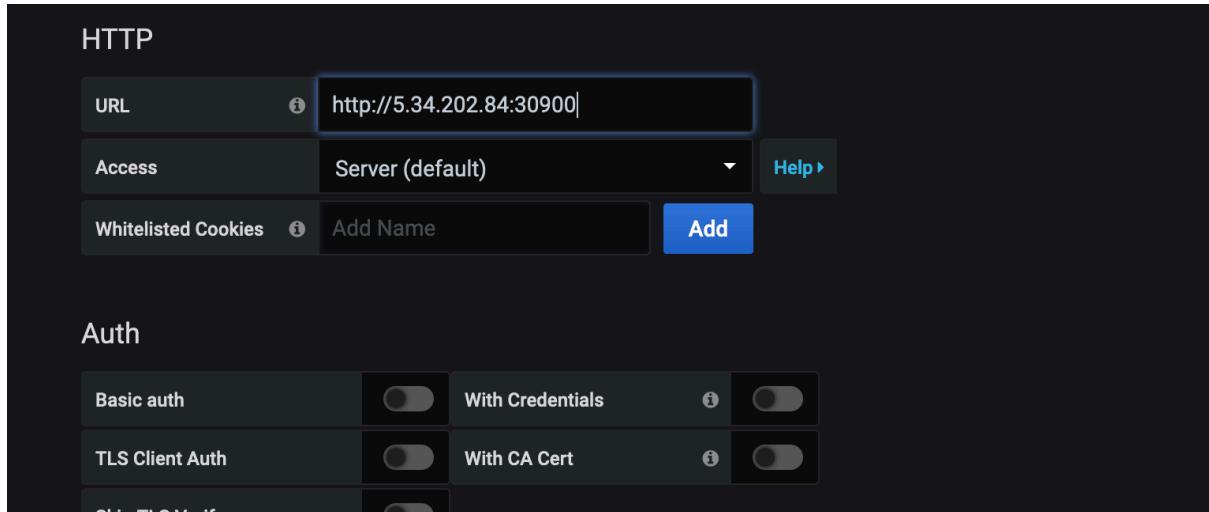
You should add Prometheus data source like this :



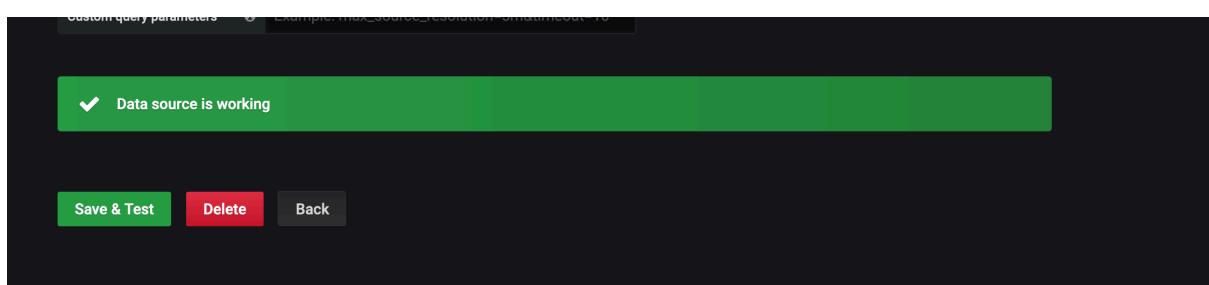
A modal dialog titled 'Time series databases' is open. It contains a search bar at the top with the placeholder 'Filter by name or type' and a 'Cancel' button. Below the search bar is a list of four items:

- Prometheus**: Open source time series database & alerting. This item is highlighted with a blue border. To its right are 'Learn more' and 'Select' buttons.
- Graphite**: Open source time series database
- OpenTSDB**: Open source time series database
- InfluxDB**: Open source time series database

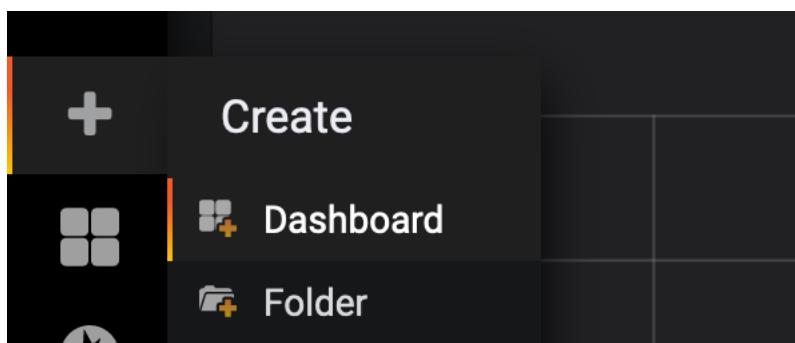
And write your node Ip in url field and port:



Then click on test:



Then I create dashboard like this:



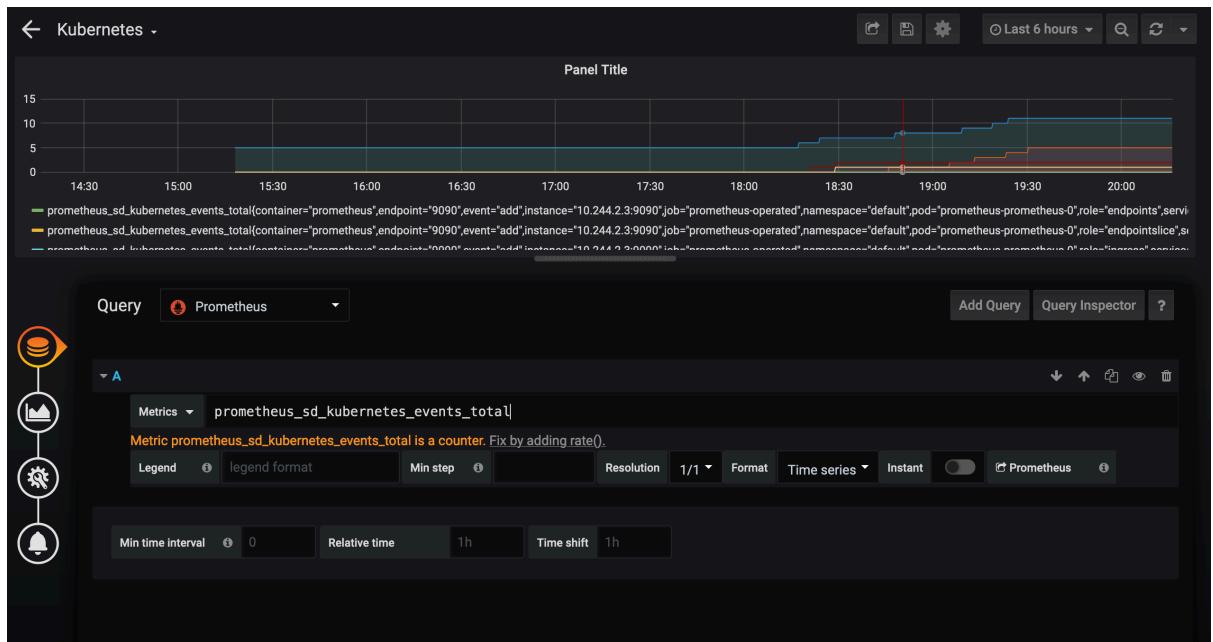
And I copy one metric from Prometheus:

The screenshot shows the Prometheus web interface at the URL `188.121.114.8:30900/graph?g0.expr=prometheus_sd_kubernetes_http_request_total&g0.tab=1&g0.slp=1`. The interface has a dark theme. At the top, there are navigation links for Alerts, Graph, Status, and Help. Below that is a toolbar with checkboxes for "Use local time", "Enable query history", "Enable autocomplete" (which is checked), "Enable highlighting" (which is checked), and "Enable linter" (which is checked). A search bar contains the query `|prometheus_sd_kubernetes_http_request_total`. Below the search bar are two tabs: "Table" (selected) and "Graph". Under the "Table" tab, there is an "Evaluation time" selector with arrows. The main content area displays the query results as a table:

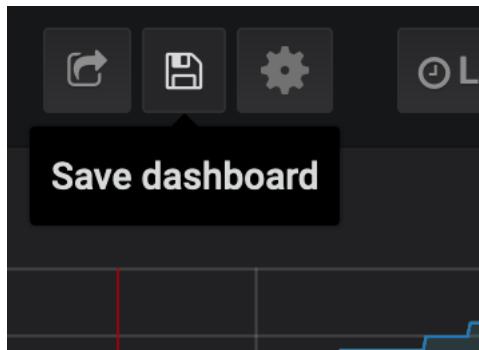
Container	Endpoint	Instance	Job	Service	Status Code
prometheus	9090	10.244.2.3:9090	prometheus-operated	prometheus-operated	200

At the bottom left, there is a blue button labeled "Add Panel".

And add it to panel and choose the data source:



Then I save dashboard and choose name:



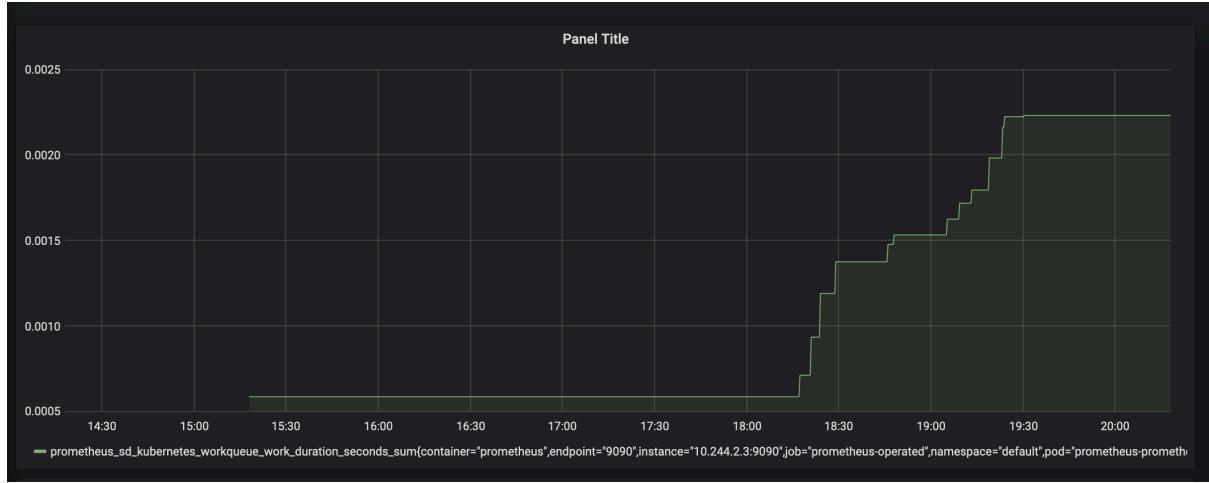
You can add other metrics as well:

A screenshot of the Grafana metric browser. It lists various Prometheus metrics under the category "kuber". The metrics are:

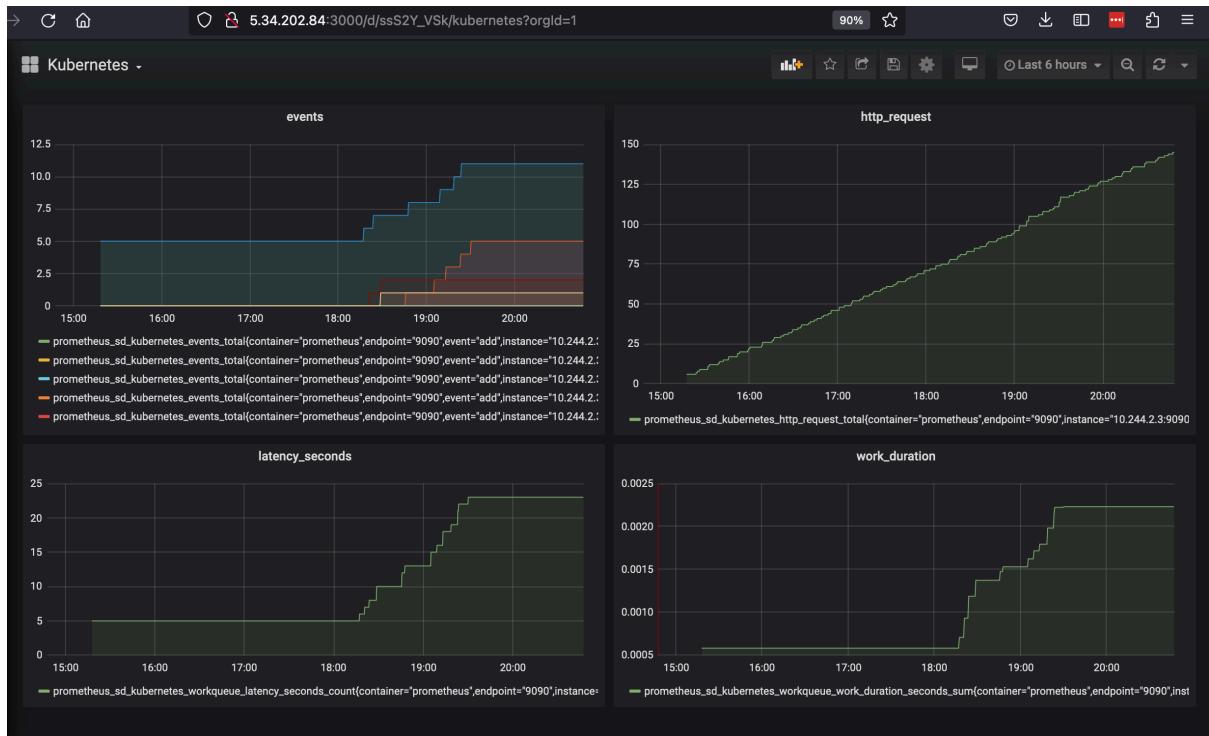
- prometheus_sd_kubernetes_events_total (counter)
- prometheus_sd_kubernetes_workqueue_depth (gauge)
- prometheus_sd_kubernetes_http_request_total (counter)
- prometheus_sd_kubernetes_workqueue_items_total (counter)
- prometheus_sd_kubernetes_workqueue_latency_seconds_sum (counter)
- prometheus_sd_kubernetes_workqueue_latency_seconds_count (counter)
- prometheus_sd_kubernetes_http_request_duration_seconds_sum (counter)
- prometheus_sd_kubernetes_workqueue_unfinished_work_seconds (gauge)
- prometheus_sd_kubernetes_http_request_duration_seconds_count (counter)
- prometheus_sd_kubernetes_workqueue_work_duration_seconds_sum (counter)
- prometheus_sd_kubernetes_workqueue_work_duration_seconds_count (counter)
- prometheus_sd_kubernetes_workqueue_longest_running_processor_seconds (gauge)
- topK(__rank_number__, __input_vector__) (snippet)
- bottomK(__rank_number__, __input_vector__) (snippet)

A tooltip is visible on the right side of the screen, pointing to the first metric, stating "The number of Ku".

For instance I add another parameter to another panel:

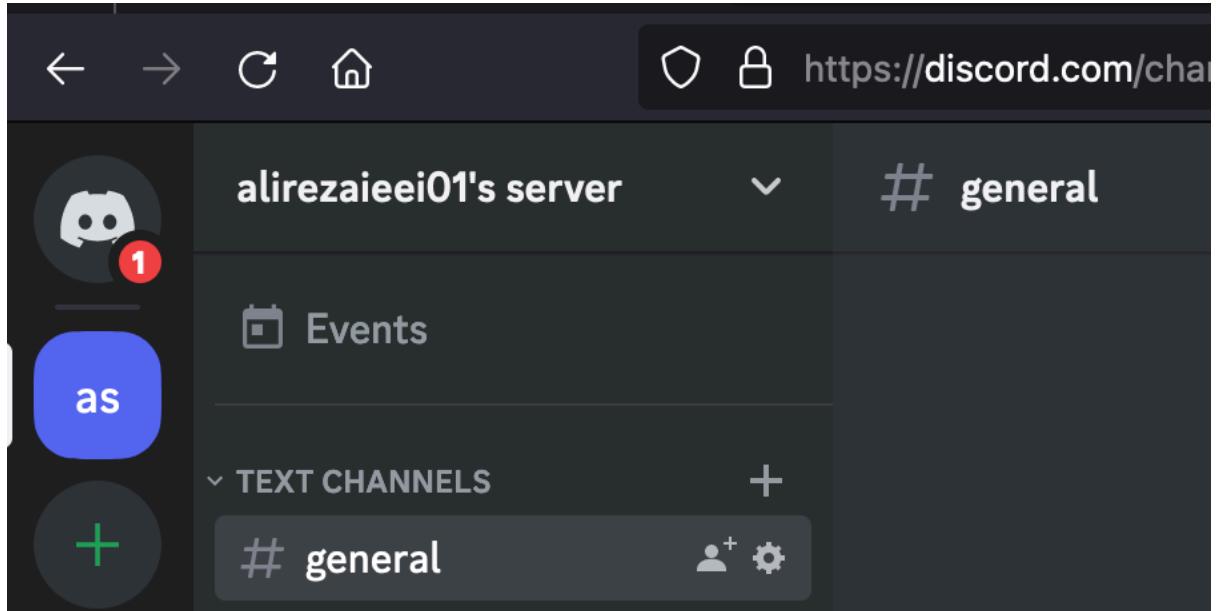


Finally their Kubernetes dashboard is prepared like this :

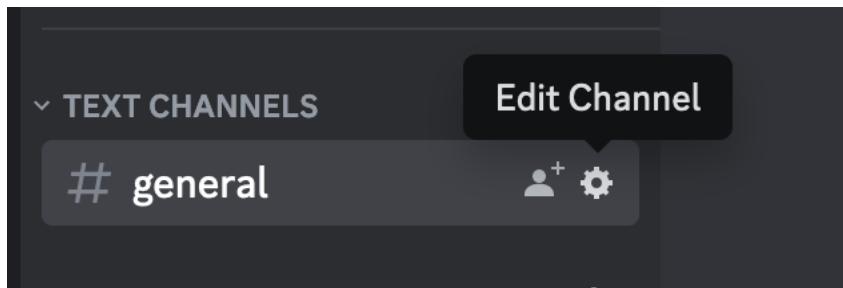


Phase 4(alerting):

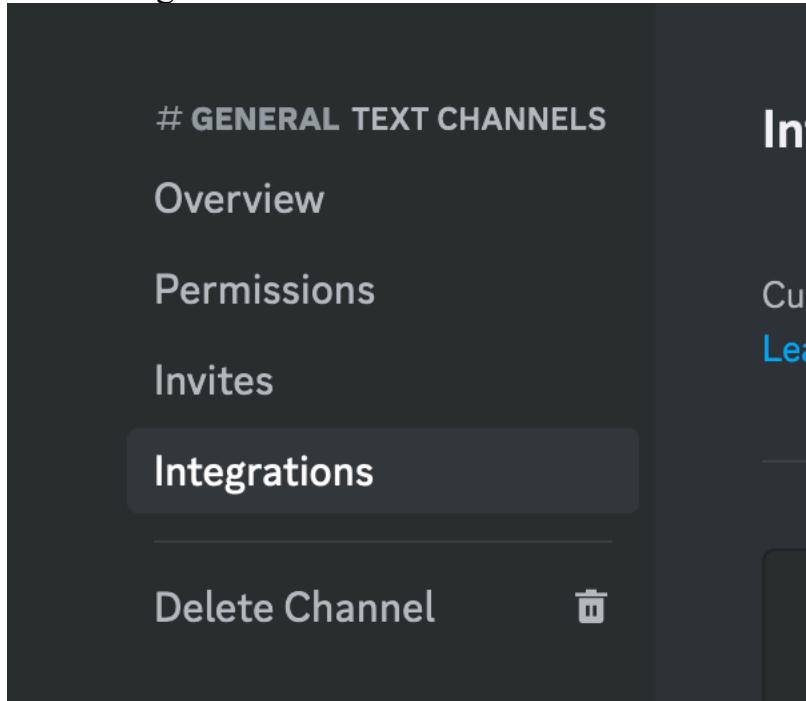
First we should create channel in discord in our own server like this:



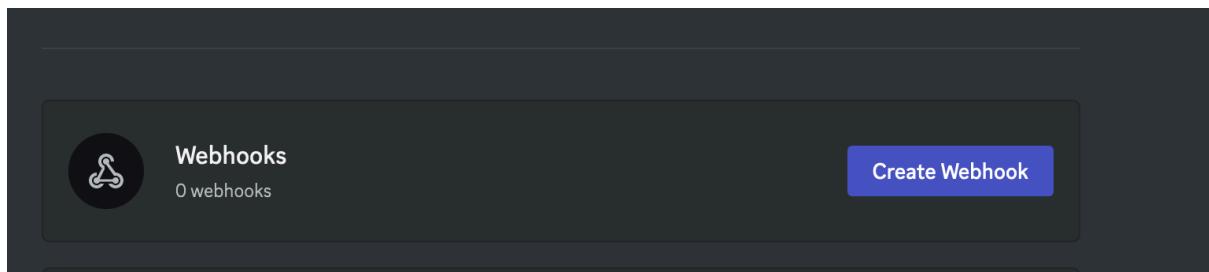
and click on edit channel:



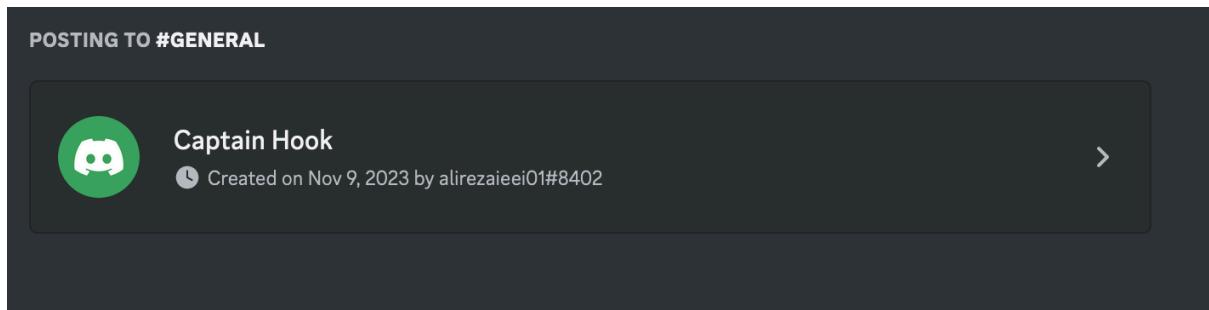
Go to integrations tab :



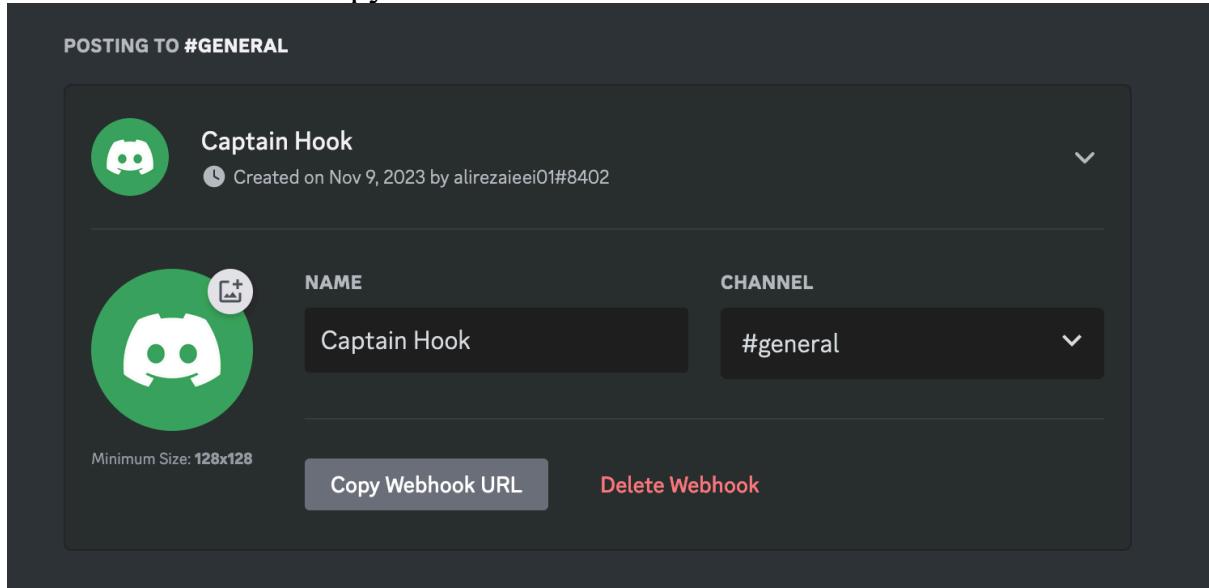
And choose create webhook:



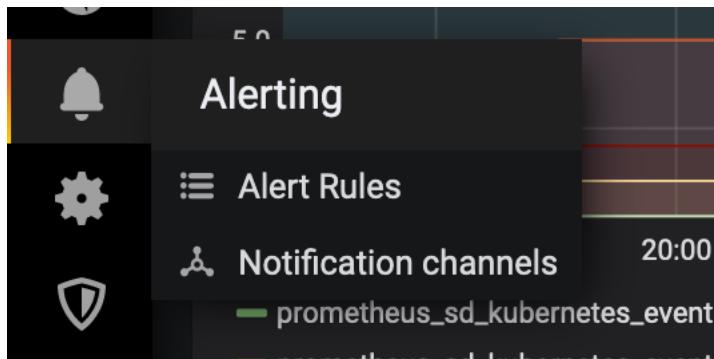
And see that is created:



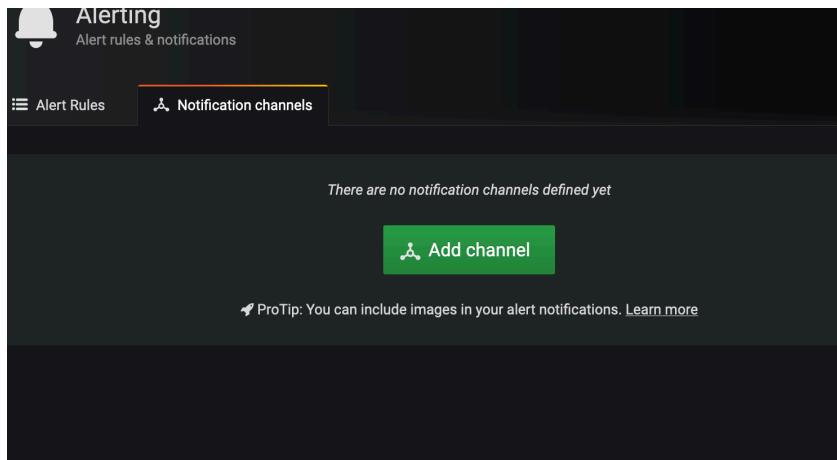
Then click on it and copy the url:



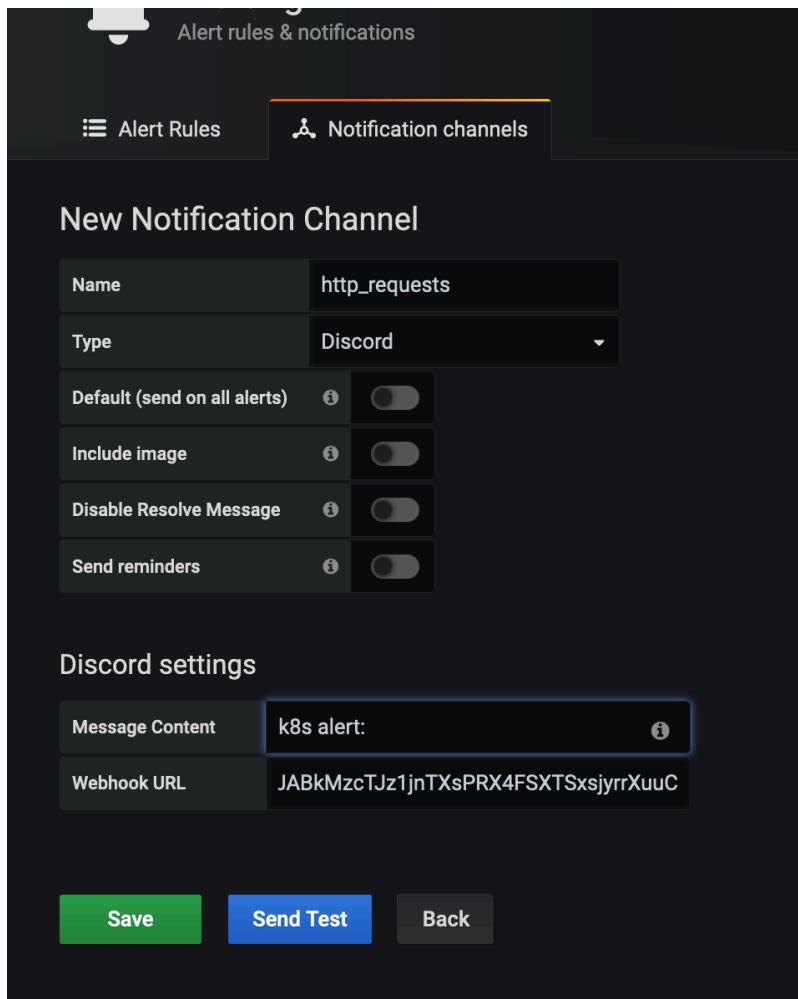
Then go to Grafana and choose notification channels:



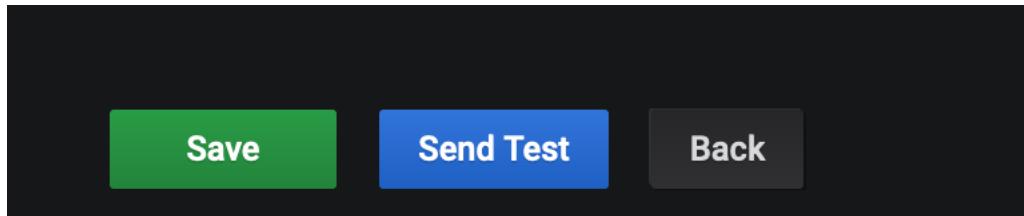
And click on add channel:



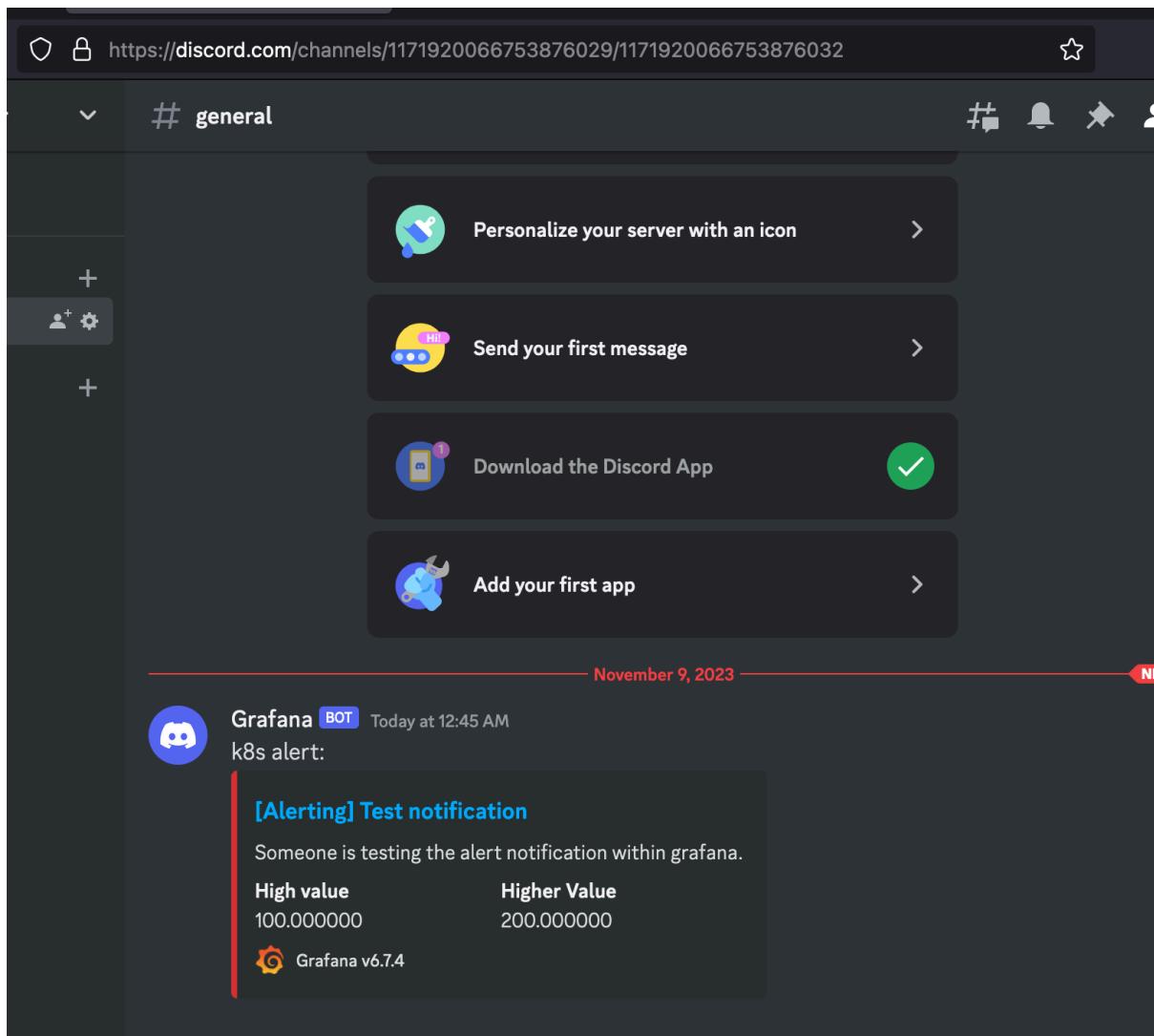
And select discord in type and paste the webhook:



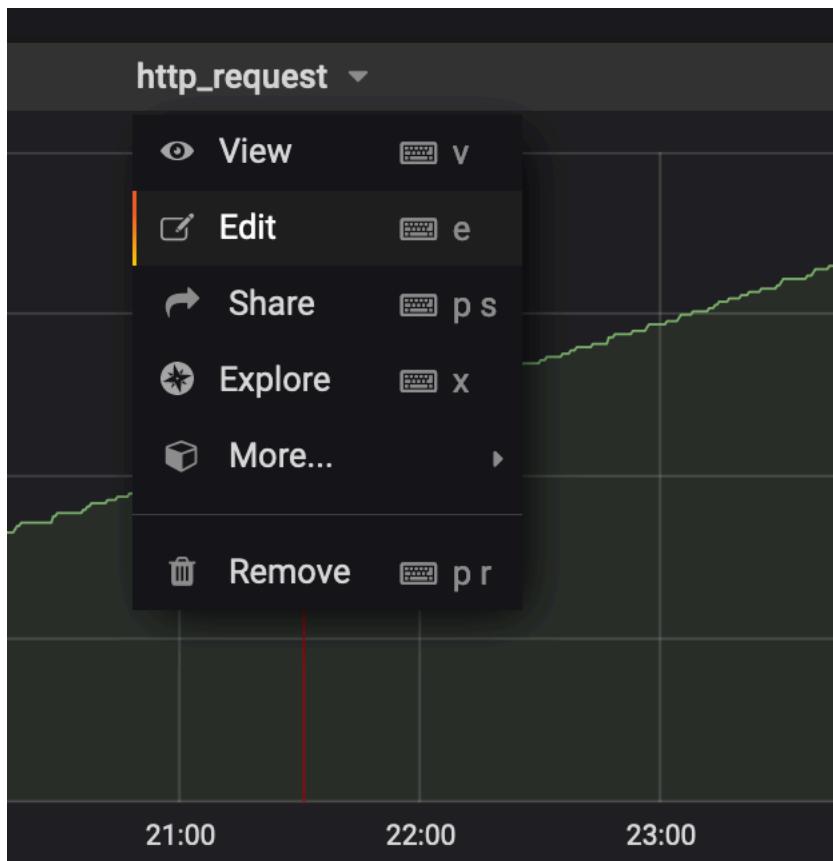
Then you can test it :



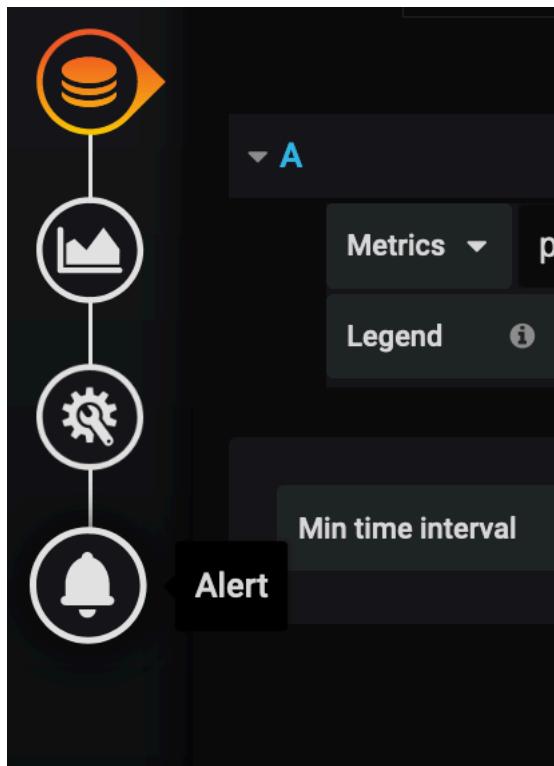
And see the result in channel:



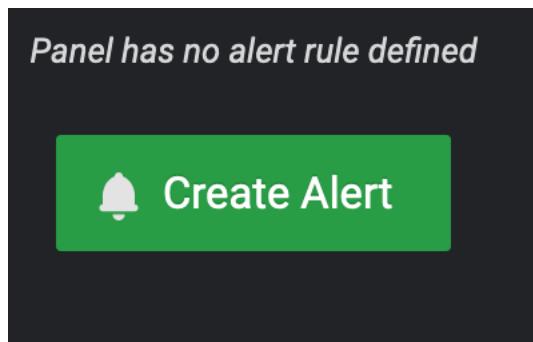
After it we should select the dashboard and click on edit:



And click on alert:



And create alert:



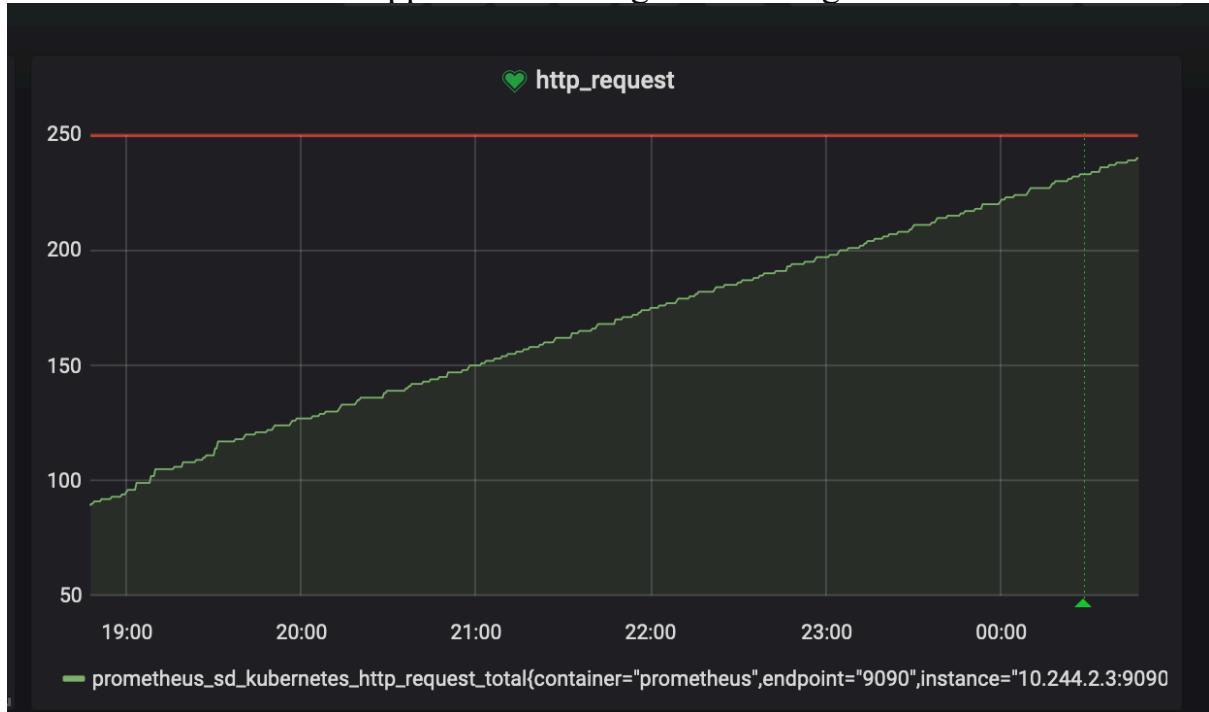
We can set condition like below that is alert when the metric go above 250:

A screenshot of the "Rule" configuration screen. It shows a condition setup: "WHEN avg () OF query (A, 5m, now) IS ABOVE 250". The "IS ABOVE" and "250" parts are highlighted with a blue selection bar.

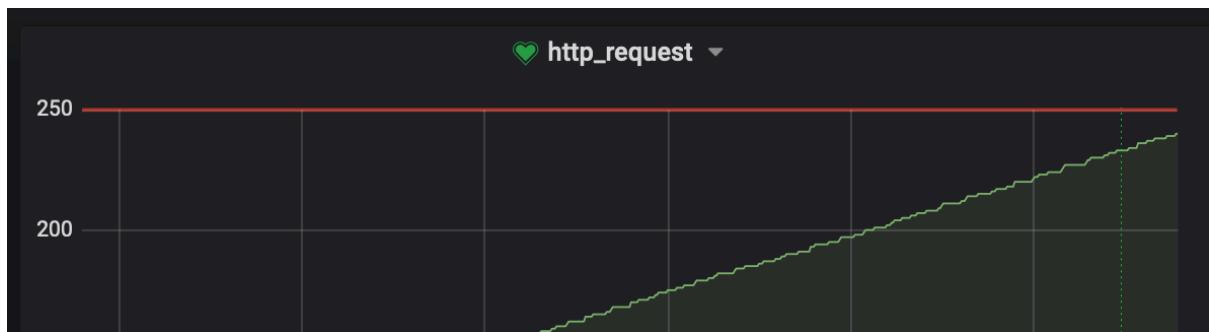
And we can set massage for it :

A screenshot of the "Notifications" configuration screen. It shows a message being sent to the "http_requests" channel. The message content is "http requests go over 250".

After we should see the upper bound 250 go above diagram:



And we can check the healthy by the color of the heart that is located on the top of the dashboard:



Phase 5(Postgres cluster):

I use this link:

<https://sweetcode.io/how-to-deploy-postgresql-instance-to-kubernetes/>

I create db-persistent-volume.yaml.

```
apiVersion: v1
# Kind for volume chain
kind: PersistentVolume
metadata:
  # Name the persistent chain
  name: postgresdb-persistent-volume
  # Labels for identifying PV
  labels:
    type: local
    app: postgresdb
spec:
  storageClassName: manual
  capacity:
    # PV Storage capacity
    storage: 8Gi
  # A db can write and read from volumes to multiple pods
  accessModes:
    - ReadWriteMany
  # Specify the path to persistent the volumes
  hostPath:
    path: "/data/db"
```

And I create db-volume-claim.yaml

```
apiVersion: v1
# define a resource for volume chain
kind: PersistentVolumeClaim
metadata:
  # Name the volume chain
  name: db-persistent-volume-claim
spec:
  storageClassName: manual
  accessModes:
    # Allow ReadWrite to multiple pods
    - ReadWriteMany
  # PVC requesting resources
  resources:
    requests:
      # the PVC storage
      storage: 8Gi
```

And I create db-configmap.yaml

```
apiVersion: v1
# Kind for kubernets ConfigMap
kind: ConfigMap
metadata:
  # Name your ConfigMap
  name: db-secret-credentials
  labels:
    app: postgresdb
data:
  # User DB
  POSTGRES_DB: testDB
  # Db user
  POSTGRES_USER: testUser
  # Db password
  POSTGRES_PASSWORD: testPassword
```

I create db-deployment.yaml:

```
# Kubernetes API version
apiVersion: apps/v1
# Deployment object
kind: Deployment
metadata:
  # The name of the Deployment
  name: postgresdb
spec:
  # Replicas for this Deployment
  replicas: 3
  selector:
    # labels the pods
    matchLabels:
      app: postgresdb
  template:
    metadata:
      labels:
        # The label the pods created from the pod template shc
        app: postgresdb
    spec:
      containers:
        # The container name to execute pods
        - name: postgresdb
          # pull postgresimage from docker hub
          image: postgres
          ports:
            # Assign ports to expose container
            - containerPort: 5432
      envFrom:
        # Load the environment variables/PostgreSQL creden
        - configMapRef:
            # This should be the ConfigMap name created ea
            name: db-secret-credentials
      volumeMounts:
        # The volume mounts for the container
        - mountPath: /var/lib/postgres/data
          name: db-data
```

I create db-service.yaml

```
apiVersion: v1
# Kind for service
kind: Service
metadata:
    # Name your service
    name: postgresdb
    labels:
        app: postgresdb
spec:
    # Choose how to expose your service
    type: NodePort
    ports:
        # The port number to expose
        - port: 5432
    # Pod to route service traffic
    selector:
        app: postgresdb
```

And apply all of them with this order:

```
[root@kuber3:~# kubectl apply -f db-persistent-volume.yaml
persistentvolume/postgresdb-persistent-volume created
[root@kuber3:~# kubectl apply -f db-volume-claim.yaml
persistentvolumeclaim/db-persistent-volume-claim created
[root@kuber3:~# kubectl apply -f db-configmap.yaml
configmap/db-secret-credentials created
[root@kuber3:~# kubectl apply -f db-deployment.yaml
deployment.apps/postgresdb created
[root@kuber3:~# kubectl apply -f db-service.yaml
service/postgresdb created
```

And you can see this in result:

```
[root@kuber3:~# kubectl get all
NAME                               READY   STATUS    RESTARTS   AGE
pod/postgresdb-5cf94bf6fb-55tpf   1/1     Running   0          12m
pod/postgresdb-5cf94bf6fb-9hmq9   1/1     Running   0          12m
pod/postgresdb-5cf94bf6fb-xzh6s   1/1     Running   0          12m
pod/prometheus-operator-6c9b57bcb8-9kg5g  1/1     Running   0          11h
pod/prometheus-prometheus-0       2/2     Running   0          11h

NAME                         TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)        AGE
service/kubernetes            ClusterIP   10.96.0.1      <none>           443/TCP       24h
service/postgresdb             NodePort    10.109.251.47  <none>           5432:31376/TCP 12m
service/prometheus              NodePort    10.104.145.106 <none>           9090:30900/TCP 7h33m
service/prometheus-operated   ClusterIP   None           <none>           9090/TCP      11h
service/prometheus-operator    ClusterIP   None           <none>           8080/TCP      11h

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/postgresdb        3/3     3           3           12m
deployment.apps/prometheus-operator 1/1     1           1           11h

NAME                           DESIRED   CURRENT   READY   AGE
replicaset.apps/postgresdb-5cf94bf6fb  3         3         3       12m
replicaset.apps/prometheus-operator-6c9b57bcb8  1         1         1       11h

NAME                           READY   AGE
statefulset.apps/prometheus-prometheus  1/1     11h
root@kuber3:~# ]
```

And you can connect to it with this command by copy one pod id:

```
[root@kuber3:~# kubectl exec -it postgresdb-5b9bf77c46-6xhx2 -- psql -h localhost -U testUser --password -p 5432 testDB
Error from server (NotFound): pods "postgresdb-5b9bf77c46-6xhx2" not found
root@kuber3:~# kubectl exec -it postgresdb-5cf94bf6fb-55tpf -- psql -h localhost -U testUser --password -p 5432 testDB
Password:
```

The password is : POSTGRES_PASSWORD

And see the result :

```
[root@kuber3:~# kubectl exec -it postgresdb-5cf94bf6fb-55tpf -- psql -h localhost -U testUser --password -p 5432 testDB
Password:
psql (16.0 (Debian 16.0-1.pgdg120+1))
Type "help" for help.

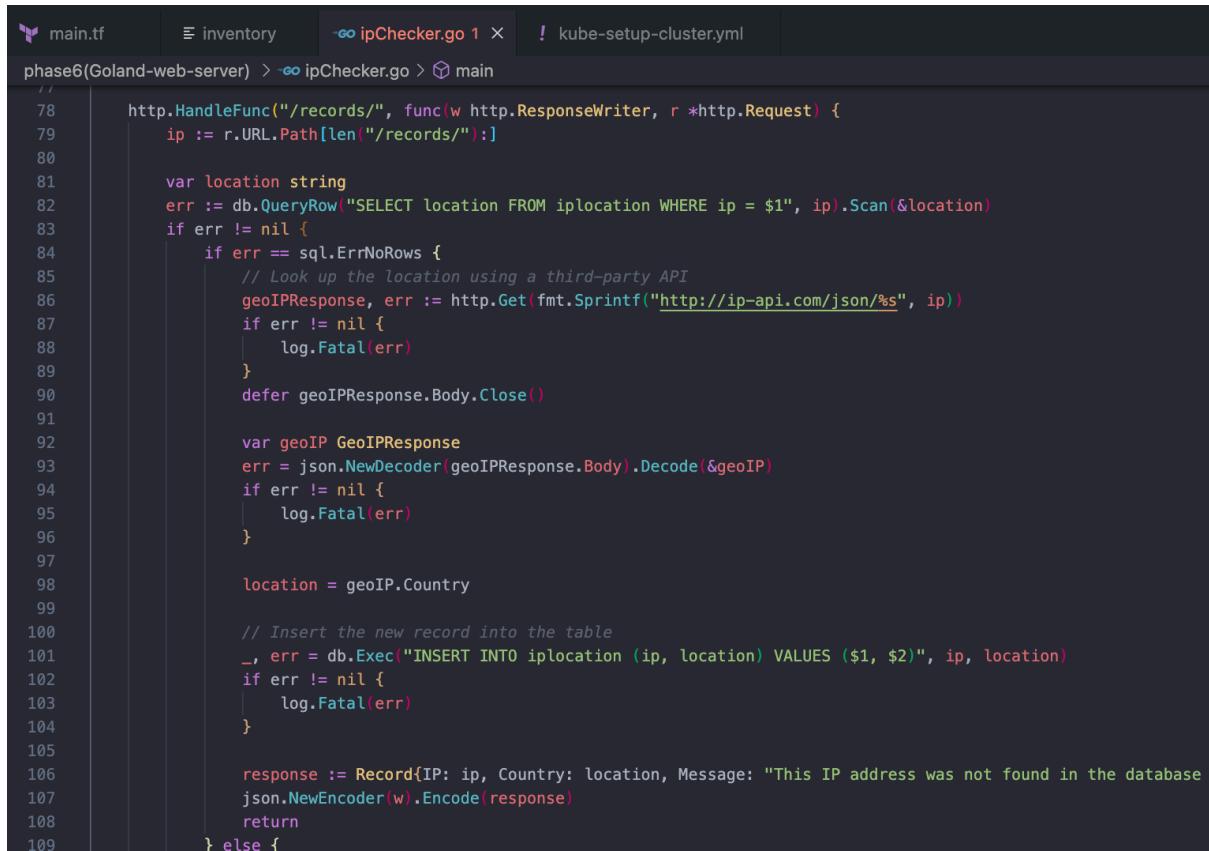
testDB=# ]
```

Phase 6(Golang web api):

In this code first the server is connected to pgdb by reading the os variable, and then if the table does not exist create the table “iplocation” with columns: ip and location. After this I implement two api that first one return the number of ip that is exist in our table because I want to use it for monitoring and the second one return the country where the ip is from, but first check that it is exist in our database or not and send the massage if exist. Lets see the part of code that you can see all of it in phase 6:

I use this website to give the location of ip :

<https://members.ip-api.com>



```
main.tf      inventory    ipChecker.go 1 ×  kube-setup-cluster.yml
phase6(Goland-web-server) > ipChecker.go > main
// ...
78     http.HandleFunc("/records/", func(w http.ResponseWriter, r *http.Request) {
79         ip := r.URL.Path[len("/records/"):]
80
81         var location string
82         err := db.QueryRow("SELECT location FROM iplocation WHERE ip = $1", ip).Scan(&location)
83         if err != nil {
84             if err == sql.ErrNoRows {
85                 // Look up the location using a third-party API
86                 geoIPResponse, err := http.Get(fmt.Sprintf("http://ip-api.com/json/%s", ip))
87                 if err != nil {
88                     log.Fatal(err)
89                 }
90                 defer geoIPResponse.Body.Close()
91
92                 var geoIP GeoIPResponse
93                 err = json.NewDecoder(geoIPResponse.Body).Decode(&geoIP)
94                 if err != nil {
95                     log.Fatal(err)
96                 }
97
98                 location = geoIP.Country
99
100                // Insert the new record into the table
101                _, err = db.Exec("INSERT INTO iplocation (ip, location) VALUES ($1, $2)", ip, location)
102                if err != nil {
103                    log.Fatal(err)
104                }
105
106                response := Record{IP: ip, Country: location, Message: "This IP address was not found in the database"}
107                json.NewEncoder(w).Encode(response)
108                return
109            } else {
```

How can we run it:

Firstly, we should set the os variable like this :

```
● user@users-MacBook-Pro phase6(Goland-web-server) % export DB_HOST=5.34.202.84
● user@users-MacBook-Pro phase6(Goland-web-server) % export DB_PORT=5432
● user@users-MacBook-Pro phase6(Goland-web-server) % export DB_NAME=testDB
● user@users-MacBook-Pro phase6(Goland-web-server) % export DB_USER=testUser
● user@users-MacBook-Pro phase6(Goland-web-server) % export DB_PASSWORD=POSTGRES_PASSWORD
```

Then we should create the go.mod and go.sum and get the pq library:

```
● user@users-MacBook-Pro phase6(Goland-web-server) % go mod init ipchecker
go: creating new go.mod: module ipchecker
go: to add module requirements and sums:
    go mod tidy
● user@users-MacBook-Pro phase6(Goland-web-server) % go get github.com/lib/pq
go: added github.com/lib/pq v1.10.9
```

And we can run it now:

```
✉ user@users-MacBook-Pro phase6(Goland-web-server) % go run ipChecker.go
2023/11/10 12:59:10 read tcp 192.168.149.66:50103->5.34.202.84:5432: read: connection reset by peer
exit status 1
```

The problem is for the wrong port that we can see the right port mapping by the this command that is 31376:

```
root@kuber3:~# kubectl get services
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
kubernetes     ClusterIP  10.96.0.1    <none>        443/TCP         2d11h
postgresdb     NodePort   10.109.251.47 <none>        5432:31376/TCP  35h
prometheus     NodePort   10.104.145.106  <none>        9090:30900/TCP  42h
prometheus-operated ClusterIP None          <none>        9090/TCP         46h
prometheus-operator ClusterIP None          <none>        8080/TCP         46h
root@kuber3:~#
```

Then I should change the os variable again:

```
✉ user@users-MacBook-Pro phase6(Goland-web-server) % export DB_PORT=31376
✉ user@users-MacBook-Pro phase6(Goland-web-server) % go run ipChecker.go
```

Now I see this :

```
✉ user@users-MacBook-Pro phase6(Goland-web-server) % go run ipChecker.go
2023/11/10 13:39:03 pq: password authentication failed for user "testUser"
exit status 1
```

So I connect to pq that is located in pod by this command :

```
root@kuber3:~# kubectl exec -it postgresdb-5b9bf77c46-6xhx2 -- psql -h localhost -U testUser --password -p 5432 testDB
Error from server (NotFound): pods "postgresdb-5b9bf77c46-6xhx2" not found
root@kuber3:~# kubectl exec -it postgresdb-5cf94bf6fb-55tpf -- psql -h localhost -U testUser --password -p 5432 testDB
Password:
```

The password is: POSTGRES_PASSWORD

```
root@kuber3:~# kubectl exec -it postgresdb-5cf94bf6fb-55tpf -- psql -h localhost -U testUser --password -p 5432 testDB
Password:
psql (16.0 (Debian 16.0-1.pgdg120+1))
Type "help" for help.

testDB=#
```

And create new user:

```
[testDB=# create user postgres with superuser password 'postgres';
CREATE ROLE
testDB=#
```

And change the var again:

```
exit status 1
● user@users-MacBook-Pro phase6(Goland-web-server) % export DB_PASSWORD=postgres
● user@users-MacBook-Pro phase6(Goland-web-server) % export DB_USER=postgres
```

Then it return this:

```
✖ user@users-MacBook-Pro phase6(Goland-web-server) % go run ipChecker.go
2023/11/10 13:02:34 pq: SSL is not enabled on the server
exit status 1
○ user@users-MacBook-Pro phase6(Goland-web-server) % █
```

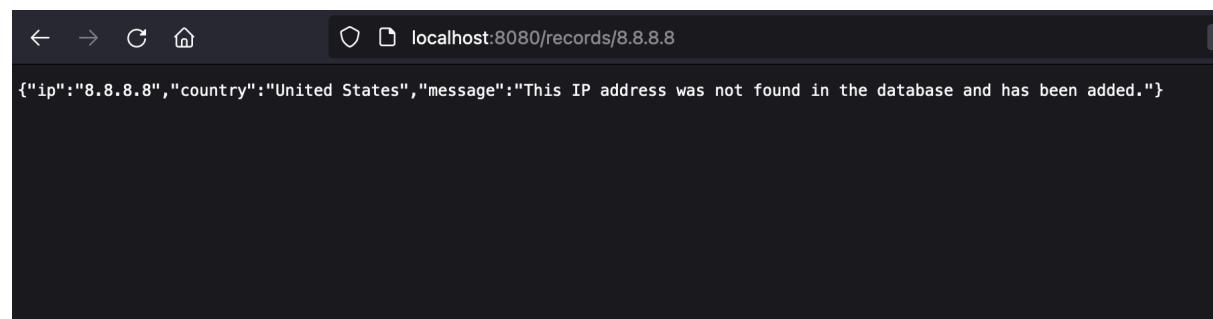
So I can disable ssl like this in code:

```
db, err := sql.Open("postgres", dbURI+"?sslmode=disable")
if err != nil {
```

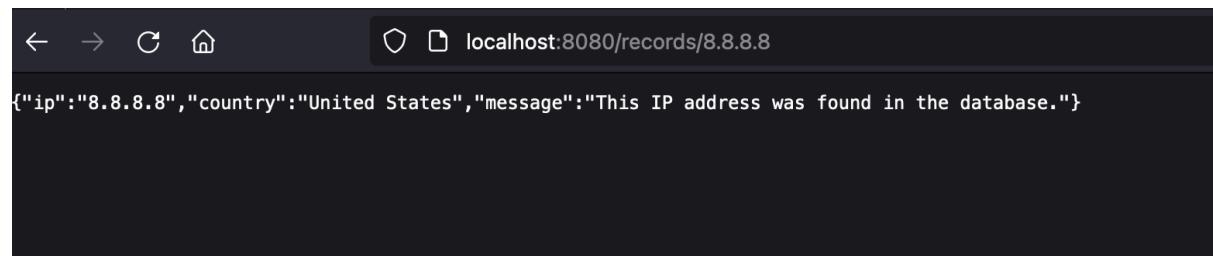
Now you can run it:

```
○ user@users-MacBook-Pro phase6(Goland-web-server) % go run ipChecker.go
█
```

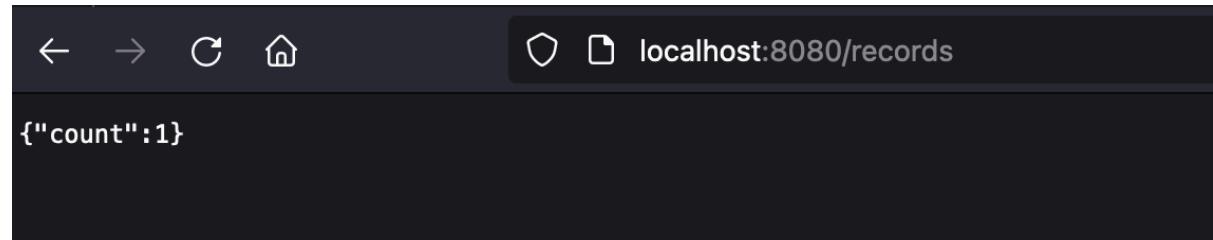
And see the result if the ip doesn't exist:



And see the result if the ip exist:



And the number of entries that we will use it for monitoring in future:

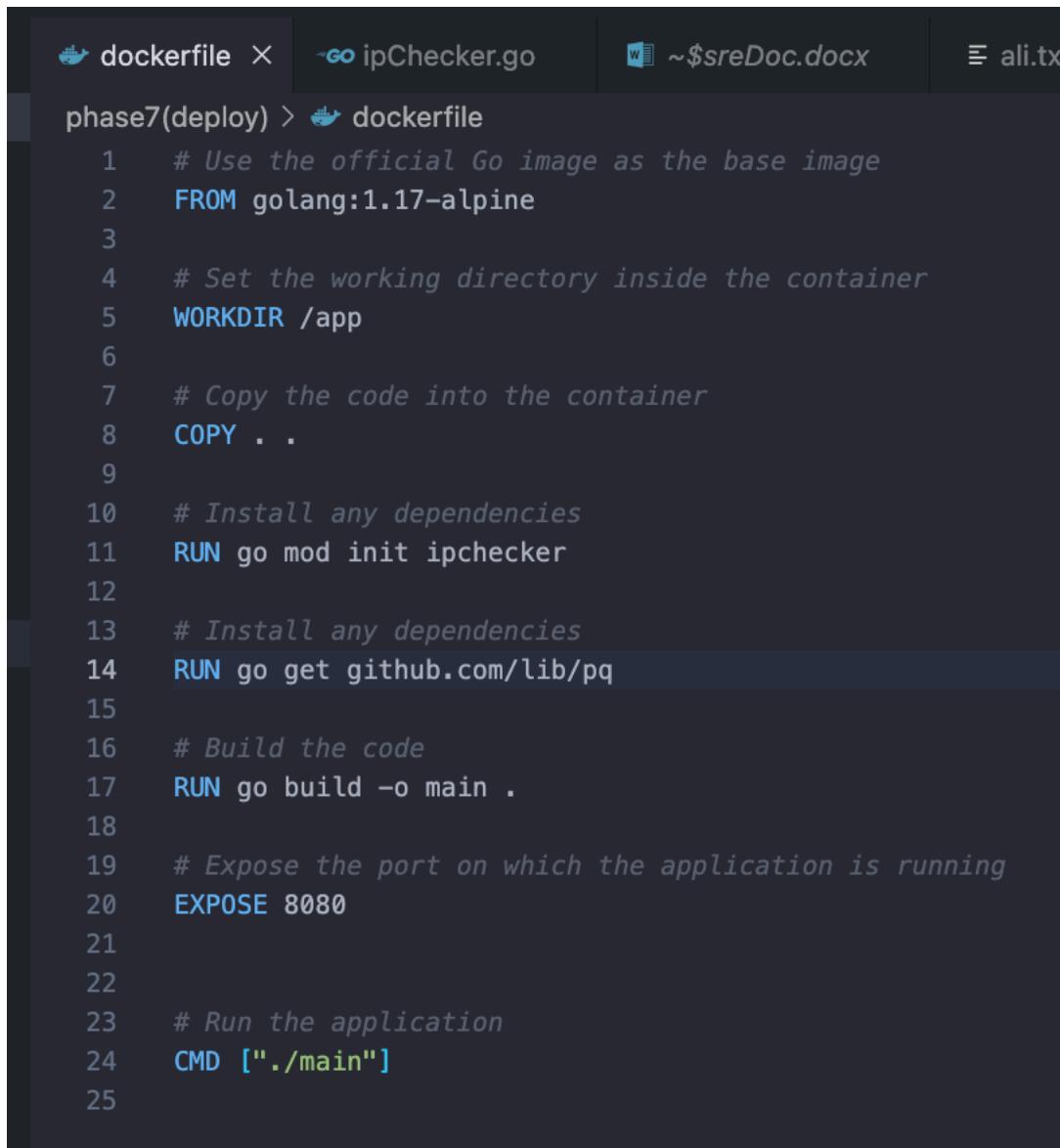


Phase 7(deploy):

First you should set shekan on your pc and creat dockerhub account like this:

<https://hub.docker.com/repositories/alirezaiiei>

then create docker file for your server like this:



The screenshot shows a terminal window with several tabs open at the top. The active tab is titled "dockerfile". Below the tabs, the terminal content is as follows:

```
phase7(deploy) > dockerfile
1 # Use the official Go image as the base image
2 FROM golang:1.17-alpine
3
4 # Set the working directory inside the container
5 WORKDIR /app
6
7 # Copy the code into the container
8 COPY .
9
10 # Install any dependencies
11 RUN go mod init ipchecker
12
13 # Install any dependencies
14 RUN go get github.com/lib/pq
15
16 # Build the code
17 RUN go build -o main .
18
19 # Expose the port on which the application is running
20 EXPOSE 8080
21
22
23 # Run the application
24 CMD ["./main"]
25
```

And build it :

```
● user@users-MacBook-Pro phase7(deploy) % docker build -t alirezaiiei/apichecker:1.0 .
[+] Building 10.5s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 37B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/golang:1.17-alpine
=> [1/6] FROM docker.io/library/golang:1.17-alpine@sha256:99ddec1bbfd6d6bca3f9804c02363daee8c8524dae50d
=> [internal] load build context
=> => transferring context: 507B
=> CACHED [2/6] WORKDIR /app
=> [3/6] COPY . .
=> [4/6] RUN go mod init ipchecker
=> [5/6] RUN go get github.com/lib/pq
=> [6/6] RUN go build -o main .
=> exporting to image
=> => exporting layers
=> => writing image sha256:bc8c97cb967eaecdf5f3d378461e91b460cca53048ab98853ec944f1b28bc529
=> => naming to docker.io/alirezaiiei/apichecker:1.0
○ user@users-MacBook-Pro phase7(deploy) %
```

And push it on dockerhub:

```
● user@users-MacBook-Pro phase7(deploy) % docker image push alirezaiiei/apichecker:1.0
The push refers to repository [docker.io/alirezaiiei/apichecker]
7748b6426fa7: Layer already exists
b1d99eceaa5e0: Layer already exists
3e9436f8d89f: Layer already exists
c4ad3b140634: Layer already exists
b8de6377084f: Layer already exists
fed9565c145c: Layer already exists
60af18e613ad: Layer already exists
925e9633fd86: Layer already exists
7b35f2def65d: Pushed
ec34fcc1d526: Layer already exists
1.0: digest: sha256:622f5c24406ee4eb6bd343cca4b874901ba38c1f6f2391f30996ba1b0217ffb0 size: 2408
○ user@users-MacBook-Pro phase7(deploy) %
```

And see the result:

The screenshot shows the Docker Hub interface for the repository 'alirezaiiei/apichecker'. The top navigation bar includes links for 'Explore', 'Repositories', 'Organizations', 'Help', and 'Upgrade'. The user's profile 'alirezaiiei' is visible on the right. The repository path 'alirezaiiei / Repositories / apichecker / General' is shown at the top left. A message indicates 'Using 0 of 1 private repositories. [Get more](#)'. Below this, there are tabs for 'General', 'Tags', 'Builds', 'Collaborators', 'Webhooks', and 'Settings'. The 'General' tab is selected. A note says 'Add a short description for this repository' with a 'Update' button. Another note says 'Last pushed: a few seconds ago'. The main content area shows the repository name 'alirezaiiei / apichecker'. It has a 'Docker commands' section with the command 'docker push alirezaiiei/apichecker:tagname'. There is a 'Public View' button. The 'Tags' section shows one tag: '1.0' (Image type). The 'Automated Builds' section provides instructions for connecting GitHub or Bitbucket to automatically build and tag images.

Then create deployment file:

```
! deployment.yaml × ! service.yaml dockerfile
phase7(deploy) > ! deployment.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: ipchecker-deployment
5  spec:
6    replicas: 1
7    selector:
8      matchLabels:
9        app: ipchecker
10   template:
11     metadata:
12       labels:
13         app: ipchecker
14   spec:
15     containers:
16       - name: ipchecker
17         image: alirezaei/apichecker:1.0
18         ports:
19           - containerPort: 8080
20         env:
21           - name: DB_HOST
22             value: postgresdb
23           - name: DB_PORT
24             value: "5432"
25           - name: DB_NAME
26             value: testDB
27           - name: DB_USER
28             value: postgres
29           - name: DB_PASSWORD
30             value: postgres
```

And service file :

```
! service.yaml × ! deployment.yaml dockerfile
phase7(deploy) > ! service.yaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: ipchecker-service
5  spec:
6    selector:
7      app: ipchecker
8    ports:
9      - name: http
10     port: 80
11     targetPort: 8080
12   type: LoadBalancer
13
```

And apply them in server by kubectl :

```
[root@kuber3:~# vim deployment.yaml
[root@kuber3:~# vim service.yaml
root@kuber3:~# kubectl apply -f deployment.yaml
deployment.apps/ipchecker-deployment created
root@kuber3:~# kubectl apply -f service.yaml
service/ipchecker-service created
```

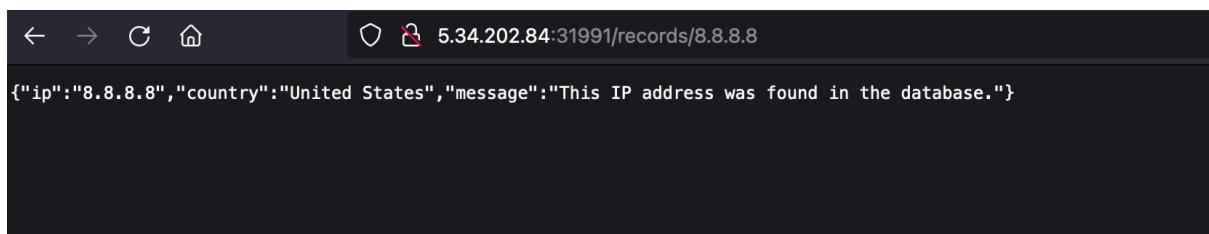
Then check the pod status:

```
[root@kuber3:~# kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
ipchecker-deployment-79b5445f67-4brbc  1/1    Running   0          5s
postgresdb-5cf94bf6fb-55tpf        1/1    Running   0          3d11h
postgresdb-5cf94bf6fb-9hmq9        1/1    Running   0          3d11h
postgresdb-5cf94bf6fb-xzh6s        1/1    Running   0          3d11h
prometheus-operator-6c9b57bcb8-9kg5g  1/1    Running   0          3d22h
prometheus-prometheus-0            2/2    Running   0          3d22h
root@kuber3:~#
```

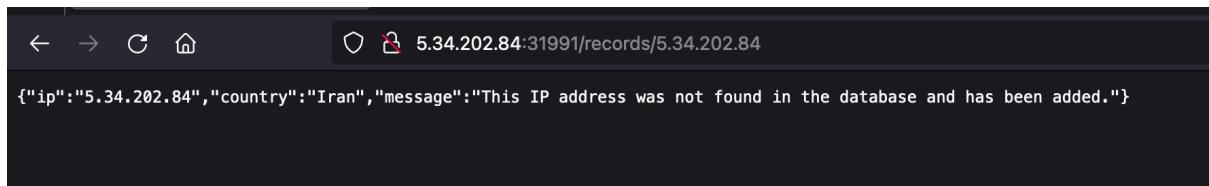
Now see the port map in your service:

```
PORT(S)
80:31991/TCP
```

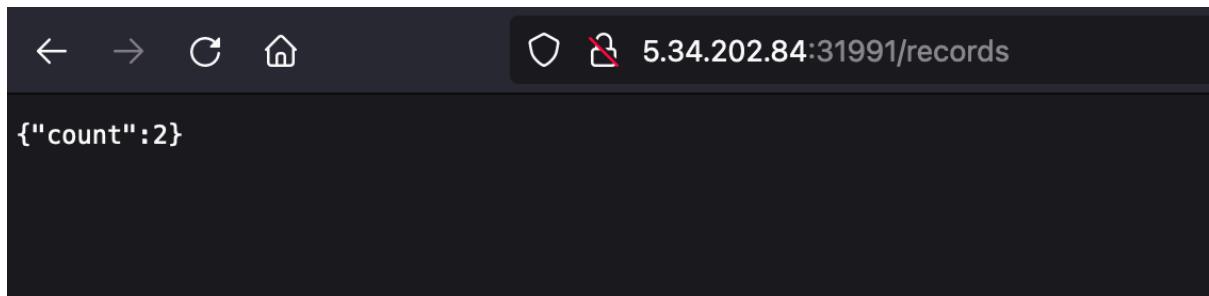
Finally it is deployed in Kubernetes cluster and you can see the result:



```
5.34.202.84:31991/records/8.8.8.8
{"ip": "8.8.8.8", "country": "United States", "message": "This IP address was found in the database."}
```



```
5.34.202.84:31991/records/5.34.202.84
{"ip": "5.34.202.84", "country": "Iran", "message": "This IP address was not found in the database and has been added."}
```



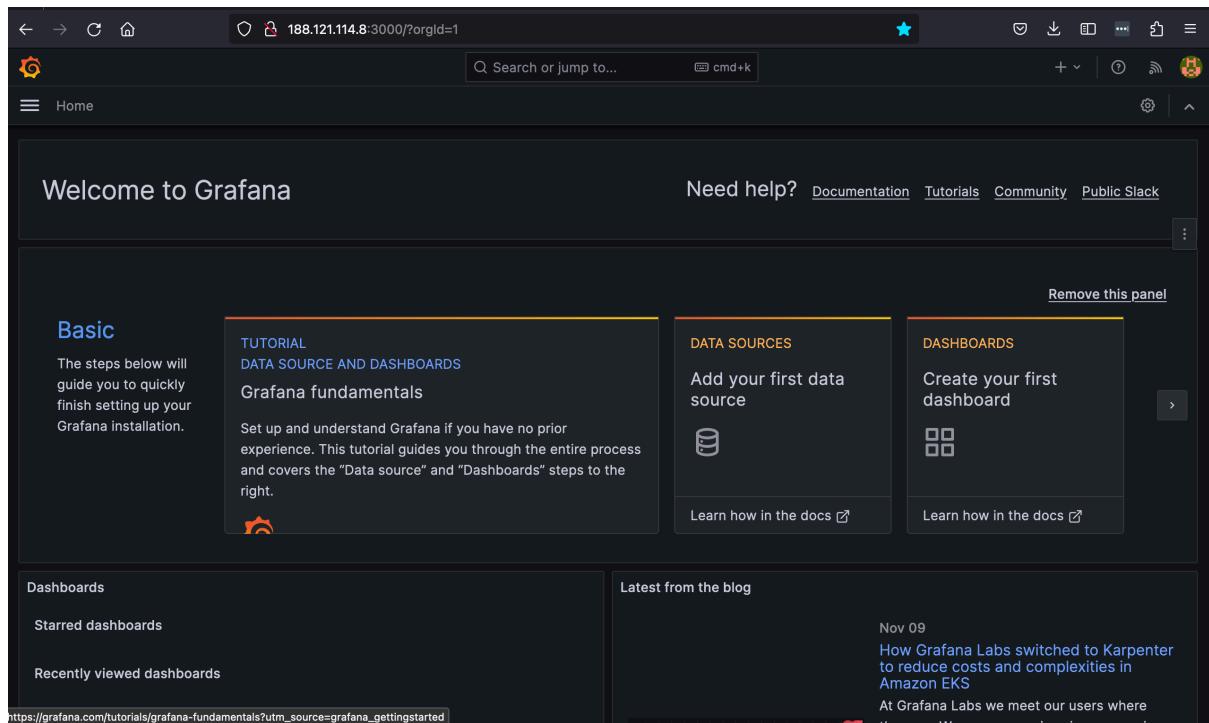
```
5.34.202.84:31991/records
{"count": 2}
```

Phase 8(service-monitoring):

My Grafana that was installed by snap didn't have http source and it cannot possible to add it, so I install Grafana by apt get in another server by this link:

<https://grafana.com/docs/grafana/latest/setup-grafana/installation/debian/>

here is the new grafan:



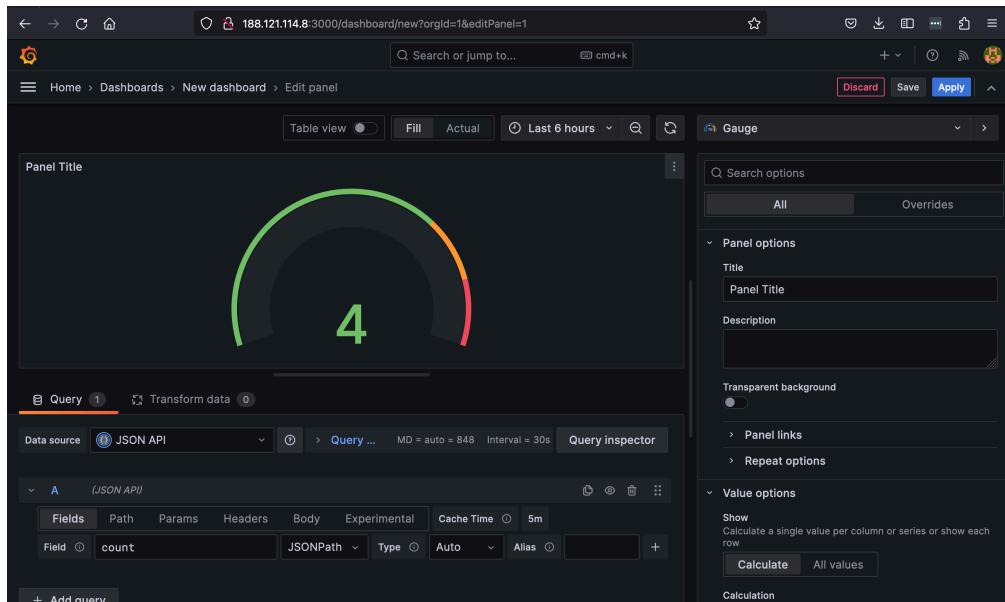
And now we should install this data source:

The screenshot shows the Grafana Marketplace interface. On the left sidebar, there's a 'Connections' section with a 'Data sources' item highlighted. In the main content area, a card for the 'JSON API' plugin is displayed. The card includes the plugin icon, name ('JSON API'), version ('1.3.8'), author ('Marcus Olsson'), download count ('10,639,761'), dependency requirement ('Grafana >=8.5.13'), and a 'Signed' badge. Below the card, a brief description states: 'A data source plugin for loading JSON APIs into Grafana.' It also links to 'Documentation', 'Website', and 'License'. A navigation bar at the bottom of the card has 'Overview' selected. The overall theme is dark.

And add it:

The screenshot shows the Grafana Settings page for the 'JSON API' data source. The left sidebar has a 'Data sources' item highlighted. The main content area shows the 'Settings' tab for the 'JSON API' data source. It includes a note 'Alerting not supported'. The 'HTTP' section contains fields for 'URL' (set to 'http://5.34.202.84:31991/records'), 'Allowed cookies' (with a 'New tag' input field and an 'Add' button), and 'Timeout' (set to 'Timeout in seconds'). The 'Auth' section includes options for 'Basic auth' (disabled), 'With Credentials' (disabled), 'TLS Client Auth' (disabled), 'With CA Cert' (disabled), and 'Skip TLS Verify' (disabled). At the top right, there are 'Explore data' and 'Build a dashboard' buttons.

And create dashboard for it :



Now their service monitoring is ready:

