DevOps Project Report

Arthur BILLEBAUT, Lou BRUNESSEAUX, Pauline DAVID, Hugo PANEL

Photo by - Manouar from Pexels

Professor: M. Lazhar HAMEL
DevOps Course — SE1 Promo. 2025, Efrei Paris

Table of Contents

Introduction

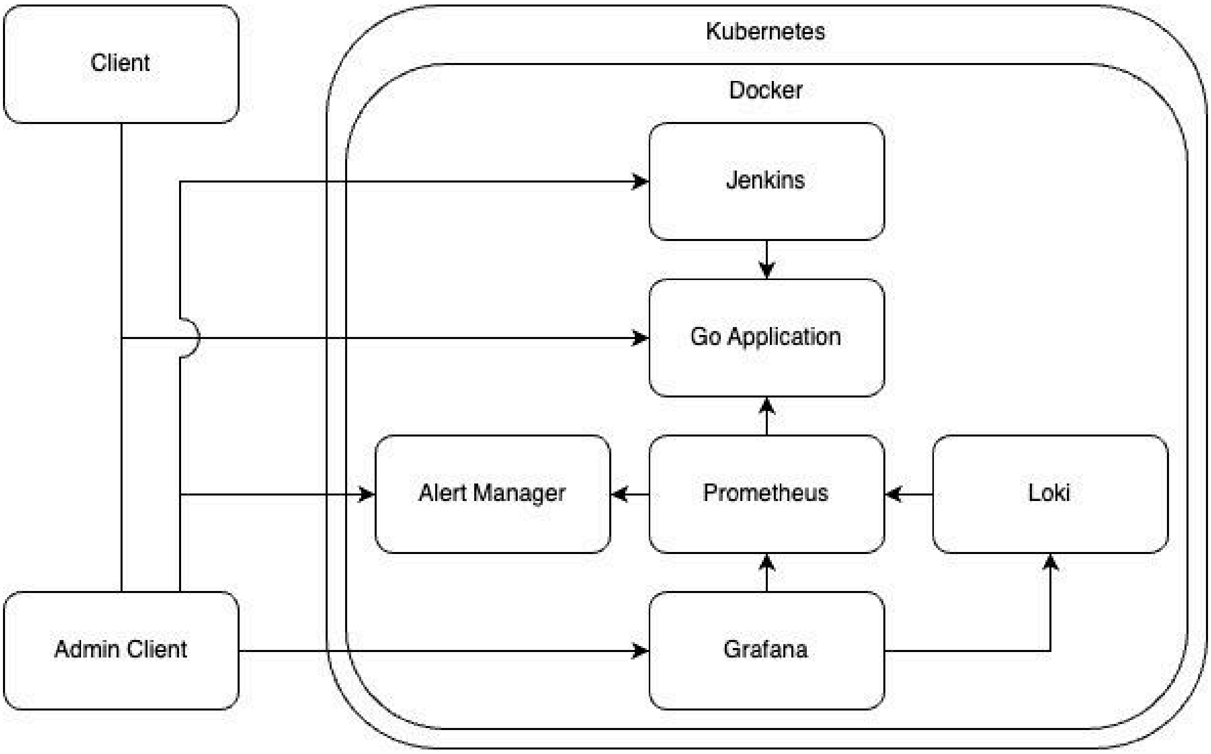
In this report, we will go through everything we did to realize the project. For each question, we will provide screenshots or explanations on how we did it, and how you can do it too in your own environment.

We created a GitHub repository with the files and commands we used. You can access it here : https://github.com/paulinedavid/project_dev_ops.git

Steps taken

Part One – Build and Deploy an application using Docker / Kubernetes and Jenkins pipeline.

1. Here is our diagram :

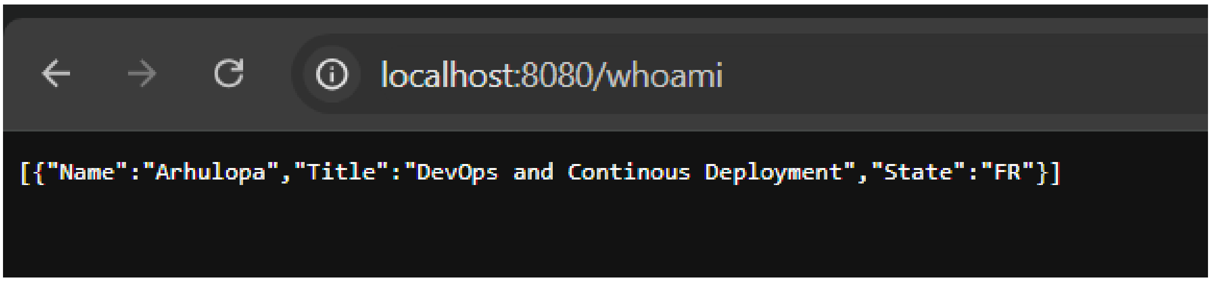


We can imagine two types of users for our project: regular users, and administrators. The regular user ("client") can only access the Go application whereas the administrator ("admin client") can also access Jenkins and Grafana to manage and monitor the application.

As we can see from the graph, the Go application, Jenkins, Prometheus, Alert Manager, Loki and Grafana are all inside of the Kubernetes and Docker containers. This is because they are run as Kubernetes services using the Docker driver.

Now that we know what we want to achieve, let's start building this project with the following questions...

2. We customized the app so that the /whoami endpoint displays our team name (Arhulopa):

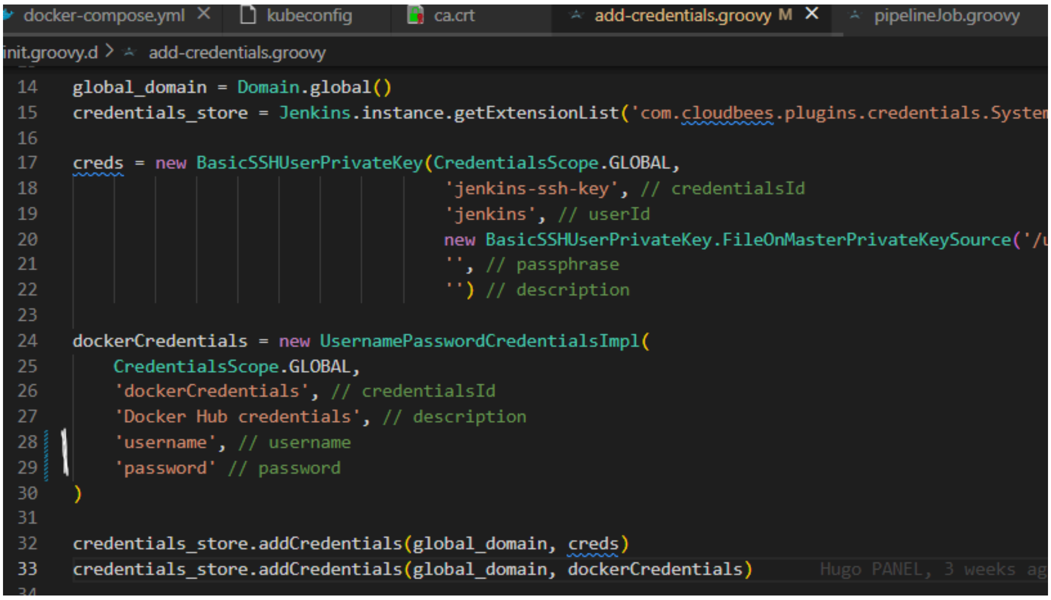


```
[{"Name":"Arhulopa","Title":"DevOps and Continous Deployment","State":"FR"}]
```

Then, we created the pipelineJob.groovy to represent the pipeline job that will build the image of the app and deploy it using docker engine.
In order to test this, you can simply clone the github project on your machine, then in a terminal at the root of the project execute the following commands :

```
docker run -d -p 8080:8080 -p 50000:50000 -v /usr/bin/docker:/usr/bin/docker -v /run/docker.sock:/run/docker.sock --name jenkins --restart unless-stopped jenkins/jenkins:lts-jdk17
```

Then, update the add-credentials.groovy file with your own docker credentials :



```
global_domain = Domain.global()
credentials_store = Jenkins.instance.getExtensionList('com.cloudbees.plugins.credentials.Systen

creds = new BasicSSHUserPrivateKey(CredentialsScope.GLOBAL,
                                   'jenkins-ssh-key', // credentialsId
                                   'jenkins', // userId
                                   new BasicSSHUserPrivateKey.FileOnMasterPrivateKeySource('/u
                                   '', // passphrase
                                   '') // description

dockerCredentials = new UsernamePasswordCredentialsImpl(
    CredentialsScope.GLOBAL,
    'dockerCredentials', // credentialsId
    'Docker Hub credentials', // description
    'username', // username
    'password' // password
)

credentials_store.addCredentials(global_domain, creds)
credentials_store.addCredentials(global_domain, dockerCredentials)
```

Then run this command to update the content of the container :

```
docker-compose up --build
```

<u>Good to know:</u>

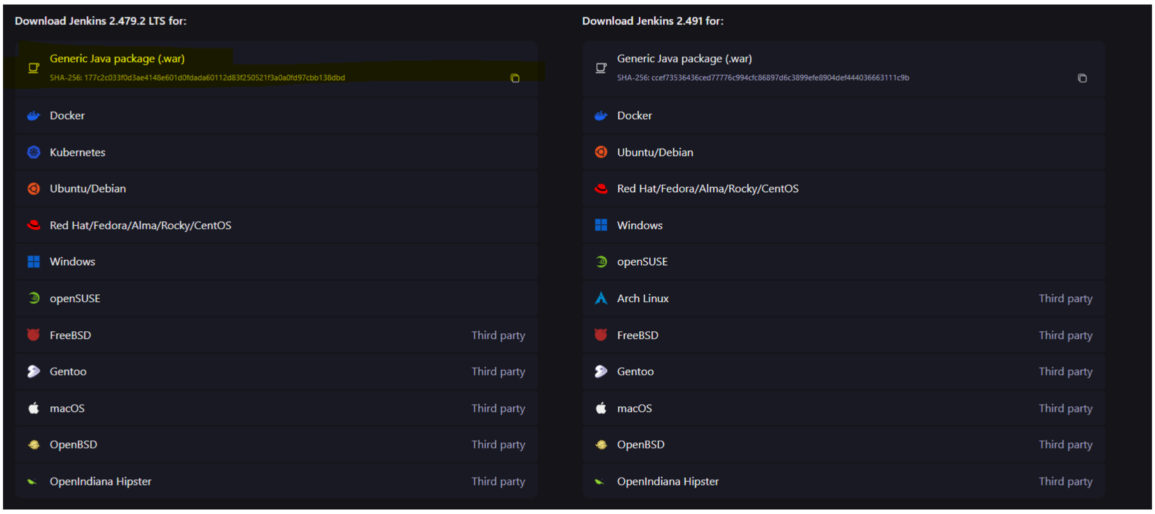
We also have two more .groovy files in the init.groovy.d folder: add-security.groovy and basic-security.groovy.
The add-agent.groovy file adds a new agent to Jenkins, as using the built-in node is considered bad practice.

The basic-security.groovy file is used to change the admin credentials. This allows you to use “admin”:”admin” to log in with the admin account, instead of having to use the default Jenkins password which changes every time.
The init.groovy.d folder is a special folder that is automatically run by Docker when the Jenkins container is started for the first time. This is why we put our .groovy scripts there, so they can be run when we add Jenkins to our project.

3. After many and many tries to do this part inside a docker container, we decided to do this question with jenkins and kubernetes directly installed on our computers as we couldn’t achieve it in a different way.

If you want to install jenkins locally as we did, you can follow the following instructions :

Start by downloading the .war file from here : https://www.jenkins.io/download/



Download Jenkins 2.479.2 LTS for:

Generic Java package (.war) SHA-256: 177c2c033f0d3ae4148e601d0fdada60112d83f250521f3a0a0fd97cbb138dbd	
Docker	
Kubernetes	
Ubuntu/Debian	
Red Hat/Fedora/Alma/Rocky/CentOS	
Windows	
openSUSE	
FreeBSD	Third party
Gentoo	Third party
macOS	Third party
OpenBSD	Third party
OpenIndiana Hipster	Third party

Download Jenkins 2.491 for:

Generic Java package (.war) SHA-256: ccef73536436ced77776c994cfc86897d6c3899efe8904def444036663111c9b	
Docker	
Ubuntu/Debian	
Red Hat/Fedora/Alma/Rocky/CentOS	
Windows	
openSUSE	
Arch Linux	Third party
FreeBSD	Third party
Gentoo	Third party
macOS	Third party
OpenBSD	Third party
OpenIndiana Hipster	Third party

Then, go to where it’s downloaded and run `java -jar jenkins.war`

If you need to free your port 8080, here are the commands to do so :

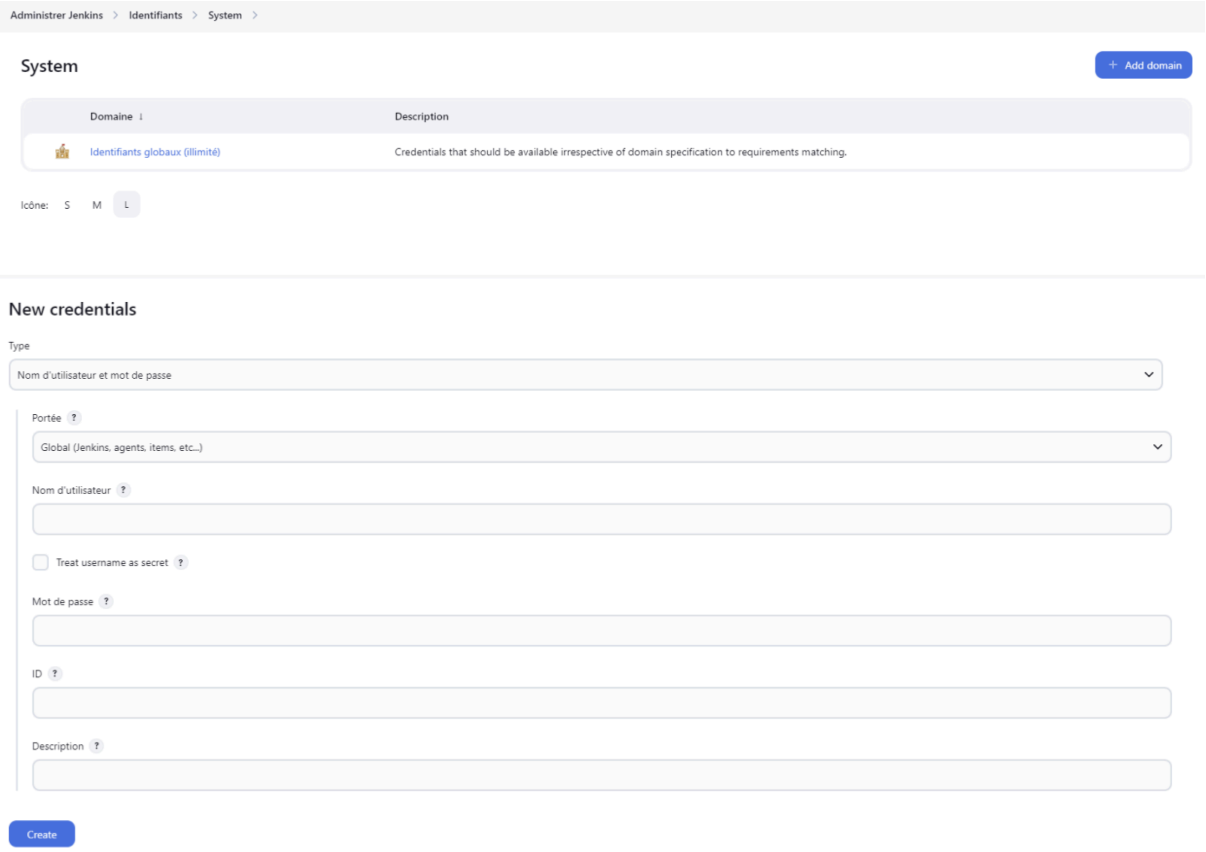
- Windows : `netstat -ano | findstr :8080`, find the id of the process and then `taskkill /PID <process-id> /F`
- Linux/Mac : `lsof -i :8080`, find the id of the process and then `kill -9 <process-id>`
- Docker : `docker ps | grep 8080`, find the id of the container and then `docker stop <container-id>`

Store your initial admin password somewhere you can easily access (so that you can connect to your admin account).
If you need, it should be stored at the following path :

- Windows : C:\Program Files (x86)\Jenkins\secrets\initialAdminPassword
- Linux/Mac : /var/lib/jenkins/secrets/initialAdminPassword

Then in Jenkins (http://localhost:8080) :

- Create dockerCredentials (manage jenkins -> credentials -> system -> global -> add credentials) : add your Docker credentials and save them under the name “dockerCredentials”



System

Domaine ↓	Description
Identifiants globaux (illimité)	Credentials that should be available irrespective of domain specification to requirements matching.

New credentials

Type: Nom d'utilisateur et mot de passe

Portée ?: Global (Jenkins, agents, items, etc...)

Nom d'utilisateur ?

□ Treat username as secret ?

Mot de passe ?

ID ?

Description ?

- Add laptop label to built-in node (manage jenkins -> node -> controller -> configure).

- Install ‘Docker pipeline” in the plugins (manage jenkins -> plugins -> plugins to install -> search Docker pipeline, select it and install)

- Create pipeline in jenkins (create job -> name = “pipeline” -> type pipeline -> create -> configure -> copy paste the content of the file corresponding to your OS :
    - Linux : “pipelineKubernetesLinux.groovy”
    - Windows : “pipelineKubernetesWindows.groovy”
    - MacOS : “pipelineKubernetesMacOS.groovy”)

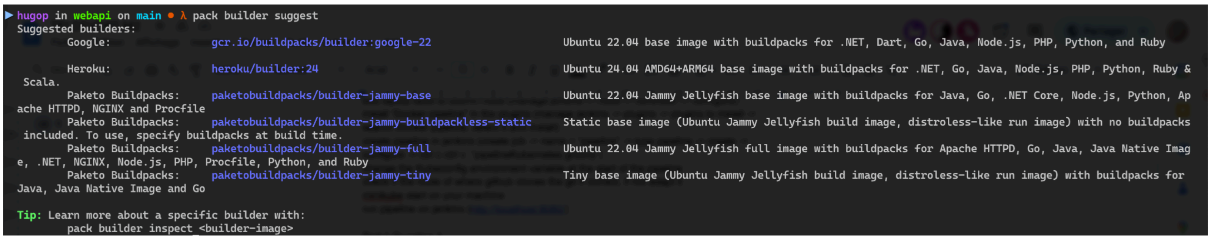
- Change the Kubeconfig environment variable at the start of the pipeline (it should refer to the location of the config of kubernetes, for examples : "C:\\Users\\me\\.kube\\config")
- Check if the route of where github clones the git if correct, if not adapt it
- `minikube start` on your machine
- Run your pipeline job : the build should conclude in a success

4. We started by installing the pack CLI. To do so we used the following commands based on our OS :

- Windows : `choco install pack --version=0.36.0`

- MacOS : `brew install buildpacks/tap/pack`

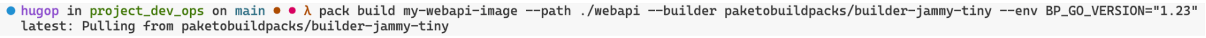
Then, we used the command `pack builder suggest` to get a list of suggested builders for our app :



```
▶ hugop in webapi on main ● λ pack builder suggest
Suggested builders:
        Google:                gcr.io/buildpacks/builder:google-22                       Ubuntu 22.04 base image with buildpacks for .NET, Dart, Go, Java, Node.js, PHP, Python, and Ruby

        Heroku:                heroku/builder:24                                         Ubuntu 24.04 AMD64+ARM64 base image with buildpacks for .NET, Go, Java, Node.js, PHP, Python, Ruby &
 Scala.
        Paketo Buildpacks:     paketobuildpacks/builder-jammy-base                       Ubuntu 22.04 Jammy Jellyfish base image with buildpacks for Java, Go, .NET Core, Node.js, Python, Ap
ache HTTPD, NGINX and Procfile
        Paketo Buildpacks:     paketobuildpacks/builder-jammy-buildpackless-static       Static base image (Ubuntu Jammy Jellyfish build image, distroless-like run image) with no buildpacks
 included. To use, specify buildpacks at build time.
        Paketo Buildpacks:     paketobuildpacks/builder-jammy-full                       Ubuntu 22.04 Jammy Jellyfish full image with buildpacks for Apache HTTPD, Go, Java, Java Native Imag
e, .NET, NGINX, Node.js, PHP, Procfile, Python, and Ruby
        Paketo Buildpacks:     paketobuildpacks/builder-jammy-tiny                       Tiny base image (Ubuntu Jammy Jellyfish build image, distroless-like run image) with buildpacks for
Java, Java Native Image and Go

Tip: Learn more about a specific builder with:
        pack builder inspect <builder-image>
```

We selected the image paketobuildpacks/builder-jammy-tiny since it is small and supports Go.

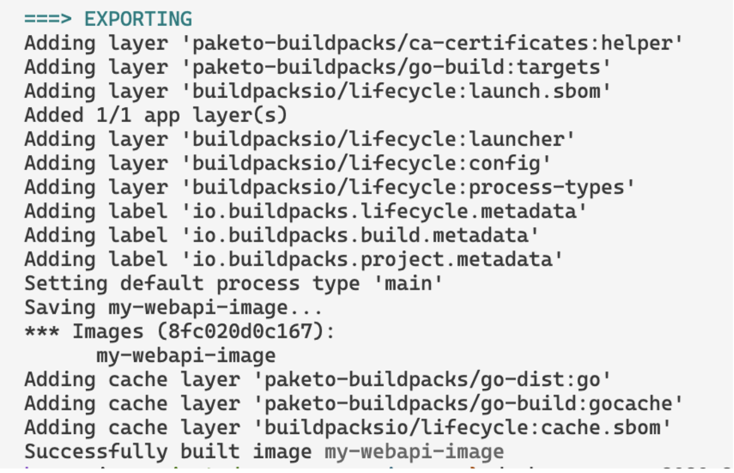
Once we have the name of the builder we want to use, we can build the image with `pack build`:



```
● hugop in project_dev_ops on main ● ● λ pack build my-webapi-image --path ./webapi --builder paketobuildpacks/builder-jammy-tiny --env BP_GO_VERSION="1.23"
  latest: Pulling from paketobuildpacks/builder-jammy-tiny
```

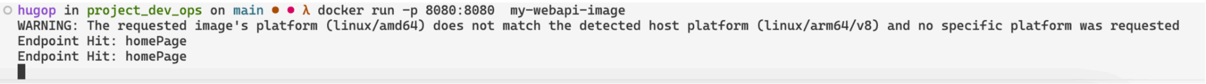
Note: With this command, we also show how we can define environment variables to force the builder to use a specific version of Go.

Here is a sample of the command's output :



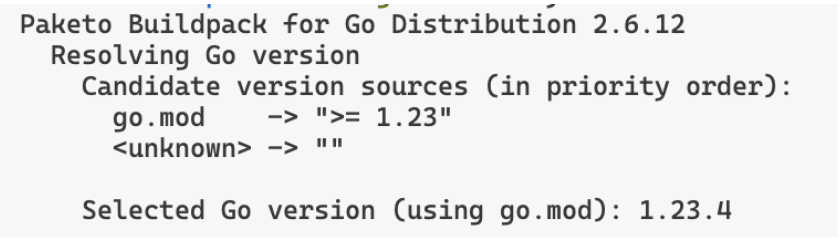
```
===> EXPORTING
Adding layer 'paketo-buildpacks/ca-certificates:helper'
Adding layer 'paketo-buildpacks/go-build:targets'
Adding layer 'buildpacksio/lifecycle:launch.sbom'
Added 1/1 app layer(s)
Adding layer 'buildpacksio/lifecycle:launcher'
Adding layer 'buildpacksio/lifecycle:config'
Adding layer 'buildpacksio/lifecycle:process-types'
Adding label 'io.buildpacks.lifecycle.metadata'
Adding label 'io.buildpacks.build.metadata'
Adding label 'io.buildpacks.project.metadata'
Setting default process type 'main'
Saving my-webapi-image...
*** Images (8fc020d0c167):
      my-webapi-image
Adding cache layer 'paketo-buildpacks/go-dist:go'
Adding cache layer 'paketo-buildpacks/go-build:gocache'
Adding cache layer 'buildpacksio/lifecycle:cache.sbom'
Successfully built image my-webapi-image
```

Once the image is built, we can create a container from it with docker run:



```
○ hugop in project_dev_ops on main ● ● λ docker run -p 8080:8080  my-webapi-image
  WARNING: The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) and no specific platform was requested
  Endpoint Hit: homePage
  Endpoint Hit: homePage
  █
```

The main difference lies in how dependencies are handled. With the Dockerfile, we specify the name of the base image and its version ourselves ("FROM golang:1.23"). According to the buildpacks documentation, this can create problems on large projects if each team is responsible for their own Dockerfiles. Instead, buildpacks chooses the appropriate version automatically. If we run the same command as before (`pack build`) but without specifying a Go version as an environment variable, we get the following in the output:



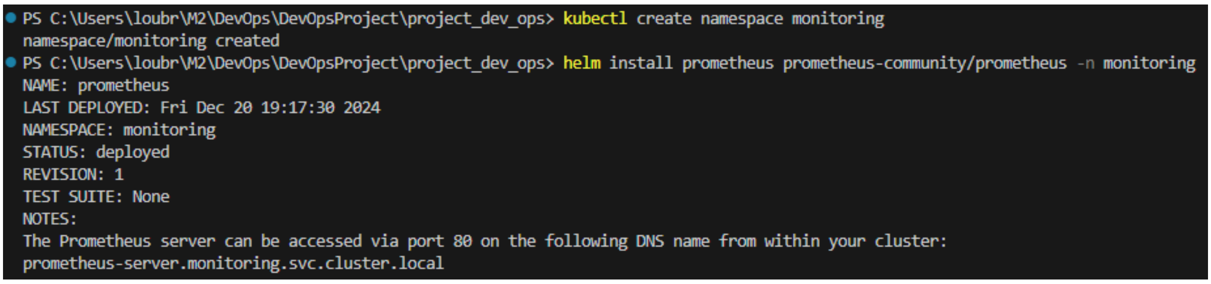
```
Paketo Buildpack for Go Distribution 2.6.12
  Resolving Go version
    Candidate version sources (in priority order):
      go.mod    -> ">= 1.23"
      <unknown> -> ""

    Selected Go version (using go.mod): 1.23.4
```

As we can see, the Go version was chosen not from the Dockerfile or from a custom configuration, but from the project's go.mod file directly.

Part Two – Monitoring and Incident Management for containerized application

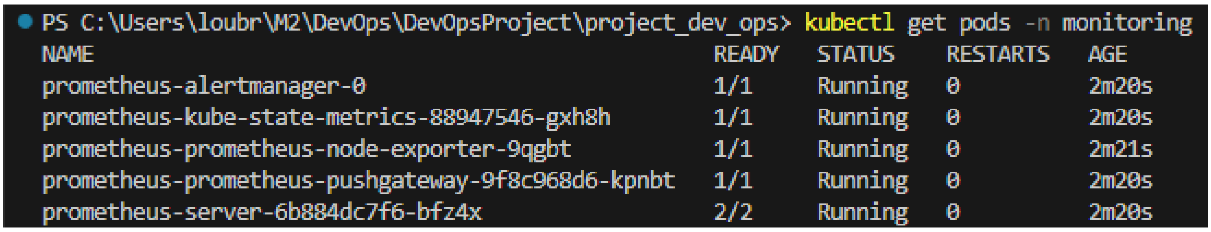
1. We start by installing the grafana and prometheus helm charts (we create a monitoring namespace dedicated to stacks)

```
kubectl create namespace monitoring
helm install prometheus prometheus-community/prometheus -n monitoring
```



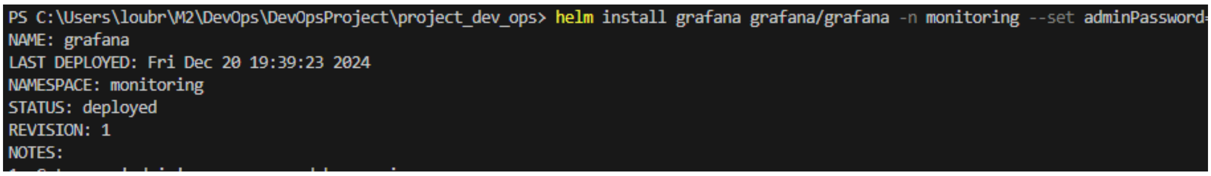
```
PS C:\Users\loubr\M2\DevOps\DevOpsProject\project_dev_ops> kubectl create namespace monitoring
namespace/monitoring created
PS C:\Users\loubr\M2\DevOps\DevOpsProject\project_dev_ops> helm install prometheus prometheus-community/prometheus -n monitoring
NAME: prometheus
LAST DEPLOYED: Fri Dec 20 19:17:30 2024
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
```

```
kubectl get pods -n monitoring
```



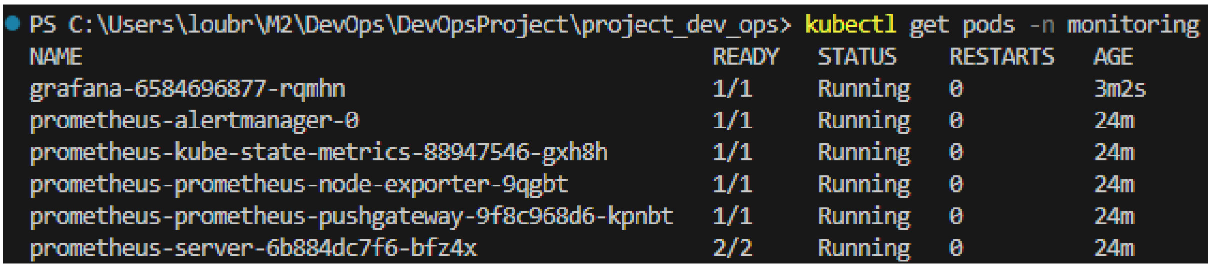
```
PS C:\Users\loubr\M2\DevOps\DevOpsProject\project_dev_ops> kubectl get pods -n monitoring
NAME                                                 READY   STATUS    RESTARTS   AGE
prometheus-alertmanager-0                            1/1     Running   0          2m20s
prometheus-kube-state-metrics-88947546-gxh8h         1/1     Running   0          2m20s
prometheus-prometheus-node-exporter-9qgbt            1/1     Running   0          2m21s
prometheus-prometheus-pushgateway-9f8c968d6-kpnbt    1/1     Running   0          2m20s
prometheus-server-6b884dc7f6-bfz4x                   2/2     Running   0          2m20s
```

```
helm install grafana grafana/grafana -n monitoring -set adminPassword=<your password>
```



```
PS C:\Users\loubr\M2\DevOps\DevOpsProject\project_dev_ops> helm install grafana grafana/grafana -n monitoring --set adminPassword=
NAME: grafana
LAST DEPLOYED: Fri Dec 20 19:39:23 2024
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
NOTES:
```

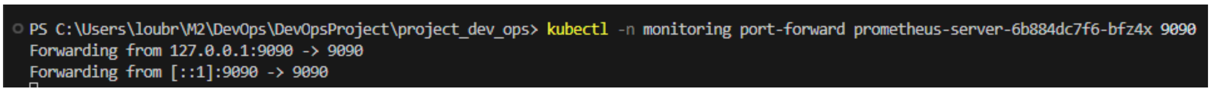
```
kubectl get pods -n monitoring
```



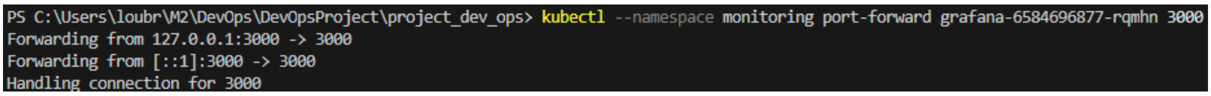
```
PS C:\Users\loubr\M2\DevOps\DevOpsProject\project_dev_ops> kubectl get pods -n monitoring
NAME                                                READY   STATUS    RESTARTS   AGE
grafana-6584696877-rqmhn                            1/1     Running   0          3m2s
prometheus-alertmanager-0                           1/1     Running   0          24m
prometheus-kube-state-metrics-88947546-gxh8h        1/1     Running   0          24m
prometheus-prometheus-node-exporter-9qgbt           1/1     Running   0          24m
prometheus-prometheus-pushgateway-9f8c968d6-kpnbt   1/1     Running   0          24m
prometheus-server-6b884dc7f6-bfz4x                  2/2     Running   0          24m
```

In order to access Grafana and Prometheus, we expose them and get their URL:

```
kubectl -n monitoring port-forward <prometheus-podname> 9090
kubectl -n monitoring port-forward <grafana-podname> 3000
```

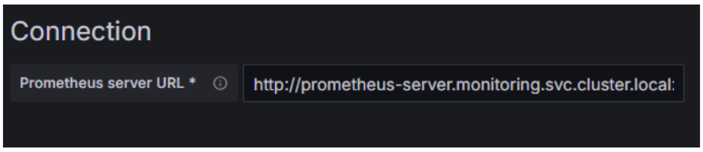


```
PS C:\Users\loubr\M2\DevOps\DevOpsProject\project_dev_ops> kubectl -n monitoring port-forward prometheus-server-6b884dc7f6-bfz4x 9090
Forwarding from 127.0.0.1:9090 -> 9090
Forwarding from [::1]:9090 -> 9090
```



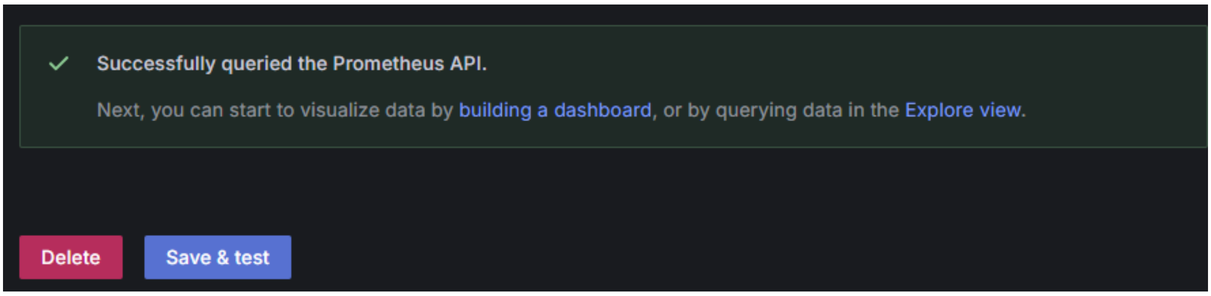
```
PS C:\Users\loubr\M2\DevOps\DevOpsProject\project_dev_ops> kubectl --namespace monitoring port-forward grafana-6584696877-rqmhn 3000
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
Handling connection for 3000
```

We can then access Grafana Web UI and configure a data source with the deployed Prometheus service URL.



Connection

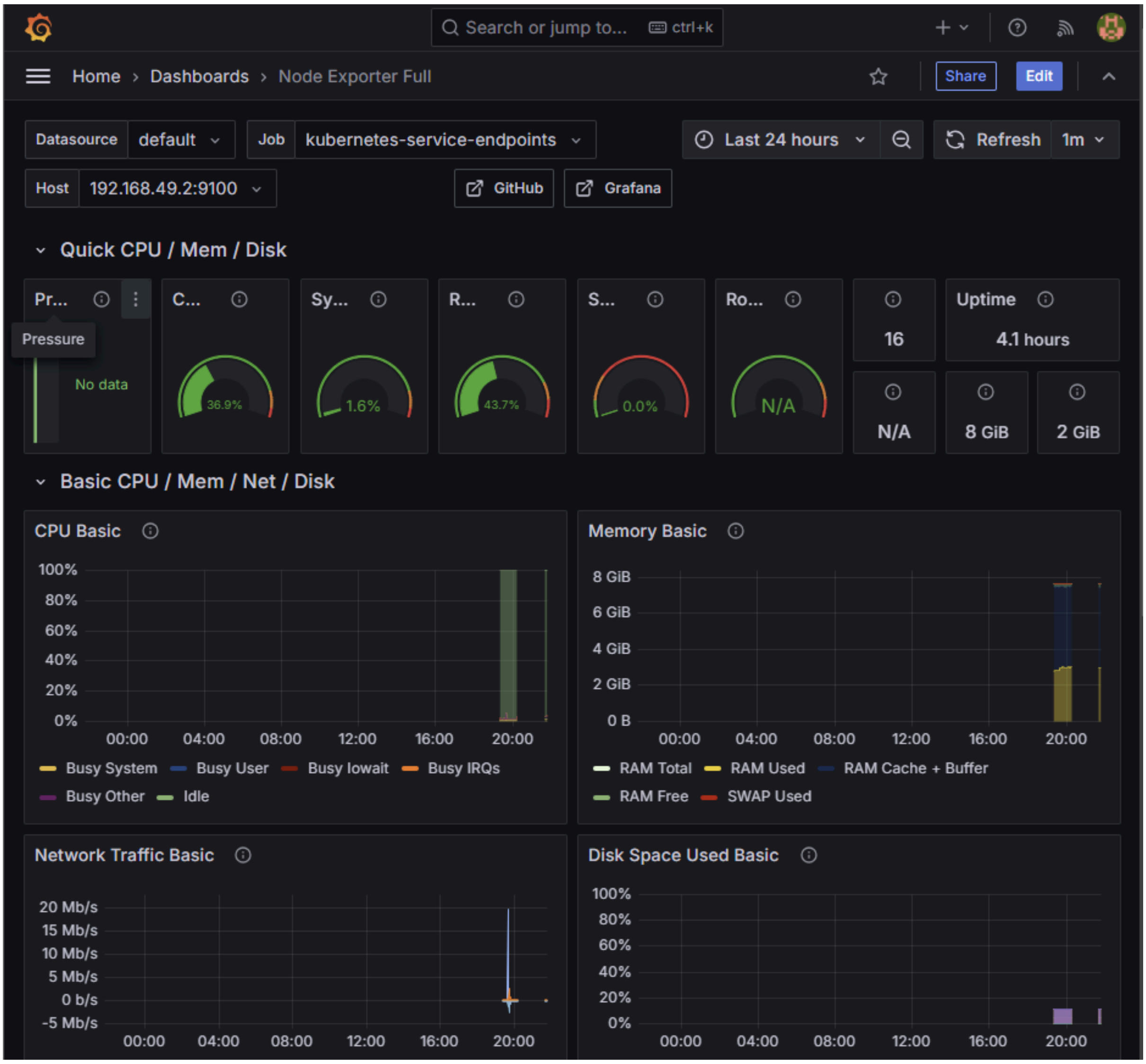
Prometheus server URL * : http://prometheus-server.monitoring.svc.cluster.local:



✓ Successfully queried the Prometheus API.

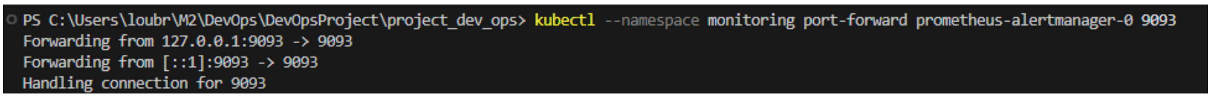
Next, you can start to visualize data by building a dashboard, or by querying data in the Explore view.

Make sure to use the correctly configured datasource in a new dashboard to visualize system data



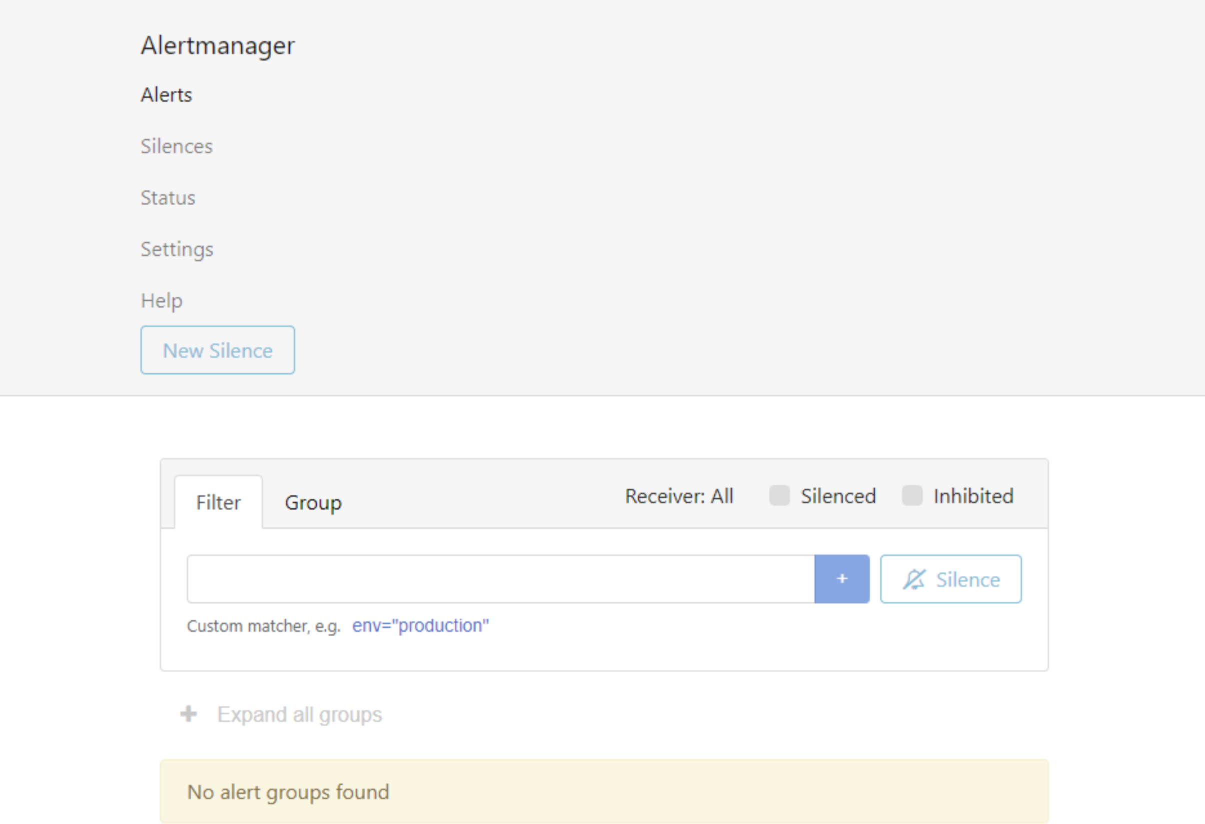
2. We then configured the prometheus alert management pod:

```
kubectl --namespace monitoring port-forward prometheus-alertmanager-0 9093
```



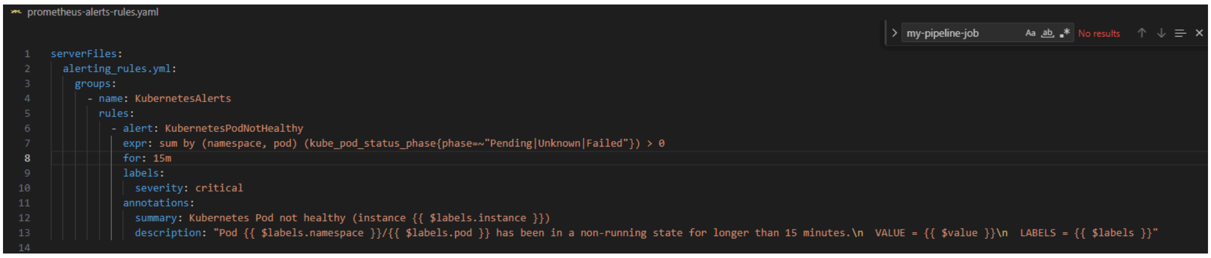
```
PS C:\Users\loubr\M2\DevOps\DevOpsProject\project_dev_ops> kubectl --namespace monitoring port-forward prometheus-alertmanager-0 9093
Forwarding from 127.0.0.1:9093 -> 9093
Forwarding from [::1]:9093 -> 9093
Handling connection for 9093
```

We then connected to it using the provided port👏



No alert groups found

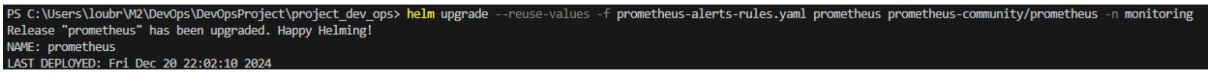
We went on to create a prometheus-alert-rules.yaml file to specify what alert the alert manager should watch out for :



```yaml
serverFiles:
  alerting_rules.yml:
    groups:
      - name: KubernetesAlerts
        rules:
          - alert: KubernetesPodNotHealthy
            expr: sum by (namespace, pod) (kube_pod_status_phase{phase=~"Pending|Unknown|Failed"}) > 0
            for: 15m
            labels:
              severity: critical
            annotations:
              summary: Kubernetes Pod not healthy (instance {{ $labels.instance }})
              description: "Pod {{ $labels.namespace }}/{{ $labels.pod }} has been in a non-running state for longer than 15 minutes.\n  VALUE = {{ $value }}\n  LABELS = {{ $labels }}"

```

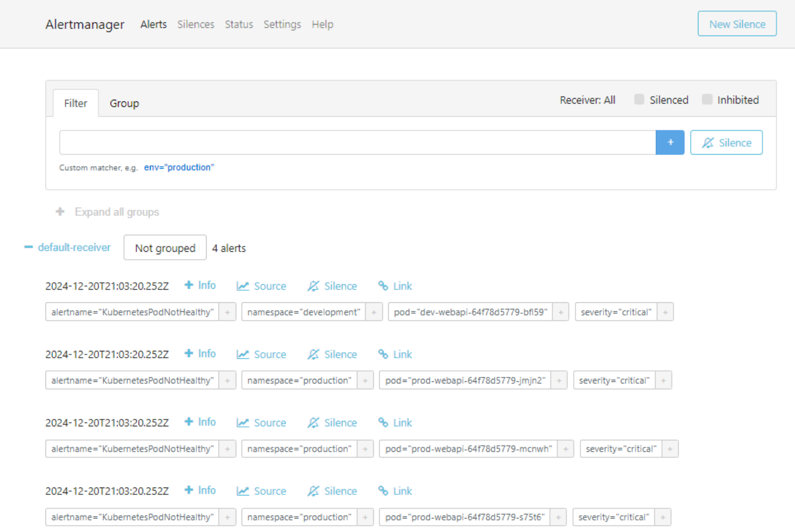
Once it is created, we update helm chart to make sure this new file is taken into consideration :

```
helm upgrade --reuse-values -f prometheus-alerts-rules.yaml prometheus prometheus-community/prometheus -n monitoring
```



```
PS C:\Users\loubr\M2\DevOps\DevOpsProject\project_dev_ops> helm upgrade --reuse-values -f prometheus-alerts-rules.yaml prometheus prometheus-community/prometheus -n monitoring
Release "prometheus" has been upgraded. Happy Helming!
NAME: prometheus
LAST DEPLOYED: Fri Dec 20 22:02:10 2024
```

Then, we wait for the alert conditions to be met, and see new alerts appear in Alert Manager :



default-receiver Not grouped 4 alerts

2024-12-20T21:03:20.252Z Info Source Silence Link
alertname="KubernetesPodNotHealthy" namespace="development" pod="dev-webapi-64f78d5779-bfl59" severity="critical"

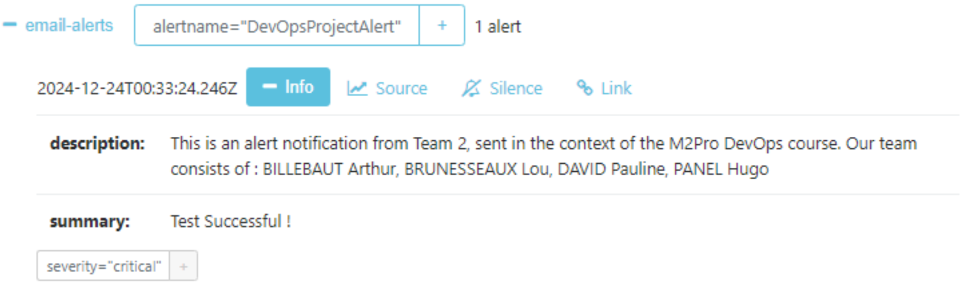
2024-12-20T21:03:20.252Z Info Source Silence Link
alertname="KubernetesPodNotHealthy" namespace="production" pod="prod-webapi-64f78d5779-jmjn2" severity="critical"

2024-12-20T21:03:20.252Z Info Source Silence Link
alertname="KubernetesPodNotHealthy" namespace="production" pod="prod-webapi-64f78d5779-mcnwh" severity="critical"

2024-12-20T21:03:20.252Z Info Source Silence Link
alertname="KubernetesPodNotHealthy" namespace="production" pod="prod-webapi-64f78d5779-s75t6" severity="critical"

3. Bonus (Mails)

We then added an alert that would send an email to your email address (and our own) every time it would be triggered. You can see its configuration and the mail it sends below :



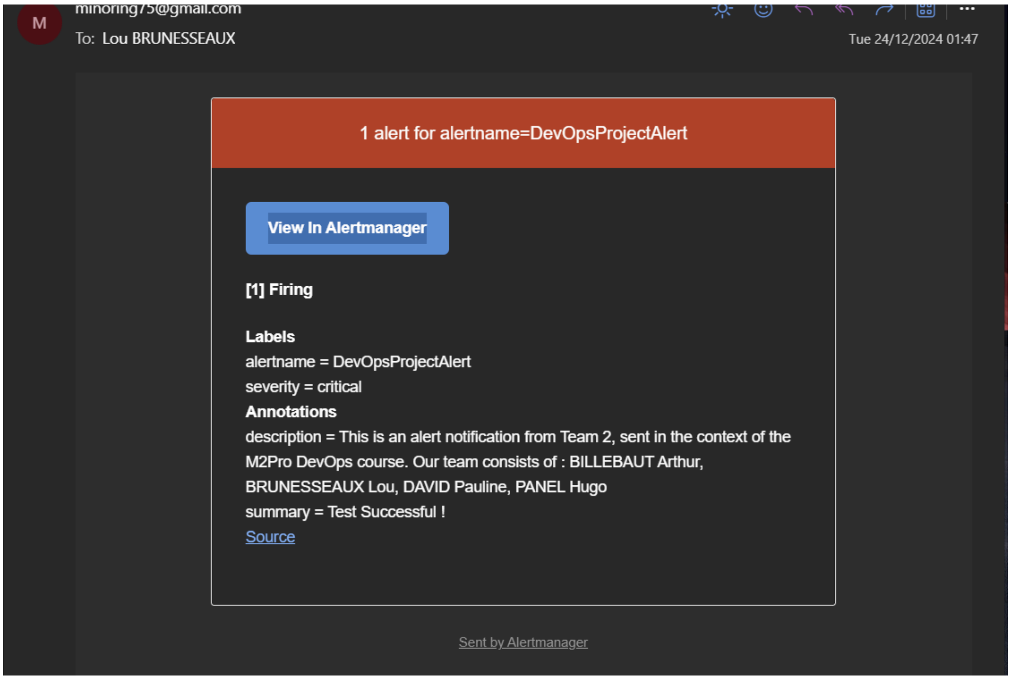
email-alerts alertname="DevOpsProjectAlert" 1 alert

2024-12-24T00:33:24.246Z Info Source Silence Link

description: This is an alert notification from Team 2, sent in the context of the M2Pro DevOps course. Our team consists of : BILLEBAUT Arthur, BRUNESSEAUX Lou, DAVID Pauline, PANEL Hugo

summary: Test Successful !

severity="critical"



minoring75@gmail.com
To: Lou BRUNESSEAUX
Tue 24/12/2024 01:47

1 alert for alertname=DevOpsProjectAlert

View In Alertmanager

[1] Firing

Labels
alertname = DevOpsProjectAlert
severity = critical
Annotations
description = This is an alert notification from Team 2, sent in the context of the M2Pro DevOps course. Our team consists of : BILLEBAUT Arthur, BRUNESSEAUX Lou, DAVID Pauline, PANEL Hugo
summary = Test Successful !
Source

Sent by Alertmanager

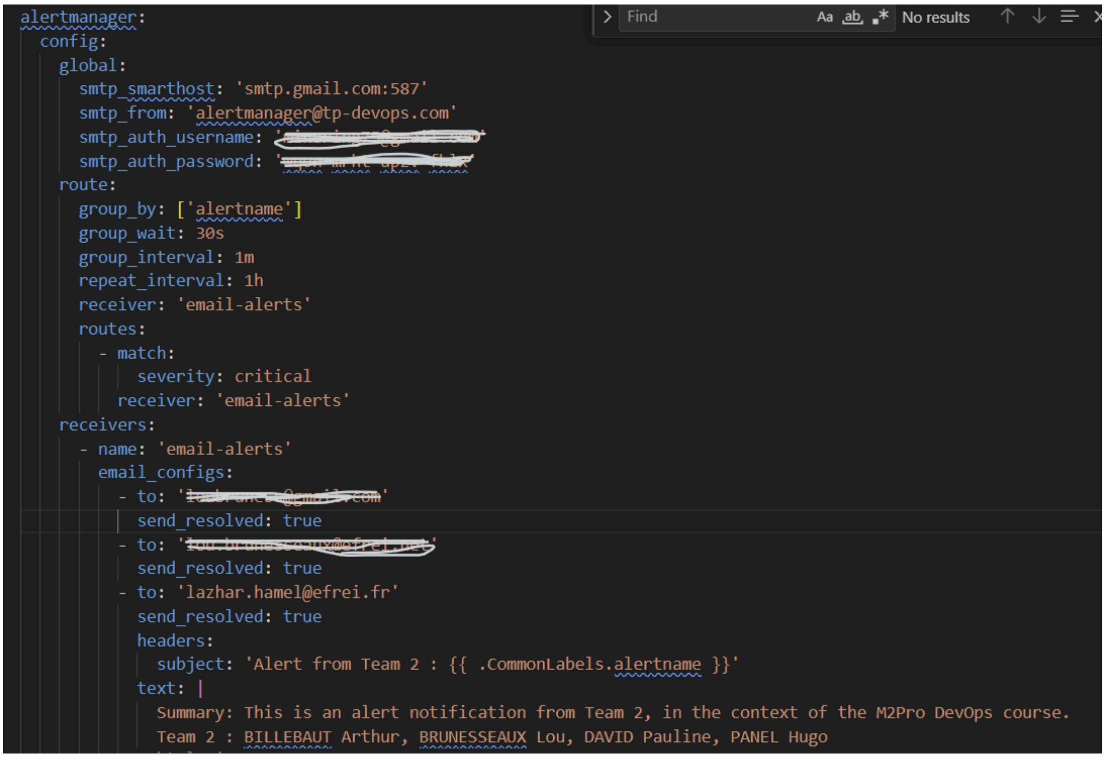
To do this step, we tried many things such as creating alertmanager-config.yaml files, creating specific alert files for prometheus, using prometheus-stack, creating configmaps, multiple forced configs.

What worked for us was :

- open the values that power prometheus in a new values.yaml file:

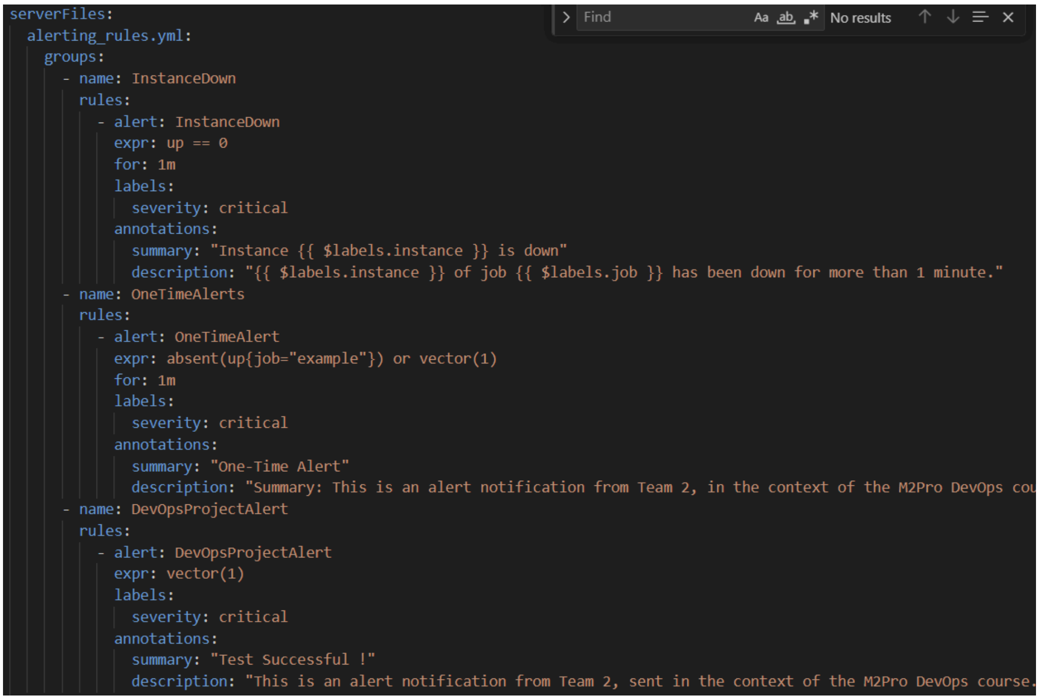
- execute the command : `helm show values prometheus-community/prometheus > values.yaml`

- Add the necessary parts in the values.yaml file : the alertmanager spec configuration lets us define the addresses that send and receive the alert messages, as well as the subjects and contents of the alert mails.



```yaml
alertmanager:
  config:
    global:
      smtp_smarthost: 'smtp.gmail.com:587'
      smtp_from: 'alertmanager@tp-devops.com'
      smtp_auth_username: '[redacted]'
      smtp_auth_password: '[redacted]'
    route:
      group_by: ['alertname']
      group_wait: 30s
      group_interval: 1m
      repeat_interval: 1h
      receiver: 'email-alerts'
      routes:
        - match:
            severity: critical
          receiver: 'email-alerts'
    receivers:
      - name: 'email-alerts'
        email_configs:
          - to: '[redacted]'
            send_resolved: true
          - to: '[redacted]'
            send_resolved: true
          - to: 'lazhar.hamel@efrei.fr'
            send_resolved: true
            headers:
              subject: 'Alert from Team 2 : {{ .CommonLabels.alertname }}'
            text: |
              Summary: This is an alert notification from Team 2, in the context of the M2Pro DevOps course.
              Team 2 : BILLEBAUT Arthur, BRUNESSEAUX Lou, DAVID Pauline, PANEL Hugo
```

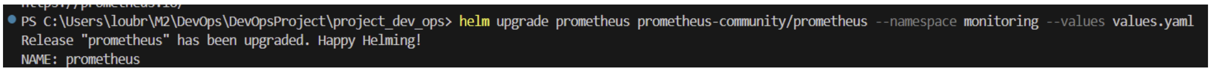
- Add custom alerts in the serverfiles :



```yaml
serverFiles:
  alerting_rules.yml:
    groups:
      - name: InstanceDown
        rules:
          - alert: InstanceDown
            expr: up == 0
            for: 1m
            labels:
              severity: critical
            annotations:
              summary: "Instance {{ $labels.instance }} is down"
              description: "{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 1 minute."
      - name: OneTimeAlerts
        rules:
          - alert: OneTimeAlert
            expr: absent(up{job="example"}) or vector(1)
            for: 1m
            labels:
              severity: critical
            annotations:
              summary: "One-Time Alert"
              description: "Summary: This is an alert notification from Team 2, in the context of the M2Pro DevOps cou
      - name: DevOpsProjectAlert
        rules:
          - alert: DevOpsProjectAlert
            expr: vector(1)
            labels:
              severity: critical
            annotations:
              summary: "Test Successful !"
              description: "This is an alert notification from Team 2, sent in the context of the M2Pro DevOps course.
```

We then made sure that all of it was applied with the following command:

```
helm upgrade prometheus prometheus-community/prometheus --namespace monitoring --values values.yaml
```



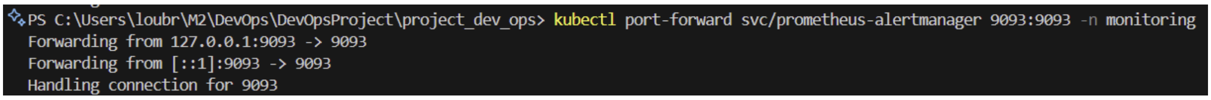
```
PS C:\Users\loubr\M2\DevOps\DevOpsProject\project_dev_ops> helm upgrade prometheus prometheus-community/prometheus --namespace monitoring --values values.yaml
Release "prometheus" has been upgraded. Happy Helming!
NAME: prometheus
```

A lot of synchronisation issues can happen, the most important thing to do is to check that the configmap file of the alertmanager reflects the changes of the prometheus values.yaml file. Here is the command to access it (do not modify it directly, simply check and adapt the values.yaml until they match):

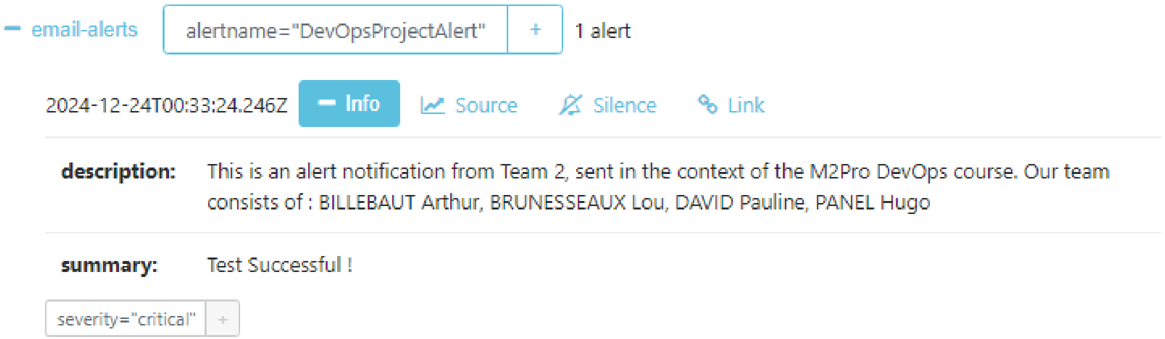
```
kubectl describe configmap prometheus-alertmanager -n monitoring
```

Once all is set, you can use this command to visualize your alerts :

```
kubectl port-forward svc/prometheus-alertmanager 9093:9093 -n monitoring
```



```
PS C:\Users\loubr\M2\DevOps\DevOpsProject\project_dev_ops> kubectl port-forward svc/prometheus-alertmanager 9093:9093 -n monitoring
Forwarding from 127.0.0.1:9093 -> 9093
Forwarding from [::1]:9093 -> 9093
Handling connection for 9093
```



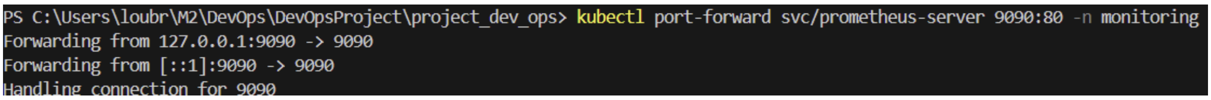
description: This is an alert notification from Team 2, sent in the context of the M2Pro DevOps course. Our team consists of : BILLEBAUT Arthur, BRUNESSEAUX Lou, DAVID Pauline, PANEL Hugo

summary: Test Successful !

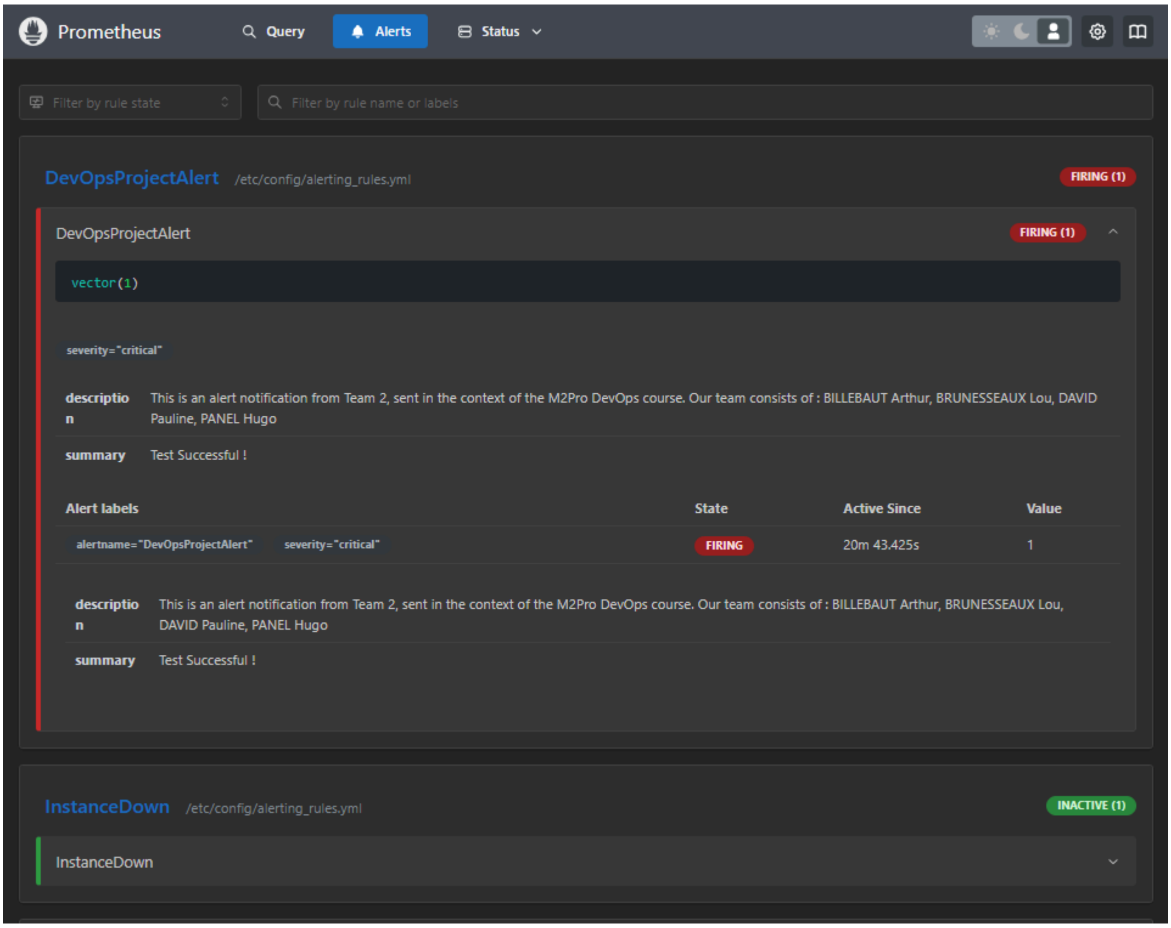
severity="critical"

And use the same principle to go check the nature of your alerts on prometheus :

```
kubectl port-forward svc/prometheus-server 9090:80 -n monitoring
```



```
PS C:\Users\loubr\M2\DevOps\DevOpsProject\project_dev_ops> kubectl port-forward svc/prometheus-server 9090:80 -n monitoring
Forwarding from 127.0.0.1:9090 -> 9090
Forwarding from [::1]:9090 -> 9090
Handling connection for 9090
```



DevOpsProjectAlert /etc/config/alerting_rules.yml FIRING (1)

DevOpsProjectAlert FIRING (1)

```
vector(1)
```

severity="critical"

description This is an alert notification from Team 2, sent in the context of the M2Pro DevOps course. Our team consists of : BILLEBAUT Arthur, BRUNESSEAUX Lou, DAVID Pauline, PANEL Hugo

summary Test Successful !

Alert labels	State	Active Since	Value
alertname="DevOpsProjectAlert" severity="critical"	FIRING	20m 43.425s	1

description This is an alert notification from Team 2, sent in the context of the M2Pro DevOps course. Our team consists of : BILLEBAUT Arthur, BRUNESSEAUX Lou, DAVID Pauline, PANEL Hugo

summary Test Successful !

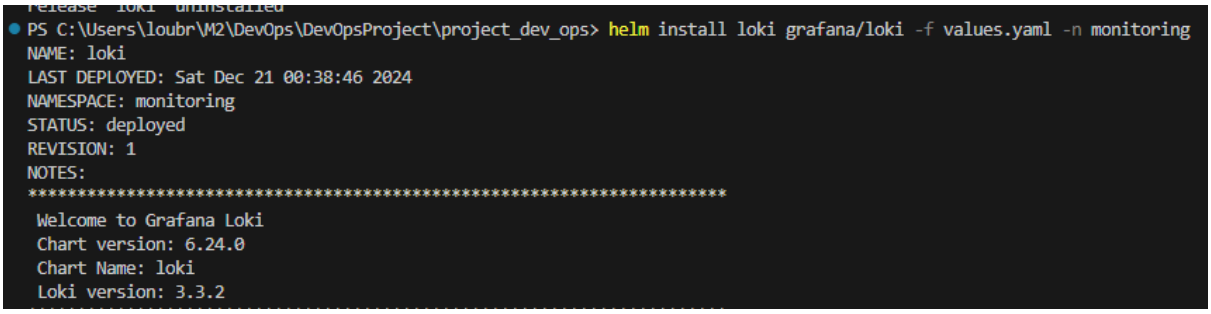
InstanceDown /etc/config/alerting_rules.yml INACTIVE (1)

InstanceDown

Part Three – Logs Management

1. We started by installing the grafana/loki chart from Grafana Official Helm Chart

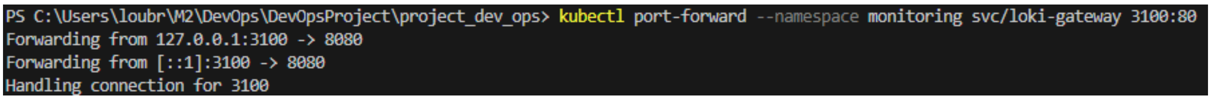
```
helm install loki grafana/loki -f values.yaml -n monitoring
```



```
release "loki" uninstalled
PS C:\Users\loubr\M2\DevOps\DevOpsProject\project_dev_ops> helm install loki grafana/loki -f values.yaml -n monitoring
NAME: loki
LAST DEPLOYED: Sat Dec 21 00:38:46 2024
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
NOTES:
***********************************************************************
 Welcome to Grafana Loki
 Chart version: 6.24.0
 Chart Name: loki
 Loki version: 3.3.2
***********************************************************************
```

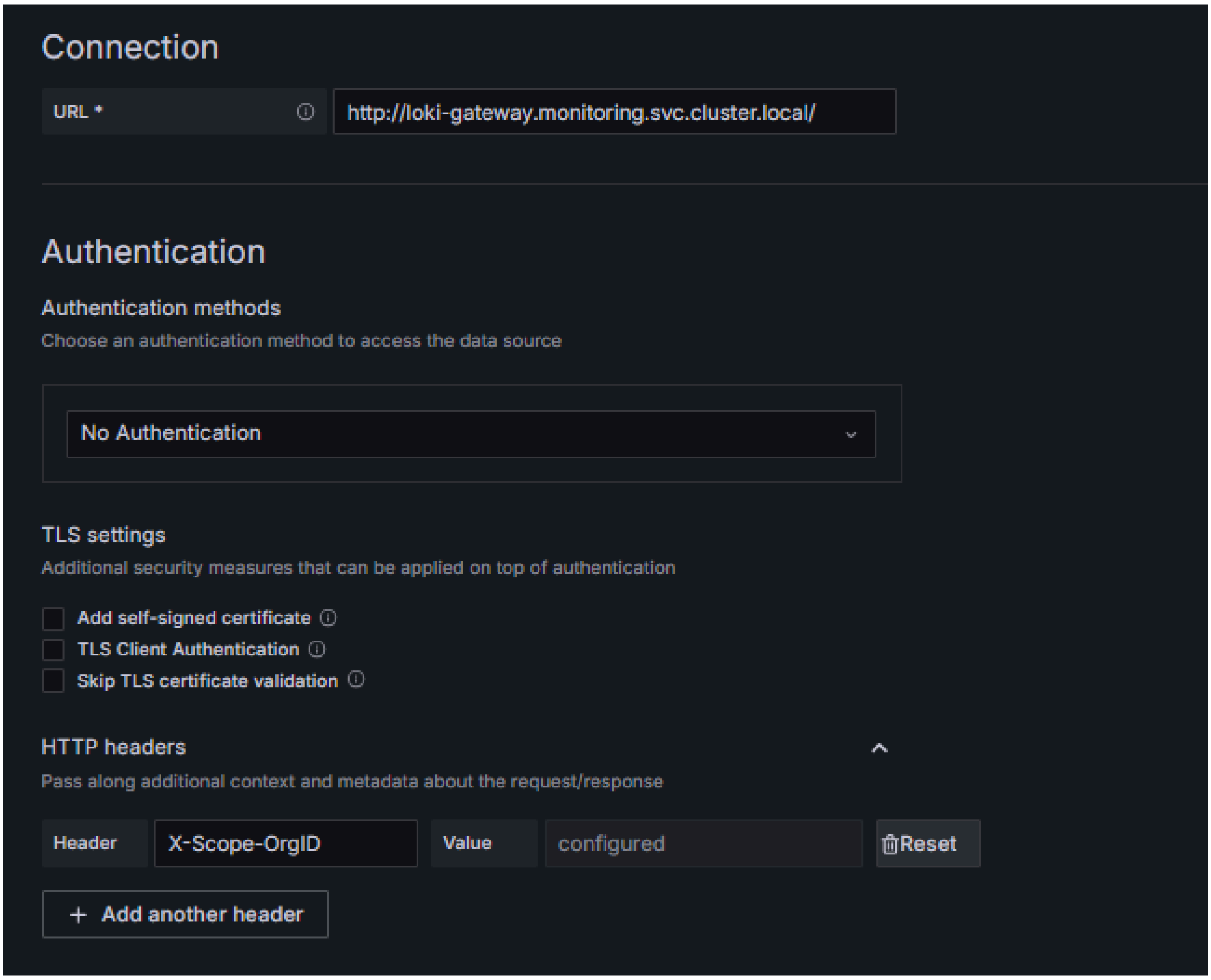
we can then connect to it to check if everything is ok 👍

`kubectl port-forward --namespace monitoring svc/loki-gateway 3100:80`



```
PS C:\Users\loubr\M2\DevOps\DevOpsProject\project_dev_ops> kubectl port-forward --namespace monitoring svc/loki-gateway 3100:80
Forwarding from 127.0.0.1:3100 -> 8080
Forwarding from [::1]:3100 -> 8080
Handling connection for 3100
```

We add loki as a data source in grafana by using the endpoint as an url and specifying a Header with the value "default" so that we don't get the output "no org id" which blocks access (very important).



Connection

URL * : http://loki-gateway.monitoring.svc.cluster.local/

Authentication

Authentication methods

Choose an authentication method to access the data source

No Authentication

TLS settings

Additional security measures that can be applied on top of authentication

□ Add self-signed certificate

□ TLS Client Authentication

□ Skip TLS certificate validation

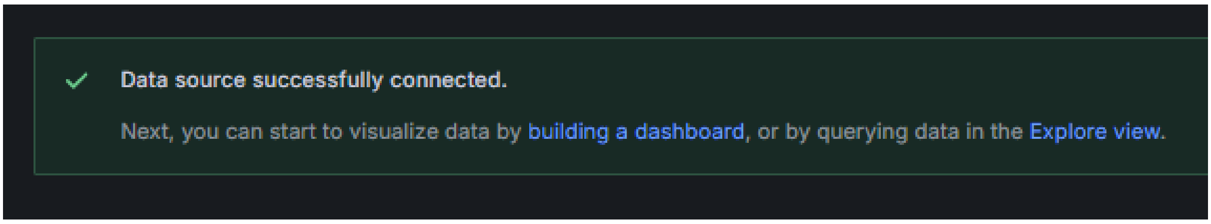
HTTP headers

Pass along additional context and metadata about the request/response

Header: X-Scope-OrgID Value: configured

This allows us to connect 👍

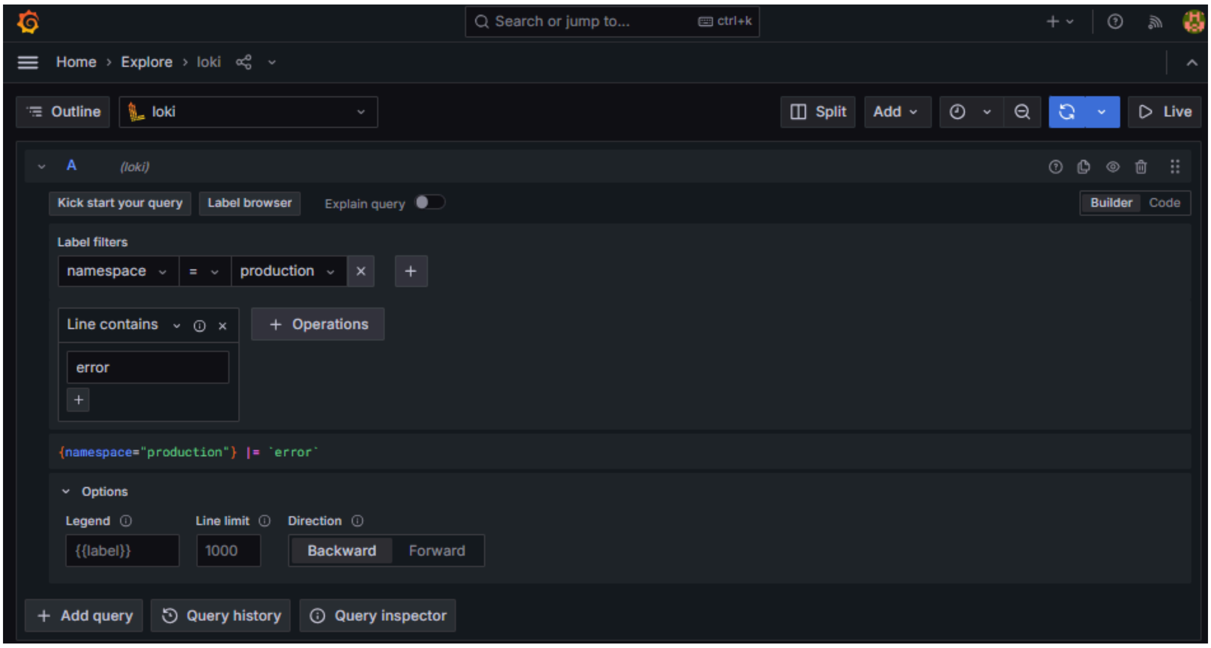
(We were stuck on this for a while. We tried initiating loki with a custom values.yaml file, rely on error-loggers and outside input like promtail +ansible, using port forwarding for the data source url, using temporary urls with minikube tunnels, reconfiguring the host file on windows, reconfiguring dockerHub's service range, reconfiguring minikube container's address on the docker Hub and many more.)



✓ Data source successfully connected.

Next, you can start to visualize data by building a dashboard, or by querying data in the Explore view.

We then add a query for logs with the word "error" in the *production* namespace:



A (loki)

Label filters: namespace = production

Line contains: error

```
{namespace="production"} |= `error`
```

Options

Legend: {{label}}

Line limit: 1000

Direction: Backward [✓] Forward

And finally, we create an error-logger.yaml file to create errors.

All the steps are now done, and everything works in harmony !