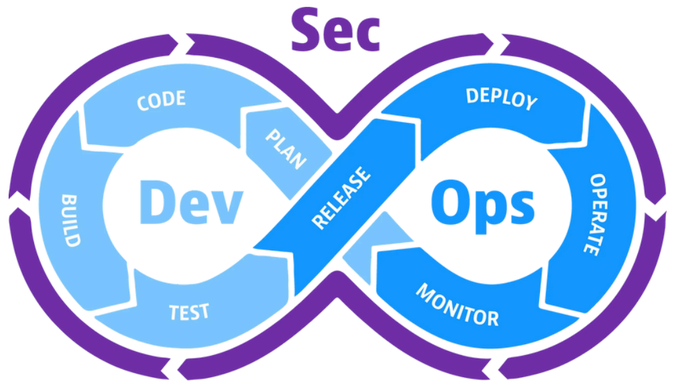
DEVSECOPS

PROJECT



DONE BY:

Amine Lachegur
Ait wahmane ELhoussaine
Ismail Iaraben
Abdellah Igajane
Abdelghafour Bouhdyd

LED BY :

Mr. ALLAKI Driss

PLAN

01 BACKLOG

02 CI/CD PIPELINE

03 SECURITY TESTS

04 MONITORING

05 KUBERNETES DEPLOYMENT

06 REPOSITORIES

1-BACKLOG

INTRODUCTION

In order to successfully carry out this project, a working methodology has been established within the team, utilizing the JIRA tool, specifically employing SCRUM boards for project management

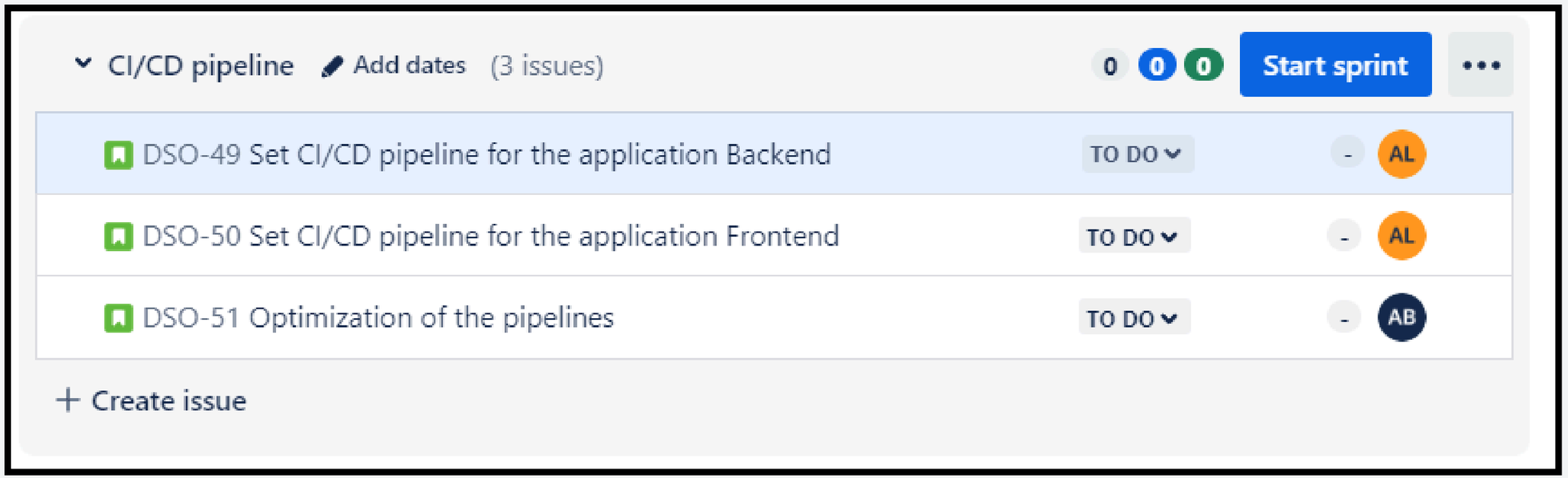
Initially, the project was divided into four sprints

Product Backlog

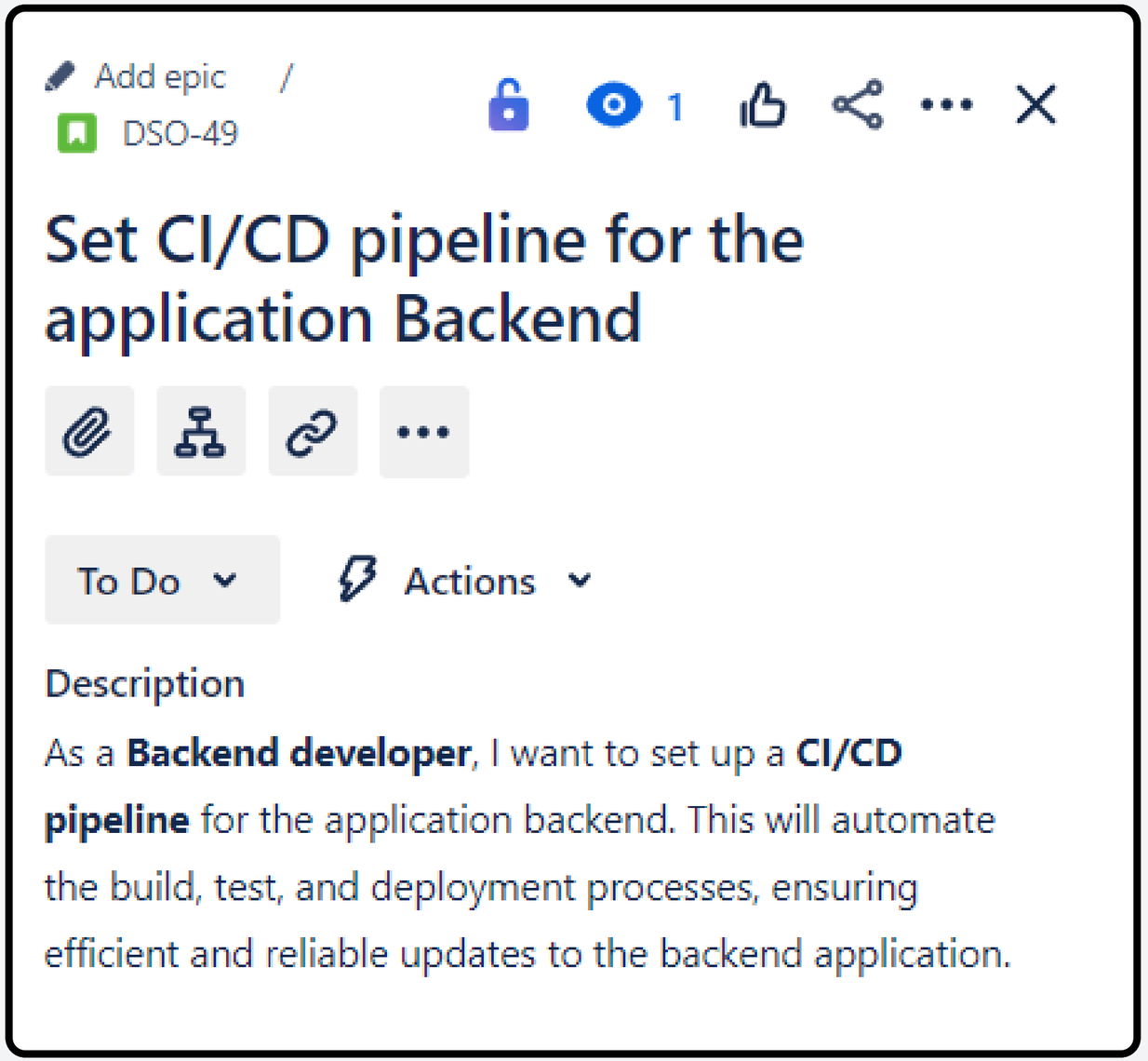
Sprint 1 : CI/CD pipeline

Sprint 1 includes 3 tickets :

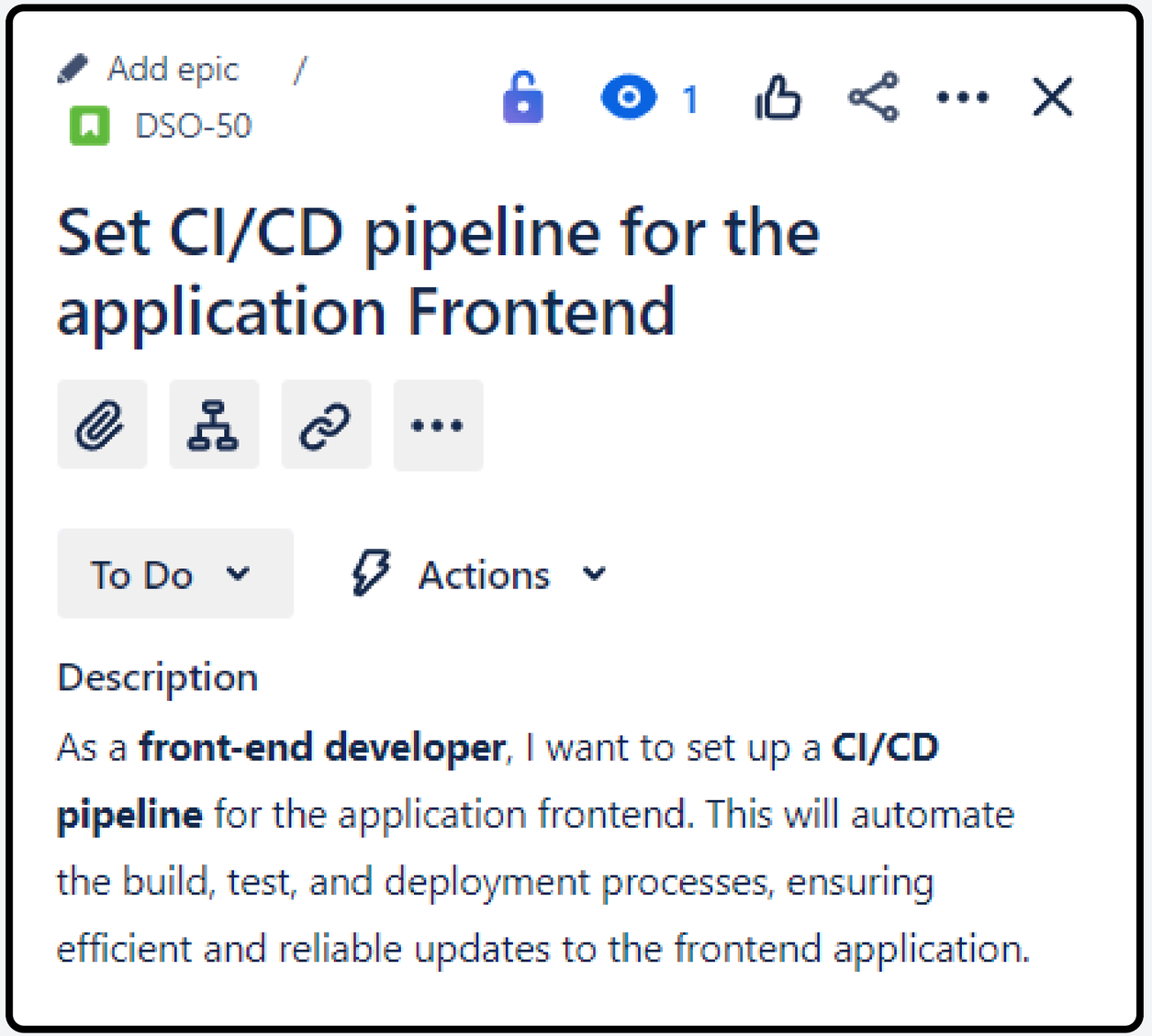
- Set CI/CD Pipeline for backend.
- Set CI/CD Pipeline for frontend.
- Pipelines optimizations.



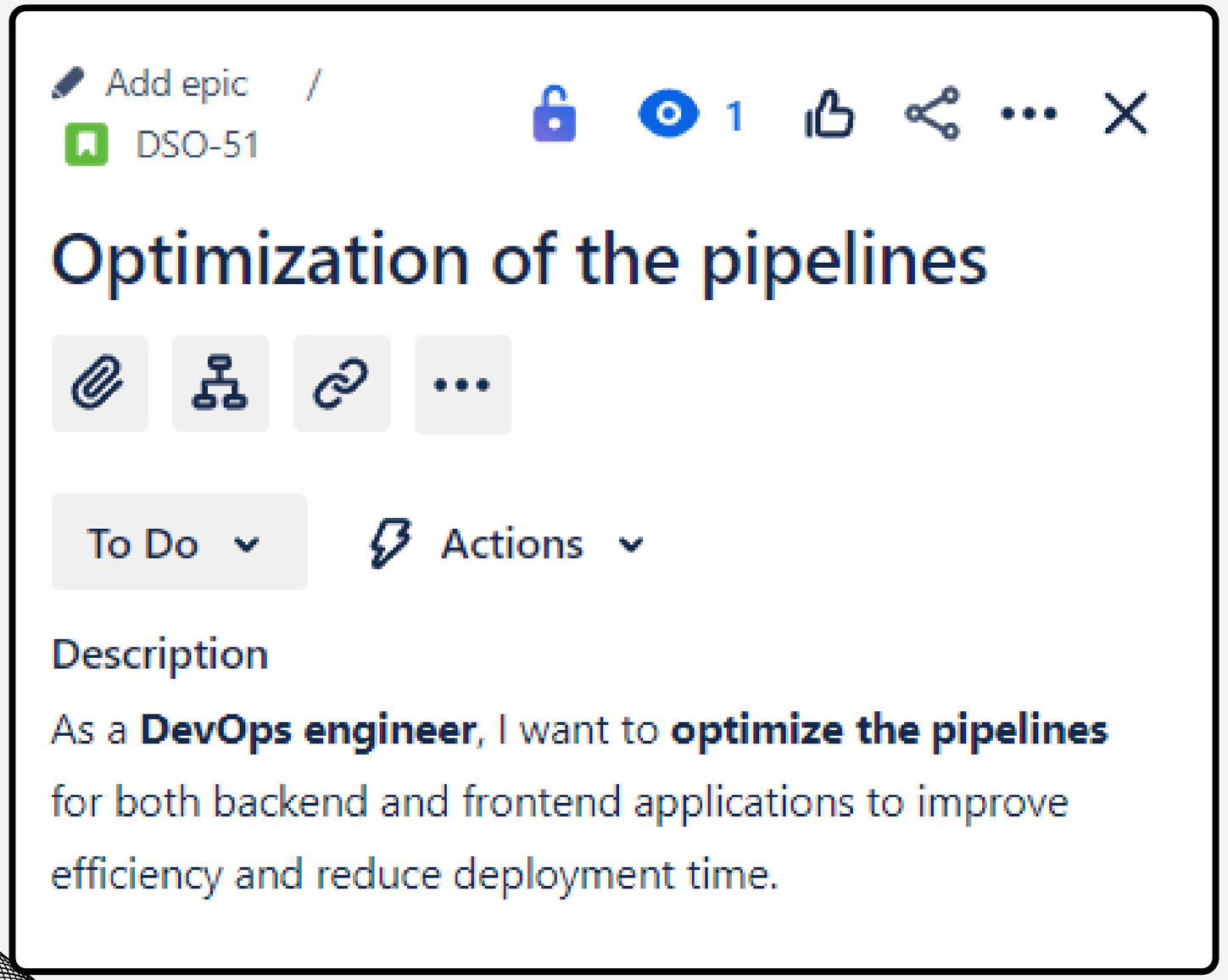
Ticket 1 : Set CI/CD Pipeline for backend.



Ticket 2 : Set CI/CD Pipeline for frontend.



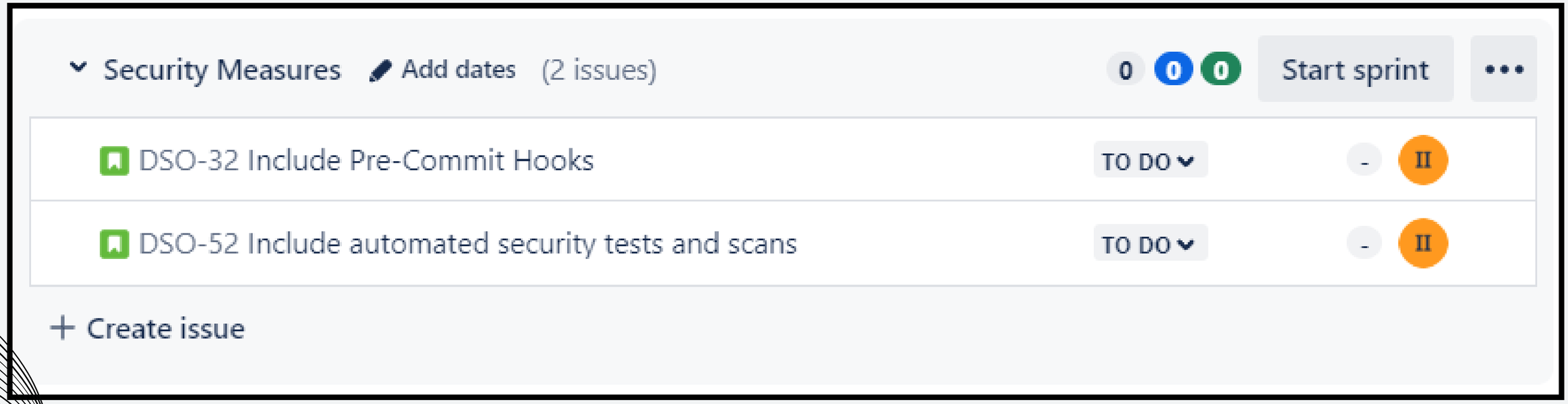
Ticket 3 : Pipelines optimization.



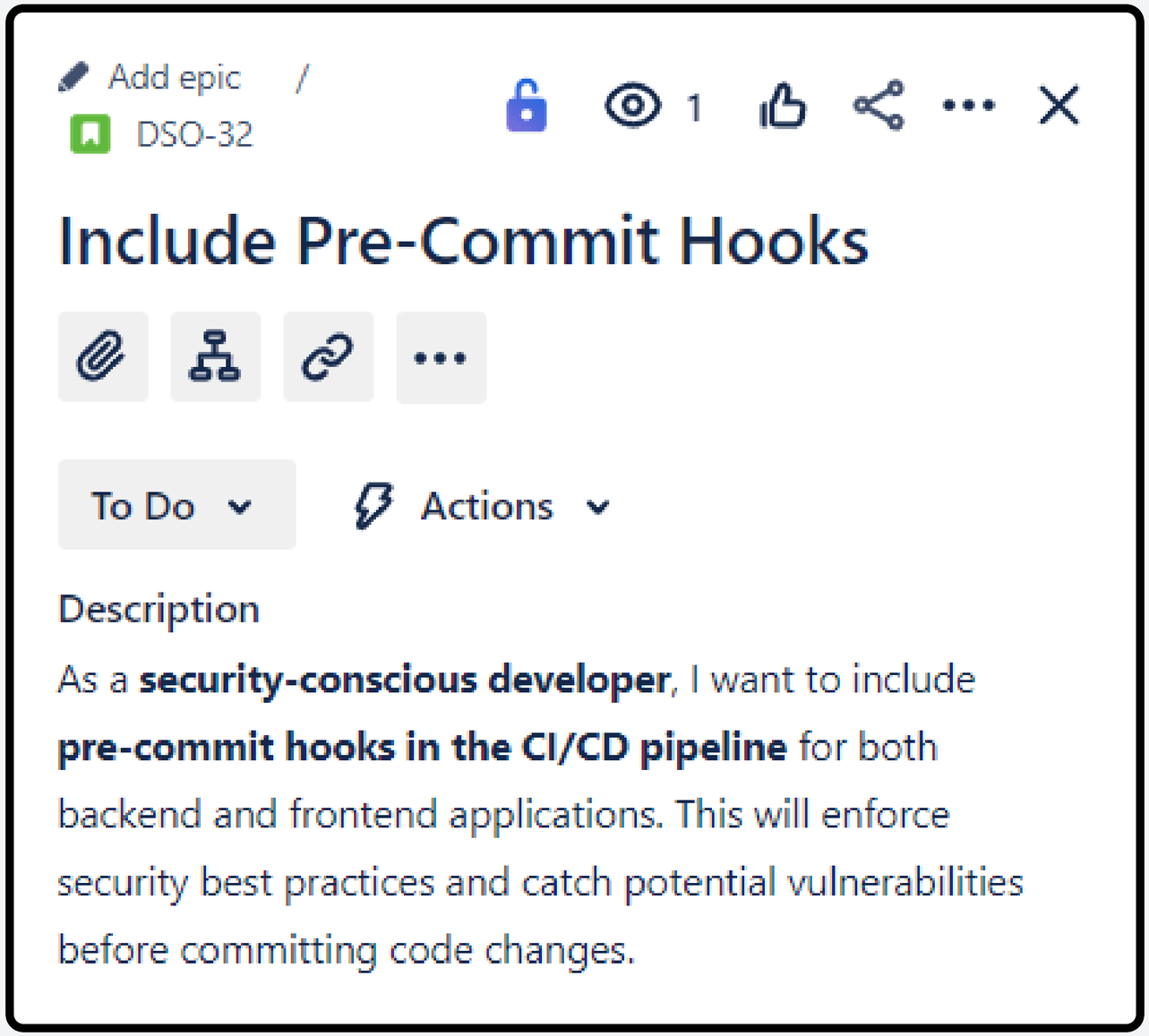
Sprint 2 : Security Measures

Sprint 2 includes 2 tickets :

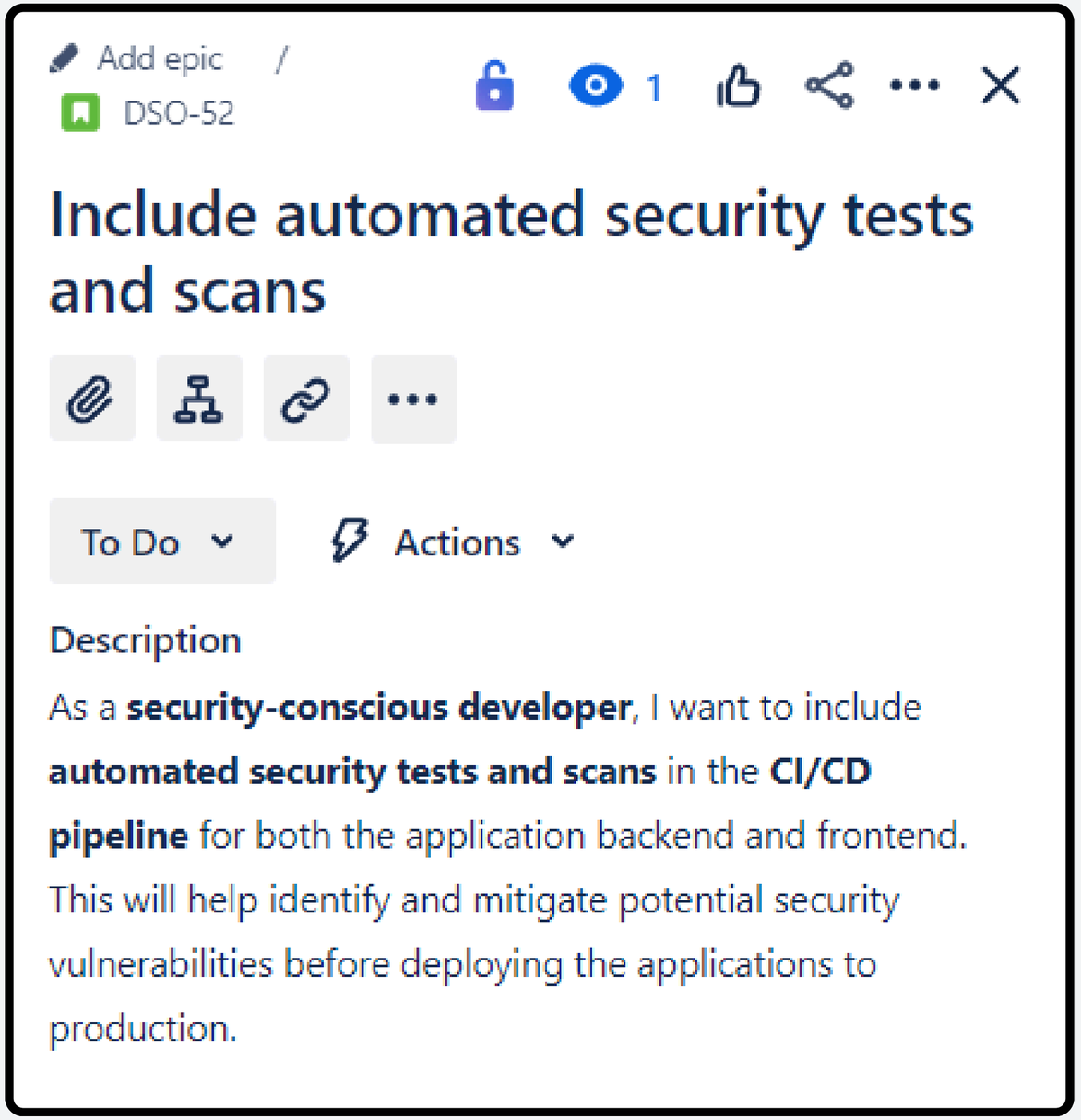
- Include Pre-commit Hooks.
- Include automated security tests and scans.



Ticket 1 : Include Pre-commit Hooks



Ticket 2 : Include automated security tests and scans



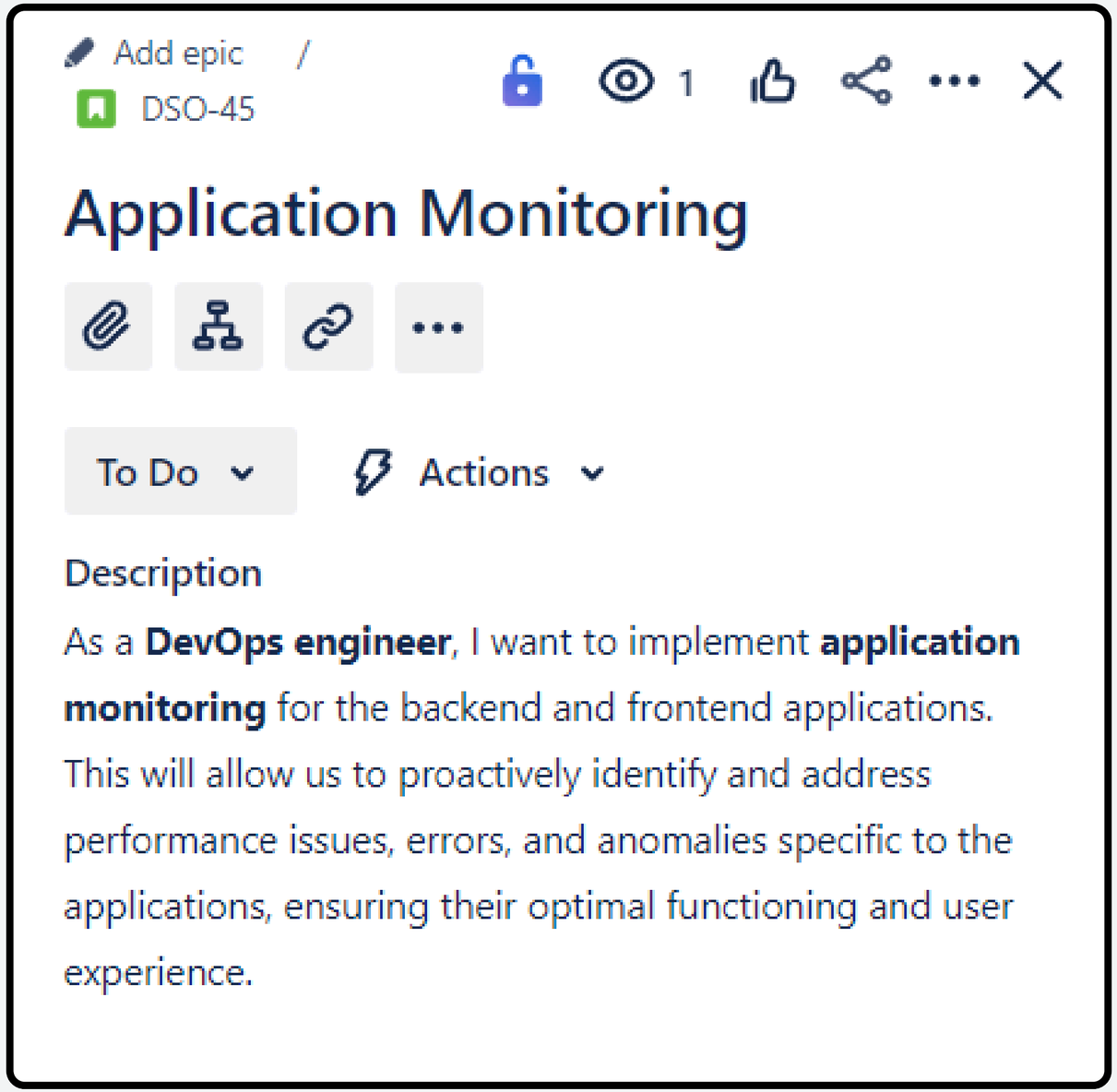
Sprint 3 : Monitoring

Sprint 3 includes 3 tickets :

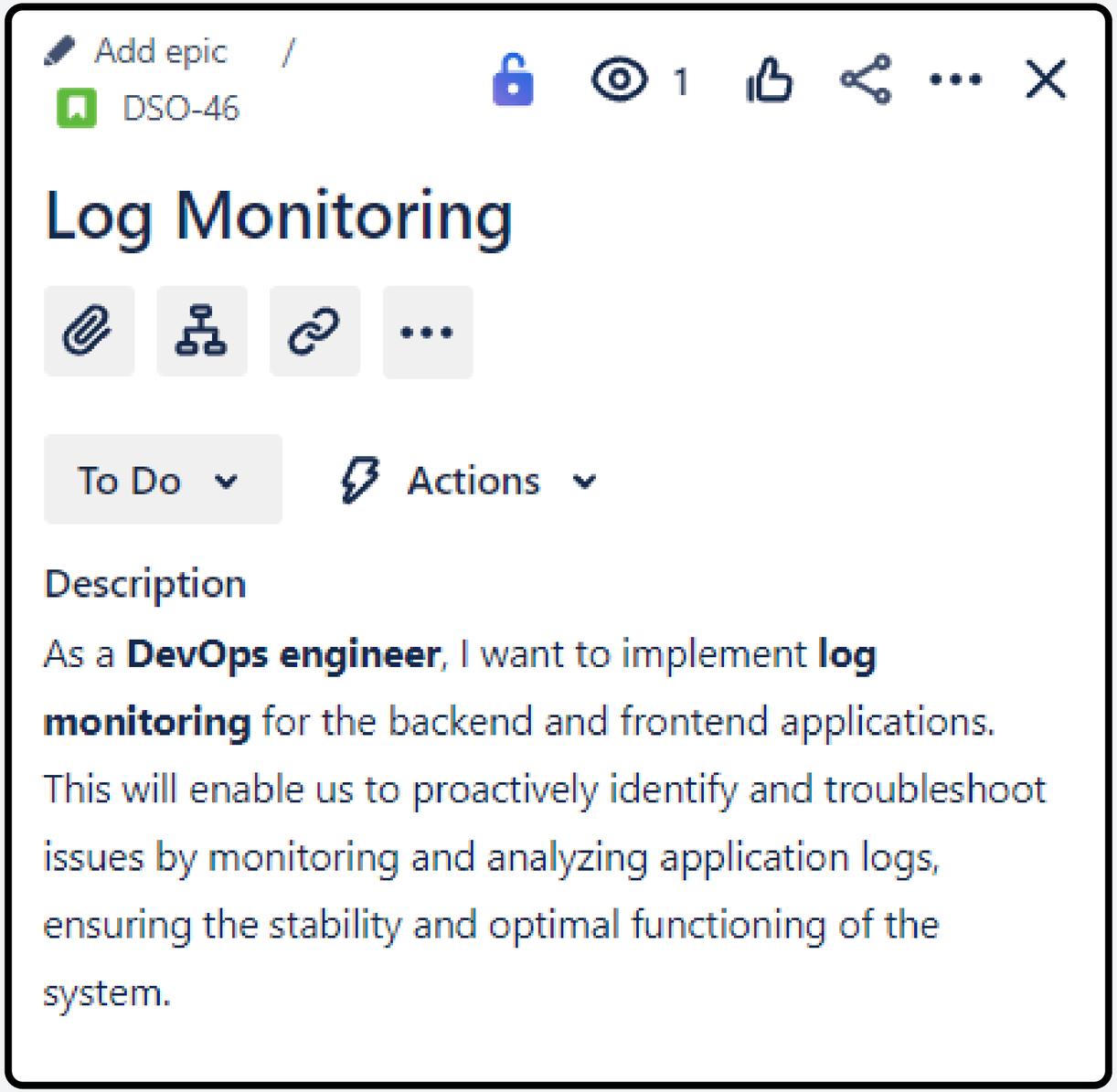
- System Monitoring.
- Application Monitoring.
- Log Monitoring.

Ticket 1 : System Monitoring

Ticket 2 : Application Monitoring



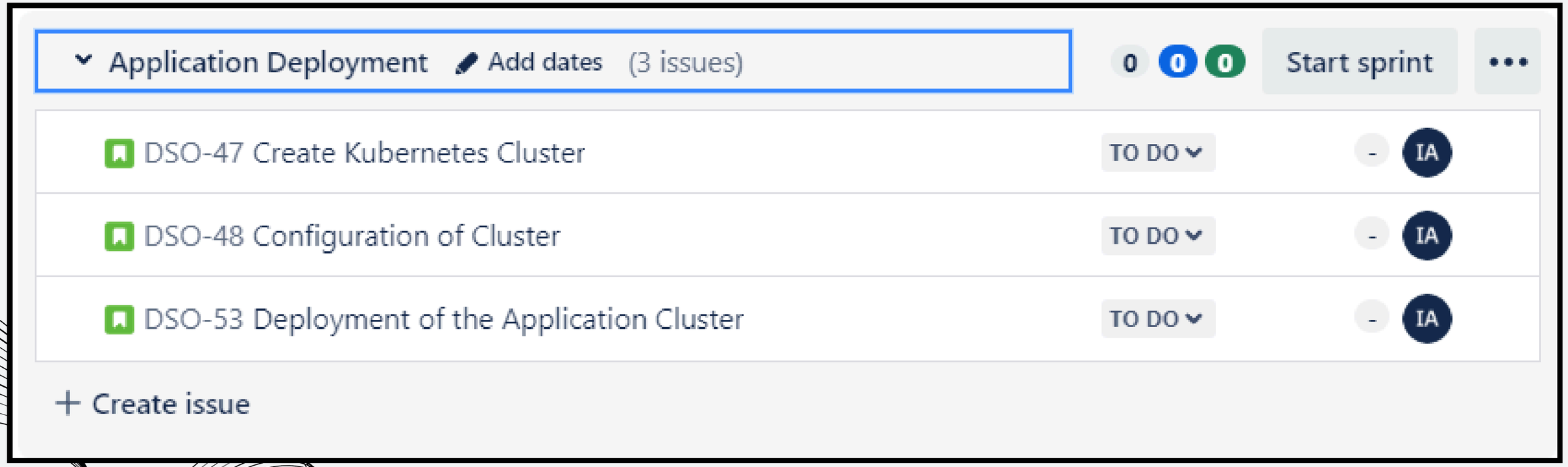
Ticket 3 : Log Monitoring



Sprint 4 : Security Measures

Sprint 4 includes 2 tickets :

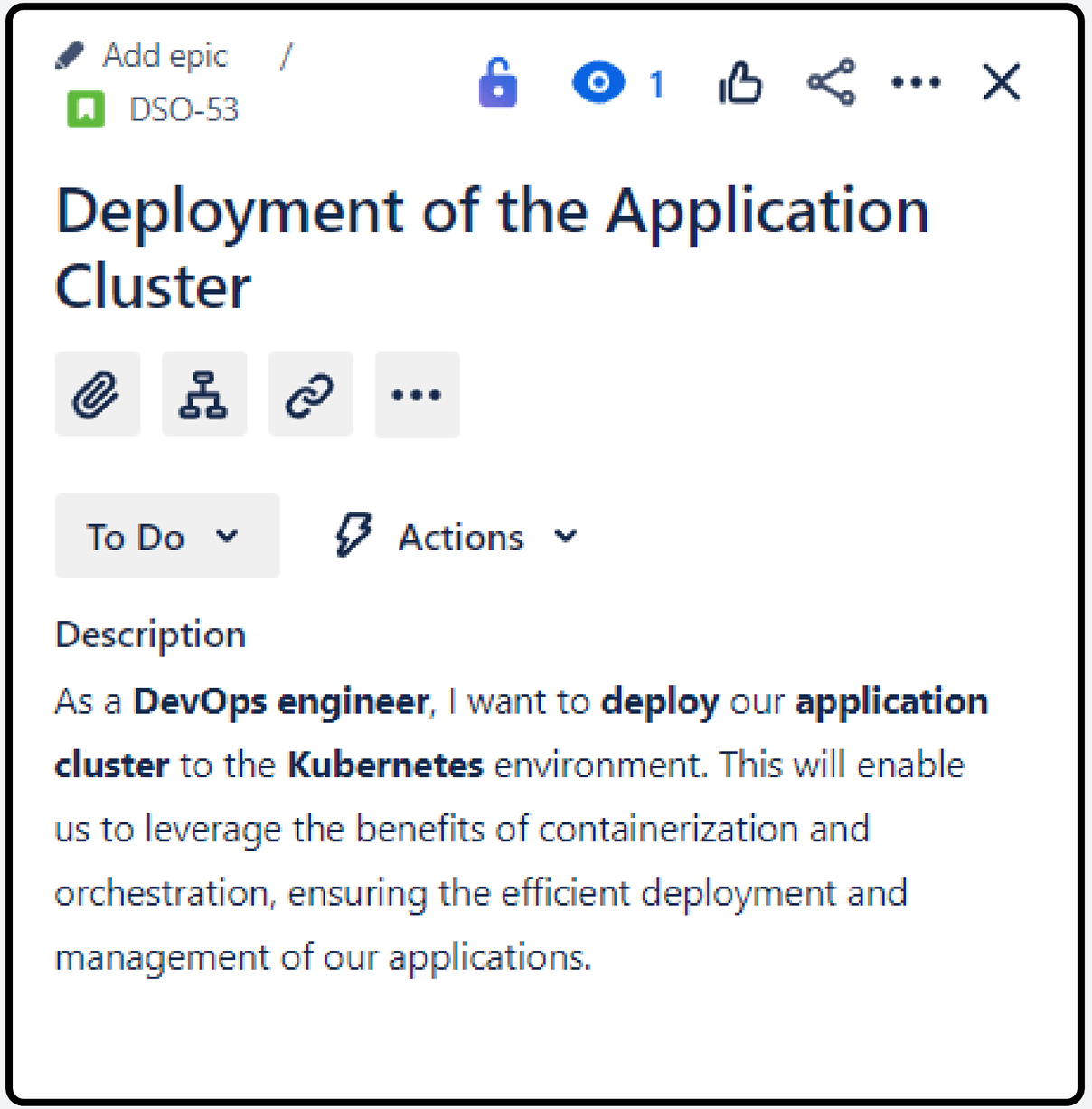
- Create Kubernetes cluster.
- Configuration of the cluster.
- Deployment of the Application’s cluster.



Ticket 1: Create Kubernetes Cluster

Ticket 2 : Configuration of the cluster

Ticket 3 : Deployment of the Application’s cluster



2-CI/CD PIPELINE

GitLab proved to be the ideal choice for our CI/CD needs due to its comprehensive set of features and seamless integration with source code management. By utilizing GitLab, we were able to streamline our software delivery process, improve collaboration among team members, and achieve faster and more reliable deployments.

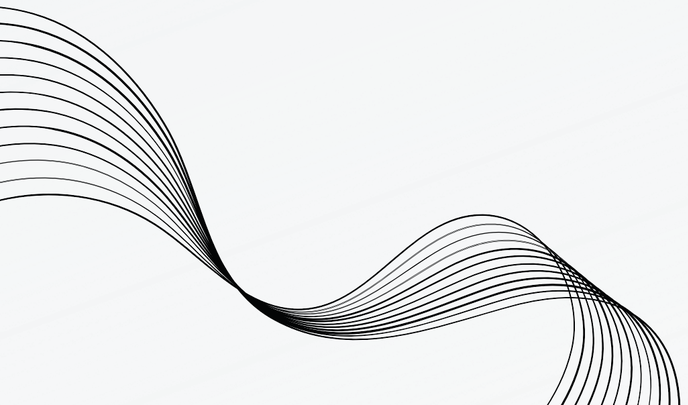
1) Backend CI/CD pipeline

First of all , we have to create a **Dockerfile** to define the instructions and configuration needed to build a **Docker image**.

Dockerfile 171 B

Blame | Edit | Replace | Delete

```
1  FROM openjdk:17-jdk-slim
2  WORKDIR /app
3  COPY target/spring-boot-data-jpa-0.0.1-SNAPSHOT.jar /app/spring-boot-app.jar
4  EXPOSE 8080
5  CMD ["java", "-jar", "spring-boot-app.jar"]
```



1) Backend CI/CD pipeline

Let’s explain the content of the ***Dockerfile*** :

- **Line 1** : specifies the base image for the Docker image. In this case, it uses the OpenJDK 17 image.
- **Line 2** : sets the working directory inside the Docker image to /app. This is the directory where subsequent commands will be executed.
- **Line 3** : copies the JAR file from the target directory of the host machine into the /app directory of the Docker image.
- **Line 4** : This line specifies that the Docker container will listen on port 8080.
- **Line 5** : defines the command that will be executed when the Docker container starts.

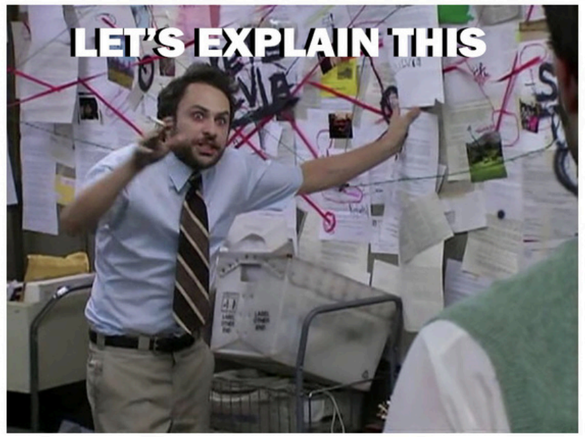
1) Backend CI/CD pipeline

alse , we have to configure our pipeline using the YAML file :

.gitlab-ci.yml 882 B

Blame | Edit | Replace | Delete

```yaml
image: docker:latest
services:
  - docker:dind

stages:
  - build
  - package
  - deploy

maven-build:
  image: maven:latest
  stage: build
  script: "mvn clean package -B"
  artifacts:
    paths:
      - target/*.jar

docker-build:

  stage: package
  rules:
    - exists:
        - Dockerfile
  before_script:
    - docker login -u 4bd3lgh4f0r -p $DOCKER_HUB_PASSWORD
  script:
    - docker build -t 4bd3lgh4f0r/devsecops_backend  .
    - docker push 4bd3lgh4f0r/devsecops_backend

docker-deploy:
  stage: deploy
  before_script:
    - apk add --update curl && rm -rf /var/cach/apk/*
  script:
    - docker network create docker_net

    - docker run -d --name mysql_name --network docker_net -e MYSQL_ROOT_PASSWORD=$MYSQL_ROOT_PASSWORD -e MYSQL_DATABASE=testdb  mysql

    - docker run -d --name backend_sb --network docker_net -p 8080:8080 4bd3lgh4f0r/devsecops_backend:1.0
```



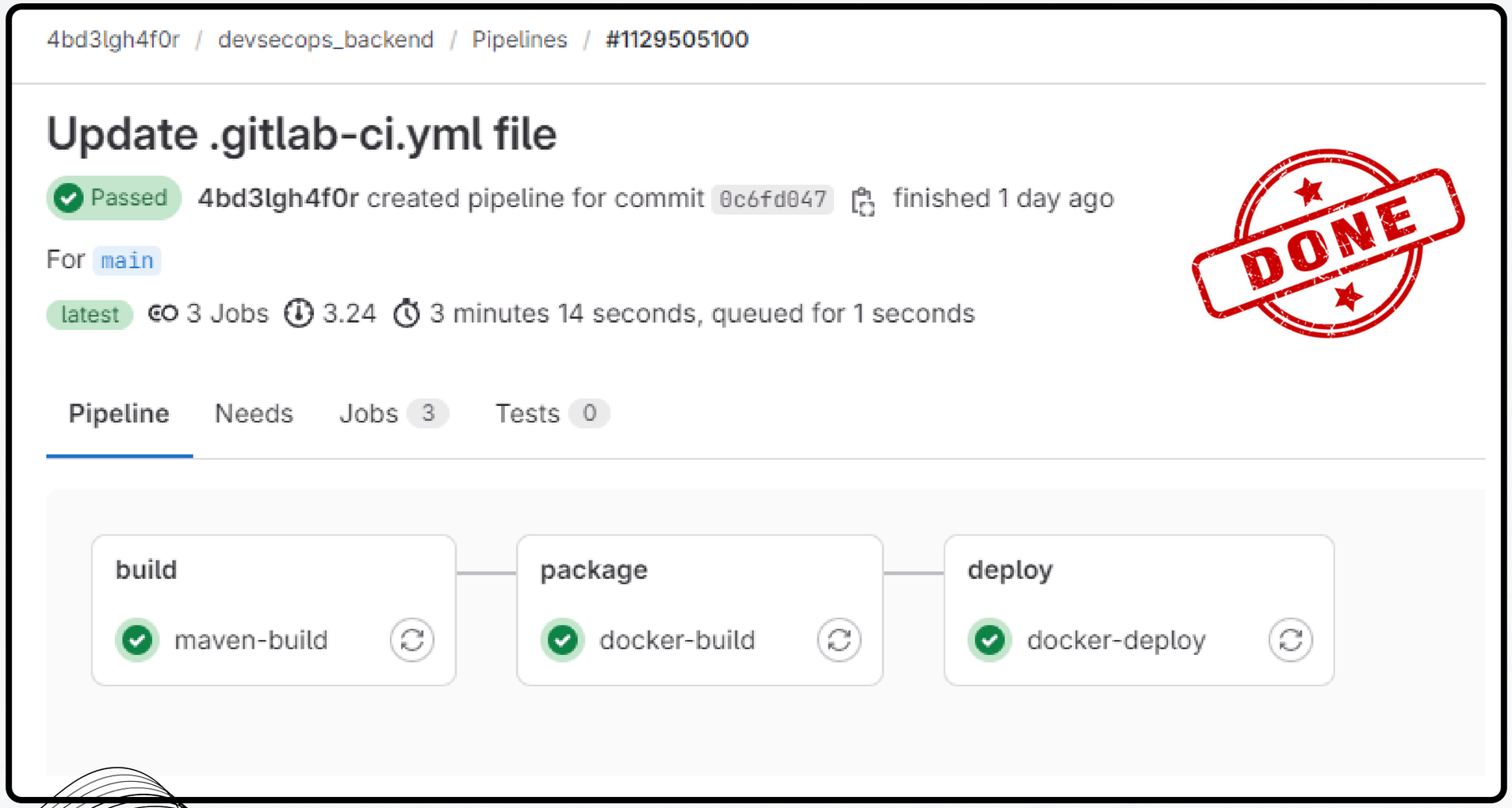
1) Backend CI/CD pipeline

Here are some key values used on the YAML file:

- The image directive specifies the Docker image that will be used as the base environment for the pipeline jobs. In this case, it is set to **docker:latest**.
- The services section defines additional services that will be available during the pipeline execution. Here, **docker:dind** service is specified, which provides a Docker-in-Docker environment.
- The stages section lists the different stages of the pipeline, which are :
    - ***Build*** : responsible for compiling and packaging the project.
    - ***Package*** : responsible for the creation and preparation of the Docker image.
    - ***Deploy*** : focuses on the deployment of the application.
- The YAML file defines 3 jobs :
    - ***maven-build***
    - ***docker-build***
    - ***docker-deploy***

1) Backend CI/CD pipeline

Now let’s run the backend pipeline



The pipeline ran successfully

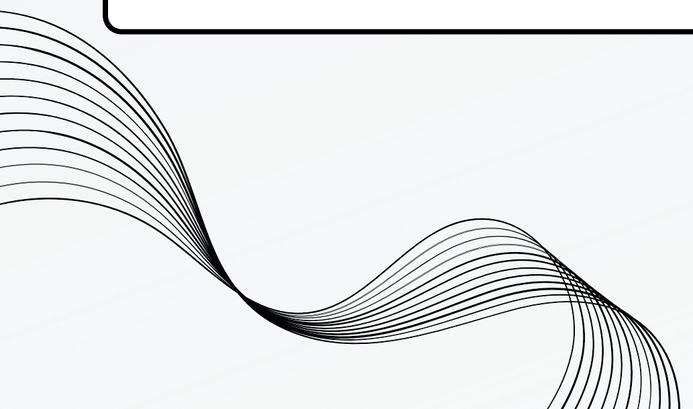
2) Frontend CI/CD pipeline

let’s create a **Dockerfile** to define the instructions and configuration needed to build a **Docker image**.

Dockerfile 215 B

Blame | Edit | Replace | Delete

```
1  FROM node:latest
2
3  WORKDIR /app
4  ENV PATH /app/node_modules/.bin:$PATH
5
6  COPY package.json  /app/package.json
7  RUN npm install
8  RUN npm install -g @angular/cli@7.3.9
9
10 COPY . /app
11
12 CMD ng serve --host 0.0.0.0 --port 4200
```



2) Frontend CI/CD pipeline

Let’s explain the content of the ***Dockerfile*** :

- **Line 1** : specifies the base image for the Docker image. In this case, it uses the latest Node.js image.
- **Line 3** : sets the working directory inside the Docker image to /app. This is the directory where subsequent commands will be executed.
- **Line 4** : sets an environment variable PATH to allow access to locally installed Node.js binaries within the project.
- **Line 6** : copies the **package.json** file from the host machine to the /app directory in the Docker image.
- **Line 7** : runs the ***npm install*** command inside the Docker image installs all the required packages.
- **Line 8** : installs the Angular CLI globally in the Docker image.
- **Line 10** : copies the remaining project files and directories from the host machine to the /app directory in the Docker image.
- **Line 11** : uses the ***ng serve*** command to start the Angular development server.

2) Frontend CI/CD pipeline

Now we configure the pipeline file :

.gitlab-ci.yml 723 B

Blame Edit

```yaml
image : docker:latest
include:
  - template: Security/Dependency-Scanning.gitlab-ci.yml
  - template: Security/SAST.gitlab-ci.yml
services:
  - docker:dind
stages:
  - test
  - package
  - deploy
docker-build:
  stage: package
  before_script:
    - docker login -u 4bd3lgh4f0r -p $DOCKER_USER_PASSWORD
  script:
    - docker build -t  4bd3lgh4f0r/devsecops_frontend  .
    - docker push 4bd3lgh4f0r/devsecops_frontend

docker-deploy:
  stage: deploy
  before_script :
    - apk add --update curl && rm -rf /var/cache/apk/*|
  script :
    - docker run -d -p 9001:4200 --name frontend 4bd3lgh4f0r/devsecops_frontend
    - sleep 30
    - docker logs frontend
    - curl -v "http://docker:9001"
    - docker logs frontend
```

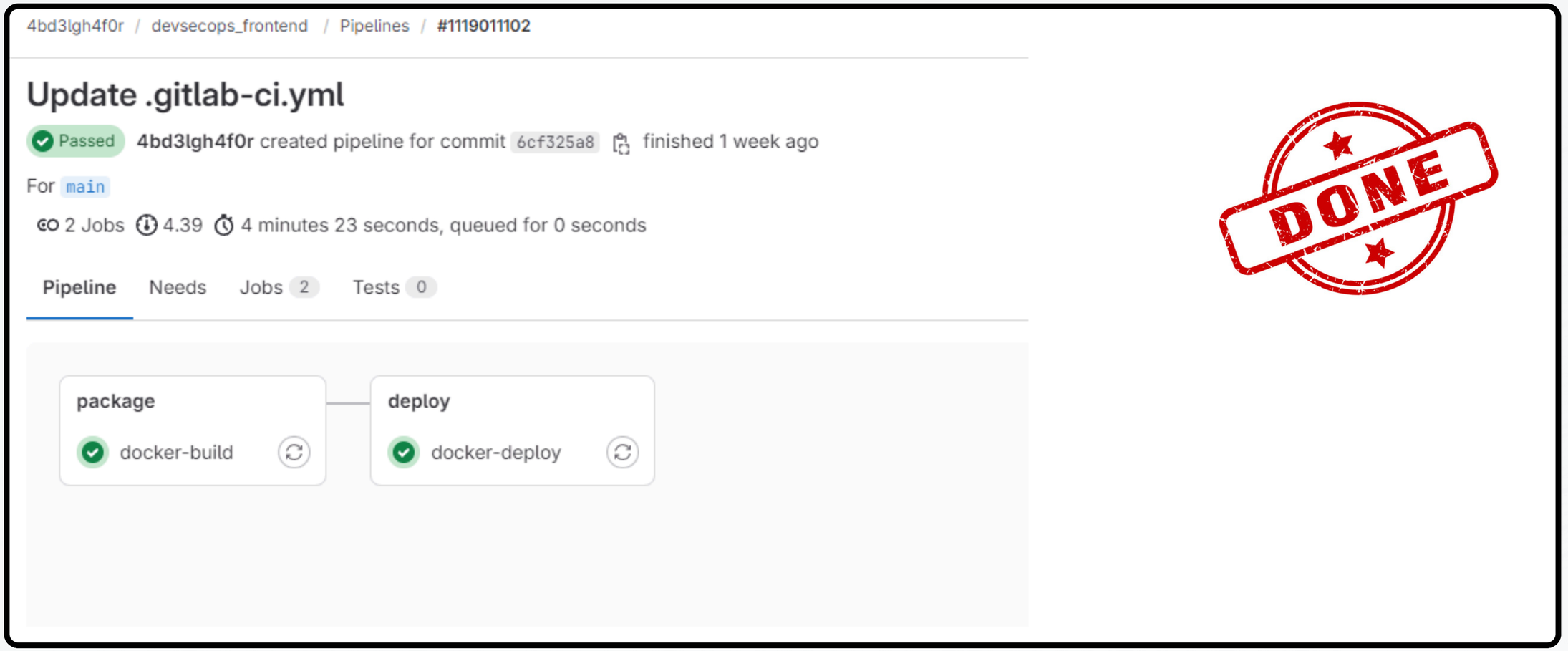
2) Frontend CI/CD pipeline

Here are some key values used on the YAML file:

- The image directive specifies the Docker image that will be used as the base environment for the pipeline jobs. In this case, it is set to **docker:latest**.
- The services section defines the service **docker:dind**, which provides a Docker-in-Docker environment to be used during the pipeline execution.
- The stages section lists the different stages of the pipeline, which are :
    - ***Test***
    - ***Package***
    - ***Deploy***
- The YAML file defines 3 jobs :
    - ***docker-build :*** responsible for building and pushing a Docker image to Dockerhub.
    - ***docker-deploy :*** responsible for deploying the Docker image to the target environment.

2) Frontend CI/CD pipeline

Now let’s run the frontend pipeline



The pipeline ran successfully

3) Deployment of the images:

As we can see , the 2 images were successfully pushed to ***DockerHub***

4bd3lgh4f0r / **devsecops_backend**

Description

This repository does not have a description

Last pushed: a few seconds ago

Tags

This repository contains 2 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	---	a few seconds ago

4bd3lgh4f0r / **devsecops_frontend**

Description

This repository does not have a description

Last pushed: 12 hours ago

Tags

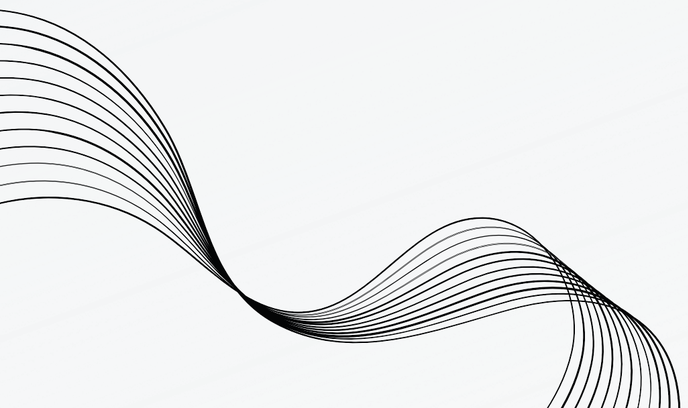
This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	3 hours ago	12 hours ago

PIPELINES OPTIMIZATION

After establishing the two pipelines, we will endeavor to optimize them.

But first we will add some rules to the pipelines



RULES

Refers to a section in the CI/CD configuration file where you define conditions that determine when a particular job should be executed

These are the rules that we have used in the 2 pipelines

the building stage

```
docker-build:
  stage: build
  rules:
    - exists:
      - Dockerfile
    - exists:
      - $CI_COMMIT_BRANCH == "main"
```

Concerning the docker-build job we have created 2 rules :

the first : this rule ensures that the associated job is executed only when the "Dockerfile" is present in the project

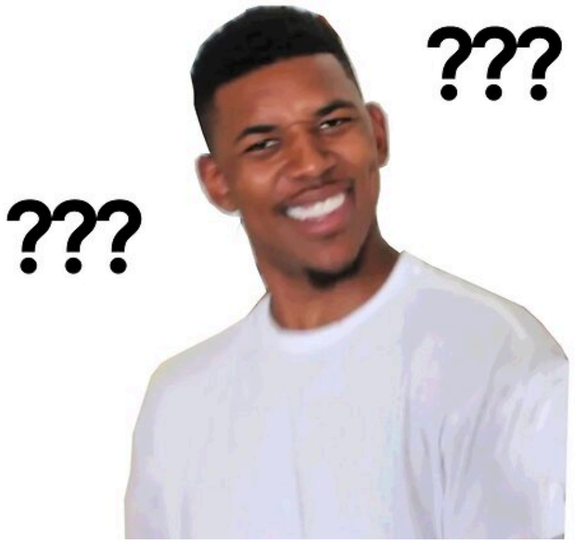
the second : the job will be executed only on main branch

```yaml
docker-deploy:
  stage: deploy
  rules:
    - needs:
        - job: "docker-build"
    - exists:
        - $CI_COMMIT_BRANCH == "main"
```

Concerning the docker-deploy job we have created 2 rules :
the first : ensures that the deployment occurs only after the successful completion of the Docker image build
the second : the job will be executed only on main branch

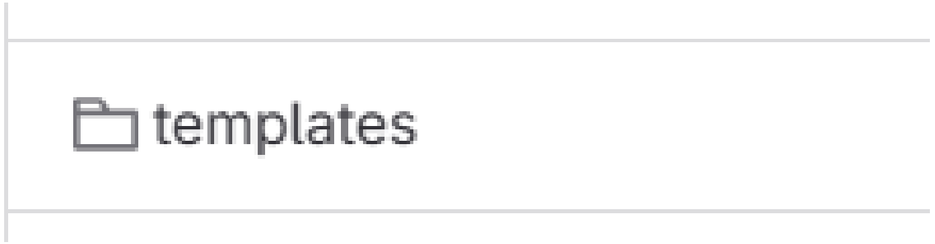
After adding some rules, we will Enhance GitLab CI/CD by adjusting **generic Jobs**
Store job configurations in an external YAML file for easy management and reuse in shared job definitions.

what are generic jobs



generic jobs is about storing job configurations in an external YAML file for easy management and reuse in shared job definitions.

First we create a directory named “templates” where we will store YAML files



then we create the YAML files for each job :

for the frontend:

docker-build.yml 430 B Blame Edit

```yaml
docker-build:
  stage: build
  rules:
    - exists:
        - Dockerfile
    - exists:
        - $CI_COMMIT_BRANCH == "main"
  before_script:
    - docker login -u 4bd3lgh4f0r -p $DOCKER_USER_PASSWORD
  script:
    - docker pull 4bd3lgh4f0r/devsecops_frontend:latest || true
    - docker build --cache-from 4bd3lgh4f0r/devsecops_frontend:latest -t 4bd3lgh4f0r/devsecops_frontend .
    - docker push 4bd3lgh4f0r/devsecops_frontend

```

docker-deploy.yml 728 B

```yaml
docker-deploy:
  stage: deploy
  rules:
    - needs:
        - job: "docker-build"
    - exists:
        - $CI_COMMIT_BRANCH == "main"
  before_script:
    - apk add --update curl && rm -rf /var/cache/apk/*
  script:
    - docker run -d -p 9001:4200 --name frontend 4bd3lgh4f0r/devsecops_frontend
    - sleep 30
    - docker logs frontend
    - curl -v "http://docker:9001"
    - docker logs frontend
```

then we Specify the paths for each job in the YAML pipeline file:

- templates/docker-build.yml
- templates/docker-deploy.yml

for the backend:

docker-package.yml 130 B

```yaml
maven-build:
  image: maven:latest
  stage: package
  script: "mvn clean package -B"
  artifacts:
    paths:
      - target/*.jar
```

docker-deploy.yml 493 B Blame Edit

```yaml
docker-deploy:
  stage: deploy
  rules:
  - needs:
        - job: "docker-build"
  - exists:
        - $CI_COMMIT_BRANCH == "main"
  before_script:
    - apk add --update curl && rm -rf /var/cach/apk/*
  script:
    - docker network create docker_net

    - docker run -d --name mysql_name --network docker_net -e MYSQL_ROOT_PASSWORD=$MYSQL_ROOT_PASSWORD -

    - docker run -d --name backend_sb --network docker_net -p 8080:8080 4bd3lgh4f0r/devsecops_backend

```

docker-build.yml 313 B

```yaml
docker-build:

  stage: build
  rules:
    - exists:
        - Dockerfile
    - exists:
        - $CI_COMMIT_BRANCH == "main"
  before_script:
    - docker login -u 4bd3lgh4f0r -p $DOCKER_HUB_PASSWORD
  script:
    - docker build -t 4bd3lgh4f0r/devsecops_backend .
    - docker push 4bd3lgh4f0r/devsecops_backend

```

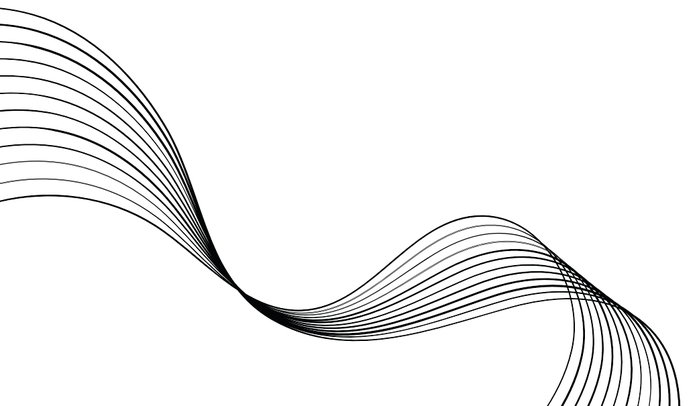
the paths :

- templates/docker-package.yml
- templates/docker-build.yml
- templates/docker-deploy.yml

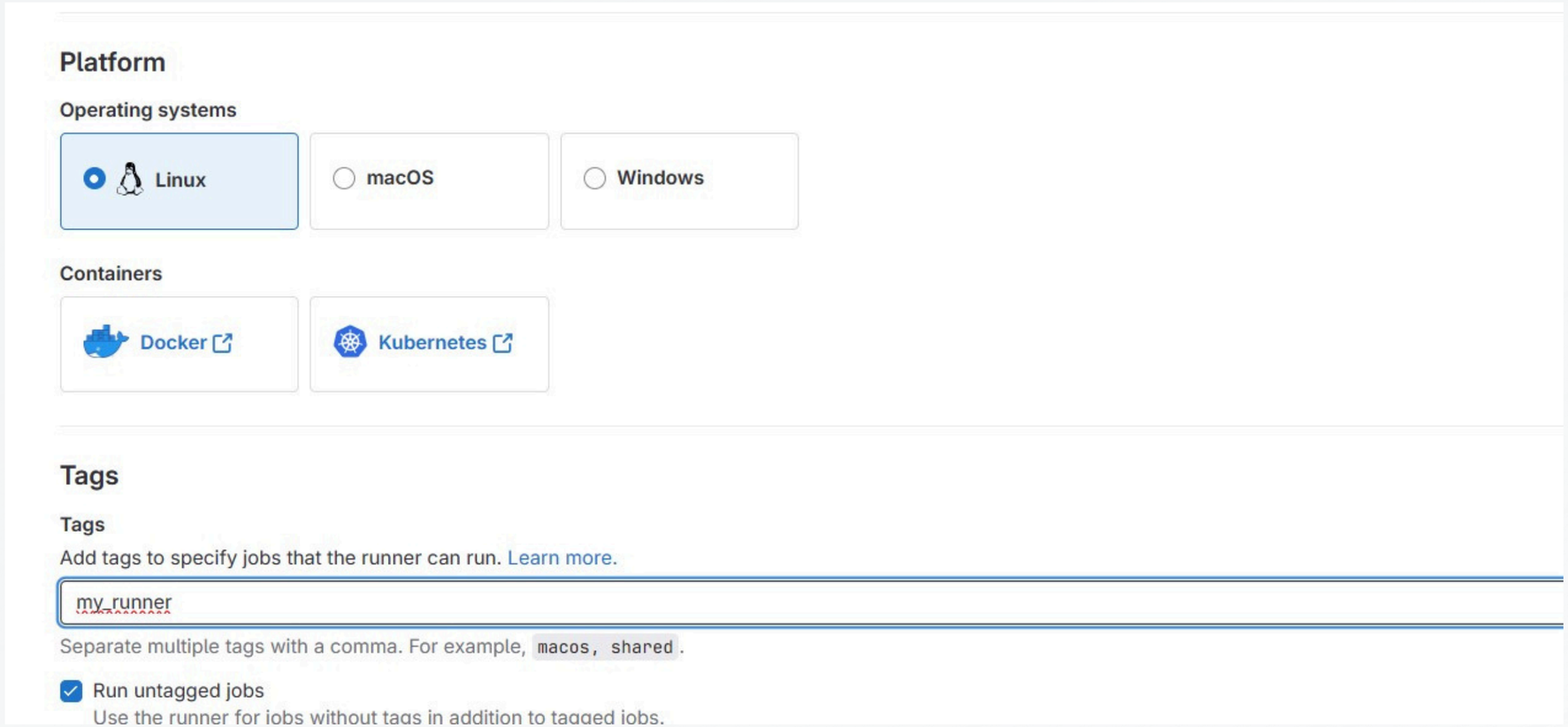
GITLAB-RUNNER

GitLab Runner is the component responsible for running the jobs defined in a GitLab CI/CD pipeline. It can be installed on various platforms, such as Linux, macOS, and Windows. Runners can be shared among projects or dedicated to a specific project

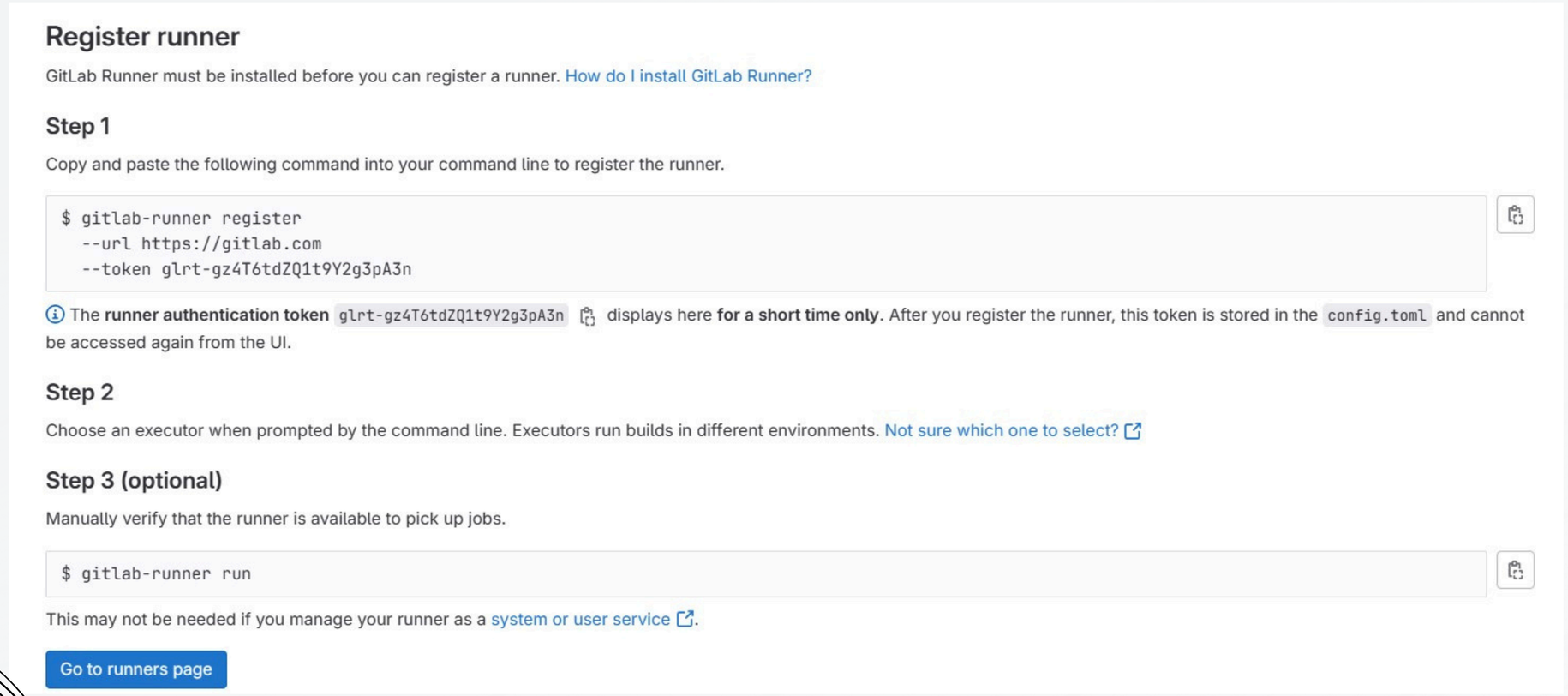
in this task our goal is to configure our GitLab Runner to enhance the efficiency of job execution.



Initially, we select the operating system, and in our case , we are opting for Linux



Here is the relevant information pertaining to our GitLab Runner configuration :



Register runner

GitLab Runner must be installed before you can register a runner. How do I install GitLab Runner?

Step 1

Copy and paste the following command into your command line to register the runner.

```
$ gitlab-runner register
  --url https://gitlab.com
  --token glrt-gz4T6tdZQ1t9Y2g3pA3n
```

The **runner authentication token** `glrt-gz4T6tdZQ1t9Y2g3pA3n` displays here **for a short time only**. After you register the runner, this token is stored in the `config.toml` and cannot be accessed again from the UI.

Step 2

Choose an executor when prompted by the command line. Executors run builds in different environments. Not sure which one to select?

Step 3 (optional)

Manually verify that the runner is available to pick up jobs.

```
$ gitlab-runner run
```

This may not be needed if you manage your runner as a system or user service.

Go to runners page

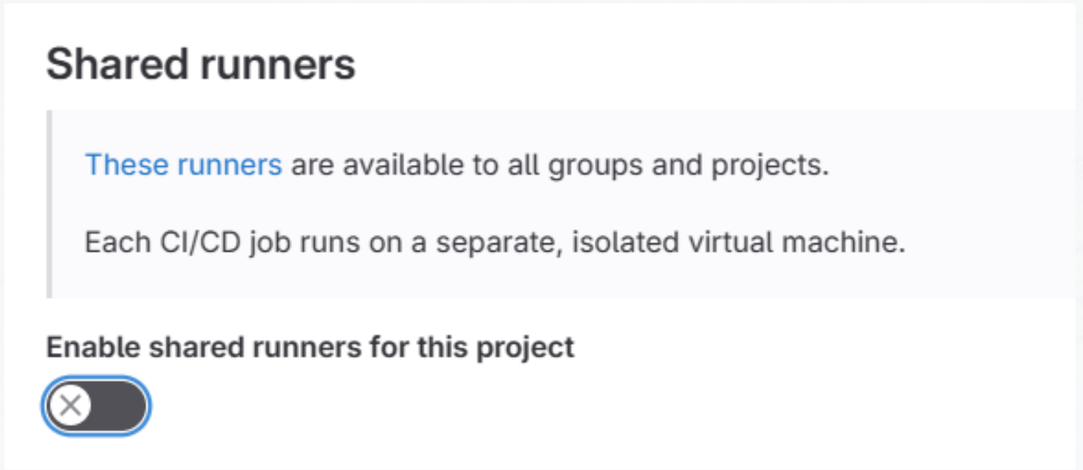
now in the terminal of the virtual machine (we have used an ubuntu VM)
we register our runner :

```
abdelghafour@abdelghafour:~$ gitlab-runner register
Runtime platform                                    arch=amd64 os=linux pid=4552 revision=102c81ba version=16.7.0
WARNING: Running in user-mode.
WARNING: The user-mode requires you to manually start builds processing:
WARNING: $ gitlab-runner run
WARNING: Use sudo for system-mode:
WARNING: $ sudo gitlab-runner...

Enter the GitLab instance URL (for example, https://gitlab.com/):
 https://gitlab.com
Enter the registration token:
glrt-gz4T6tdZQ1t9Y2g3pA3n
Verifying runner... is valid                        runner=gz4T6tdZQ
Enter a name for the runner. This is stored only in the local config.toml file:
[abdelghafour]: runner
Enter an executor: docker-windows, ssh, kubernetes, docker-autoscaler, docker+machine, instance, custom, docker, parallels, shell
virtualbox:
docker
Enter the default Docker image (for example, ruby:2.7):
docker:latest
Runner registered successfully. Feel free to start it, but if it's running already the config should be automatically reloaded!

Configuration (with the authentication token) was saved in "/home/abdelghafour/.gitlab-runner/config.toml"
abdelghafour@abdelghafour:~$
```

after registring the runner we add the following lines into the YAML file :

```
variables:
  DOCKER_HOST: tcp://172.17.0.1:2375
  DOCKER_DRIVER: overlay2
```

then we disable the shared runner



and finally we start the runner :

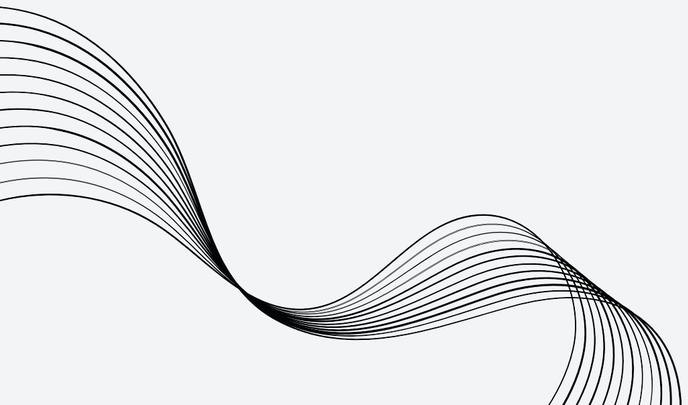
```
abdelghafour@abdelghafour:~$ gitlab-runner run
Runtime platform                                    arch=amd64 os=linux pid=4562 revision=102c81ba version=16.7.0
Starting multi-runner from /home/abdelghafour/.gitlab-runner/config.toml...  builds=0 max_builds=0
WARNING: Running in user-mode.
WARNING: Use sudo for system-mode:
WARNING: $ sudo gitlab-runner...

Configuration loaded                                builds=0 max_builds=1
listen_address not defined, metrics & debug endpoints disabled  builds=0 max_builds=1
[session_server].listen_address not defined, session endpoints disabled  builds=0 max_builds=1
Initializing executor providers                     builds=0 max_builds=1
```

3-Security Tests

Pre-commit Hooks

Pre-commit hooks are scripts or actions that are executed automatically before a developer's changes are committed to the version control system (e.g., Git). The purpose of pre-commit hooks is to catch issues early in the development process and prevent code that doesn't meet certain criteria from being committed.

In our project, we will use pre-commit hooks to detect if hard coded secrets (passwords, credentials, api keys...) exist in our code, so we remove them before they are commited publicly.



Tool used:

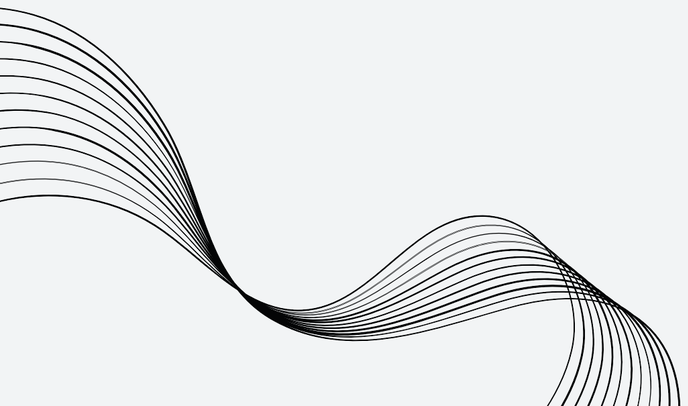
Link: https://github.com/gitleaks/gitleaks

Configuration file:

.pre-commit-config.yaml 103 B

Blame | Edit | Lock | Replace | Delete

```yaml
repos:
  - repo: https://github.com/gitleaks/gitleaks
    rev: v8.18.0
    hooks:
      - id: gitleaks

```



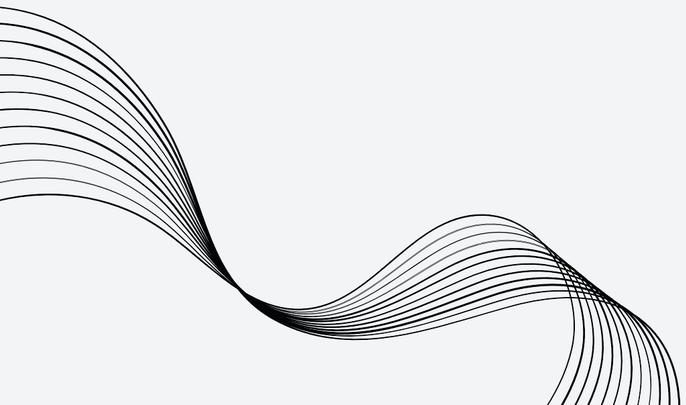
Result:

```
┌──(ismail㉿kali)-[~/devsecops/devsecops_backend]
└─$ echo "password: dokcer1234uuuu" >> password

┌──(ismail㉿kali)-[~/devsecops/devsecops_backend]
└─$ git add password

┌──(ismail㉿kali)-[~/devsecops/devsecops_backend]
└─$ git commit -m "testing pre-commit hooks"
[WARNING] Unstaged files detected.
[INFO] Stashing unstaged files to /home/ismail/.cache/pre-commit/patch1703809669-19071.
Detect hardcoded secrets.....(no files to check)Skipped
[INFO] Restored changes from /home/ismail/.cache/pre-commit/patch1703809669-19071.
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    .pre-commit-config.yml
        deleted:    docker_credentials

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .pre-commit-config.yaml

no changes added to commit (use "git add" and/or "git commit -a")
```

SAST

Static Application Security Testing (SAST) is a critical component of a comprehensive security strategy in software development. SAST involves analyzing the source code or compiled binaries of an application to identify and mitigate security vulnerabilities and coding flaws early in the development process. It helps developers catch issues such as SQL injection, cross-site scripting (XSS), and other vulnerabilities before the code is deployed.

We will use Gitlab for all security scans.

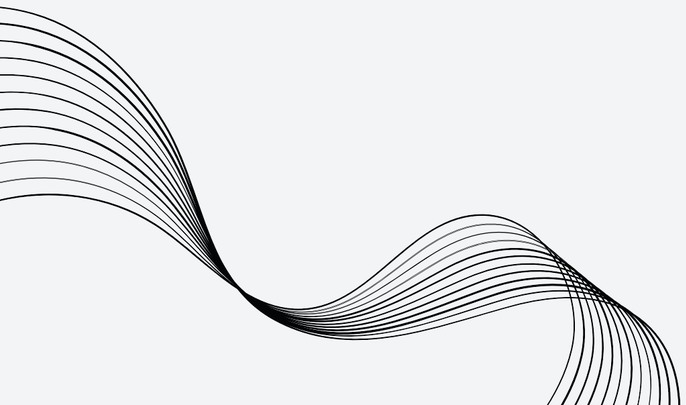


To add SAST scans to our pipeline, we should simply include Gitlab SAST file to our YAML file:

```yaml
include:
  - template: Security/SAST.gitlab-ci.yml
```

More info: https://docs.gitlab.com/ee/user/application_security/sast/

After running our pipeline, we can now look at the scan results by downloading artifact, or using Gitlab vulnerability management tool.



Results

Vulnerability report

+ Submit vulnerability | Export

The Vulnerability Report shows results of successful scans on your project's default branch, manually added vulnerability records, and vulnerabilities found from scanning operational environments. Learn more.

Development vulnerabilities 0 | Operational vulnerabilities 0

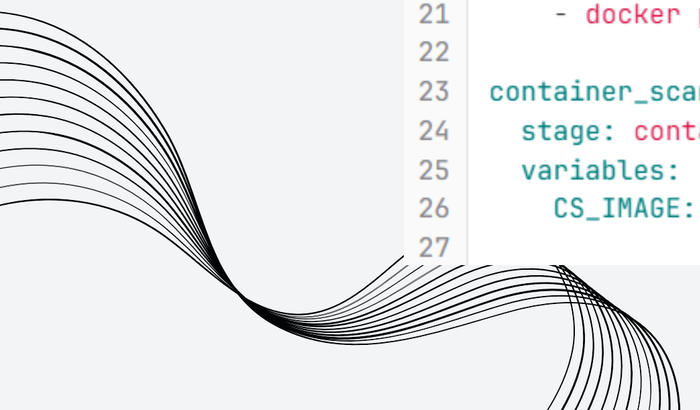
Security reports last updated 1 week ago #1121952268 • **SBOMs last updated** 1 week ago #1121952268

Critical	High	Medium	Low	Info	Unknown
0	0	0	0	0	0

Status: Needs triage +1 more | **Severity**: All severities | **Tool**: SAST | **Activity**: Still detected

The same way, we add scans for dependency scanning and container security, here is the full configuration file:

```
stages:
  - test
  - build
  - container_scanning
  - deploy

include:
  - template: Security/Dependency-Scanning.gitlab-ci.yml
  - template: Security/SAST.gitlab-ci.yml
  - template: Security/Container-Scanning.gitlab-ci.yml

docker-build:
  stage: build
  before_script:
    - docker login -u 4bd3lgh4f0r -p $DOCKER_USER_PASSWORD
  script:
    - docker build -t  4bd3lgh4f0r/devsecops_frontend  .
    - docker push 4bd3lgh4f0r/devsecops_frontend

container_scanning:
  stage: container_scanning
  variables:
    CS_IMAGE: 4bd3lgh4f0r/devsecops_frontend

```



Results

Vulnerability report

+ Submit vulnerability | Export

The Vulnerability Report shows results of successful scans on your project's default branch, manually added vulnerability records, and vulnerabilities found from scanning operational environments. Learn more.

Development vulnerabilities 782 | Operational vulnerabilities 0

Security reports last updated 1 week ago #1121952268 • **SBOMs last updated** 1 week ago #1121952268

Critical	High	Medium	Low	Info	Unknown
4	55	233	484	0	6

Status: Needs triage +1 more | Severity: All severities | Tool: All tools | Activity: Still detected

Group by: None

Detected	Status	Severity	Description	Identifier	Tool	Activity
2023-12-28	Needs Triage	Critical	CVE-2023-45853 in :zlib1g 4bd3lgh4f0r/devsecops_frontend	CVE-2023-45853	Container Scanning	
2023-12-28	Needs Triage	Critical	CVE-2023-51385 in :openssh-client 4bd3lgh4f0r/devsecops_frontend	CVE-2023-51385	Container Scanning	

DAST

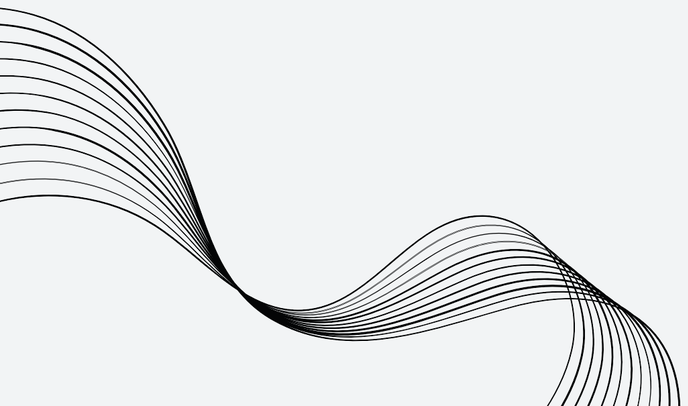
Dynamic Application Security Testing (DAST) is a security testing methodology that assesses the vulnerability of a software application by actively examining its running version. Unlike static analysis that reviews the source code, DAST operates during runtime, simulating real-world attack scenarios to identify potential vulnerabilities. DAST is crucial for ensuring the security of software as it provides a dynamic and comprehensive assessment of an application's security posture, detecting issues such as input validation flaws, authentication weaknesses, and other runtime vulnerabilities. By mimicking the behavior of malicious actors, DAST helps uncover vulnerabilities that may not be evident in static analysis, allowing developers and security teams to address weaknesses and enhance the overall security of the software.

Tool used:

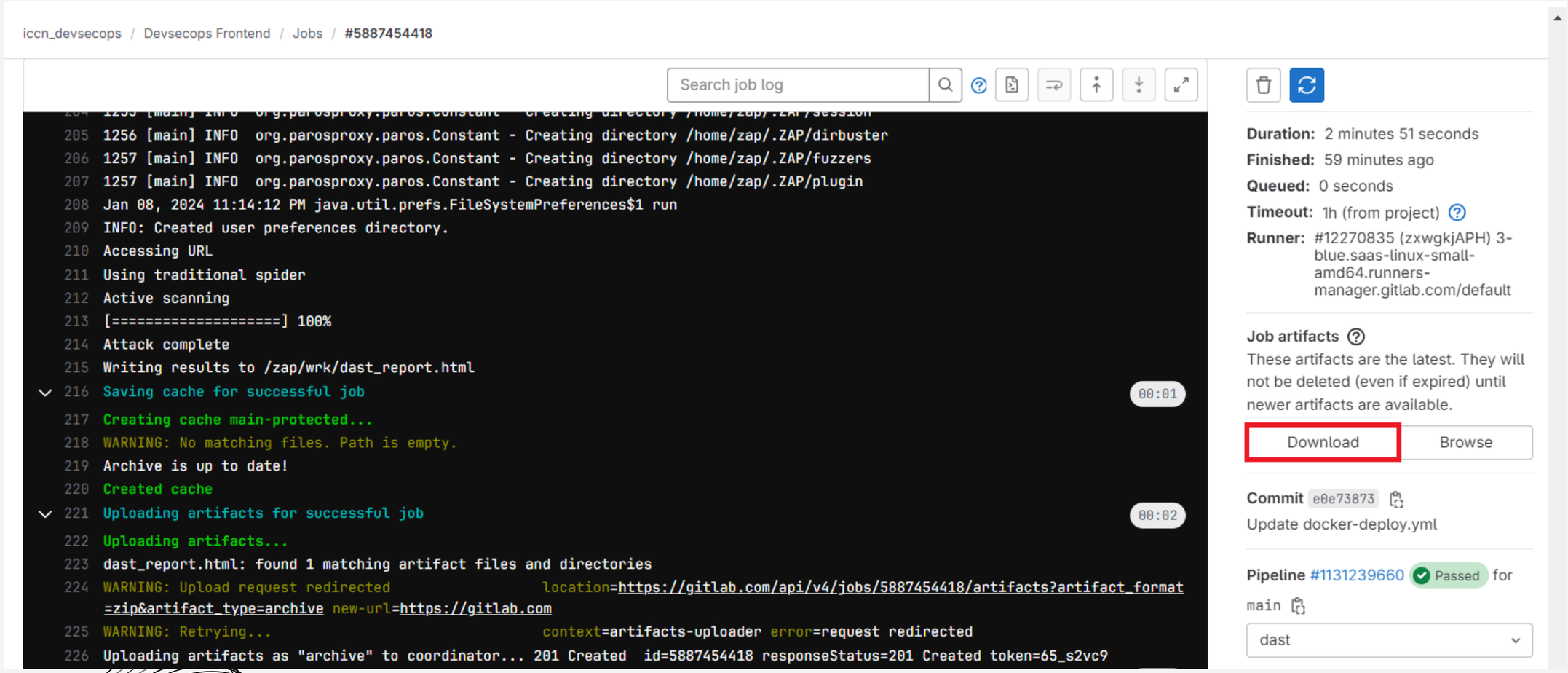
OWASP
Zed Attack Proxy

Configuration:

```
dast:
  stage: dast
  rules:
    - needs:
        - job: "docker-build"
        - job: "container_scanning"
    - exists:
        - $CI_COMMIT_BRANCH == "main"
  script:
    - docker network create my_network
    - docker run -d -p 9001:4200 --name frontend --network my_network 4bd3lgh4f0r/devsecops_frontend
    - docker pull softwaresecurityproject/zap-stable
    - docker run --name dast --network my_network --rm -v "$(pwd):/zap/wrk" -t softwaresecurityproject/zap-stable zap.sh -cmd -quickurl http://
frontend:4200 -quickprogress -quickout /zap/wrk/dast_report.html

  artifacts:
    paths:
      - dast_report.html
```



We download the artifact after the job is finish:



Result:

ZAP Report

Site: http://frontend:4200

Generated on Mon, 8 Jan 2024 23:14:36

ZAP Version: 2.14.0

Summary of Alerts

Risk Level	Number of Alerts
High	1
Medium	3
Low	2
Informational	0
False Positives:	0

Alerts

Name	Risk Level	Number of Instances
Cloud Metadata Potentially Exposed	High	1
.htaccess Information Leak	Medium	1
Content Security Policy (CSP) Header Not Set	Medium	3
Missing Anti-clickjacking Header	Medium	3
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	3
X-Content-Type-Options Header Missing	Low	3

The same thing is done to backend pipline:

Result:

Vulnerability report

+ Submit vulnerability | Export

The Vulnerability Report shows results of successful scans on your project's default branch, manually added vulnerability records, and vulnerabilities found from scanning operational environments. Learn more.

Development vulnerabilities 180 | Operational vulnerabilities 0

Security reports last updated 1 day ago #1132323145 • **SBOMs last updated** 1 day ago #1132323145

Critical	High	Medium	Low	Info	Unknown
12	45	49	74	0	0

Status: Needs triage +1 more | **Severity**: All severities | **Tool**: All tools | **Activity**: Still detected

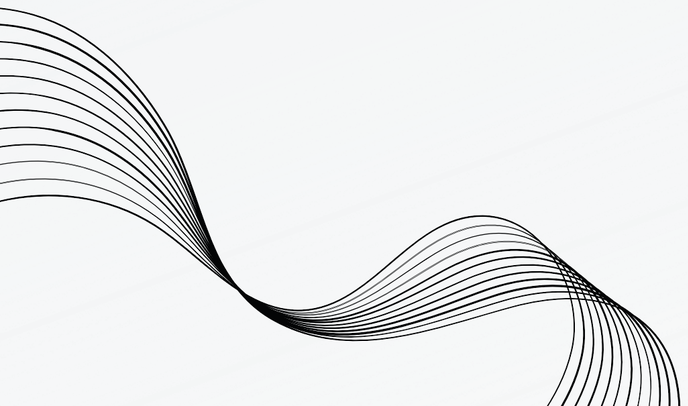
Group by: None

Detected	Status	Severity	Description	Identifier	Tool	Activity
2024-01-08	Needs Triage	Critical	CVE-2022-1292 in :openssl 4bd3lgh4f0r/devsecops_backend	CVE-2022-1292	Container Scanning	
2024-01-08	Needs Triage	Critical	CVE-2019-8457 in :libdb5.3 4bd3lgh4f0r/devsecops_backend	CVE-2019-8457	Container Scanning	

4-MONITORING

What is Monitoring?

Monitoring is an essential aspect of managing and ensuring the health and performance of our cluster and applications. Monitoring allows us to collect and analyze data about various components in our Kubernetes environment.



How to do it?

There are several tools to monitor your environment in Kubernetes and ensure the tracking of your system resources and in this project we will use:

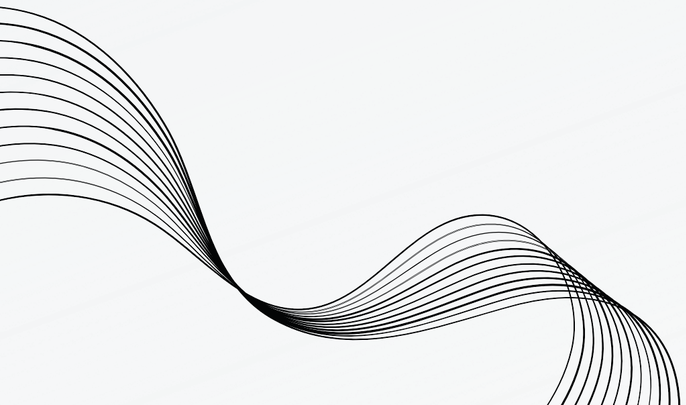
- **Prometheus**
- **Grafana**

Prometheus

Prometheus is an open-source monitoring system that collects and analyzes time series data, providing powerful querying, alerting, and integration capabilities for monitoring applications and infrastructure in dynamic environments like Kubernetes.

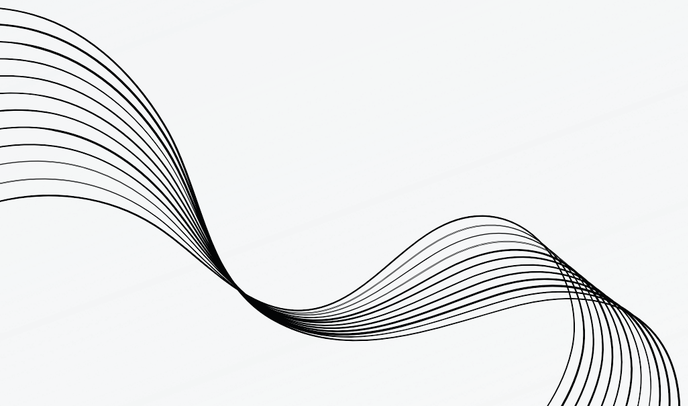
Grafana

Grafana is an open-source visualization tool that provides customizable dashboards for monitoring and analyzing metrics, while Prometheus is an open-source monitoring system that collects and stores time series data from various sources.



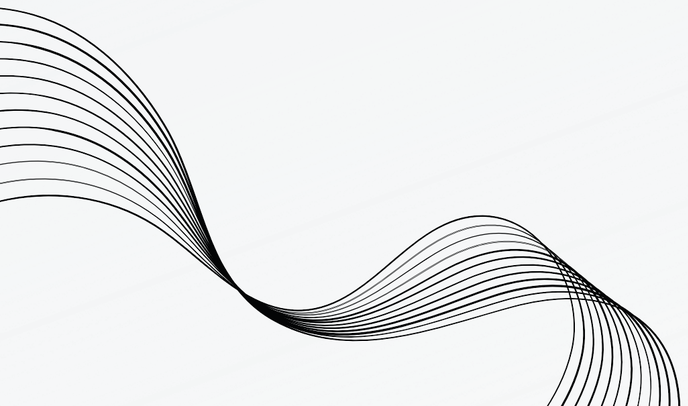
Adding prometheus repo with helm

```
C:\Users\DELL>helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
"prometheus-community" already exists with the same configuration, skipping
```



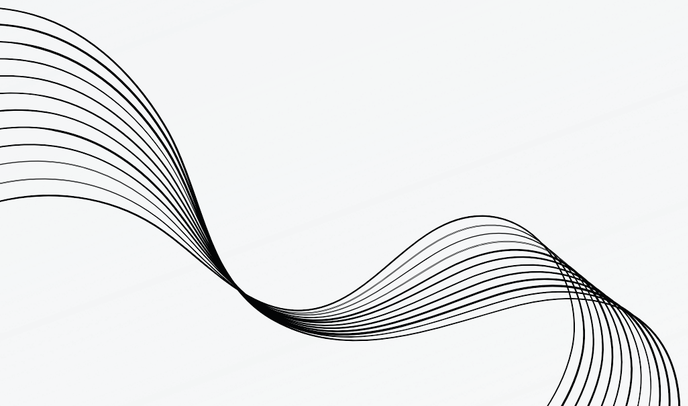
Adding the charts repo to my repositories

```
C:\Users\DELL>helm repo add stable https://charts.helm.sh/stable
"stable" has been added to your repositories
```



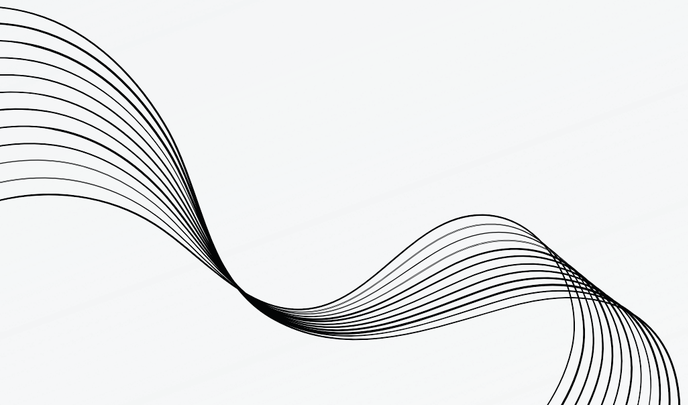
Updating repo with helm

```
C:\Users\DELL>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 "stable" chart repository
Update Complete. ⎈Happy Helming!⎈
```



Installing Prometheus

```
C:\Users\DELL>helm install prometheus prometheus-community/kube-prometheus-stack
NAME: prometheus
LAST DEPLOYED: Fri Dec 29 11:19:07 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace default get pods -l "release=prometheus"

Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmanager and Prometheu
s instances using the Operator.
```

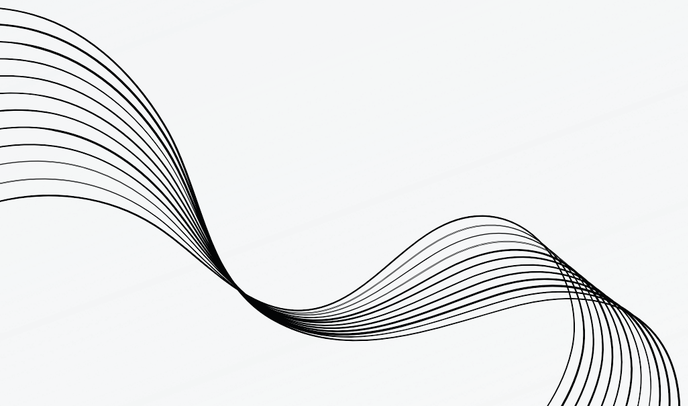


List allresources in Kubernetes cluster

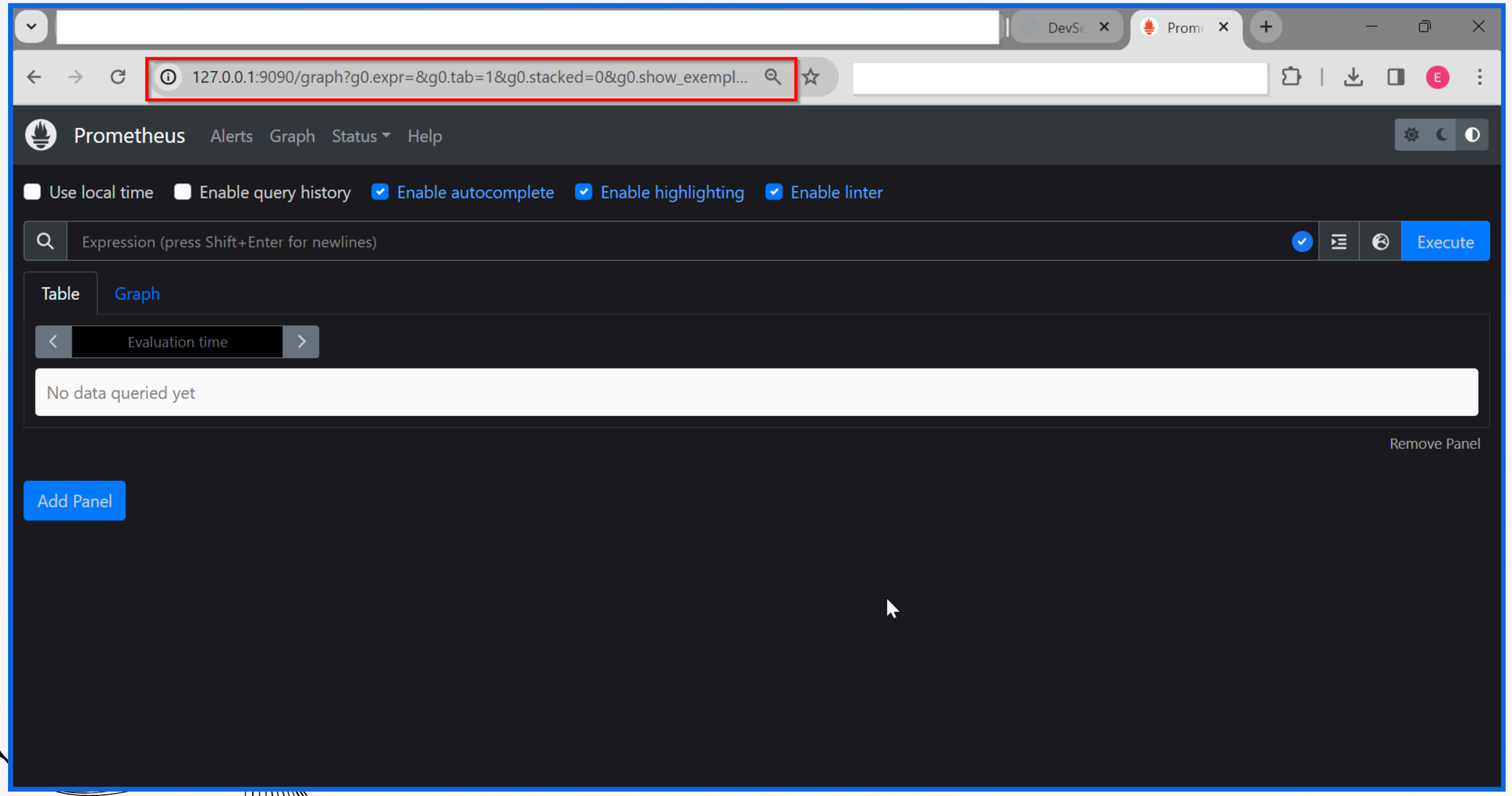
```
C:\Users\DELL>minikube kubectl get all
NAME                                                   READY   STATUS              RESTARTS   AGE
pod/alertmanager-prometheus-kube-prometheus-alertmanager-0   0/2     Init:0/1            0          6s
pod/prometheus-grafana-57cc5d6996-2mz7k                0/3     ContainerCreating   0          41s
pod/prometheus-kube-prometheus-operator-558444bb59-qsw5p     1/1     Running             0          41s
pod/prometheus-kube-state-metrics-6cd846d5cf-f8w99           1/1     Running             0          41s
pod/prometheus-prometheus-kube-prometheus-prometheus-0       0/2     Init:0/1            0          6s
pod/prometheus-prometheus-node-exporter-l74gq                1/1     Running             0          41s

NAME                                             TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
service/alertmanager-operated                    ClusterIP   None             <none>        9093/TCP,9094/TCP,9094/UDP   6s
service/kubernetes                               ClusterIP   10.96.0.1        <none>        443/TCP                      7m6s
service/prometheus-grafana                       ClusterIP   10.105.192.224   <none>        80/TCP                       42s
service/prometheus-kube-prometheus-alertmanager  ClusterIP   10.107.248.98    <none>        9093/TCP,8080/TCP            42s
service/prometheus-kube-prometheus-operator      ClusterIP   10.99.104.83     <none>        443/TCP                      42s
service/prometheus-kube-prometheus-prometheus    ClusterIP   10.109.153.168   <none>        9090/TCP,8080/TCP            42s
service/prometheus-kube-state-metrics            ClusterIP   10.105.97.250    <none>        8080/TCP                     42s
service/prometheus-operated                      ClusterIP   None             <none>        9090/TCP                     6s
service/prometheus-prometheus-node-exporter      ClusterIP   10.107.141.222   <none>        9100/TCP                     42s

NAME                                                DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
daemonset.apps/prometheus-prometheus-node-exporter   1         1         1       1            1           kubernetes.io/os=linux   41s

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/prometheus-grafana            0/1     1            0           41s
deployment.apps/prometheus-kube-prometheus-operator   1/1     1            1           41s
deployment.apps/prometheus-kube-state-metrics         1/1     1            1           41s

NAME                                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/prometheus-grafana-57cc5d6996          1         1         0       41s
replicaset.apps/prometheus-kube-prometheus-operator-558444bb59   1         1         1       41s
replicaset.apps/prometheus-kube-state-metrics-6cd846d5cf         1         1         1       41s

NAME                                                              READY   AGE
statefulset.apps/alertmanager-prometheus-kube-prometheus-alertmanager   0/1     6s
statefulset.apps/prometheus-prometheus-kube-prometheus-prometheus       0/1     6s
```

Running Prometheus using port-forward

```
C:\Users\DELL\Desktop\DevSecOps>kubectl -- port-forward prometheus-prometheus-kube-prometheus-prometheus-0 9090 -n webapp
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
```



Prometheus Web-UI

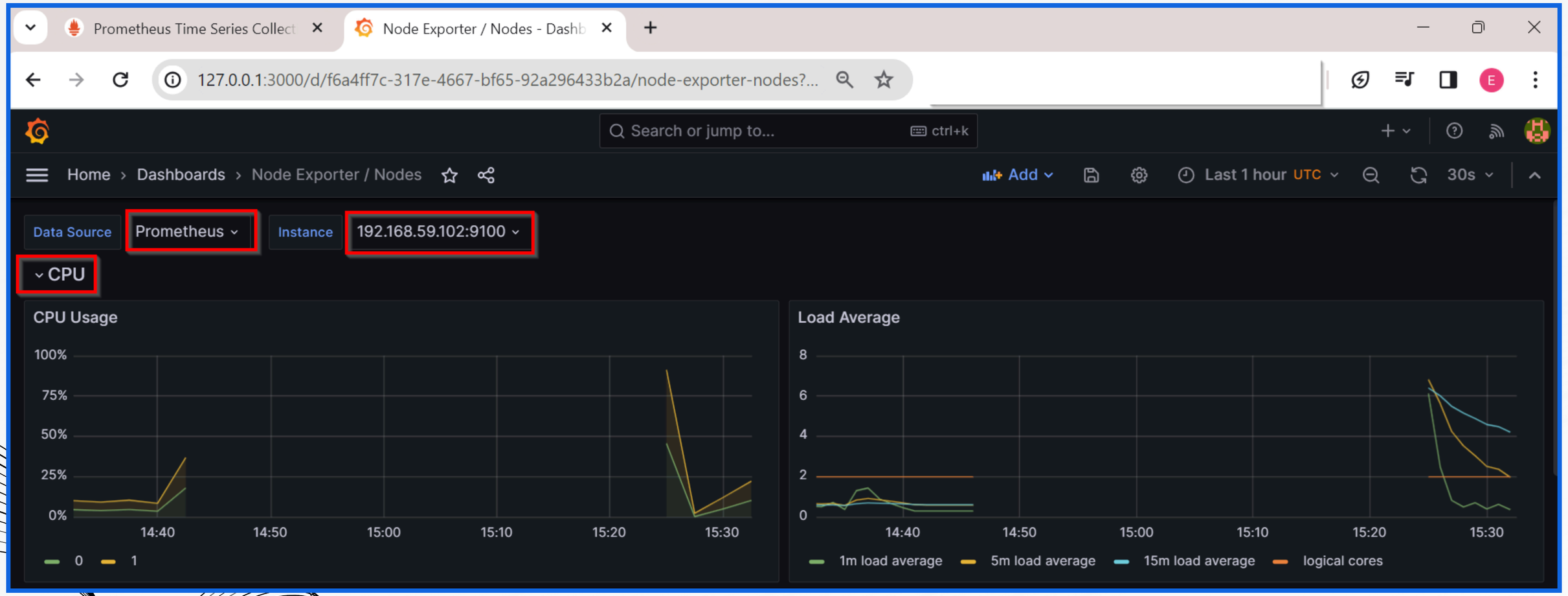


Running Grafana using port-forward

```
C:\Users\DELL\Desktop\DevSecOps>minikube kubectl port-forward deployment/prometheus-grafana 3000
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
```

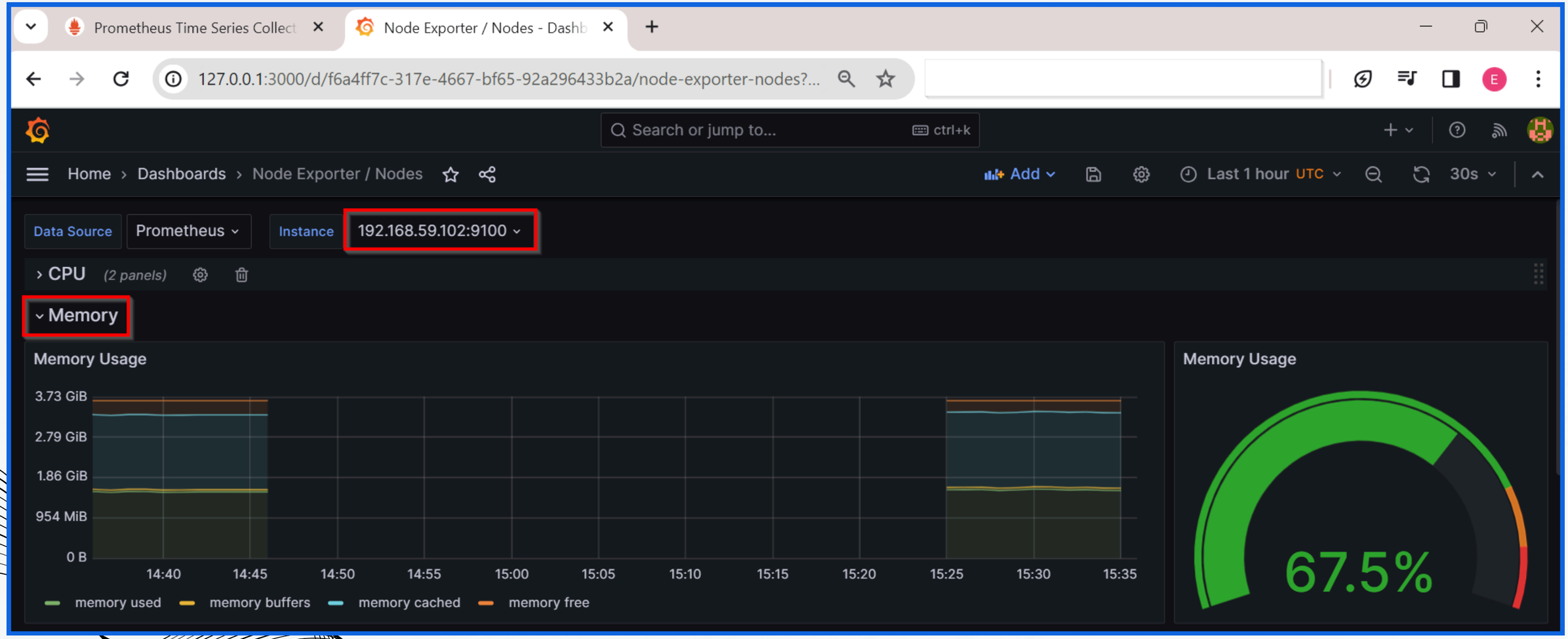
System Monitoring

=> CPU:



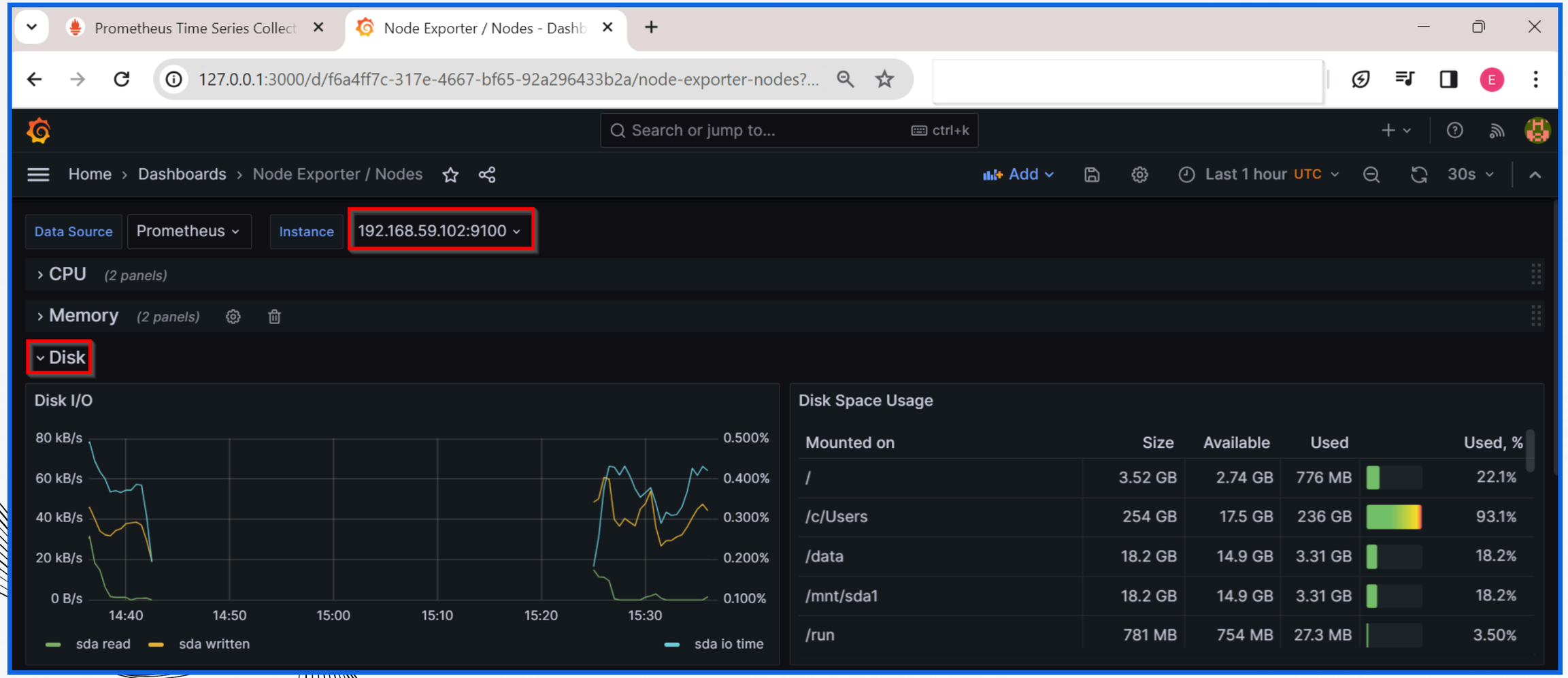
System Monitoring

=> RAM:



System Monitoring

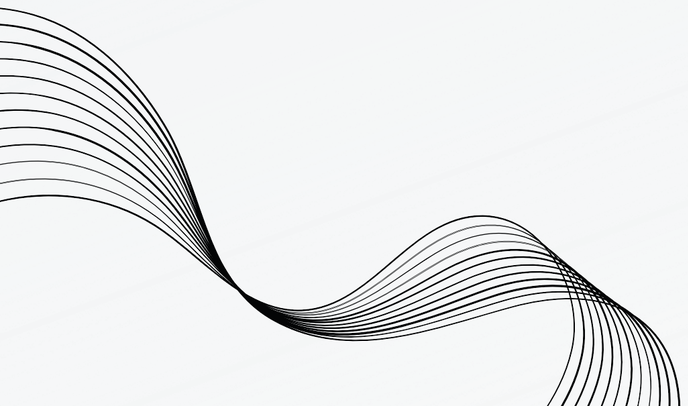
=> Disk:



Application Monitoring

=> Mysql deployment:

```
C:\Users\DELL\Desktop\DevSecOps>kubectl -- apply -f mysql-deployment.yaml -n webapp
deployment.apps/mysql-deployment created
service/mysql-service unchanged
```



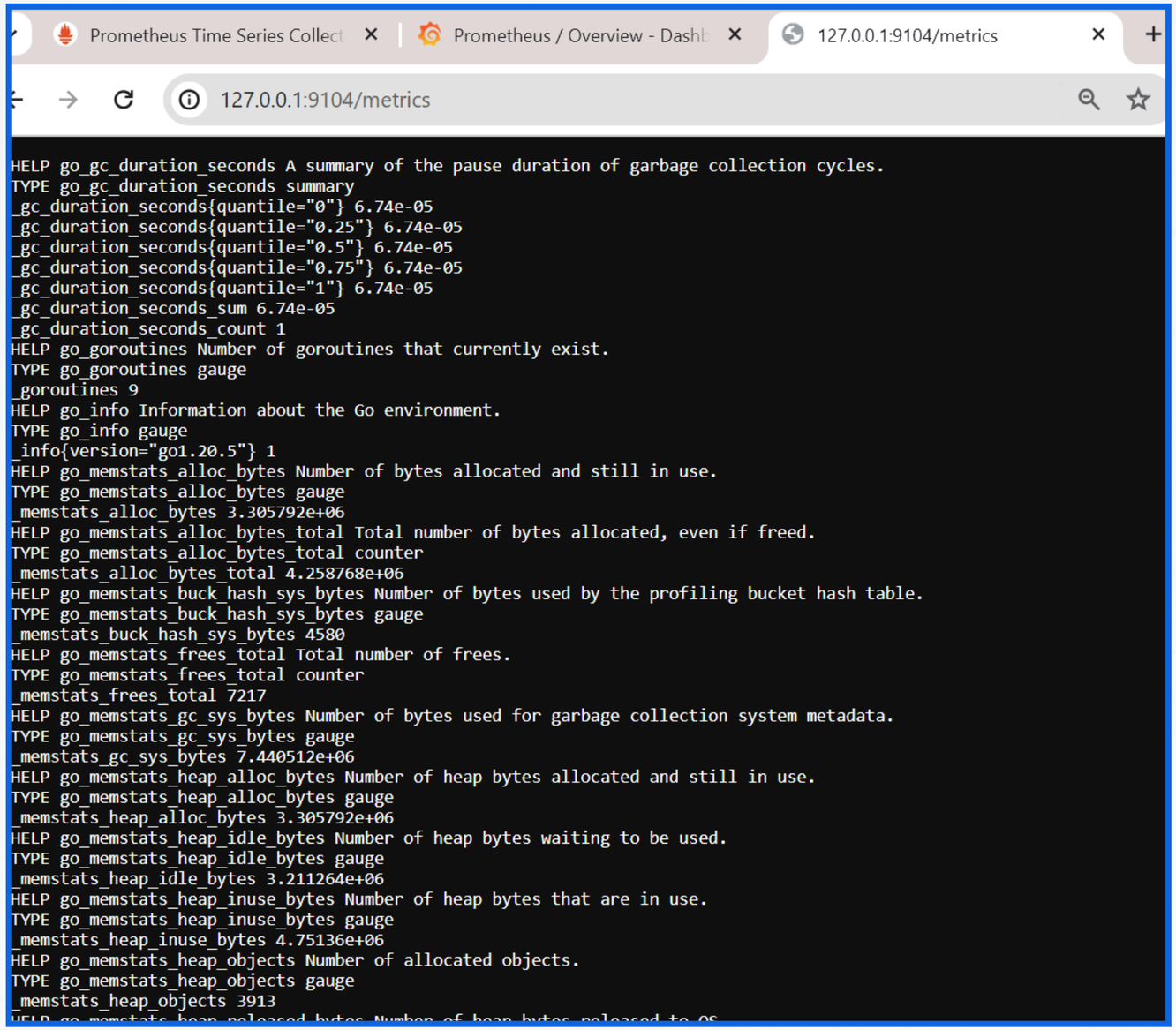
Application Monitoring

=> Installation of mysql-exporter:

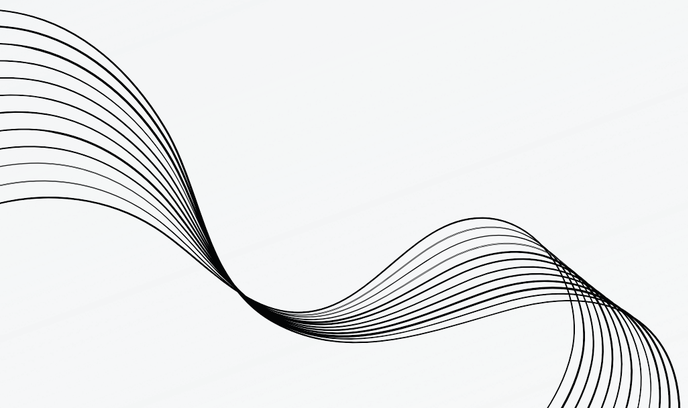
```
C:\Users\DELL\Desktop\DevSecOps>helm install mysql-exporter prometheus-community/prometheus-mysql-exporter -f values.yaml -n webapp
NAME: mysql-exporter
LAST DEPLOYED: Sun Jan  7 18:17:22 2024
NAMESPACE: webapp
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace webapp -l "app.kubernetes.io/name=prometheus-mysql-exporter,app.kubernetes.io/instan
ce=mysql-exporter" -o jsonpath="{.items[0].metadata.name}")
  echo "Visit http://127.0.0.1:9104 to use your application"
  kubectl --namespace webapp port-forward $POD_NAME 9104
```

Application Monitoring

=> Collected metrics of mysql:

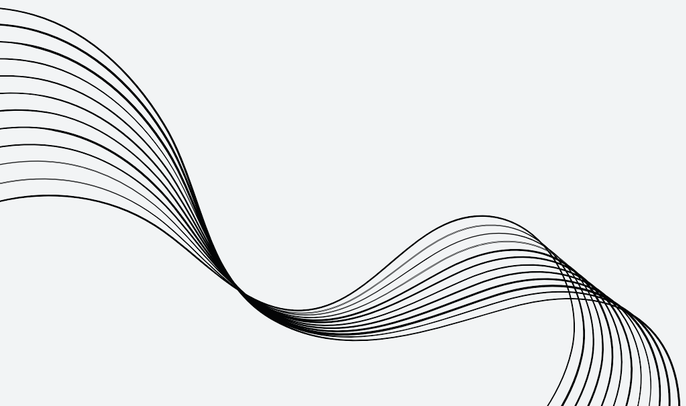


5-KUBERNETES DEPLOYMENT



Minikube

Minikube is a lightweight tool that enables developers to run and manage a single-node Kubernetes cluster on their local machine, facilitating the development and testing of containerized applications.

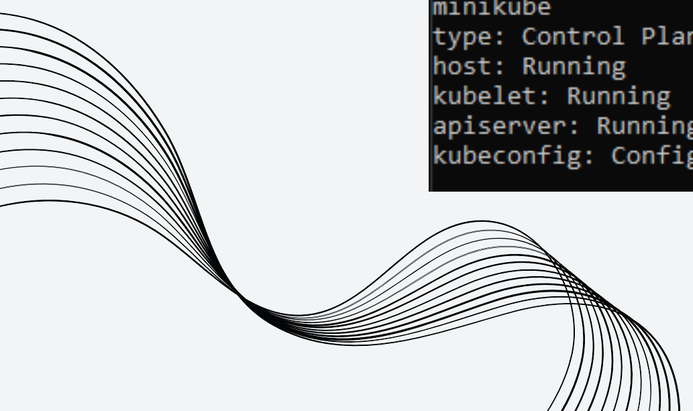


minikube installation with choco :

```
PS C:\WINDOWS\system32> choco install minikube
Chocolatey v2.2.2
Installing the following packages:
minikube
By installing, you accept licenses for the packages.
Progress: Downloading kubernetes-cli 1.29.0... 100%

kubernetes-cli v1.29.0 [Approved]
kubernetes-cli package files install completed. Performing other installation steps.
The package kubernetes-cli wants to run 'chocolateyInstall.ps1'.
Note: If you don't run this script, the installation will fail.
Note: To confirm automatically next time, use '-y' or consider:
choco feature enable -n allowGlobalConfirmation
Do you want to run the script?([Y]es/[A]ll - yes to all/[N]o/[P]rint): y

Extracting 64-bit C:\ProgramData\chocolatey\lib\kubernetes-cli\tools\kubernetes-client-windows-amd64.tar.gz to C:\Progra
mData\chocolatey\lib\kubernetes-cli\tools...
C:\ProgramData\chocolatey\lib\kubernetes-cli\tools
Extracting 64-bit C:\ProgramData\chocolatey\lib\kubernetes-cli\tools\kubernetes-client-windows-amd64.tar to C:\ProgramDa
ta\chocolatey\lib\kubernetes-cli\tools...
C:\ProgramData\chocolatey\lib\kubernetes-cli\tools
 ShimGen has successfully created a shim for kubectl-convert.exe
 ShimGen has successfully created a shim for kubectl.exe
 The install of kubernetes-cli was successful.
  Software installed to 'C:\ProgramData\chocolatey\lib\kubernetes-cli\tools'
Progress: Downloading Minikube 1.32.0... 100%

Minikube v1.32.0 [Approved]
Minikube package files install completed. Performing other installation steps.
 ShimGen has successfully created a shim for minikube.exe
 The install of Minikube was successful.
  Software installed to 'C:\ProgramData\chocolatey\lib\Minikube'

Chocolatey installed 2/2 packages.
 See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
```

=> Starting a Kubernetes instance locally using Minikube. :

```
PS C:\WINDOWS\system32> minikube start
W1228 21:40:46.991056   31312 main.go:291] Unable to resolve the current Docker CLI context "default": context "default"
: context not found: open C:\Users\LENOVO\.docker\contexts\meta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a
33f0688f\meta.json: The system cannot find the path specified.
* minikube v1.32.0 on Microsoft Windows 11 Pro 10.0.22631.2861 Build 22631.2861
* Automatically selected the docker driver. Other choices: virtualbox, ssh
* Using Docker Desktop driver with root privileges
* Starting control plane node minikube in cluster minikube
* Pulling base image ...
* Downloading Kubernetes v1.28.3 preload ...
    > gcr.io/k8s-minikube/kicbase...:  453.90 MiB / 453.90 MiB  100.00% 1.41 Mi
    > preloaded-images-k8s-v18-v1...:  403.35 MiB / 403.35 MiB  100.00% 1.19 Mi
* Creating docker container (CPUs=2, Memory=4000MB) ...
* Preparing Kubernetes v1.28.3 on Docker 24.0.7 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring bridge CNI (Container Networking Interface) ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
PS C:\WINDOWS\system32> minikube status
W1228 21:49:33.151781   38048 main.go:291] Unable to resolve the current Docker CLI context "default": context "default"
: context not found: open C:\Users\LENOVO\.docker\contexts\meta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a
33f0688f\meta.json: The system cannot find the path specified.
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```



Creating the 'webapp' namespace:

```
PS C:\WINDOWS\system32> kubectl create namespace webapp
>>
namespace/webapp created
```

We created a new Kubernetes namespace named 'webapp.' The output "namespace/webapp created" confirms the successful creation of the 'webapp' namespace, providing an isolated environment for managing resources within a Kubernetes cluster.

Use of environment variables

Before starting the creation of manifests for deployment, small modifications need to be made to the source code of the "application.properties" file. We will use environment variables for the database connection.

application.properties 422 B

```
spring.datasource.url= jdbc:mysql://${SPRING_DATASOURCE_URL}:3306/testdb?useSSL=false
spring.datasource.username= root
spring.datasource.password= root

spring.datasource.url=${SPRING_DATASOURCE_URL}
spring.datasource.username=${SPRING_DATASOURCE_USERNAME}
spring.datasource.password=${MYSQL_ROOT_PASSWORD}

spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.ddl-auto= update

```

Deployment

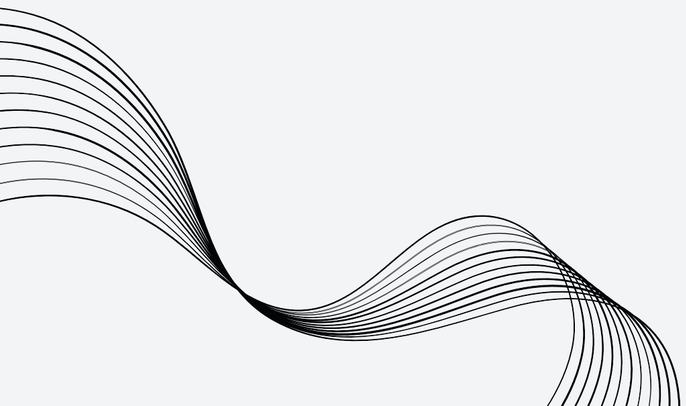
=> backend.yaml (deployment) :

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: backend-container
          image: 4bd3lgh4f0r/devsecops_backend:latest
          ports:
            - containerPort: 8080
          env:
            - name: SPRING_DATASOURCE_URL
              value: "jdbc:mysql://frontend-service:3306/testdb"
            - name: FRONTEND_IP
              value: "frontend-service"
```

Deployment

=> frontend.yaml (deployment) :

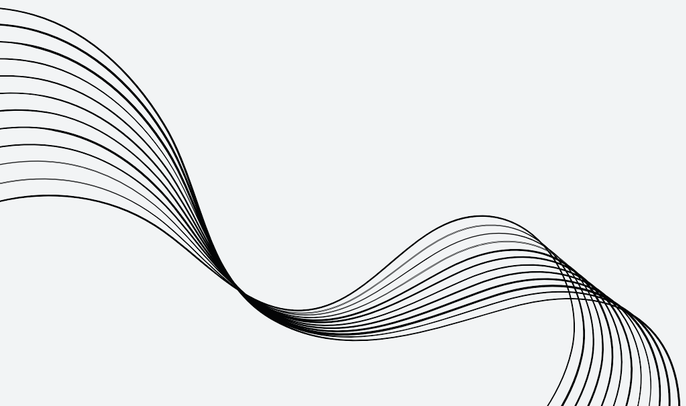
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend-container
        image: 4bd3lgh4f0r/devsecops_frontend
        imagePullPolicy: Always
        ports:
          - name: http
            containerPort: 4200
        env:
          - name: backendIP
            value: "backend-service"
---
```



Deployment

=> mysq.yaml (deployment) :

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
  labels:
    app: mysql
  namespace: webapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:latest
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: root_password
        - name: MYSQL_DATABASE
          value: webapp_database
---
```



Service

=> Backend.yaml (Service) :

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend-service
  namespace: webapp
  labels:
    app: backend
spec:
  type: ClusterIP
  selector:
    app: backend
  ports:
  - name: http
    port: 8080
    targetPort: 8080
```

Service

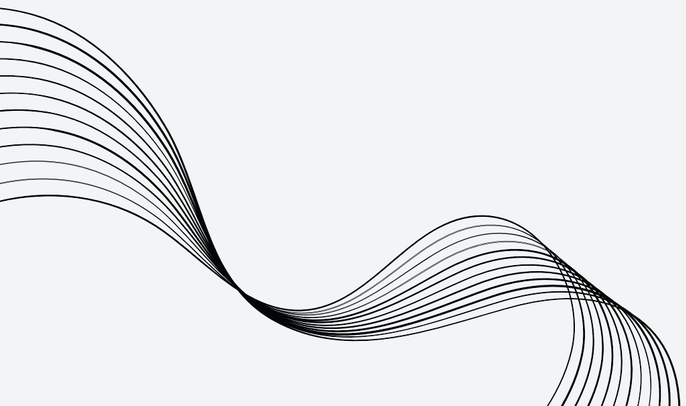
=> Frontend.yaml (Service) :

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
  namespace: webapp
spec:
  type: ClusterIP
  selector:
    app: frontend
  ports:
  - name: http
    port: 4200
    targetPort: 4200
```

Service

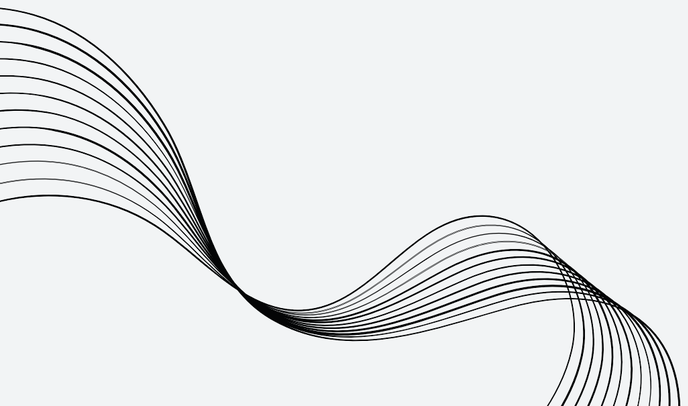
=> mysql.yaml (Service) :

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
  namespace: webapp
spec:
  selector:
    app: mysql
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
  type: ClusterIP
```



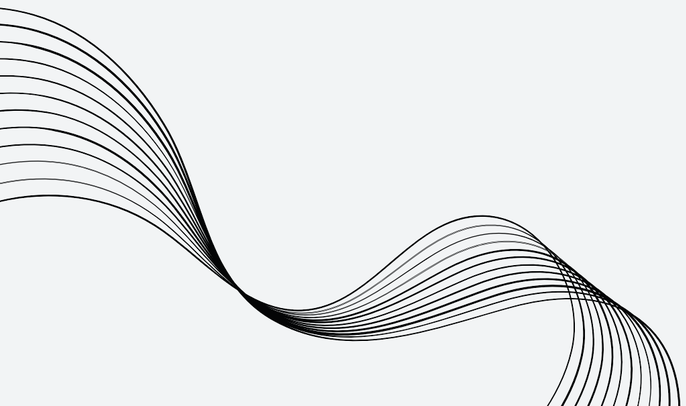
Application of Configurations in the Kubernetes Cluster:

```
PS C:\Users\LENOVO\webapp-k8s-config-main> kubectl apply -f backend.yaml -n webapp
deployment.apps/backend-deployment created
service/backend-service created
PS C:\Users\LENOVO\webapp-k8s-config-main> kubectl apply -f frontend.yaml -n webapp
deployment.apps/frontend-deployment created
service/frontend-service created
PS C:\Users\LENOVO\webapp-k8s-config-main> kubectl apply -f mysql.yaml -n webapp
deployment.apps/mysql-deployment created
service/mysql-service created
```



the pods are currently running

```
PS C:\Users\LENOVO\webapp-k8s-config-main> kubectl get pods -n webapp
NAME                                       READY   STATUS    RESTARTS      AGE
backend-deployment-786b959994-n894m        1/1     Running   1 (48s ago)   99s
frontend-deployement-55b4f46d6c-jxk2p      1/1     Running   1 (21m ago)   22m
frontend-deployment-55b4f46d6c-6dwqb       1/1     Running   0             20m
mysql-deployment-67d6cd6bbc-rwk4p          1/1     Running   0             20m
```



Accessing the Application Locally with kubectl port-forward

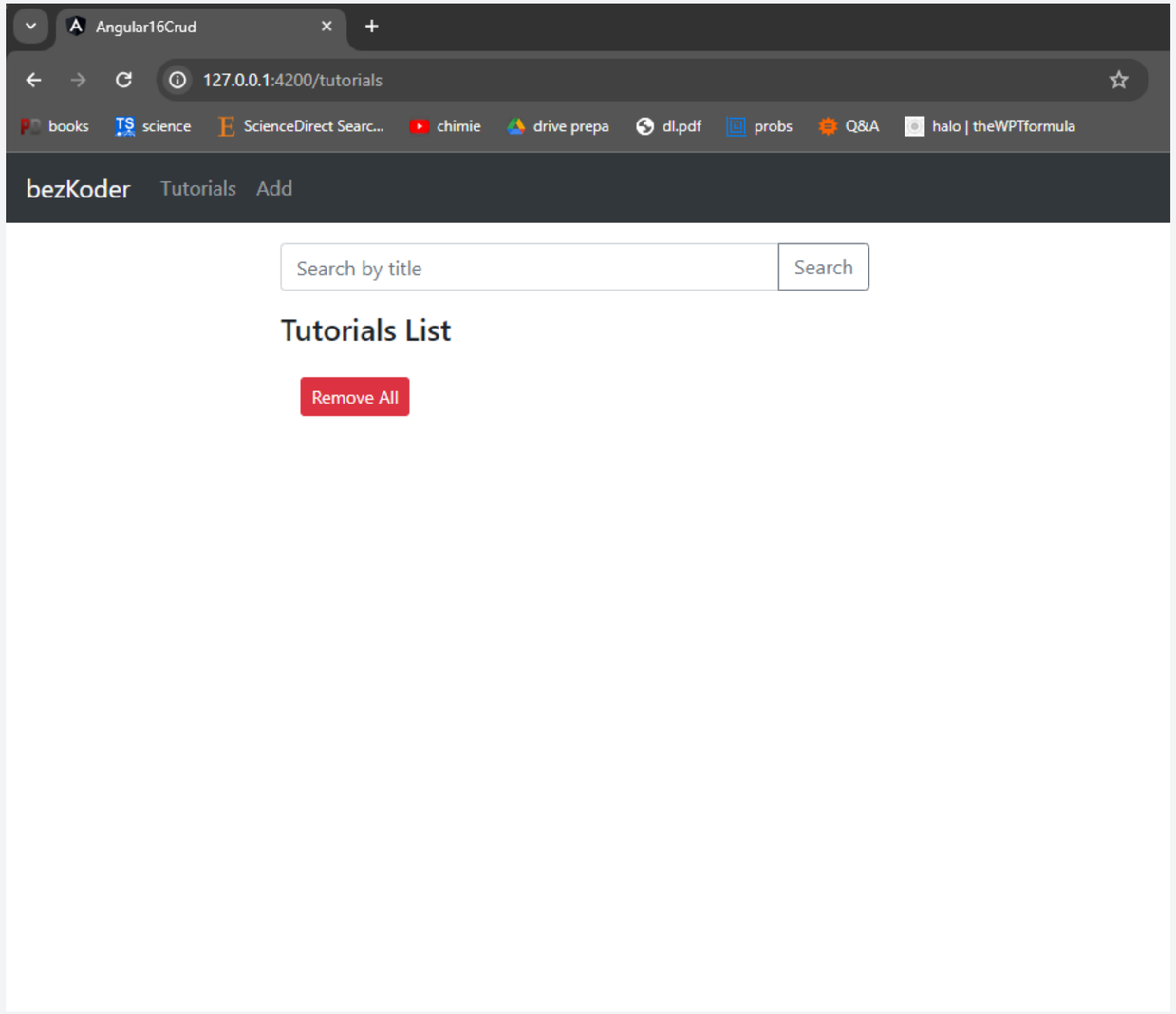
backend:

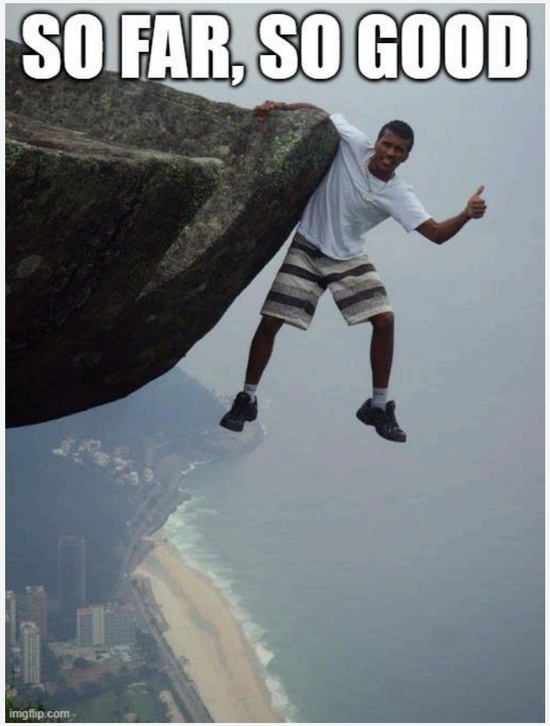
```
PS C:\Users\LENOVO> kubectl port-forward service/backend-service 8080:8080 -n webapp
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

frontend:

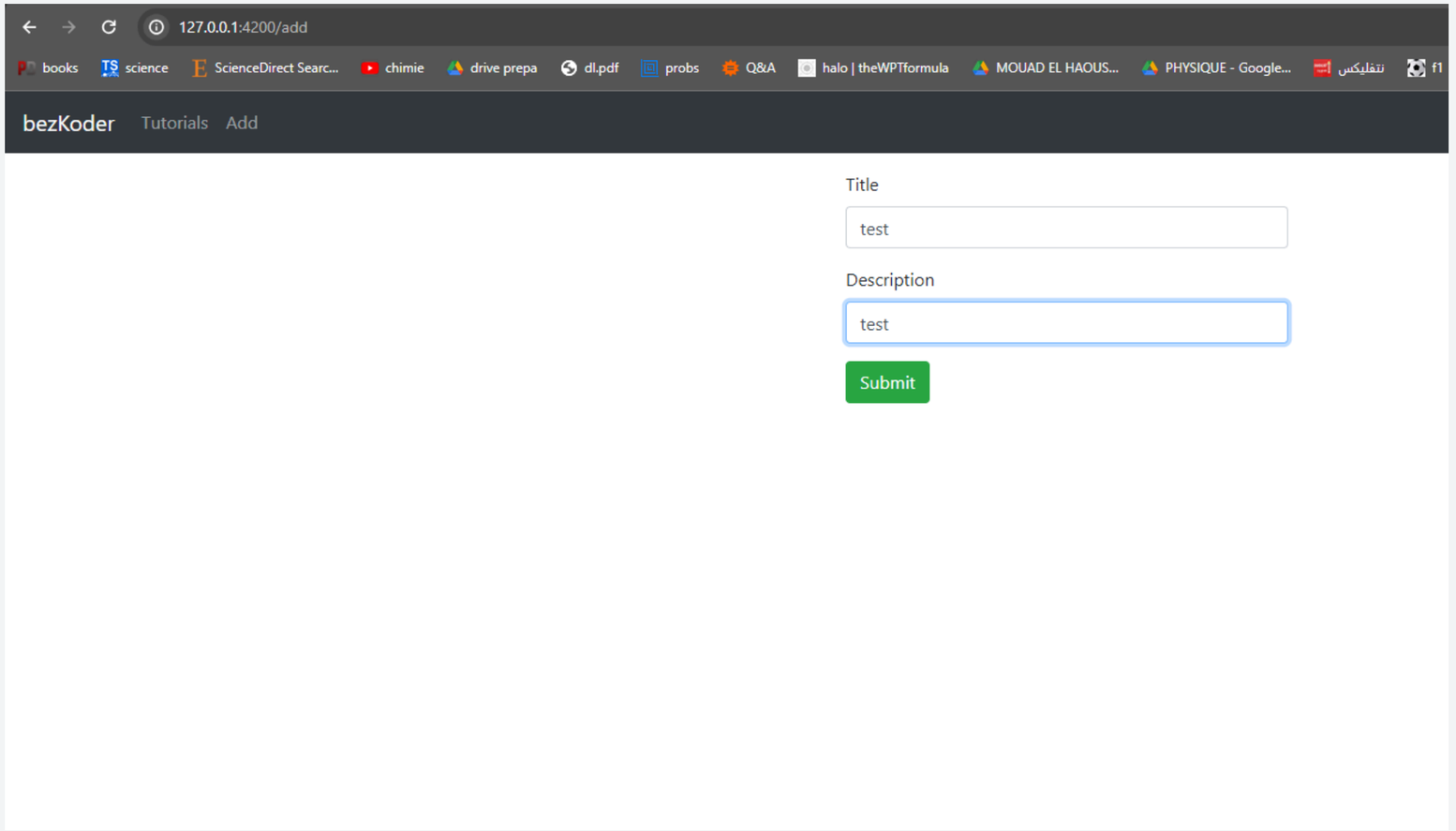
```
PS C:\Users\LENOVO\webapp-k8s-config-main> kubectl port-forward service/frontend-service 4200:4200 -n webapp
Forwarding from 127.0.0.1:4200 -> 4200
Forwarding from [::1]:4200 -> 4200
Handling connection for 4200
```

the frontend is succesfully accessed:



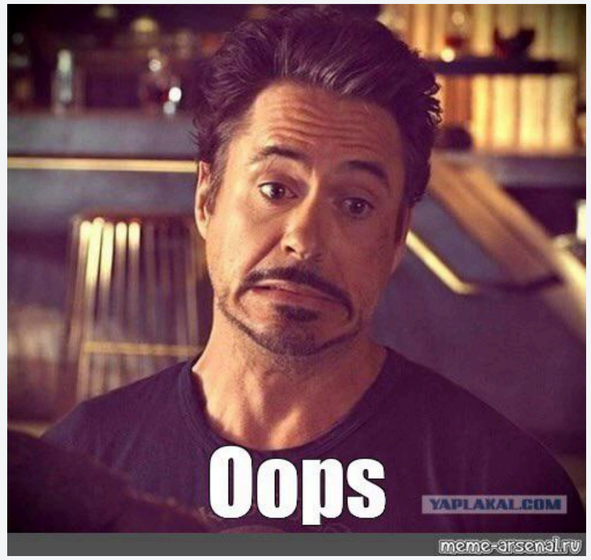


we try to interact with the database:

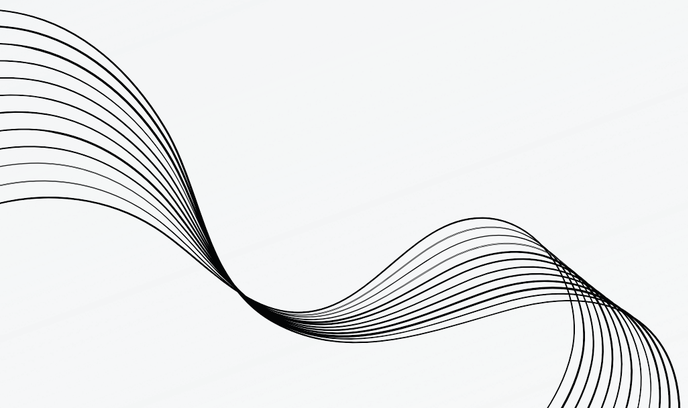


when we try to interact with the database, the connection with the pod gets lost:

```
Forwarding from [::1]:8080 -> 8080
Handling connection for 8080
E0109 23:30:26.490146   21804 portforward.go:409] an error occurred forwarding 8080 -> 8080: error forwarding port 8080 to pod 45efe105776168c7b33044ce4581f888e653ee3f82ecbc58c79655980b00a8a3, uid : exit statu
s 1: 2024/01/09 22:30:26 socat[354564] E connect(5, AF=2 127.0.0.1:8080, 16): Connection refused
error: lost connection to pod
PS C:\Users\LENOVO>
```



6-REPOSITORIES



The gitlab and dockerhub repositories:

gitlab : https://gitlab.com/iccn_devsecops

dockerhub : https://hub.docker.com/repositories/4bd3lgh4f0r