How to Embed Dependency-Check into GitLab CI/CD

Throughout this content, you will learn how to integrate a Dependency Check tool into GitLab, enabling the job to fail even when multiple issues are detected by the tool.

A simple CI/CD pipeline:

Let’s download the code using git clone in your machine. I prefer unix/linux

Once done, we need to push our source code into GitLab. Let’s download the code using git clone in DevSecOps Box.

git clone https://github.com/WebGoat/WebGoat.git
cd webgoat

=> Rename git url to the new one.

git remote rename origin old-origin
git remote add origin https://gitlab.com/tanmoy.xxx.xxx/WebGoat.git

=> Then, push the code into the repository.

git push -u origin --all

Username Your Username
Password Your Password

Let’s log into Gitlab using your accounts details

Name	Value
Gitlab URL	**https://gitlab.com/tanmoy.xxx.xxx/WebGoat.git**
Username	You
Password	Yours’

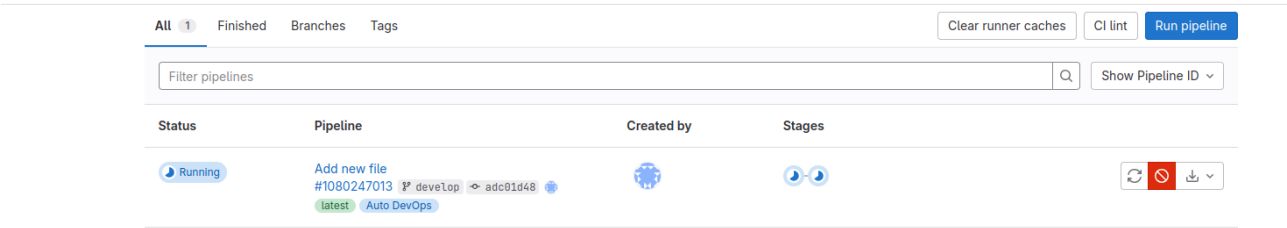
Next, we need to create a CI/CD pipeline by adding the **.gitlab-ci.yml file.** Click on the + (plus) button, then click New File and copy the above CI script (use Control+A and Control+V).

```
image: docker:20.10  # To run all jobs in this pipeline, use a latest docker image

services:
  - docker:dind       # To run all jobs in this pipeline, use a docker image which contains a docker daemon running inside (dind - docker in docker). Reference: https://forum.gitlab.com/t/why-services-docker-dind-is-needed-while-already-having-image-docker/43534

stages:
  - build
  - test
  - release
  - preprod
  - integration
  - prod

build:
  stage: build
  script:
    - echo "This is a build step"

test:
  stage: test
  script:
    - echo "This is a test step"
```

Save changes to the file using the Commit changes button.

As soon as a change is made to our repository, the pipeline starts to execute the defined jobs.

Check out the results of this pipeline by visiting https://link/path/webgoat/pipelines and click on the appropriate job name under the pipeline to see the job output.



As discussed in the **Software Component Analysis using Dependency Check** discussion, we can put Dependency Check in our CI/CD pipeline. However, do remember you need to run the command manually before you embed OAST in the pipeline.

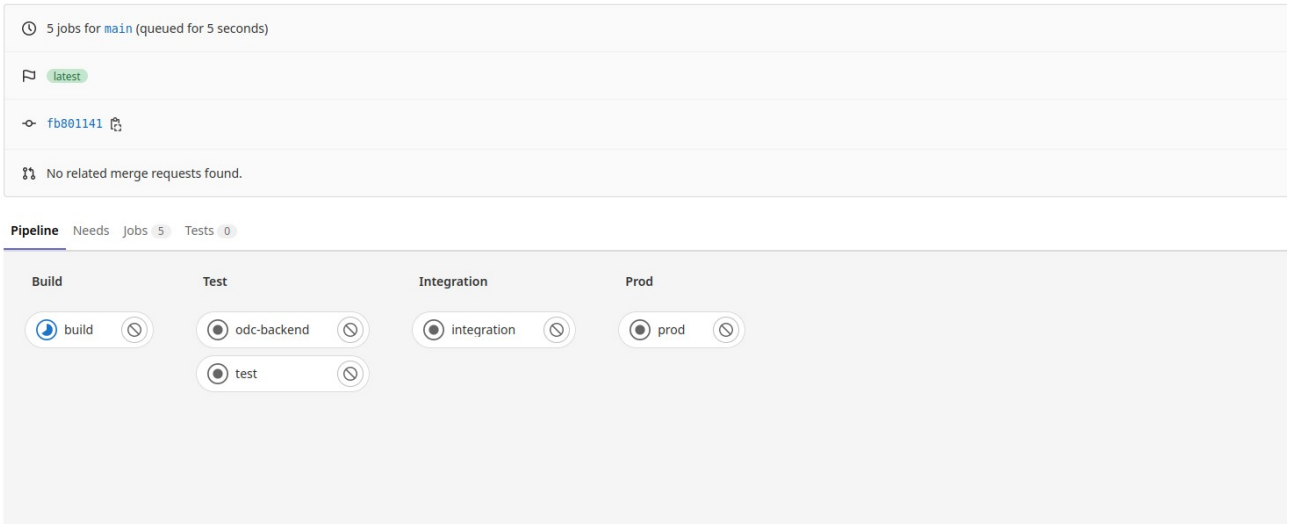
Visit your webgoat link https://webgoat.gitlab.com/root/webgoat to create a new file called `run-depcheck.sh` with the following contents.

```
#!/bin/sh

DATA_DIRECTORY="$PWD/data"
REPORT_DIRECTORY="$PWD/reports"

if [ ! -d "$DATA_DIRECTORY" ]; then
  echo "Initially creating persistent directories"
  mkdir -p "$DATA_DIRECTORY"
  chmod -R 777 "$DATA_DIRECTORY"

  mkdir -p "$REPORT_DIRECTORY"
  chmod -R 777 "$REPORT_DIRECTORY"
fi

# mvn install -Dmaven.test.skip=true
docker run --rm \
  --volume $(pwd):/src \
  --volume "$DATA_DIRECTORY":/usr/share/dependency-check/data \
  --volume "$REPORT_DIRECTORY":/report \
  owasp/dependency-check \
  --scan /src \
  --format "CSV" \
  --project "Webgoat" \
  --failOnCVSS 4 \
  --out /report
```

Then, add the following content to the **.gitlab-ci.yml** file.

```
image: docker:20.10  # To run all jobs in this pipeline, use a latest docker image

services:
  - docker:dind       # To run all jobs in this pipeline, use a docker image which contains a docker daemon running inside (dind - docker in docker). Reference: https://forum.gitlab.com/t/why-services-docker-dind-is-needed-while-already-having-image-docker/43534

stages:
  - build
  - test
  - release
  - preprod
  - integration
  - prod
```

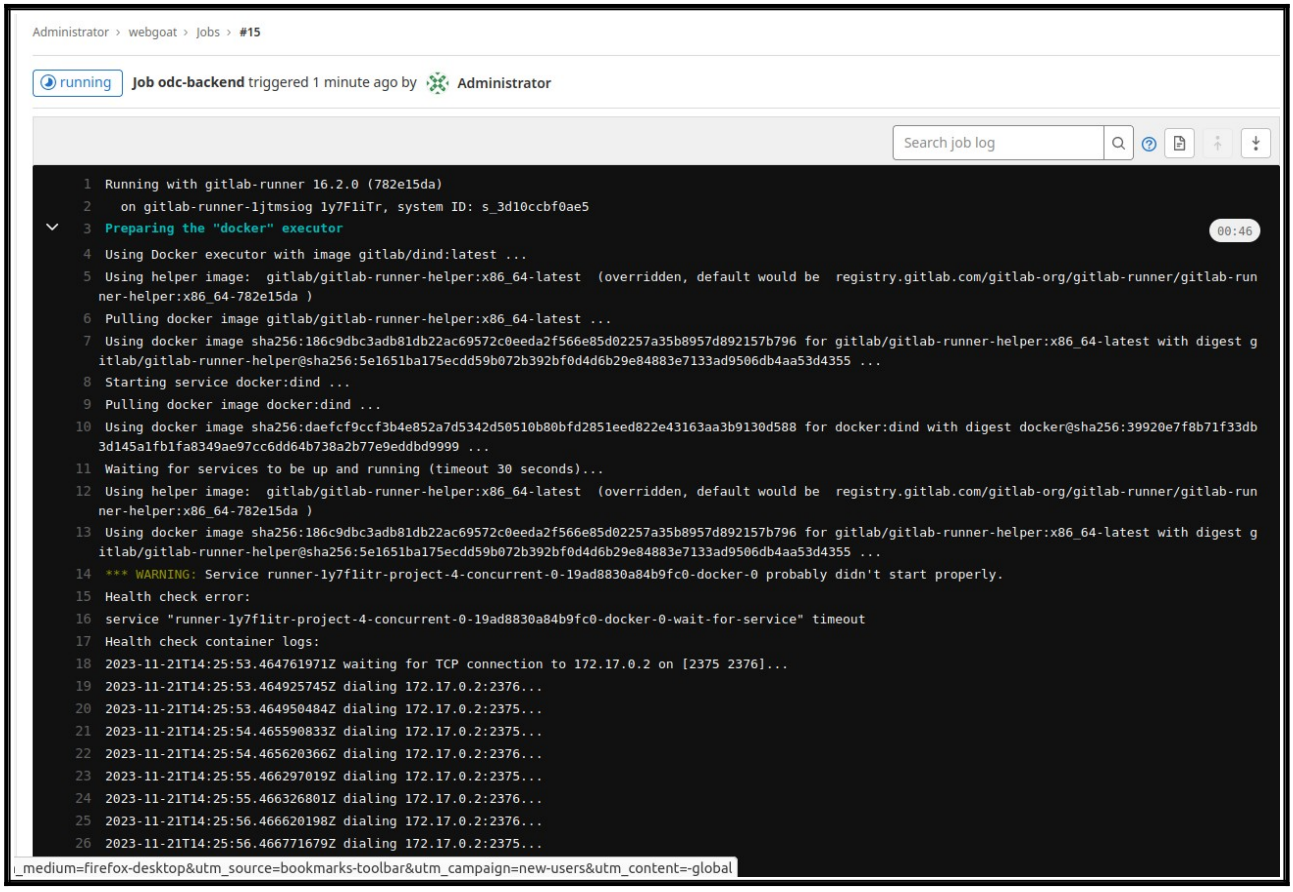
```
build:
  stage: build
  script:
    - echo "This is a build step"

test:
  stage: test
  script:
    - echo "This is a test step"

odc-backend:
  stage: test
  image: gitlab/dind:latest
  script:
    - chmod +x ./run-depcheck.sh
    - ./run-depcheck.sh
  artifacts:
    paths:
      - reports/dependency-check-report.csv
    when: always
    expire_in: one week

integration:
  stage: integration
  script:
    - echo "This is an integration step"
    - exit 1
  allow_failure: true # Even if the job fails, continue to the next stages

prod:
  stage: prod
  script:

    - echo "This is a deploy step."
  when: manual # Continuous Delivery
```



As discussed, any change to the repo kick starts the pipeline.

We can see the results of this pipeline by visiting https://webgoat.gitlab.com/root/webgoat/pipelines.

Click on the appropriate job name to see the output.



It is failed.In the **next step**, I will guide you why you should not fail the build.

In DevSecOps Maturity Levels 1 and 2, we do not want to fail the builds, jobs, or scans. This is because security tools often generate a significant number of false positives.

In DevSecOps Maturity Levels 1 and 2, the focus is primarily on integrating security practices into the development and deployment process. During these stages, it is common to encounter a significant number of false positives generated by security tools.

To ensure the smooth progression of development and deployment, it is generally not recommended to fail the builds or jobs based solely on the results of security scans. This is because false positives can disrupt the development cycle and hinder the efficiency and agility of the team.

You can use the **allow_failure** tag to prevent the build from failing, even if the tool identifies issues.

The pipeline would look like the following.

```
image: docker:20.10  # To run all jobs in this pipeline, use a latest docker image
services:
  - docker:dind       # To run all jobs in this pipeline, use a docker image which contains a docker daemon running inside (dind - docker in docker). Reference: https://forum.gitlab.com/t/why-services-docker-dind-is-needed-while-already-having-image-docker/43534

stages:
  - build
  - test
  - release
  - preprod
  - integration
  - prod

build:
  stage: build
  script:
    - echo "This is a build step"

test:
  stage: test
  script:
    - echo "This is a test step"

odc-backend:
  stage: test
  image: gitlab/dind:latest
  script:
    - chmod +x ./run-depcheck.sh
    - ./run-depcheck.sh
  artifacts:
    paths:
      - reports/dependency-check-report.csv
    when: always
    expire_in: one week
  allow_failure: true

integration:
```

```
  stage: integration
  script:
    - echo "This is an integration step"
    - exit 1
  allow_failure: true # Even if the job fails, continue to the next stages

prod:
  stage: prod
  script:
    - echo "This is a deploy step."
  when: manual # Continuous Delivery
```