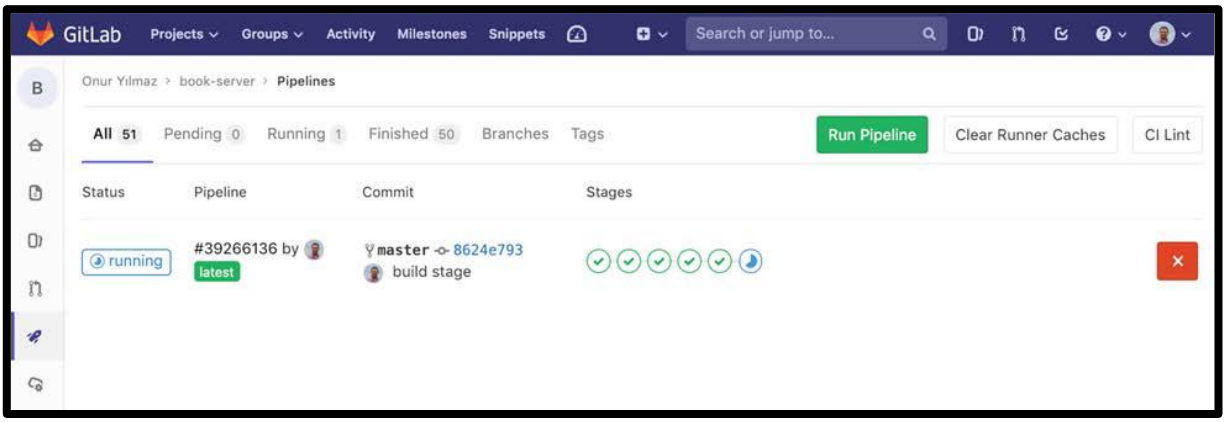
Cloud-Native Continuous Integration and Delivery

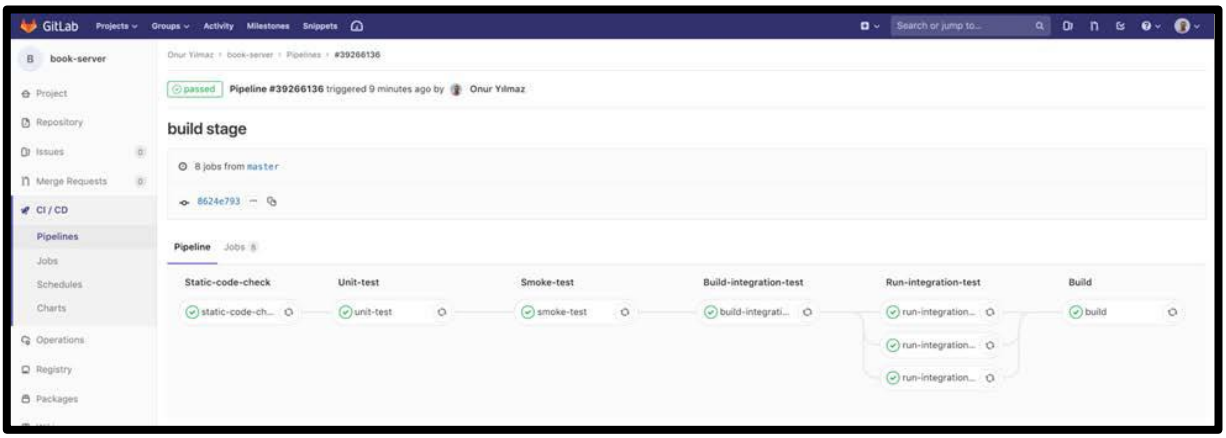
Activity 1: Building a CI Pipeline for Cloud-Native Microservices

You have been tasked with creating a CI pipeline for a book-server application that is designed to be built and tested with containers.

To complete this activity, all previous exercises across all lessons will need to be completed. Use the test and build functions from the previous exercises with the help of GitLab CI/CD pipelines. Once completed, you should have a complete pipeline, starting with static code analysis, passing all tests, and finally building the production-ready container. The pipeline stages and their statuses should be checked from the GitLab web interface:



The pipeline should be green upon successful completion of the activity.



Perform the following steps to complete this activity:

1. Fork the book-server code to your GitLab project from the **`book-server`** repository (https://gitlab.com/TrainingByPackt/book-server).

2. Delete the existing code in the **`.gitlab-ci.yml`** file. We will be creating all of the stages in this file in the following steps. You should already have the test and build Dockerfiles, along with the source code in the repository once you fork it to your own namespace.

3. Define the following stages in the **`.gitlab-ci.yml`** file to run in Docker-in-Docker, namely the **`docker:dind`** service: static-code-check, unit-test, smoketest, build-integration-test, run-integration-test, and build.

4. Create the **`static-code`** check stage.

5. Create the **`unit-test`** stage.

6. Create the **`smoke-test`** stage.

7. Create the **`build-integration-test`** check stage.

8. Create the **`run-integration-test`** check stage.

9. Create the **`build`** stage.

10. Commit the **`.gitlab-ci.yml`** file to the repository.

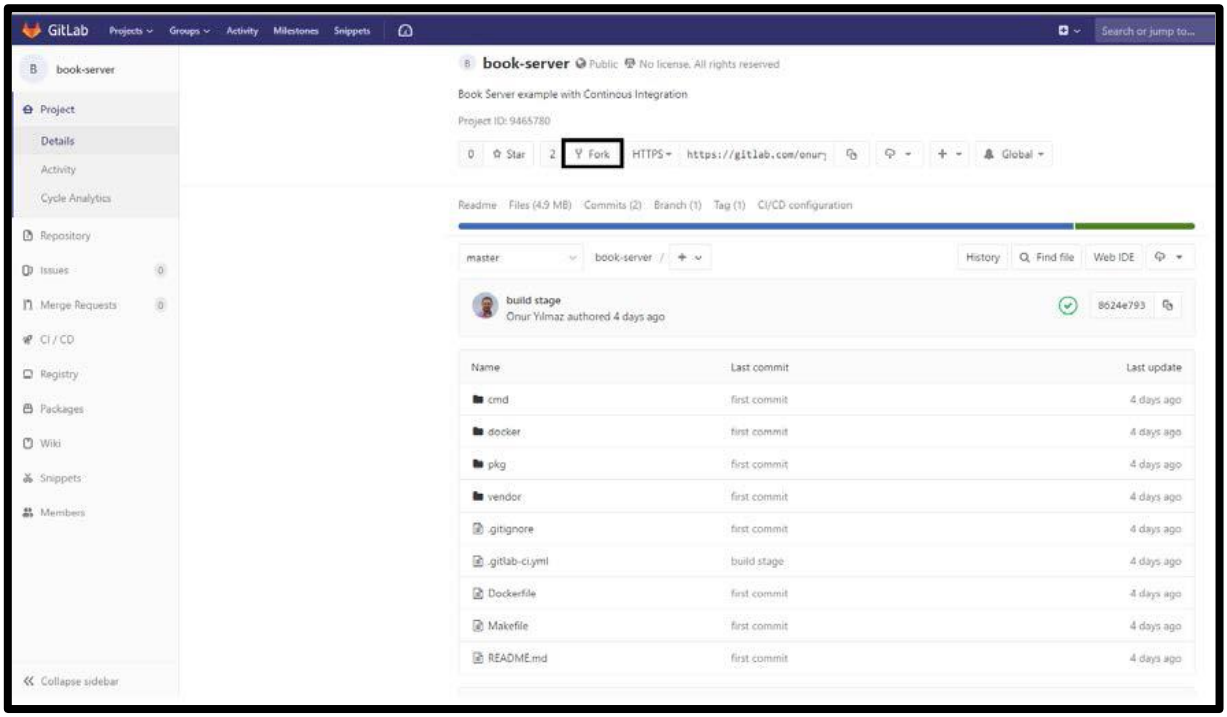
11. Click the **CI/CD** tab on the GitLab interface and then click on the **Run Pipeline** tab.

12. Click on **Create pipeline** to build the CI/CD pipeline.

Solution: Building a CI Pipeline for Cloud-Native Microservices

Execute the following steps to complete the exercise:

1. Fork the book-server code to your GitLab project from the **`book-server`** repository (https://gitlab.com/onuryilmaz/book-server) by clicking the **Fork** button:



Forking the repository on GitLab

2. Delete the existing code in the **`.gitlab-ci.yml`** file. We will be creating all of the stages in this file in the following steps. You should already have the test and build Dockerfiles, along with the source code in the repository once you fork it to your own namespace.
3. Define the stages in the following order in the **`.gitlab-ci.yml`** file to run within Docker-in-Docker, namely **`docker:dind`** service:

```
image: docker:latest

services:
- docker:dind

stages:
- static-code-check
- unit-test
- smoke-test
- build-integration-test
- run-integration-test
- build
```

4. Create the **`static-code-check`** by typing in the following command in the **`.gitlab-ci.yml`** file:

```
static-code-check:
  stage: static-code-check
  script:
  - docker build --rm -f docker/Dockerfile.static-code-check
```

5. Create the **smoke-test** stage by typing in the following code in the same file, in continuation with the code added in the previous step:

```
smoke-test:
   services:
   - docker:dind
   stage: smoke-test
   script:
   - docker build -f docker/Dockerfile.smoke-test -t $CI_REGISTRY_IMAGE/
smoke-test:$CI_COMMIT_SHA .
   - docker run -d -p 5432:5432 --name postgres postgres
   - docker run --rm --link postgres:postgres gesellix/wait-for
postgres:5432
   - docker run -e DATABASE="postgresql://postgres:postgres@
postgres:5432/postgres?sslmode=disable" --link postgres $CI_REGISTRY_
IMAGE/smoke-test:$CI_COMMIT_SHA
```

6. Create the **unit-test** stage by typing in the following code in the same file, in continuation with the code added in the previous step:

```
unit-test:
   stage: unit-test
   script:
   - docker build --rm -f docker/Dockerfile.unit-test
```

7. Create the **build-integration-test** stage by typing in the following code in the same file, in continuation with the code added in the previous step:

```
build-integration-test:
   stage: build-integration-test
   script:
   - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN $CI_REGISTRY
   - docker build -f docker/Dockerfile.integration-test -t $CI_REGISTRY_
IMAGE/integration-test:$CI_COMMIT_SHA
   - docker push $CI_REGISTRY_IMAGE/integration-test:$CI_COMMIT_SHA
```

8. Create a **run-integration** stage with three parallel jobs for running the tests against MySQL, PostgreSQL, and MSSQL by typing the following code in the same file, in continuation with the code mentioned in the previous step:

```
run-integration-test-postgresql:
   services:
   - docker:dind
   stage: run-integration-test
   script:
   - docker run -d -p 5432:5432 --name postgres postgres
   - docker run --rm --link postgres:postgres gesellix/wait-for
   postgres:5432

      - docker run -e DATABASE="postgresql://postgres:postgres@
   postgres:5432/postgres?sslmode=disable" --link postgres $CI_REGISTRY_
   IMAGE/integration-test:$CI_COMMIT_SHA

   run-integration-test-mysql:
     services:
```

```
  - docker:dind
  stage: run-integration-test
  script:
  - docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=password -e
MYSQL_DATABASE=default --name mysql mysql
  - docker run --rm --link mysql:mysql gesellix/wait-for mysql:3306 -t
30
  - docker run -e DATABASE="mysql://root:password@mysql:3306/default"
--link mysql $CI_REGISTRY_IMAGE/integration-test:$CI_COMMIT_SHA

run-integration-test-mssql:
  services:
  - docker:dind
  stage: run-integration-test
  script:
  - docker run -d -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=Password!" -p
1433:1433 --name mssql mcr.microsoft.com/mssql/server:2017-latest
  - docker run --rm --link mssql:mssql gesellix/wait-for mssql:1433
  - docker run -e DATABASE="mssql://sa:Password!@mssql:1433" --link
mssql $CI_REGISTRY_IMAGE/integration-test:$CI_COMMIT_SHA
```

9. Create a final **`build`** stage to create a production-ready container image by typing in the following code:

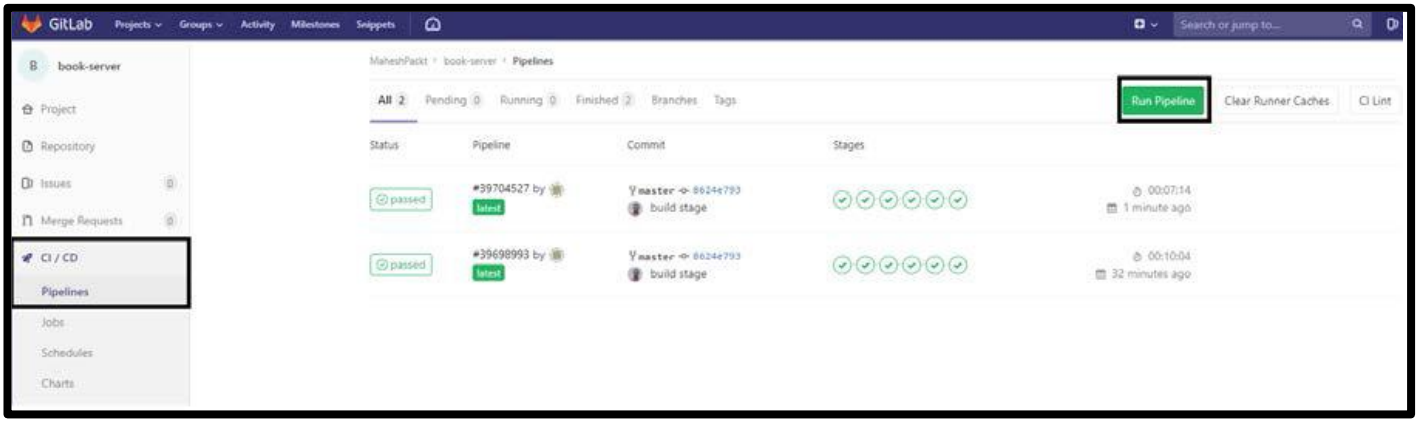
```
build:
  stage: build
  script:
  - docker build --target production -t $CI_REGISTRY_IMAGE:$CI_COMMIT_
SHA .
```

Once done, we will now commit the file to create and run the pipeline.

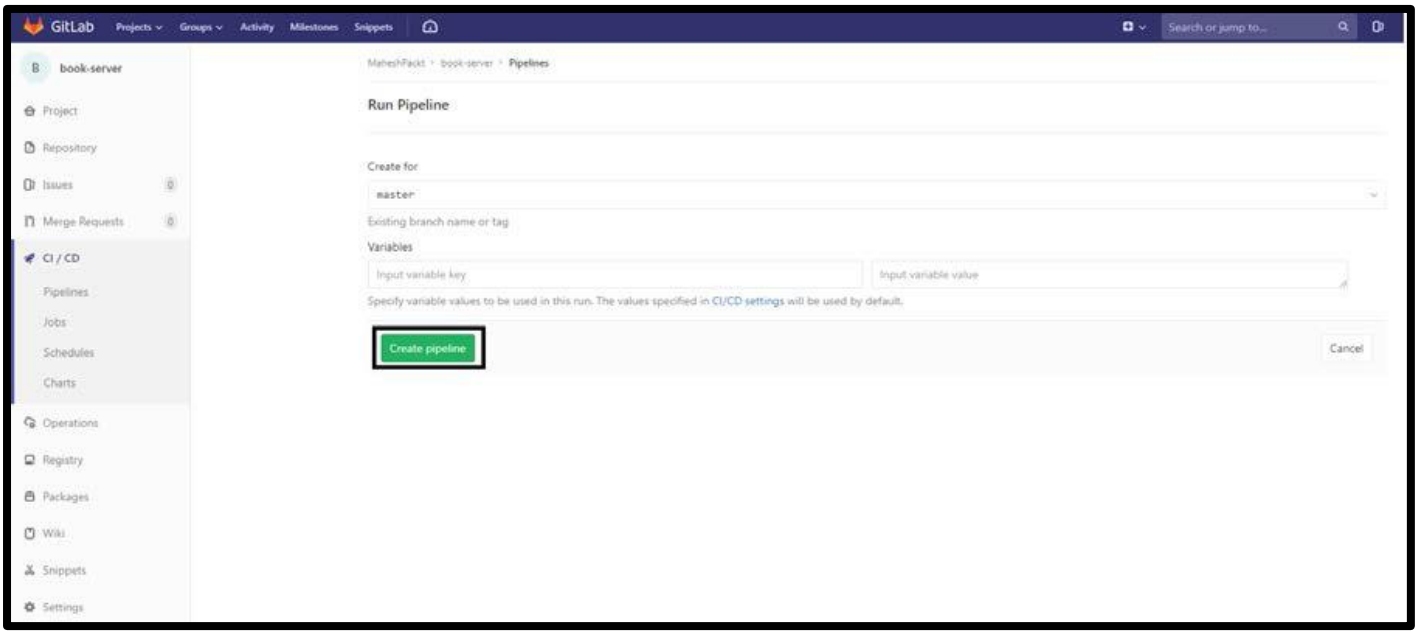
10. Commit the **`.gitlab-ci.yml`** file to the repository by running the following commands locally:

```
git add .gitlab-ci.yml
git commit -m "ci pipeline"
git push origin master
```

11. Click on the **CI/CD** tab on the GitLab interface and then click on the **Run Pipeline** tab, as shown in the following screenshot:

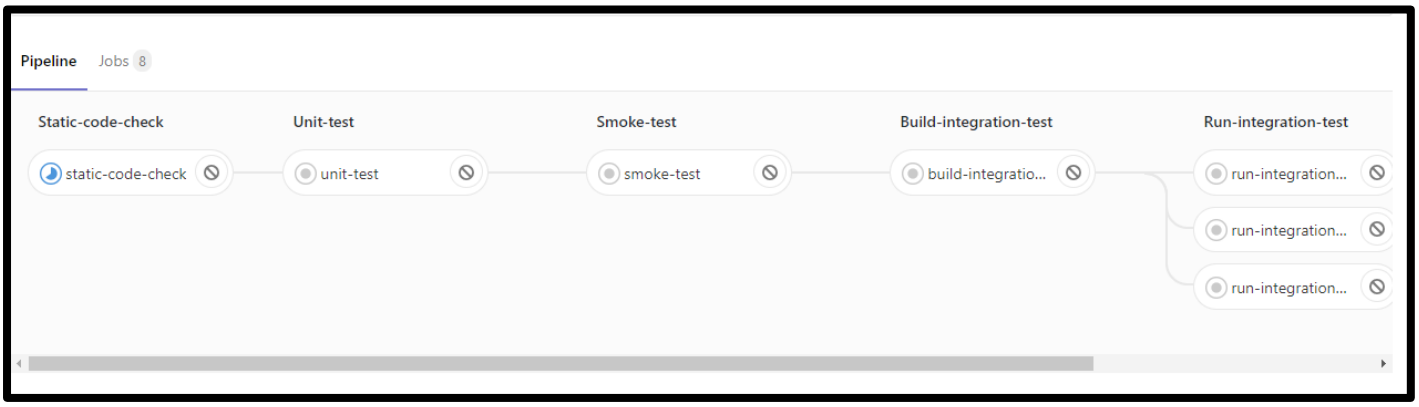


Page for running a pipeline



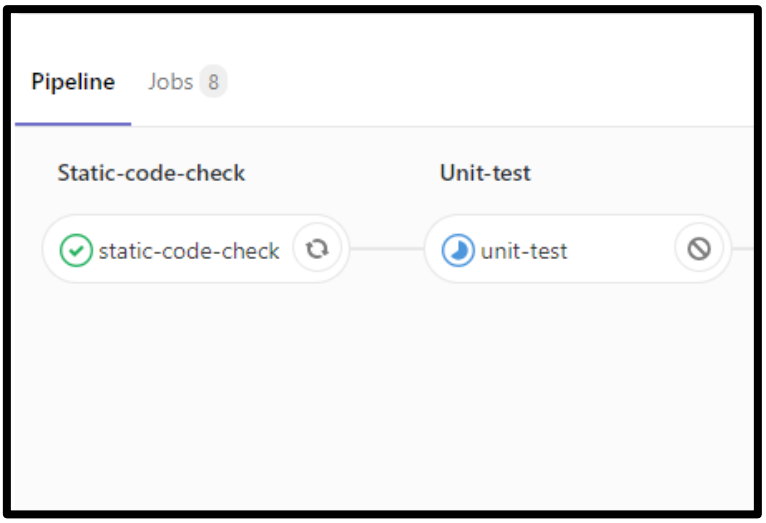
Page for creating a pipeline

12. Click on **Create pipeline**, as shown in the previous screenshot. This page navigates to the pipeline view of the jobs. You can see in the following screenshot that all of the stages are presented in the form of a pipeline, with their live status displayed:



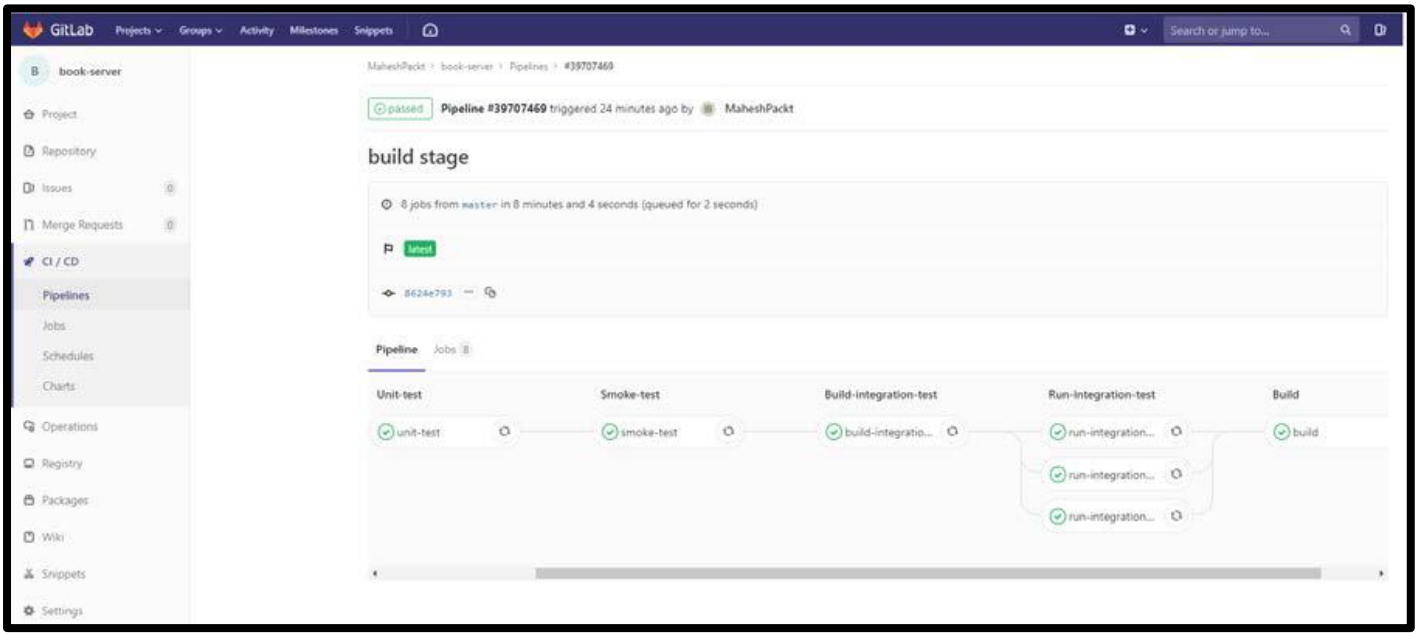
Pipeline view of the stages

As each stage is successfully completed, the status is updated, and a green tick is visible, as shown in the following screenshot:



Live status update

Once all of the stages are successfully completed, the entire pipeline can be considered to be successfully built, as shown in the following screenshot:



A successful pipeline is obtained once all the stages are passed

Note

A pipeline solution is already available in the root folder of book-server in the `.gitlab-ci.yml` file, which can be found at: https://gitlab.com/TrainingByPackt/book-server/blob/master/.gitlab-ci.yml . The complete code for this activity can be found here: https://bit.ly/2PNhuNV .