

GitLab CI/CD Documentation

1. Introduction [↗](#)

GitLab CI/CD is an integrated tool within GitLab that automates the software development lifecycle. It simplifies and accelerates the processes of building, testing, and deploying applications. This documentation covers the essentials of GitLab CI/CD, comparing it with Jenkins, and provides a step-by-step guide to getting started.

2. GitLab CI/CD vs. Jenkins [↗](#)

2.1 Overview [↗](#)

GitLab CI/CD and **Jenkins** are popular CI/CD tools that automate software development workflows. Both have their strengths and cater to different needs:

- **GitLab CI/CD:** An integrated part of GitLab, providing a seamless experience from version control to deployment. It offers built-in CI/CD features, making it easier to set up and manage pipelines directly within the GitLab interface.
- **Jenkins:** A standalone open-source automation server that supports a wide range of plugins to extend its capabilities. It requires more configuration and setup compared to GitLab CI/CD but offers greater flexibility and customization options.

2.2 Key Comparisons [↗](#)

Feature	GitLab CI/CD	Jenkins
Integration	Built into GitLab	Standalone with various integrations
Setup	Easier setup within GitLab	Requires separate setup and configuration
User Interface	Integrated UI with GitLab	Separate UI with extensive plugin options
Configuration	<code>.gitlab-ci.yml</code> file	Jenkinsfile and plugin configurations
Scalability	Scales with GitLab	Highly scalable with plugins and agents
Pipeline Visualization	Visual pipeline graphs and status	Customizable dashboard with plugins
Pricing	Included in GitLab (free and paid plans)	Free, with costs for plugins and infrastructure

3. Getting Started with GitLab CI/CD [↗](#)

3.1 Prerequisites [↗](#)

1. **GitLab Account:** Ensure you have a GitLab account and access to a project repository.
2. **GitLab Runner:** Install and register GitLab Runners to execute jobs. [GitLab Runner Installation Guide](#).

What is a GitLab Runner?

A GitLab Runner is an application that works with GitLab CI/CD to run jobs in a pipeline. It processes jobs by executing scripts defined in `.gitlab-ci.yml`. Runners can be configured to run on different platforms and are essential for automating the build, test, and deployment processes of your project.

Utility of a GitLab Runner

1. Execution of CI/CD Pipelines:

- Runners execute the various stages of your CI/CD pipeline, such as building the application, running tests, and deploying code.

2. Environment Isolation:

- Runners often use Docker containers to provide isolated environments for each job, ensuring that jobs run in clean, reproducible environments without interference from previous jobs.

3. Load Distribution:

- Runners distribute the workload across multiple machines, which helps to speed up the CI/CD process and manage resources more effectively.

4. Cross-Platform Support:

- Runners can be configured to run on different operating systems (Windows, Linux, macOS), which is useful for testing and building applications in various environments.

Types of GitLab Runners

1. Shared Runners:

- These runners are available to all projects in a GitLab instance. They are managed by the GitLab administrator and are useful for general purposes.

2. Specific Runners:

- These are dedicated to a specific project or group. They can be configured to meet the particular needs of the project, such as using certain tools or libraries.

3. Group Runners:

- These runners are shared across all projects in a group. They allow centralized runner management for multiple projects within a group.

Workflow with GitLab Runners

1. Triggering a Pipeline:

- A pipeline is triggered by an event, such as a commit or a merge request.

2. Job Assignment:

- GitLab assigns jobs to available runners. If you have configured tags, GitLab uses those tags to match jobs with appropriate runners.

3. Job Execution:

- The runner fetches the code, runs the scripts defined in `.gitlab-ci.yml`, and sends back the results (console logs, artifacts, etc.) to GitLab.

4. Results Reporting:

- GitLab displays the results of each job in the UI, allowing developers to see successes or failures and take appropriate actions.

Importance of GitLab Runners

- **Automation:** Runners automate the entire software development lifecycle, reducing manual errors and increasing efficiency.
- **Reproducibility:** By using containers and isolated environments, runners ensure that builds and tests are consistent and reproducible.
- **Continuous Integration:** Runners facilitate continuous integration by automatically testing and validating each code change.
- **Continuous Deployment:** Runners also enable continuous deployment by automatically deploying applications once tests pass, ensuring quick and reliable software delivery.

Generating a Registration Token

1. Log in to GitLab
2. Navigate to Project : the Lumen Microservice:
 - For a specific project:
 - i. Go to the project for which I want to register the runner.
 - ii. Navigate to **Settings > CI/CD**.
 - iii. Expand the **Runners** section. I found a **Registration Token**

```
C:\GitLab-Runner>.gitlab-runner-windows-386 register
Runtime platform arch=386 os=windows pid=18600 revision=9882d9c7 version=17.2.1
Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.u-cloudsolutions.xyz
Enter the registration token:
GR1348941593R8NP-Kyk5ZbXhExyk
Enter a description for the runner:
[DESKTOP-03FAKBL]: emna
Enter tags for the runner (comma-separated):

Enter optional maintenance note for the runner:

WARNING: Support for registration tokens and runner parameters in the 'register' command has been deprecated in GitLab R
runner 15.6 and will be replaced with support for authentication tokens. For more information, see https://docs.gitlab.co
m/ee/ci/runners/new_creation_workflow
Registering runner... succeeded runner=GR1348941593R8NP-
Enter an executor: custom, shell, parallels, virtualbox, docker, instance, ssh, docker-windows, docker+machine, kubernete
s, docker-autoscaler:
shell
Runner registered successfully. Feel free to start it, but if it's running already the config should be automatically re
loaded!

Configuration (with the authentication token) was saved in "C:\\GitLab-Runner\\config.toml"

C:\\GitLab-Runner>
```

```
.gitlab-runner-windows-386 install
```

```
.gitlab-runner-windows-386 start
```

and here's the service of gitlab runner is running



```
C:\GitLab-Runner>sc query gitlab-runner

SERVICE_NAME: gitlab-runner
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE                : 4   RUNNING
                        (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE       : 0   (0x0)
        SERVICE_EXIT_CODE   : 0   (0x0)
        CHECKPOINT           : 0x0
        WAIT_HINT            : 0x0

C:\GitLab-Runner>
```

and here it's connected :

Assigned project runners

 #8 (HyTgbZiCk) 
emna

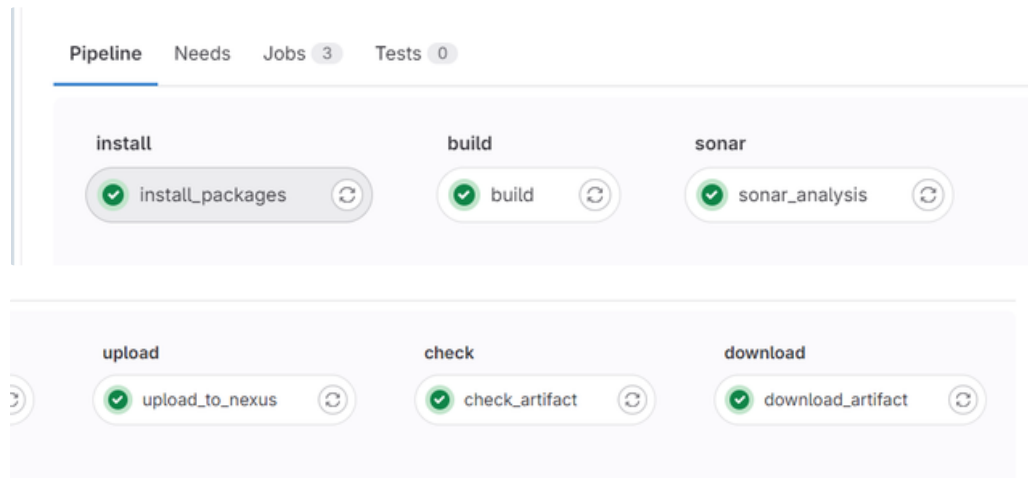
  Remove runner

GitLab CI/CD Pipeline Configuration

1. Stages

The `.gitlab-ci.yml` file is organized into stages. Each stage contains jobs that are executed in a specific order. The stages defined are:

- `install`
- `build`
- `sonar`



2. Stage: Install Packages [↗](#)

Purpose: This stage is responsible for installing necessary dependencies required for your project.

Job: `install_packages`

Configuration:

```
1 install_packages:
2   stage: install
3   script:
4     - echo "Installing packages..."
5     - composer install
6   tags:
7     - shell
8
```

- **Stage:** Specifies the pipeline stage (`install`).
- **Script:** Commands to be executed for the job.
- **Tags:** Identifies the GitLab Runner with the `shell` tag that should pick this job.

3. Stage: Build [↗](#)

Purpose: This stage is used to build or prepare your project for analysis and deployment.

Job: `build`

Configuration:

```
1 build:
2   stage: build
3   script:
4     - echo "This is a Lumen project. Build stage executed."
5   tags:
6     - shell
7
```

- **Stage:** Specifies the pipeline stage (`build`).

- **Script:** Commands to be executed for the job.
- **Tags:** Identifies the GitLab Runner with the `shell` tag that should pick this job.

SonarQube Stage Configuration

1. GitLab Runner Registration and Setup

You registered a GitLab Runner with the following key points:

- **GitLab Runner Registration:**

```
1 .\gitlab-runner-windows-386 register
```

This command is used to register a new runner with your GitLab instance.

- **GitLab Instance URL and Registration Token:**
 - **GitLab Instance URL:** `https://gitlab.u-cloudsolutions.xyz`
 - **Registration Token:** You provided this token during the registration process.
- **Description and Tags:**
 - **Description:** `sonar-runner`
 - **Tags:** `sonar-runner_emna`
- **Executor Choice:**

```
1 docker
```

You chose the Docker executor, which allows the runner to execute jobs within Docker containers.

- **Default Docker Image:**

```
1 sonarsource/sonar-scanner-cli:latest
```

```
C:\GitLab-Runner>.\gitlab-runner-windows-386 register
Runtime platform arch=386 os=windows pid=16496 revision=9882d9c7 version=17.2.1
Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.u-cloudsolutions.xyz
Enter the registration token:
GR1348941593R8NP-Kyk5zNxEyxk
Enter a description for the runner:
[DESKTOP-03FAKBL]: sonar-runner
Enter tags for the runner (comma-separated):
sonar-runner_emna
Enter optional maintenance note for the runner:

WARNING: Support for registration tokens and runner parameters in the 'register' command has been deprecated in GitLab R
unner 15.6 and will be replaced with support for authentication tokens. For more information, see https://docs.gitlab.co
m/ee/ci/runners/new_creation_workflow
Registering runner... succeeded runner=GR1348941593R8NP-
Enter an executor: custom, shell, ssh, parallels, virtualbox, instance, docker, docker-windows, docker+machine, kubernet
es, docker-autoscaler:
docker
Enter the default Docker image (for example, ruby:2.7):
sonarsource/sonar-scanner-cli:latest
Runner registered successfully. Feel free to start it, but if it's running already the config should be automatically re
loaded!
```

Steps Taken:

1. **Registering the Runner:** You registered the GitLab Runner and associated it with the `sonar-runner_emna` tag. This tag is used to identify this runner for specific jobs in your pipeline.
2. **Docker Image Configuration:** The runner is configured to use the Docker executor with the `sonarsource/sonar-scanner-cli:latest` image. This image contains the necessary tools to perform SonarQube analysis.

3. **Runner Configuration File:** After registration, the runner's configuration was saved in `C:\GitLab-Runner\config.toml`. This file contains details about the runner's settings, including the Docker image and tags.

Project runners

These runners are assigned to this project.

[New project runner](#) 

Assigned project runners

 #10 (HavfErFCv) 

  [Remove runner](#)

sonar-runner

[sonar-runner_emna](#)

 #8 (HyTgbZiCk) 

  [Remove runner](#)

emna

[shell](#)

2. SonarQube Analysis Stage in `.gitlab-ci.yml`

Configuration:

```
1 sonar_analysis:
2   stage: sonar
3   script:
4     - echo "Running SonarQube analysis..."
5     - sonar-scanner -Dsonar.projectKey=dev-pipeline-lumen -Dsonar.projectName=dev-pipeline-lumen -Dsonar.sources=
6   tags:
7     - sonar-runner_emna
8
```

Explanation:

1. Stage:

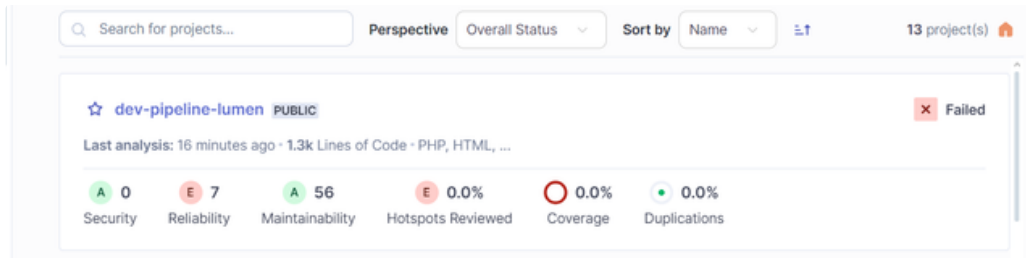
- The `sonar_analysis` job is part of the `sonar` stage, which is specifically set up for SonarQube analysis.

2. Script:

- **Echo Statement:** Outputs a message indicating that SonarQube analysis is starting.
- **SonarQube Scanner Command:**
 - `sonar-scanner` : Command to run SonarQube analysis.
 - **Parameters:**
 - `-Dsonar.projectKey` : Unique identifier for the project in SonarQube.
 - `-Dsonar.projectName` : Display name of the project.
 - `-Dsonar.sources` : Directory containing source code to analyze.
 - `-Dsonar.host.url` : URL of the SonarQube server.
 - `-Dsonar.login` : Authentication token for accessing SonarQube.
 - `-Dsonar.exclusions` : Specifies files or directories to exclude from analysis.

3. Tags:

- `sonar-runner_emna` : Ensures that this job is picked up by the runner registered with this tag.



Package Artifact [↗](#)

In this stage, the project is packaged into a ZIP file. This is a critical step before uploading the artifact to Nexus or any other repository.

Configuration:

- **Stage Name:** `package`
- **Purpose:** To create a compressed archive of the project files.
- **Script:**

```
1 package_project:
2   stage: package
3   script:
4     - echo "Packaging project..."
5     - mkdir -p artifacts
6     - zip -r artifacts/project.zip .
7   artifacts:
8     paths:
9       - artifacts/project.zip
10  tags:
11    - shell
```

Explanation:

1. **Create Artifacts Directory:** The `mkdir -p artifacts` command ensures that the `artifacts` directory exists where the ZIP file will be stored.
2. **Create ZIP Archive:** The `zip -r artifacts/project.zip .` command creates a ZIP file named `project.zip` that includes all files in the project directory.
3. **Artifacts Path:** The `artifacts` section specifies that the created ZIP file should be kept as an artifact, making it available for subsequent stages like upload.

Upload to Nexus: [↗](#)

The `upload_to_nexus` stage in the GitLab CI/CD pipeline handles the uploading of the packaged artifact (`project.zip`) to the Nexus repository. This stage ensures that the build artifacts are stored in a central repository for versioning and further use in deployment pipelines.

Pipeline Configuration:

1. **Variables Configuration:** The following variables are defined for the `upload_to_nexus` stage:
 - `NEXUS_URL` : The base URL of the Nexus repository.
 - `NEXUS_REPOSITORY` : The target repository in Nexus where artifacts will be uploaded.
 - `MAVEN_GROUP_ID` : The Maven group ID of the project, which is used to structure the repository path.
 - `MAVEN_ARTIFACT_ID` : The Maven artifact ID of the project.

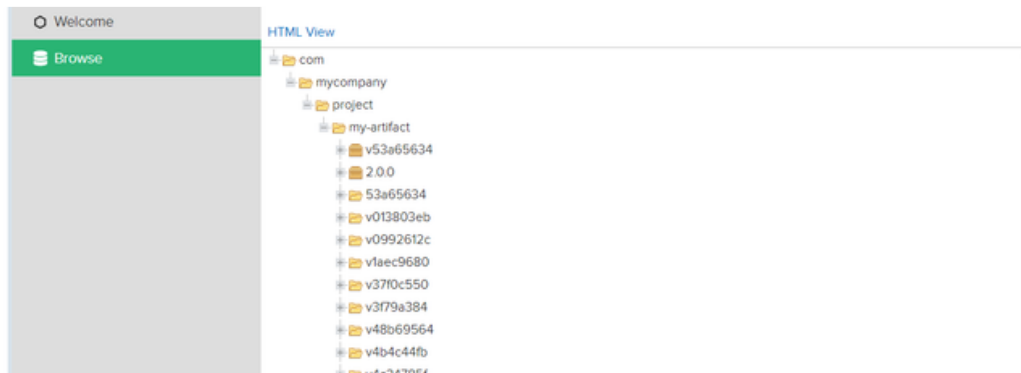
YAML Configuration:

Here is the YAML configuration for the `upload_to_nexus` stage:

```
1 upload_to_nexus:
2   stage: upload
3   variables:
4     NEXUS_URL: 'https://nexus.u-cloudsolutions.xyz'
5     NEXUS_REPOSITORY: 'student-repository'
6     MAVEN_GROUP_ID: 'com.gitlab.project'
7     MAVEN_ARTIFACT_ID: 'my-artifact'
8   script:
9     - echo "Uploading artifacts/project.zip to Nexus..."
10    - |
11      CURL_CMD="curl -u ${NEXUS_USERNAME}:${NEXUS_PASSWORD} --upload-file artifacts/project.zip"
12      UPLOAD_URL="${NEXUS_URL}/repository/${NEXUS_REPOSITORY}/${(echo ${MAVEN_GROUP_ID} | tr '.' '/')}/${MAVEN_ARTIFACT_ID}"
13      echo $CURL_CMD $UPLOAD_URL
14      $CURL_CMD $UPLOAD_URL
15    - echo "Upload complete with commit ID ${CI_COMMIT_SHA} and tag ${CI_COMMIT_REF_NAME}"
16   tags:
17     - shell
18
19
```

Notes:

- The `MAVEN_GROUP_ID` is transformed to replace dots with slashes to fit the Nexus repository path structure.
- The `curl` command uploads the `project.zip` file to the constructed URL using the provided Nexus credentials.
- The stage ensures the proper handling and storage of build artifacts in the Nexus repository, facilitating artifact management and version control.



Stage: Check Artifact in Nexus [🔗](#)

Purpose: Checks if a specific artifact exists in the Nexus Repository by making a request and inspecting the HTTP response code.

Script:

```
1 check_artifact:
2   stage: check
3   script:
4     - echo "Checking if artifact exists in Nexus..."
5     - |
6       # Replace dots with slashes in MAVEN_GROUP_ID
```



```

7     $MAVEN_GROUP_ID_PATH = $env:MAVEN_GROUP_ID -replace '\.', '/'
8
9     # Use the parameter ARTIFACT_VERSION for versioning
10    $VERSION_TAG = $env:ARTIFACT_VERSION
11
12    # Construct the Nexus URL to check the artifact
13    $NEXUS_CHECK_URL = "${env:NEXUS_URL}/repository/${env:NEXUS_REPOSITORY}/${MAVEN_GROUP_ID_PATH}/${env:MAVEN
14
15    # Print URL for debugging
16    echo "Checking URL: $NEXUS_CHECK_URL"
17
18    # Check if the artifact is available using curl
19    $RESPONSE_CODE = curl -s -o /dev/null -w "%{http_code}" -u "$env:NEXUS_USERNAME:$env:NEXUS_PASSWORD" "$NEX
20
21    if ($RESPONSE_CODE -eq "200") {
22        echo "Artifact found in Nexus."
23    } else {
24        echo "Artifact not found. HTTP response code: $RESPONSE_CODE"
25    }
26    tags:
27      - shell
28    rules:
29      - if: '$ARTIFACT_VERSION'
30        when: always

```

Explanation:

- **Replace Dots with Slashes:** Converts Maven group ID into a path format.
- **VERSION_TAG:** Fetches the version tag from the pipeline environment variables.
- **NEXUS_CHECK_URL:** Constructs the URL to check if the artifact exists.
- **Curl Command:** Performs a request to check the HTTP response code, indicating artifact presence.


Setting VERSION_TAG Manually: To manually set the `ARTIFACT_VERSION` (which represents the `VERSION_TAG`) when running the pipeline:

1. **Go to GitLab:**
 - Navigate to your project in GitLab.
2. **Navigate to CI/CD Pipelines:**
 - Click on `CI/CD` in the left sidebar.
 - Select `Pipelines`.
3. **Trigger a New Pipeline:**
 - Click on the `Run pipeline` or `Create pipeline` button.
4. **Add Pipeline Variables:**
 - In the "Variables" section, add a new variable:
 - **Key:** `ARTIFACT_VERSION`
 - **Value:** Specify the version tag you want to check, for example, `v92a8e2fc` in our case.
5. **Run the Pipeline:**
 - Click `Create pipeline` or `Run pipeline` to start the pipeline with the specified variables.

Run pipeline

Run for branch name or tag

Variables

Variable	ARTIFACT_VERSION	v92a8e2fc	
Variable	Input variable key	Input variable value	

Specify variable values to be used in this run. The variables specified in the configuration file as well as [CI/CD settings](#) are used by default. Variables specified here are **expanded** and not **masked**.

Download Artifact

Purpose: To download a specific version of the artifact from Nexus and save it to a defined path.

Configuration:

Stage Name: download

Purpose: To retrieve the packaged artifact from the Nexus repository for further use or verification.

Script:

```

1 download_artifact:
2   stage: download
3   script:
4     - echo "Downloading artifact from Nexus..."
5     - |
6       # Convert Maven group ID from dots to slashes
7       $MAVEN_GROUP_ID_PATH = $env:MAVEN_GROUP_ID -replace '\.', '/'
8
9       # Retrieve the artifact version
10      $VERSION_TAG = $env:ARTIFACT_VERSION
11
12      # Build the Nexus URL for downloading the artifact
13      $NEXUS_DOWNLOAD_URL = "${env:NEXUS_URL}/repository/${env:NEXUS_REPOSITORY}/${MAVEN_GROUP_ID_PATH}/${env:MA
14
15      # Print URL for debugging
16      echo "Downloading from URL: $NEXUS_DOWNLOAD_URL"
17
18      # Download the artifact using curl
19      curl -o "$env:DOWNLOAD_PATH" "$NEXUS_DOWNLOAD_URL"
20
21      echo "Artifact downloaded to ${env:DOWNLOAD_PATH}"
22  tags:
23    - shell
24  rules:
25    - if: '$ARTIFACT_VERSION'
26      when: always
27

```

Explanation:

1. Format Maven Group ID:

- Converts `MAVEN_GROUP_ID` from dots to slashes to match Nexus URL format.

2. Retrieve Version Tag:

- Uses `ARTIFACT_VERSION` to specify which version of the artifact to download.

3. Construct URL:

- Builds the Nexus URL for the artifact using Nexus base URL, repository name, Maven group ID path, artifact ID, and version tag.

4. Download Artifact:

- Uses `curl` to download the artifact from the constructed URL and saves it to `DOWNLOAD_PATH`.

5. Print Download Path:

- Confirms where the artifact was saved for verification.

Unzip Stage in CI/CD Pipeline

Stage Name: `unzip_artifact`

Purpose: Extract the contents of a ZIP file for further processing.

Configuration

```
1 # Stage: Unzip Artifact
2 unzip_artifact:
3   stage: unzip
4   script:
5     - echo "Unzipping artifact..."
6     - |
7       # Create a directory for unzipped files
8       $EXTRACT_PATH = "artifacts/unzipped"
9       mkdir -p $EXTRACT_PATH
10
11     # Determine which ZIP file to use
12     if ($env:ARTIFACT_VERSION) {
13       $ZIP_FILE = "$env:DOWNLOAD_PATH" # Use downloaded artifact if version is specified
14     } else {
15       $ZIP_FILE = "$env:FALLBACK_ZIP_PATH" # Fallback to packaged artifact
16     }
17
18     # Unzip the chosen artifact
19     unzip "$ZIP_FILE" -d $EXTRACT_PATH
20
21     echo "Artifact extracted to $EXTRACT_PATH"
22   tags:
23     - shell
24   rules:
25     - if: '$ARTIFACT_VERSION || $CI_COMMIT_REF_NAME'
26       when: always
27
```

Explanation

- **Stage:** Defined as `unzip`.
- **Create Directory:** Ensures `artifacts/unzipped` directory exists.
- **Determine ZIP File:**
 - If `ARTIFACT_VERSION` is set, use the downloaded artifact.
 - Otherwise, use the fallback ZIP file.
- **Unzip Artifact:** Extracts the chosen ZIP file to `artifacts/unzipped`.
- **Tags and Rules:**
 - Tagged with `shell`.
 - Runs if `ARTIFACT_VERSION` or `CI_COMMIT_REF_NAME` is set.

Build Docker Image

Stage Name: `build_image`

Job Name: `build_docker_image`

Purpose: This stage builds the Docker image from the source code in the repository. The Docker image is tagged using the commit ID to ensure that each image is uniquely identified.

Script:

```
1 build_docker_image:
2   stage: build_image
3   script:
4     - echo "Building Docker image..."
5     - |
6       # Define Docker image name and tag using the commit ID
7       $IMAGE_TAG = $env:CI_COMMIT_SHA.Substring(0, 8)
8       $IMAGE_NAME = "u-cloudsolutions/microservice:${IMAGE_TAG}"
9
10      # Build the Docker image
11      docker build -t $IMAGE_NAME .
12
13      echo "Docker image built: $IMAGE_NAME"
14  tags:
15    - shell
16  rules:
17    - if: '$CI_COMMIT_SHA'
18      when: always
19
```

Details:

- **Docker Image Name:** Formed using the environment variable `AA` and the short commit ID.
- **Docker Image Tag:** Consists of the commit ID's first 8 characters.
- **Commands:**
 - `docker build -t $IMAGE_NAME .` builds the Docker image.
 - The image is tagged with the specified name and commit ID.

Configuration:

- Ensure that the Docker daemon has the necessary permissions to build images and access the source code.

```

28 #12 22.16 32/120 [=====>-----] 26%
29 #12 22.28 40/120 [=====>-----] 33%
30 #12 22.40 50/120 [=====>-----] 41%
31 #12 22.64 66/120 [=====>-----] 55%
32 #12 22.74 73/120 [=====>-----] 60%
33 #12 23.04 88/120 [=====>-----] 73%
34 #12 23.26 101/120 [=====>-----] 84%
35 #12 23.49 114/120 [=====>-----] 95%
36 #12 23.58 120/120 [=====] 100%
37 #12 26.01 Generating optimized autoload files
38 #12 27.20 83 packages you are using are looking for funding.
39 #12 27.20 Use the `composer fund` command to find out more!
40 #12 DONE 27.4s
41 #13 exporting to image
42 #13 exporting layers
43 #13 exporting layers 1.0s done
44 #13 writing image sha256:461727cba85f62a6329c5f6835a87143b33256d5423369c1b1f931318edaca12 done
45 #13 naming to docker.io/u-cloudsolutions/microservice:4b4c44fb done
46 #13 DONE 1.0s
47 Docker image built: u-cloudsolutions/microservice:4b4c44fb
48 Cleaning up project directory and file based variables
49 Job succeeded

```

2. Push Docker Image to Nexus [↗](#)

Stage Name: push_image

Job Name: push_docker_image

Purpose: This stage publishes the Docker image built in the previous stage to the Nexus Docker registry. The image is tagged appropriately for the Nexus repository before being pushed.

Script:

```

1 push_docker_image:
2   stage: push_image
3   script:
4     - echo "Publishing Docker image to Nexus..."
5     - |
6       # Define Docker registry URL and image name
7       $DOCKER_REGISTRY_URL = "${env:NEXUS_DOCKER_URL}/${env:NEXUS_DOCKER_REPOSITORY}"
8       $IMAGE_TAG = $env:CI_COMMIT_SHA.Substring(0, 8)
9       $IMAGE_NAME = "u-cloudsolutions/microservice:${IMAGE_TAG}"
10      $IMAGE_TAGGED = "${DOCKER_REGISTRY_URL}/u-cloudsolutions/microservice:${IMAGE_TAG}"
11
12      # Log in to Docker registry securely
13      echo "${env:NEXUS_PASSWORD}" | docker login $env:NEXUS_DOCKER_URL --username $env:NEXUS_USERNAME --password
14
15      # Tag and push Docker image
16      docker tag $IMAGE_NAME $IMAGE_TAGGED
17      docker push $IMAGE_TAGGED
18
19      echo "Docker image $IMAGE_TAGGED published successfully."
20 tags:
21   - shell
22 rules:
23   - if: '$CI_COMMIT_SHA'
24     when: always
25

```

Details:

- **Docker Registry URL:** Defined by the environment variables `NEXUS_DOCKER_URL` and `NEXUS_DOCKER_REPOSITORY`.
- **Docker Image Name:** The name of the Docker image with the tag from the commit ID.
- **Commands:**
 - `docker login` logs in to the Nexus Docker registry.
 - `docker tag` tags the image with the Nexus registry URL.
 - `docker push` pushes the tagged image to the Nexus registry.

Configuration:

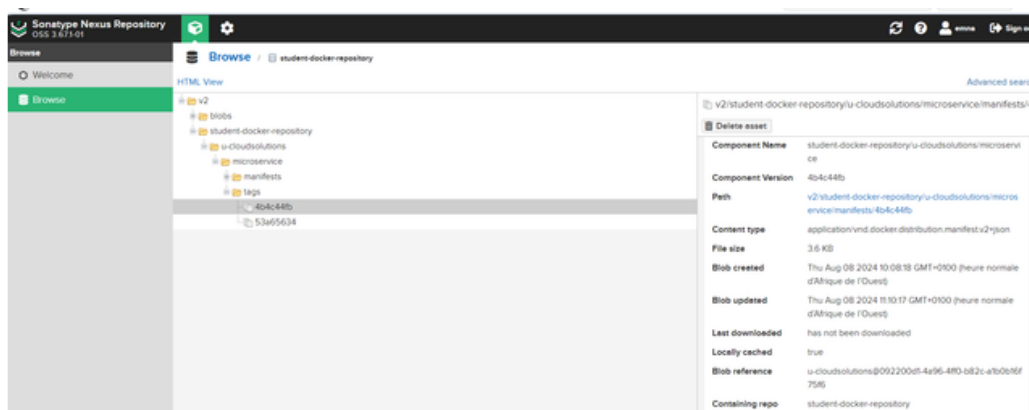
- Ensure that the Nexus Docker registry is accessible from the build environment and configured to accept incoming requests.
- Verify that the Docker registry URL, username, and password are correct and securely managed.

added "u-cloudsolutions.xyz:10001" in the insecure-registries in docker engine

```

56 4e08788c8b71: Waiting
57 66afeec14619: Waiting
58 5f70bf18a086: Layer already exists
59 905c5cb8eeb0: Layer already exists
60 66afeec14619: Layer already exists
61 801813e1b4b4: Layer already exists
62 09f0209dd517: Layer already exists
63 4e6a7aae8a7f: Layer already exists
64 bb2e2b95ad11: Layer already exists
65 f58f003cfd1a: Layer already exists
66 d6ee7296c8e0: Pushed
67 470caf92a2a: Layer already exists
68 0cab796bb000: Layer already exists
69 9908a80d2ba0: Layer already exists
70 1b6fd3ad4ce6: Layer already exists
71 f99b495681ef: Layer already exists
72 200c76d1a9fb: Pushed
73 fd1949c5c73: Pushed
74 4b4c44fb: digest: sha256:636f17cc1ddf94ba535c319cc1b094d3d833e652046ed7ae41b3844f05c70e13 size: 3671
75 Docker image 54.38.185.102:10001/student-docker-repository/u-cloudsolutions/microservice:4b4c44fb published successfully.
76 Cleaning up project directory and file based variables
77 Job succeeded

```



Build JMeter Docker Image [↗](#)

Stage: `build_jmeter_image`

Purpose: Build a Docker image specifically for running JMeter tests.

Script:

```

1 build_jmeter_image:
2   stage: build_jmeter_image
3   script:

```

```

4     - echo "Building Docker image for JMeter..."
5     - |
6     # Build Docker image for JMeter
7     docker build -f tests/Dockerfile.jmeter -t $JMETER_IMAGE_NAME .
8     echo "Docker image for JMeter built: $JMETER_IMAGE_NAME"
9     echo $JMETER_IMAGE_NAME > img.txt
10    tags:
11    - shell
12    artifacts:
13    paths:
14    - img.txt
15    rules:
16    - if: '$CI_COMMIT_SHA'
17      when: always
18

```

Details:

- **Dockerfile Location:** tests/Dockerfile.jmeter
- **Image Name:** \$JMETER_IMAGE_NAME
- **Artifact:** img.txt containing the name of the built JMeter image.

```

35 #1 DONE 0.0s
36 #2 [internal] load metadata for docker.io/library/openjdk:11-jre-slim
37 #2 DONE 1.3s
38 #3 [internal] load .dockerignore
39 #3 transferring context: 2B done
40 #3 DONE 0.0s
41 #4 [1/4] FROM docker.io/library/openjdk:11-jre-slim@sha256:93af7df2308c5141a751c4830e6b6c5717db102b3
42 b31f012ea29d842dc4f2b02
42 #4 DONE 0.0s
43 #5 [internal] load build context
44 #5 transferring context: 66B 0.0s done
45 #5 DONE 0.0s
46 #6 [2/4] RUN apt-get update && apt-get install -y wget unzip && wget https://archive.apache.
47 org/dist/jmeter/binaries/apache-jmeter-5.6.2.zip -P /tmp && unzip /tmp/apache-jmeter-5.6.2.zip
48 -d /opt && rm /tmp/apache-jmeter-5.6.2.zip
47 #6 CACHED
48 #7 [3/4] COPY tests/test_plan.jmx /tests/test_plan.jmx
49 #7 CACHED
50 #8 [4/4] WORKDIR /tests
51 #8 CACHED
52 #9 exporting to image
53 #9 exporting layers done
54 #9 writing image sha256:1717af8bb52aa8064958cb5fb6e7ab72ad508a47d54e89f0d2d4b19c1b095323 done
55 #9 naming to docker.io/library/jmeter-image done
56 #9 DONE 0.0s
57 Docker image for JMeter built: jmeter-image

```

2. Deploy Application Container

Stage: deploy

Purpose: Deploy the Docker container with the application for JMeter testing.

Script:

```

1  deploy_app_container:
2    stage: deploy
3    script:
4      - echo "Deploying Docker container..."
5      - |
6        $env:IMAGE_TAG = (Get-Content image_tag.txt).Trim()
7        $env:APP_CONTAINER_NAME = "my-app-$( $env:IMAGE_TAG )-container"
8        $env:DOCKER_IMAGE_NAME = "u-cloudsolutions/microservice:$( $env:IMAGE_TAG )"
9        Write-Host "Using Docker image: $env:DOCKER_IMAGE_NAME"
10       Write-Host "Container name: $env:APP_CONTAINER_NAME"
11
12       $ErrorActionPreference = "Stop"
13
14       function Cleanup {
15         Write-Host "Cleaning up application container..."
16         if (docker ps -q -f "name=$env:APP_CONTAINER_NAME") {
17           docker stop $env:APP_CONTAINER_NAME | Out-Null
18           docker rm $env:APP_CONTAINER_NAME | Out-Null
19           Write-Host "Application container $env:APP_CONTAINER_NAME has been removed."
20         }
21       }
22
23       try {
24         docker run -d --name $env:APP_CONTAINER_NAME -p 10088:80 $env:DOCKER_IMAGE_NAME
25         Write-Host "Docker container deployed with name: $env:APP_CONTAINER_NAME"
26         Start-Sleep -Seconds 10
27         docker exec $env:APP_CONTAINER_NAME php -S localhost:8000 > $null 2>&1 &
28       } finally {
29         # Cleanup function will be called only if necessary
30       }
31   tags:
32     - shell
33   dependencies:
34     - build_jmeter_image
35   artifacts:
36     paths:
37       - image_tag.txt
38   rules:
39     - if: '$CI_COMMIT_SHA'
40       when: always
41

```

Details:

- **Docker Image:** u-cloudsolutions/microservice:\$IMAGE_TAG
- **Container Name:** my-app-\$IMAGE_TAG-container
- **Ports:** Mapping container port 80 to host port 10088
- **Cleanup:** Function to stop and remove the container if it exists.


```
30 Using Docker Image: 0-CloudSolutions/microservice-f547b6ac
31 Container name: my-app-f547b6ac-container
32 8289185218c7791e9acb9ae674b4c7edb4e969d1a2a574dff1c72fe369f91aff
33 Docker container deployed with name: my-app-f547b6ac-container
34 Id      Name      PSJobTypeName  State      HasMoreData  Location  Command
35 --      ---      -
36 1       Job1      BackgroundJob  Running    True          localhost docker.exe
37 Uploading artifacts for successful job
38 Version: 17.2.1
39 Git revision: 9882d9c7
40 Git branch: 17-2-stable
41 GO version: go1.22.5
42 Built: 2024-07-25T17:34:51+0000
43 OS/Arch: windows/386
44 Uploading artifacts...
45 Runtime platform arch=386 os=windows pid=1504 revision=9882d9c7 version=17.2.1
46 image_tag.txt: found 1 matching artifact files and directories
47 Uploading artifacts as "archive" to coordinator... 201 Created id=25275 responseStatus=201 Created token=64_WeAA
48 Cleaning up project directory and file based variables
49 Job succeeded
```

3. Run JMeter Tests [↗](#)

Stage: `run_jmeter_tests`

Purpose: Execute JMeter tests against the deployed application container and collect results.

Script:

```
1 run_jmeter_tests:
2   stage: run_jmeter_tests
3   script:
4     - echo "Running JMeter tests..."
5     - |
6       $env:IMAGE_TAG = (Get-Content image_tag.txt).Trim()
7       $env:APP_CONTAINER_NAME = "my-app-$( $env:IMAGE_TAG )-container"
8       $env:JMETER_IMAGE_NAME = (Get-Content img.txt).Trim()
9       $env:JMETER_RESULTS_DIR = "$(pwd)/jmeter-results"
10
11     $ErrorActionPreference = "Stop"
12
13     function Cleanup {
14       Write-Host "Cleaning up application container..."
15       if (docker ps -q -f name=$env:APP_CONTAINER_NAME) {
16         docker stop $env:APP_CONTAINER_NAME | Out-Null
17         docker rm $env:APP_CONTAINER_NAME | Out-Null
18       }
19       Write-Host "Application container $env:APP_CONTAINER_NAME has been removed."
20     }
21
22     try {
23       if (-not (Test-Path -Path $env:JMETER_RESULTS_DIR)) {
24         New-Item -ItemType Directory -Path $env:JMETER_RESULTS_DIR
25       }
26
27       docker run --rm `
28         --link $env:APP_CONTAINER_NAME:my-app `
```

```

29     -v "$env:JMETER_RESULTS_DIR:/tests/results" `
30     $env:JMETER_IMAGE_NAME `
31     -n -t /tests/test_plan.jmx -l /tests/results/results.jtl -e -o /tests/results -Jhostname=my-app -Jport
32
33     Write-Host "JMeter tests completed and results are saved to $env:JMETER_RESULTS_DIR"
34
35     Write-Host "Publishing JMeter reports..."
36     Write-Host "JMeter reports published successfully."
37 } catch {
38     Write-Host "An error occurred: $_"
39     exit 1
40 }
41 tags:
42   - shell
43 artifacts:
44   paths:
45     - jmeter-results/
46     - image_tag.txt
47     - img.txt
48   when: always
49 rules:
50   - if: '$CI_COMMIT_SHA'
51     when: always
52

```

Details:

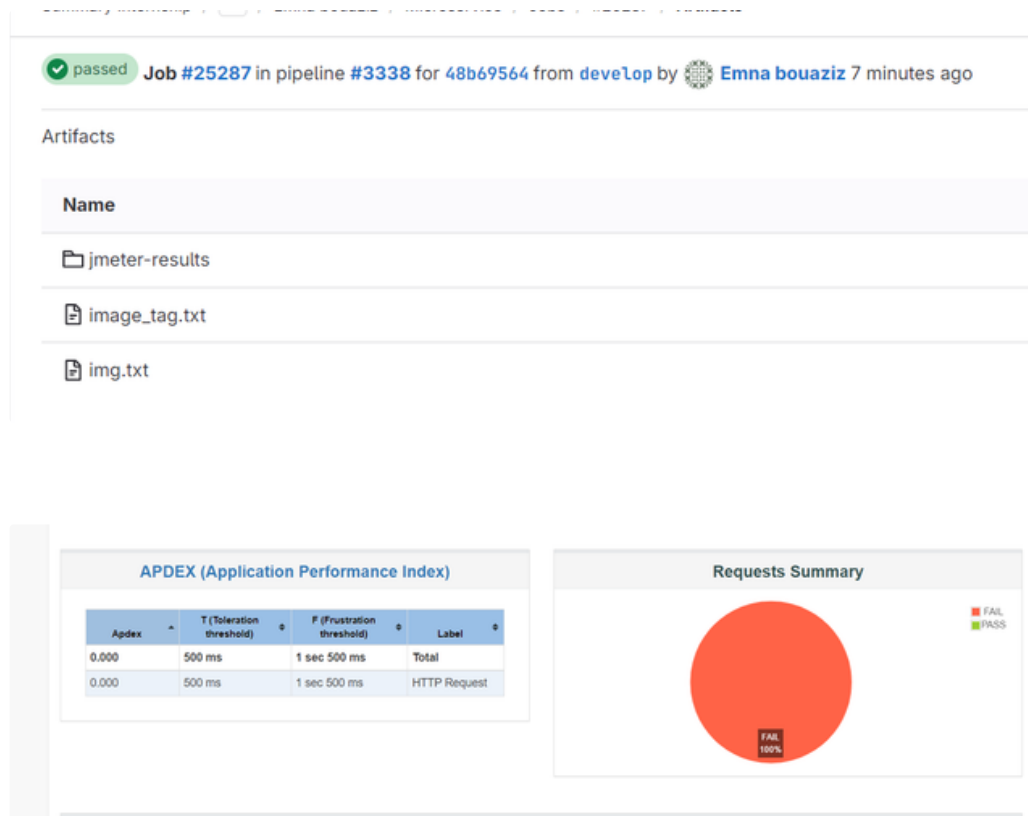
- **JMeter Image:** Obtained from `img.txt`
- **Results Directory:** `$JMETER_RESULTS_DIR`
- **JMeter Command:** Runs JMeter tests using `test_plan.jmx` and stores results in `results.jtl`

```

40 d----      08/08/2024    18:47      jmeter-results
41 WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be r
42   moved in a future release
42 WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be r
43   moved in a future release
43 WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be r
44   moved in a future release
44 WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be r
45   moved in a future release
45 Aug 08, 2024 5:47:13 PM java.util.prefs.FileSystemPreferences$1 run
46 INFO: Created user preferences directory.
47 Creating summariser <summary>
48 Created the tree successfully using /tests/test_plan.jmx
49 Starting standalone test @ 2024 Aug 8 17:47:14 UTC (1723139234647)
50 Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
51 Warning: Nashorn engine is planned to be removed from a future JDK release
52 summary =    300 in 00:00:10 =   30.2/s Avg:    1 Min:    0 Max:   41 Err:   300 (100.00%)
53 Tidying up ...    @ 2024 Aug 8 17:47:25 UTC (1723139245317)
54 ... end of run
55 JMeter tests completed and results are saved to C:\GitLab-Runner\builds\HyTgbZiC\0\summary-internshi
56   p\2024\emna-bouaziz\microservice\jmeter-results
56 Publishing JMeter reports...
57 JMeter reports published successfully.
58 Uploading artifacts for successful job

```

I added the reports as artifacts in gitlab :



4. Cleanup

Stage: cleanup

Purpose: Clean up resources to ensure no leftover containers or artifacts remain.

Script:

```
1 cleanup:
2   stage: cleanup
3   script:
4     - echo "Cleaning up application container..."
5     - |
6       $env:IMAGE_TAG = (Get-Content image_tag.txt).Trim()
7       $env:APP_CONTAINER_NAME = "my-app-$( $env:IMAGE_TAG )-container"
8       Write-Host "Using container name: $env:APP_CONTAINER_NAME"
9
10      # Stop and remove the container using the correct variable
11      docker stop $env:APP_CONTAINER_NAME
12      docker rm $env:APP_CONTAINER_NAME
13
14      Write-Host "Application container $env:APP_CONTAINER_NAME has been removed."
15  tags:
16    - shell
17  artifacts:
18    paths:
19      - image_tag.txt
20  dependencies:
21    - run_jmeter_tests
22  rules:
23    - if: '$CI_COMMIT_SHA'
24      when: always
```

Details:

- **Cleanup Action:** Stops and removes the application container if it exists.
- **Artifact:** `image_tag.txt` to maintain context across stages.

```
28 $ echo "Cleaning up application container..."
29 Cleaning up application container...
30 $ $env:IMAGE_TAG = (Get-Content image_tag.txt).Trim() # collapsed multi-line command
31 Using container name: my-app-f547b6ac-container
32 my-app-f547b6ac-container
33 my-app-f547b6ac-container
34 Application container my-app-f547b6ac-container has been removed.
35 Uploading artifacts for successful job
36 Version:      17.2.1
37 Git revision: 9882d9c7
38 Git branch:   17-2-stable
39 GO version:   go1.22.5
40 Built:        2024-07-25T17:34:51+0000
41 OS/Arch:      windows/386
42 Uploading artifacts...
43 Runtime platform arch=386 os=windows pid=3892 revision=9882d9c7 v
   ersion=17.2.1
44 image_tag.txt: found 1 matching artifact files and directories
45 Uploading artifacts as "archive" to coordinator... 201 Created id=25277 responseStatus=201 Created
   token=64_Yv1Xk
46 Cleaning up project directory and file based variables
```

00:02

00:01