**Use case 1**

Design an end-to-end ci/cd automated pipeline in githubactions (from

Engineer\_check-in to delivery\_to\_customers) for a c/c++ application with

Dockerfile that is targeted to run on windows/macos/linux.

Guidance:

- this is a c++ application which depends on a lot of 3rd party libraries and they

Are added as modules in the application and during the build it has to build all

The modules then it will build the application.

- pipeline to have conditional steps that should run only on a master branch

- arch diagram is mandatory

- adding in unit test case and code coverage metrics will be a bonus

**Flow diagram:**

Github repository

↓

Jenkins ci/cd workflow (jenkinsfile)

↓

Build docker image

↓

Code quality check (agrocd)

↓

Unit tests

↓

If (branch == "master") {

docker build

↓

push image to docker hub

}

Test artifacts

↓

Deployment target

**Suggestions:  
  
 setting up jenkins:**

i’ll install jenkins server on the preferred environment.

I’ll ensure that the necessary plugins installed, such as pipeline and docker pipeline, which are needed for this setup.

**Configuring jenkins credentials:**

I’ll configure the jenkins pipeline needs to access external resources like your git repository or docker hub, set up credentials for these services.

This ensures secure access during the pipeline execution.

**Creating a jenkins pipeline job:**

In the jenkins dashboard, create a new pipeline job.

Configure this job to use the pipeline script from your source control management (scm) system, such as git.

Point the job to the repository where your jenkinsfile is located.

**Writing the jenkins pipeline script (jenkinsfile):**

Create a jenkinsfile in the root of my project's repository.

This file will define the entire ci/cd workflow as code.

I’ll use a declarative or scripted syntax to define the stages and steps of your pipeline.

**Defining stages in the jenkins pipeline:**

Within the jenkinsfile, define stages that represent the different parts of my ci/cd process.

Common stages include checkout (to get the code), build (to compile the application), test (to run tests), dockerbuild (to create the docker image), dockerpush (to push the docker image), and deploy (to deploy to a target environment).

By using the directive to conditionally run statements for the certain stages, such as only building and pushing docker images when changes are on the master branch.  
  
Please refer one of my another project which i did and uploaded in my git hub repo for this step https://github.com/karthikeyanrajan/karthikeyanrajan-capstone-project.git

**Configuring docker build and push:**

By using the docker pipeline syntax within your jenkinsfile to define steps for building and pushing docker images.

The dockerfile in the project directory will be used to build the docker image.

Once built, the image is pushed to a docker registry like docker hub. Make sure to use the appropriate docker credentials secretes are configured in jenkins.

**Integrating code quality checks:**

Integrate code quality checking tools like agrocd into the jenkins pipeline.

Run these checks as a stage in your pipeline to ensure your code meets quality standards before proceeding further.

**implementing unit tests and code coverage:**

Write unit tests for our project c/c++ application using frameworks such as ctest.

Integrate these tests into jenkins pipeline to automatically run them during the test stage.

Optionally, integrate code coverage tools for the code coverage to gather code coverage metrics.

**Artifact generation and deployment:**

As part of your jenkins pipeline, generate artifacts such as compiled binaries or distribution packages.

These artifacts are what will be deployed to your target environment.

Define deployment steps in your pipeline to automate the deployment process, ensuring consistency and reliability.

**Triggering the pipeline:**

Set up triggers for your jenkins pipeline.

We can use the webhooks that notify jenkins of changes in your git repository, or by configuring jenkins to periodically poll your repository for changes.

Or if the jenkins server is in the private repo we can use the smee.io work payload delivery to trigger the pipeline

**Testing and validation:**

Test the jenkins pipeline by making changes to the code repository.

Ensure that the pipeline triggers automatically, builds the application, runs tests, creates docker images, and deploys to the target environment as expected.

Validate that the deployment is successful and the application behaves as intended in the target environment.

**Monitoring**

Continuously monitor and improve your jenkins pipeline.

By using the cadvisor we can monitor the docker file which deployed on the server.