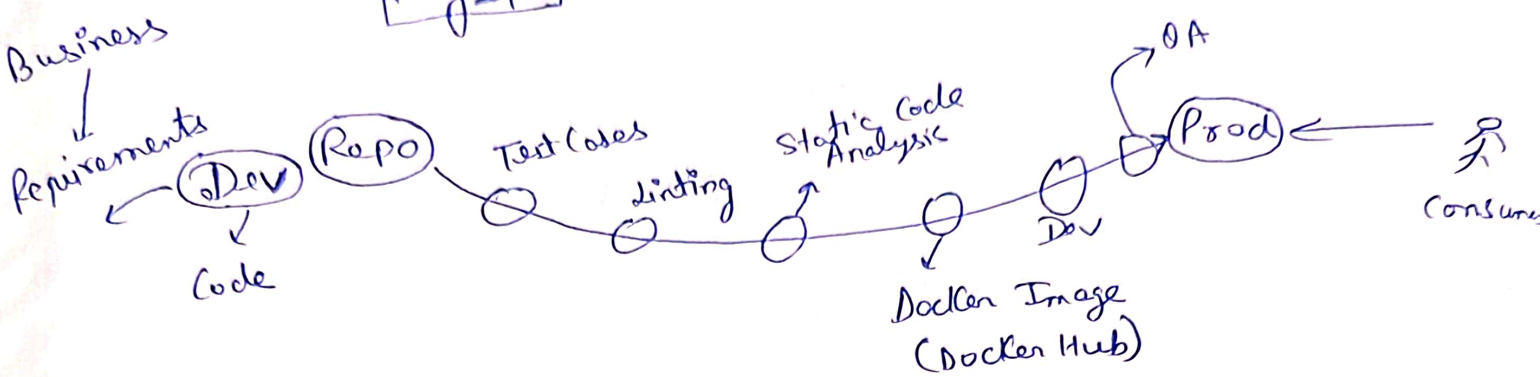
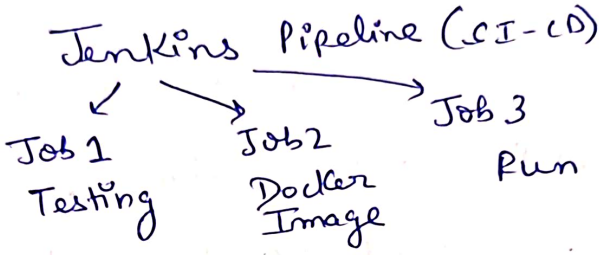
Day 15





CD ⇒ Real Deployment (to Prod env).

CI ⇒ flow until Docker Image (Docker Hub).

Pipeline As Code

Pipeline script (can be shared / tracked with Anyone).

Add all jobs inside this.

Practical

1. Launch an EC2 Instance.

→ will create Jenkins | Sonar Cube Server.

Static Code Analysis. Sonar Cube and Static Code Analysis are closely related concepts used to improve code Quality maintainibility and security.

Note Suppose you have initialized variables and not used anywhere in code / written Duplicate code.

Now this will not be covered under test cases, as test cases will only cover Business logic.

Therefore SonarCube will help us in analyzing this.

Red flag means serious issues.

Definition:- SonarCube is one of the most widely used tools for automated static code analysis. Supports multiple languages (Python, JavaScript, C# etc). and provides a web-based dashboard to track code quality over time.

→ Provides in helping High Code Quality.

→ Provides solution as well for identified issues.

→ Identifies common vulnerabilities.

(for this will add one more stage in the Jenkins Pipeline)

→ Decide priority over issues.

(Say hardcoded password in code (Serious issue)).

⇒ Connect to EC2 instance.

1> Install Docker 2> Start Docker Engine.

3> Installing Jenkins Server inside Docker Container. (As it is fast).

4> Connect to Web UI of Jenkins.

5> docker exec -it ___ bat ___ file
Copy password and proceed further.
(Build Pipeline plugin).

CI-CD

GitHub Actions

GitLab

Internal way to create CI-CD Pipelines

GitHub Actions :- Powerful way to automate tasks like running SonarCube for static code analysis every time you push code or open a pull request.

⇒ GitHub Actions automates software workflows right inside GitHub, without needing on external CI/CD tools. Entire infra structure is provided by them. Can trigger actions on code changes, Pull requests, issues, releases etc.

Benefits → Tightly Integrated with GitHub.
a> No setup needed for Integration.
b> Uses GitHub secrets, checks and artifacts Manually.
→ Built in CI/CD (Run unit tests, lint code, build, artifacts etc).

. GitHub Actions. (can use pre created images).
↓ ↳ yml file.

CI - CD workflow.

Everything Managed by GitHub

⇒ gitHub Actions

- Workflow
- yml
- jobs section
- runs-on
- steps (entire logic).

Jenkins (Need to configure everything our own)

- Pipeline script
- Groovy (JSON).
- stages section.
- agent.

⇒ . Triggers in Jenkins.

⇒ . Difference b/w Jenkins & GitHub Actions.

Actions ⇒ code created by GitHub community.

checkout → will clone the repository. (GitHub Action on web).

Benefits of GitHub Actions continued

→ Highly customizable.
a> Defines workflows in YAML file.
b> Run multiple jobs in parallel or matrix builds.

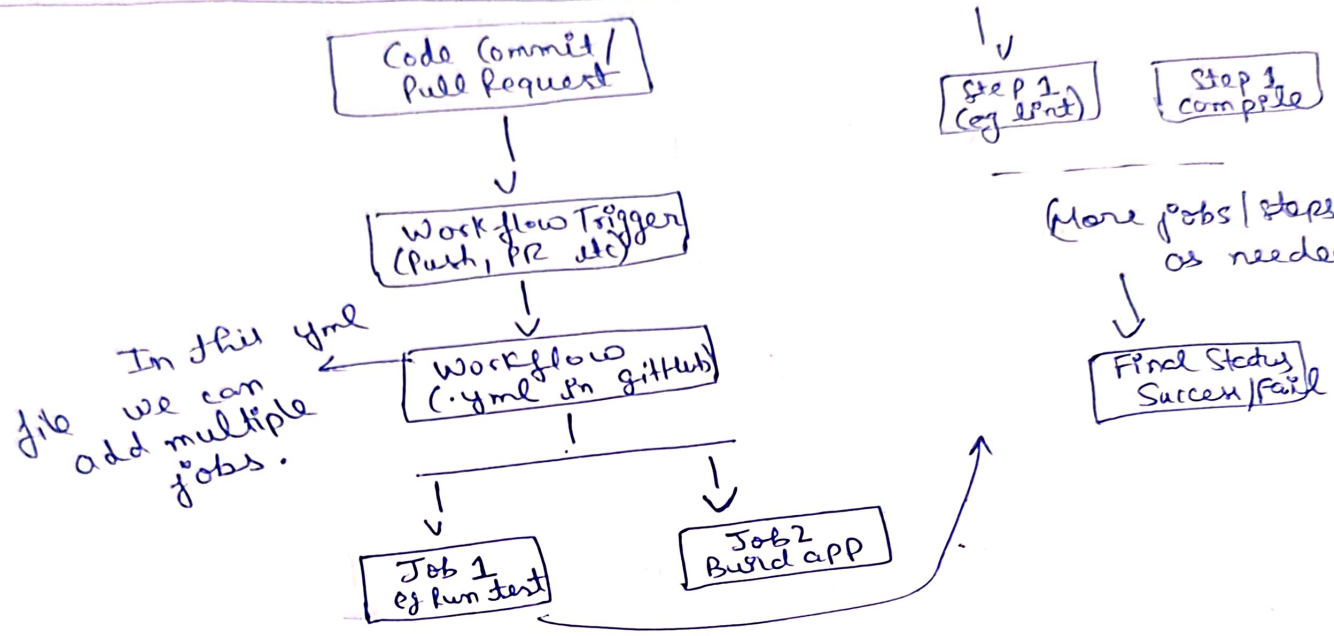
→ . Cross platform Support.
a> Runs jobs on Linux, Windows or MacOS runners.

→ . Secrets Management.
a> Securely store API keys, tokens, or environment variable

Flow Diagram. (Pay are per usage).

GitHub Actions.

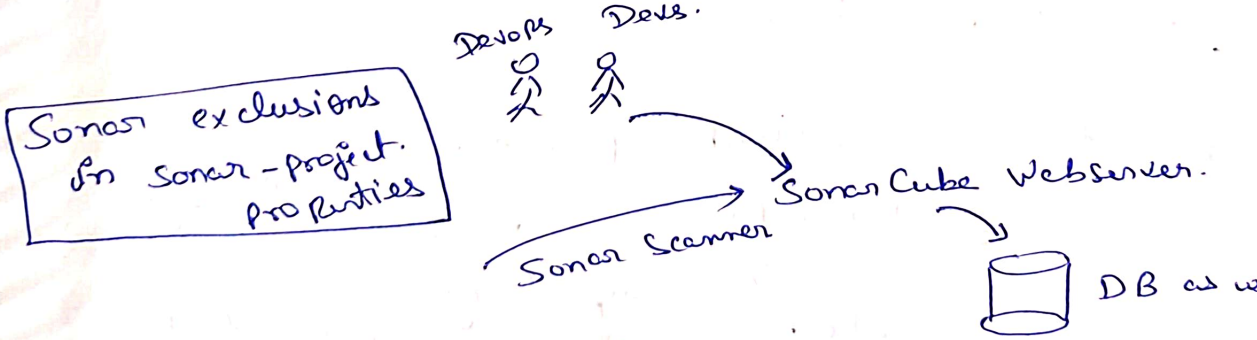
(Workers Nodes ⇒ Named as Runners).



⇒ Configure and Setup SonarCube.

→ Have their own Dedicated Web Server

→ Create A/c in this Web Server.



⇒ Will run at CI-CD Pipeline Code, Webserver will store the reports info in DB and then sent a Notification/ check.

⇒ . Sonarqube Setup steps.

→ (Have official Docker Image as well).

→ or can install from Zip file in local system.

1> . Docker Compose to setup Sonarqube server.

2> . Login to Sonarqube.

3> Create a Project/local Project

4> Create a local project

→ Project Name → Project Key → Project Branch (name).

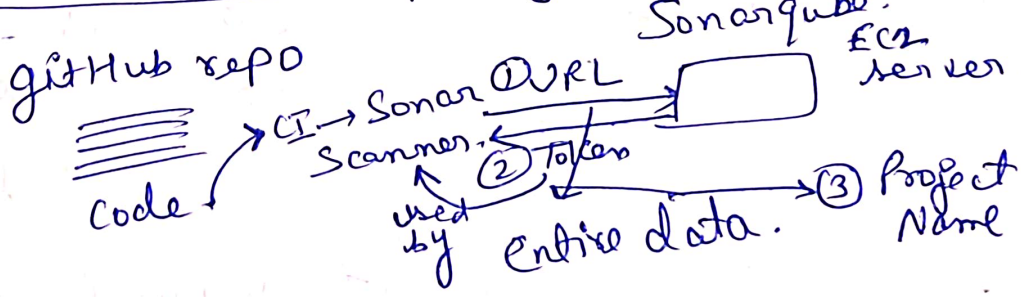
→ Use global setting (Project created).

→ which CI-CD tool using (Provide options to Select)

↓↓

With GitHub Actions

→ Configuration on GitHub repository.



• Normal Setup

↓

Repo → Settings → Secrets & Variables → Actions →

Repository Secrets → Seret Name

→ Generate Token – Copy & Paste the token.

→ Add Secret.

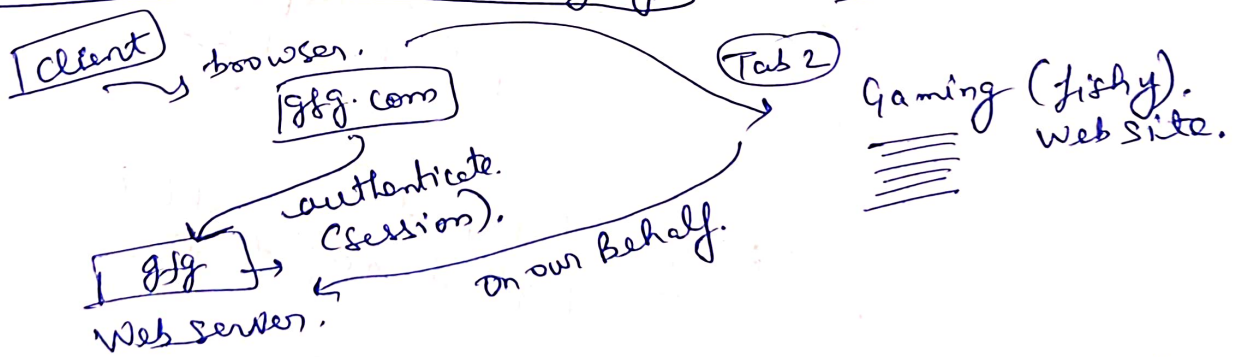
→ Sonar_host_url : (copy value).
and paste under Add Secret (In gitHub repo).

→ Select programming language.

→ VS code ⇒ Sonar-Project.properties (Base Phrase).
(Need URI, Token, Project Name)
↳ This will be read by Sonarqube server.

→ .Workflow file. (Already created).
(will add one more job). (from jobs:).

CSRF (Cross-site Request Forgery) → Automatically will be created.



Session → CSRF (Token will be created everytime
Session/CSRF token) blocked if not
Matched.

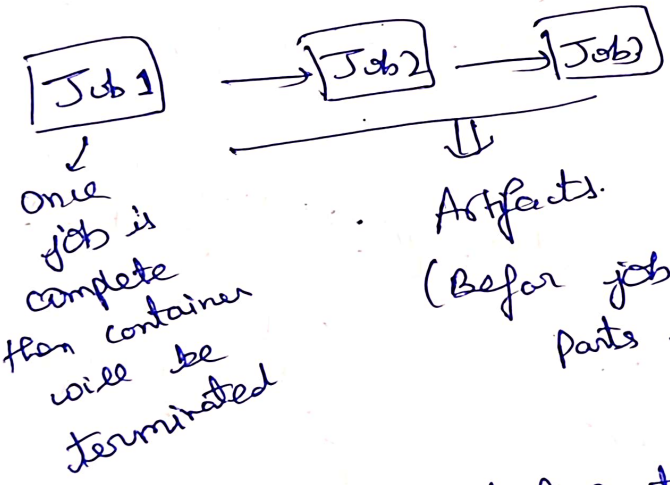
Coverage Need to create coverage report file and pass it to SonarQube (Test Cases should cover the code)

⇒ Go to workflow file

```
Pytest --cov=. --cov-report=xml:coverage.xml.
Cat coverage.xml (Help to check how much part of code
                  is being covered by Test Cases).
```

↳ Give this to Sonarqube.

⇒ **Artifacts:** When we have multiple jobs



Once job is complete then container will be terminated

(Befor job Completes, need to add parts to Artifacts).

```
- name: Upload Report to Artifact.
  uses: actions/upload-artifact@v3.
  with:
    name: coverage-report
    path: coverage.xml
```

⇒ Both jobs are running in parallel which is not correct, so, get Artifacts expired error.

↓↓

To resolve this use **needs** key

Artifacts & needs key

Artifacts are files created during software Development or CI/CD pipelines that are saved and passed to later stages (like testing or deployment).

⇒ They can include

- Test results, logs, static files, Docker Images.

Need for Artifacts

Reusability :
Reliability
Debugging
Efficiency.

Syntax for GitHub Action

```
jobs:
  build:
    runs-on: linux-latest
    steps:
      ≡
      with:
        name: my-artifact
        path: output
```

To use artifacts ←

OR

Flow in YAML Pipelines

Source Code
You start with code

↓

Build Stage
Code is compiled.
Build o/p is created

↓

Save Artifact
Build output (Artifact) is saved/stored

↓

Test Stage
Test job downloads the artifact & run tests

↓

Deploy Stage
Same artifact is used for deploy to production

⇒ needs is a keyword in YAML pipeline configurations to define job dependencies. It tells the system that a job should wait for specific earlier jobs to complete before running.

⇒ This helps in controlling job execution, artifact sharing and optimizing workflows by running jobs in parallel when possible.

Syntax

```
jobs:
  build:
    run-on: linux-latest

  test:
    needs: build
    runs-on: ubuntu-latest

  deploy:
    needs: [build, test]
    runs-on: ubuntu-latest
```

In this example, we have defined 3 Jobs, **build, test and deploy**

Now until **build** is not executed then test and deploy will not be executed.

. Understanding GitHub Actions.

⇒ GitHub actions is a continuous integration and continuous delivery (CI/CD) platform that allows to automate your build, test and deployment pipeline.

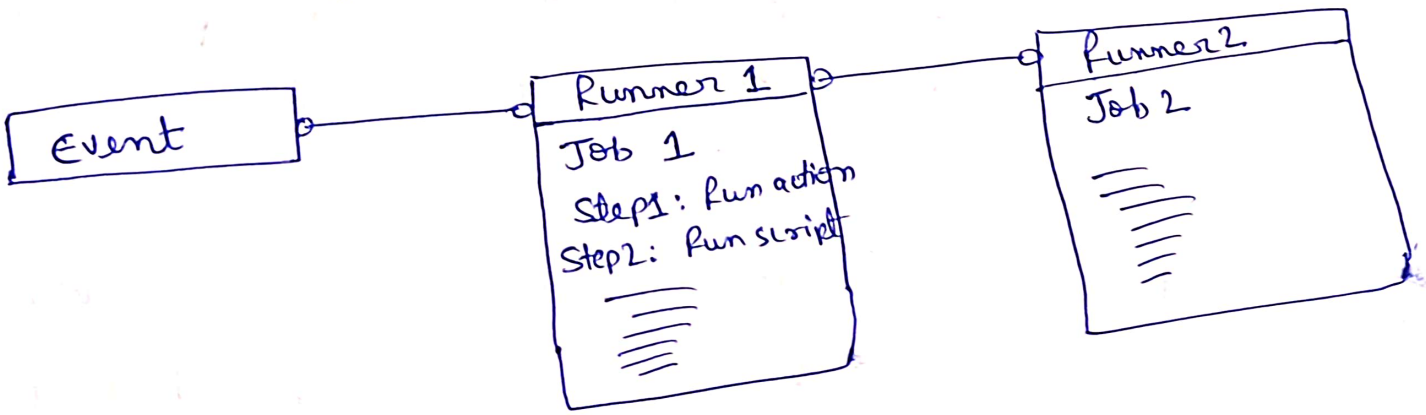
→ Create workflows that build and test every pull request to your repository, or deploy merged pull requests to production.

. Components of GitHub Actions.

→ Can configure a GitHub Action workflow to be triggered when an event occurs in your repository, such as pull request being opened or being created.

→ Workflow can contain one or more jobs which can run in sequential order or in parallel.

* Each job will run inside its own virtal machine runner, or inside a container, and has one or more steps that either run a script that you define or run an action.



Workflows: A workflow is a configurable automated process that will run one or more jobs. Workflows are defined by a YAML file checked in to your repository and will run when triggered by an event in your repository.

→ Workflows are defined in `.github/workflows` directory in a repository. A repo can have multiple workflows, each of which can perform a different set of tasks such as:

- Building & testing pull requests.
- Deploying your Application every time a release is created.

Events: An event is a specific activity in a repository that triggers a workflow. like an activity from GitHub when someone creates a pull request, opens an issue or pushes a commit to a repo. Can also trigger a workflow to run a schedule, By posting to a REST API or manually.

Jobs: A job is a set of steps in a workflow that is executed on the same runner. Each step is either a shell script that will be executed, or an action that will be run.

Steps are executed in order and are dependent on each other.

Since each step is executed on the same runner, you can share data from one step to another (and this can be done using "Artifacts").

Actions An action is a custom application for the GitHub that performs a complex but frequently repeated tasks. Use action to help reduce the amount of repetitive code that are in workflow files.

Runners A runner is a server that runs your workflow when they have/are triggered. Each runner can run a single job at a time. GitHub provides Ubuntu Linux, Windows and MacOS runners to run your workflows.

Note To create a GitHub Actions workflow, can use pre defined Workflow ~~Do~~ Templates and can do Modifications to them as per our usage.