1. **trigger in git hub actions**

Push: The push event triggers a workflow when a commit is pushed to a branch or tag in the repository.

```
on:
 push: branches:
   - main                # Triggers when code is pushed to the 'main' branch tags:
```

Pull_request : The pull_request event triggers a workflow when a pull request is opened, synchronized, or closed.

```
on:
  pull_request:
       branches:
          - main  # Triggers when a PR targets the 'main' branch
```

Schedule: The schedule event triggers a workflow on a schedule (like cron jobs). It uses a cron syntax for specifying the timing.

Example:

yaml
Copy

```
on:
  schedule:
    - cron: '0 0 * * *'  # Trigger the workflow daily at midnight UTC
```

workflow_run

The workflow_run event triggers a workflow after another workflow has completed, either successfully or with failure. This is useful for chaining workflows.

```
on:
  workflow_run:
    workflows: ["CI"]
    types:
      - completed
```

Star

The star event is similar to watch but focuses only on starred repositories.

Example:

yaml
Copy
on:star:types:-created# Trigger when a repository is starred

2. conditions used in git

1. File States
2. Branch States
5. Push and Pull Conditions
6. Commit Conditions
2. Repository Types
7. Configuration Conditions

3) jinja filter:

Jinja2 is a powerful templating engine for Python, and it's an integral part of Ansible. It's primarily used for generating text files (like configuration files) from templates, but its capabilities extend far beyond that. In Ansible, Jinja2 comes into play for variable substitution and data manipulation within playbooks.

Jinja2 uses placeholders for rendering dynamic content. These placeholders are defined in double curly braces, like {{ variable_name }}. When Ansible runs a playbook, it replaces these placeholders with actual variable values, making the content dynamic.

What makes Jinja2 especially useful in Ansible is its filtering capability. Filters in Jinja2 are used to transform the data passed to them. They can modify strings, manage lists, perform logical operations, and much more. A filter is typically applied to a variable using the pipe syntax, {{ variable_name | filter_name }}.

Types of filters are :

1. **String Filters** : Converts a string to lowercase, uppercase,capitalize, replace , trim etc ..
2. **Number Filters** : abs , round, int
3. **List and Dictionary Filters** : length , sort unique , map , select
4. **Date and Time Filters** : date and time
5. **Default filters**: {{ undefined_variable | default("Default Value") }}

4)) sonar analysis what steps will be done

1. **SonarQube server setup** (can be SonarQube Cloud or your own SonarQube instance).
2. **SonarQube scanner setup** for your GitHub Action.
3. **Maven build** setup for Java project, including test and code coverage generation (usually with tools like JaCoCo).
4. **SonarQube Quality Gate** check to ensure that the code meets specific quality standards (e.g., coverage percentage, issues count).

quality gate: In SonarQube, you can define **Quality Gates**, which specify thresholds for various quality metrics (like code coverage, number of issues, duplications, etc.). The quality gate is checked after the analysis is completed.

- Go to your SonarQube instance (or SonarCloud).
- Define a **Quality Gate** that enforces the desired thresholds for metrics like:
  - **Coverage**: e.g., 80% or higher.
  - **New Bugs**: No new bugs allowed.
  - **Duplicated Lines**: Less than 3%.

smell test: Definition: Detects maintainability issues in the code that may not be bugs but indicate potential problems.
Examples: Long methods, large classes, duplicate code.
Purpose: Helps improve code readability and maintainability.

5. Ansible collections vs roles.

Roles splits the single playbook into multiple files. Ansible roles is defined with 8 directory and with 8 files, roles must include at least contain one of above directories.

(An **Ansible Role** is a modular way to organize and reuse automation tasks. Roles help simplify complex playbooks by breaking them into smaller, reusable components. They follow a predefined directory structure that makes the organization of tasks, variables, handlers, and templates consistent.)

```
roles/
├── <role_name>/
│   ├── defaults/        # Default variables (lowest priority)
│   │   └── main.yml
│   ├── vars/            # Other variables (higher priority than defaults)
│   │   └── main.yml
│   ├── files/           # Static files to be copied to the target system
│   ├── templates/       # Jinja2 templates
│   ├── tasks/           # Main list of tasks to execute
│   │   └── main.yml
│   ├── handlers/        # Handlers to trigger (e.g., service restarts)
│   │   └── main.yml
│   ├── meta/            # Role metadata (dependencies, author info)
│   │   └── main.yml
│   ├── tests/           # Test playbooks for the role
│   │   ├── inventory
│   │   └── test.yml
│   └── README.md        # Documentation for the role
```

An Ansible Collection has a predefined directory structure:

```
collection/
├── docs/              # Documentation files
├── plugins/           # Custom plugins (modules, lookups, etc.)
```

```
|   ├── modules/
|   ├── lookup/
|   ├── filter/
|   ├── callback/
|   └── ...              # Other plugin types
├── roles/               # Roles included in the collection
|   └── my_role/
|       ├── tasks/
|       ├── vars/
|       ├── templates/
|       └── ...          # Standard role structure
├── playbooks/           # Example playbooks
├── tests/               # Test cases for the collection
├── galaxy.yml           # Metadata file for the collection
└── README.md            # Readme file for the collection
```

6) notify and handlers

Handlers: it are special type of tasks which is executed only when there is change in task to which handler is defined. Notify is used to call the handlers.

- notify: handlers name

7) 8 types of git hub actions

1. CI/CD actions
2. Testing Actions
3. Deployment Actions
4. Issue Management Actions
5. Pull Request Actions
6. Notification Actions
7. Code Quality and Security Actions
8. Workflow Management Actions

8) structure of roles

roles/

```
├── <role_name>/
│   ├── defaults/
│   │   └── main.yml      # Default variables (lowest priority)
│   ├── vars/
│   │   └── main.yml      # Variables (higher priority than defaults)
│   ├── files/
│   │   └── ...           # Static files to be copied to managed nodes
│   ├── templates/
│   │   └── ...           # Jinja2 templates for dynamic configuration
│   ├── tasks/
│   │   └── main.yml      # List of main tasks to execute
│   ├── handlers/
│   │   └── main.yml      # Handlers triggered by tasks
│   ├── meta/
│   │   └── main.yml      # Metadata about the role (dependencies, etc.)
│   ├── tests/
│   │   ├── inventory     # Test inventory file
│   │   └── test.yml      # Test playbook for the role
│   ├── README.md         # Documentation for the role
│   └── LICENSE           # License for the role (optional)
```

9) Templates directory

The **`templates` directory** in Ansible roles is used to store **Jinja2 templates**. These templates allow dynamic configuration file generation or content customization based on variables, facts, or conditions. It is a core component of Ansible's templating system.

The `templates` directory is a subdirectory within the role structure

10) What is j2 and jinja filter:

Answered above

11) what are plugins used and types of parameterized plugins.

Plugins in Jenkins are software components that extend the core functionality of Jenkins. They allow users to integrate additional tools, customize pipelines, and automate a variety of tasks. Jenkins

plugins enable seamless integration with SCMs (e.g., Git), build tools (e.g., Maven, Gradle), testing frameworks, deployment tools, and cloud services.