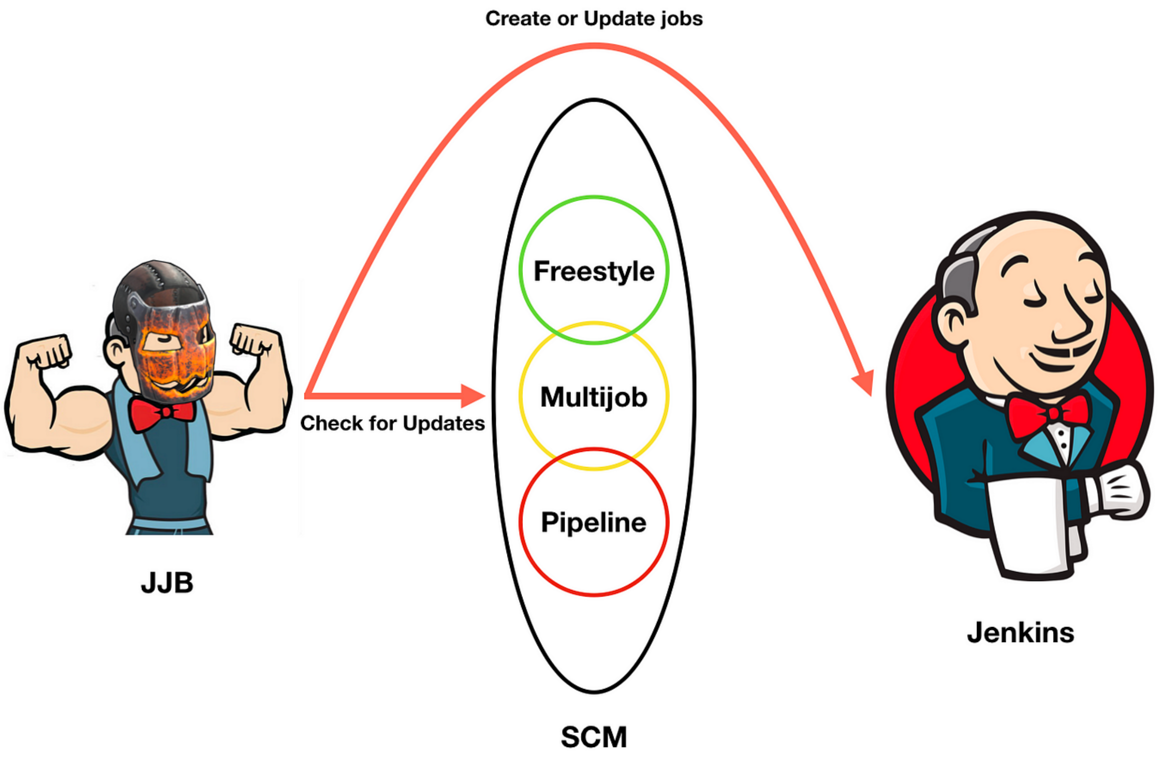
Jenkins

Session 2

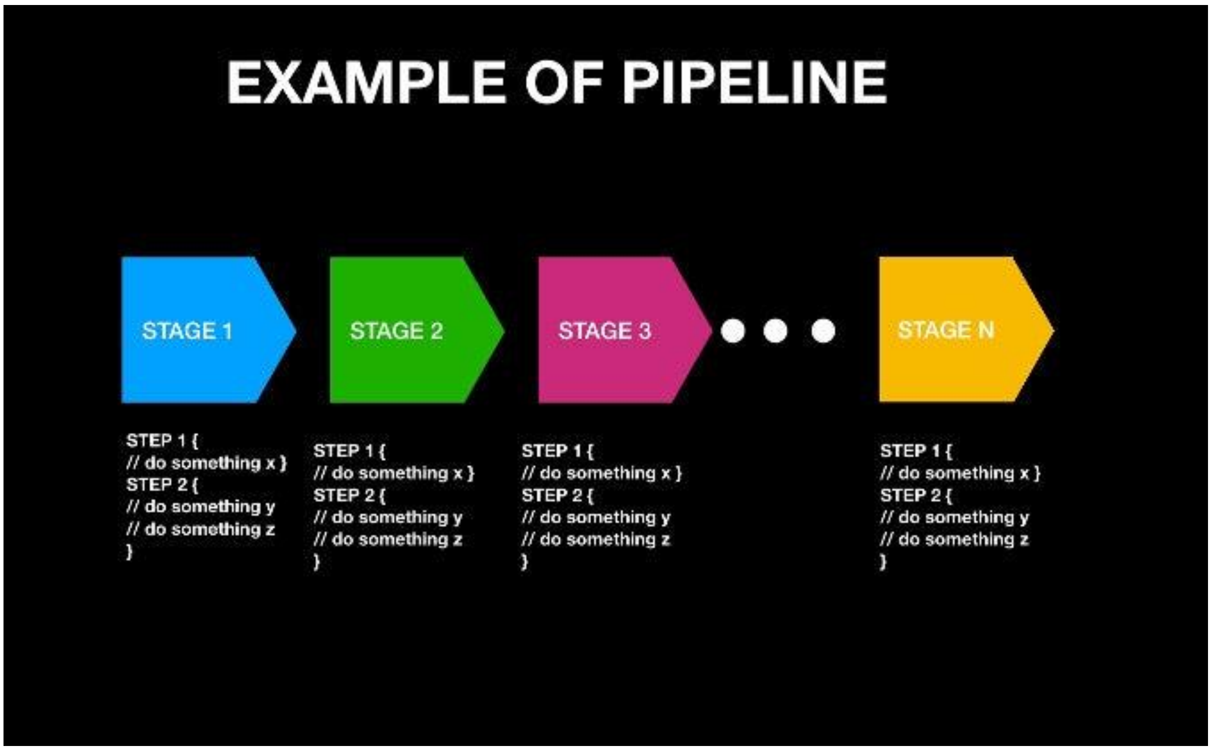
Understanding Jobs and Pipelines:

1. **Jenkins Jobs:**



- **Types of Jenkins Jobs:**
  - **Freestyle Jobs:** The simplest type, configured through the UI, allowing you to define build steps like compiling code, running tests, etc.
  - **Maven Jobs:** Tailored for Maven projects, automatically handling dependencies and build lifecycles.
  - **Multi-Configuration Jobs:** Used for running a job with different configurations (e.g., across different platforms).
  - **Pipeline Jobs:** Allow defining complex build pipelines using Groovy code in a Jenkinsfile.

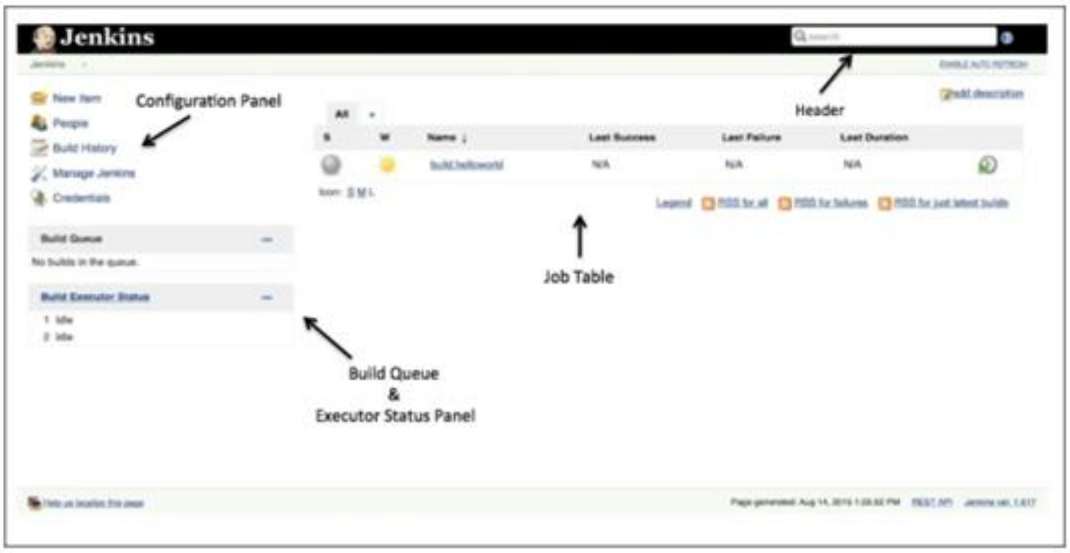
2. **Jenkins Pipelines:**


EXAMPLE OF PIPELINE
STAGE 1
STAGE 2
STAGE 3
STAGE N
STEP 1 { // do something x } STEP 2 { // do something y // do something z }
STEP 1 { // do something x } STEP 2 { // do something y // do something z }
STEP 1 { // do something x } STEP 2 { // do something y // do something z }
STEP 1 { // do something x } STEP 2 { // do something y // do something z }

- A Jenkins Pipeline is a set of build steps that are executed sequentially or in parallel, defined as code.
- Pipelines can be:
  - **Declarative Pipelines:** Have a more straightforward syntax and structure, suitable for most users.
  - **Scripted Pipelines:** Offer more flexibility and control, written entirely in Groovy.
- Pipelines are powerful for defining complex CI/CD workflows, including multiple stages, parallel jobs, and conditional executions.

Creating and Running Your First Job:

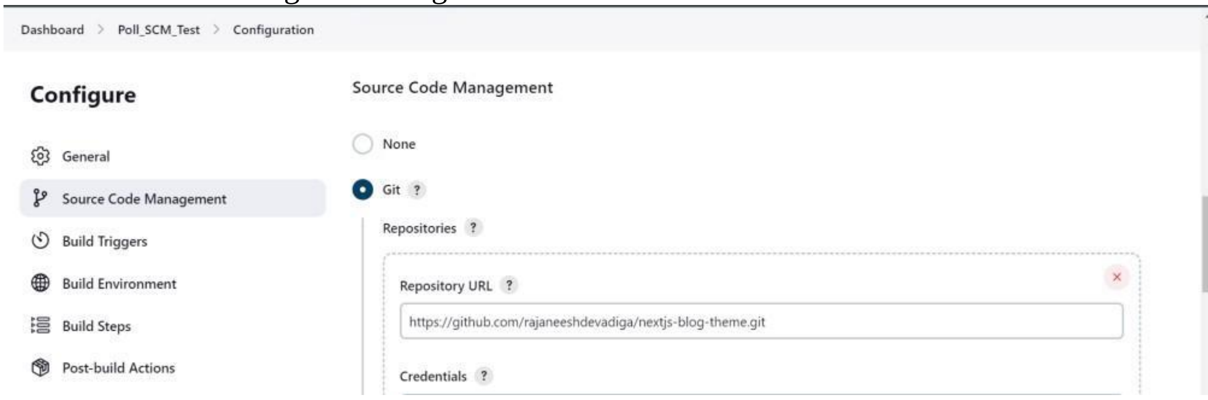
1. **Creating a Jenkins Job:**



2.
   - In Jenkins, go to the dashboard and click on "New Item."
   - Enter a name for the job and select the job type (e.g., Freestyle Project).
   - Configure the job by defining the build steps, post-build actions, etc.
   - Save the configuration.
3. **Running the Job:**
   - Once the job is created, it can be triggered manually by clicking "Build Now."
   - Jenkins will start executing the build steps defined in the job configuration.
   - The job status (success or failure) is displayed on the dashboard, and detailed build logs can be viewed for troubleshooting.

Configuring SCM (Source Code Management) with Jenkins:

1. **What is SCM in Jenkins?**
   - Source Code Management (SCM) refers to tools like Git, Subversion, and Mercurial that store and manage source code.
   - Jenkins can be integrated with these SCM tools to automatically fetch the latest code for building and testing.



Source Code Management

None [✓ Git]

Repositories

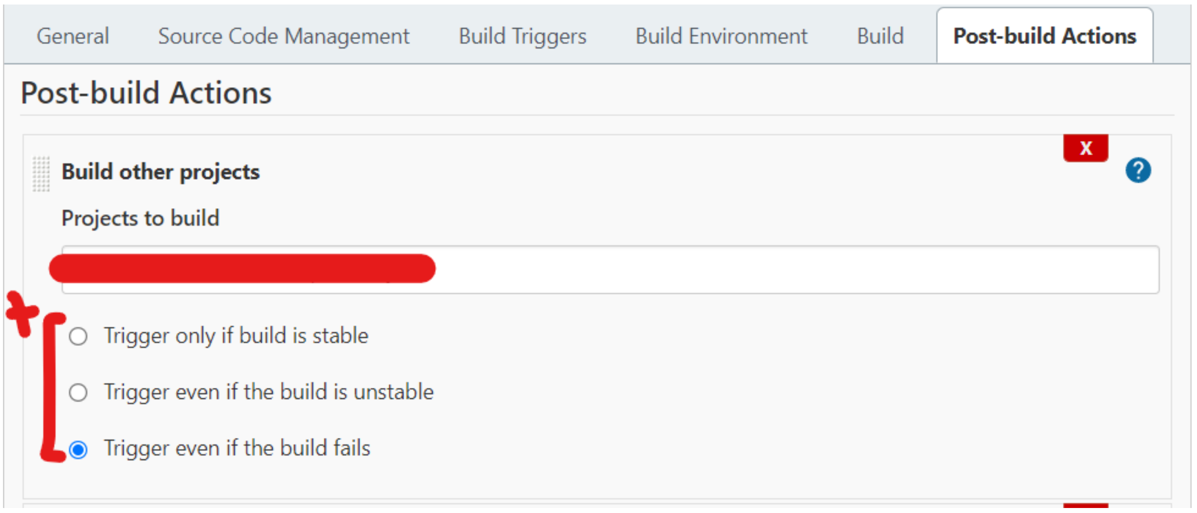
Repository URL: https://github.com/rajaneeshdevadiga/nextjs-blog-theme.git

Credentials

2. **Configuring SCM in a Jenkins Job:**
   - In the job configuration, navigate to the "Source Code Management" section.
   - Select the appropriate SCM tool (e.g., Git).
   - Provide the repository URL and, if needed, credentials for access.
   - You can also specify branches to build, commit triggers, etc.
   - This allows Jenkins to pull the latest code from the SCM system each time a build is triggered.

Understanding Build Triggers:

1. **What are Build Triggers?**
   - Build triggers define when and how a Jenkins job is started. Different types of triggers allow automation of builds based on specific conditions.



General | Source Code Management | Build Triggers | Build Environment | Build | **Post-build Actions**

Post-build Actions

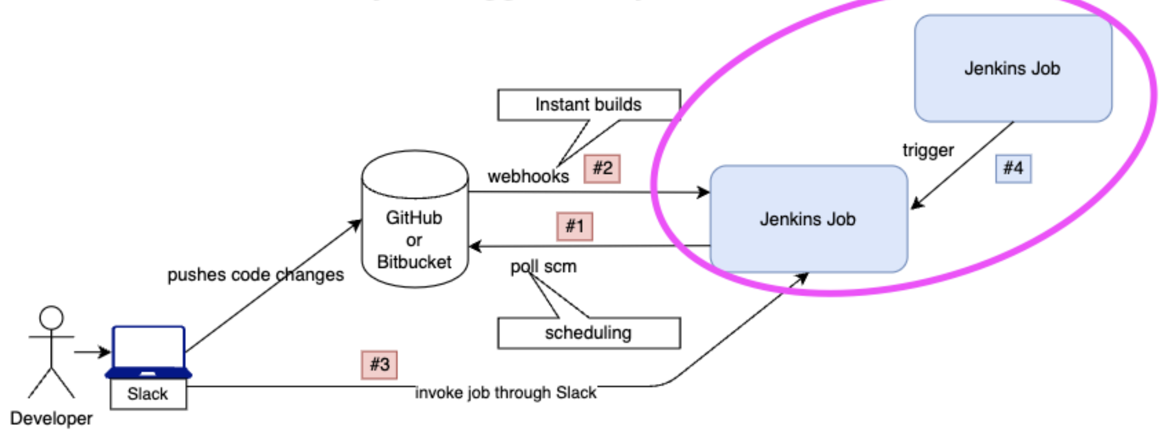
Build other projects

Projects to build

[redacted]

- ○ Trigger only if build is stable
- ○ Trigger even if the build is unstable
- ● Trigger even if the build fails

Different ways to trigger build jobs in Jenkins



2. **Types of Build Triggers:**
   - **Manual Trigger:** Users manually trigger builds by clicking "Build Now."
   - **SCM Polling:** Jenkins polls the SCM at regular intervals to check for code changes. If changes are detected, it triggers a build.
   - **Webhook Triggers (e.g., GitHub Webhooks):** SCM systems like GitHub can automatically notify Jenkins of code changes, immediately triggering a build.
   - **Scheduled Triggers (Cron Syntax):** Jobs can be scheduled to run at specific times, such as daily or weekly.
   - **Build After Other Projects Are Built:** Triggers a job based on the completion of another job.
3. **Importance of Build Triggers:**
   - Automating builds with triggers ensures timely feedback on code changes, reducing manual effort and enhancing the CI/CD process.

Remote Build

To create a remote build in Jenkins, you need to configure your Jenkins instance to accept build triggers from remote systems. Here's a step-by-step guide:

Prerequisites

1. **Jenkins Installed:** Ensure Jenkins is up and running.
2. **Job Configuration:** You should have a job already configured that you want to trigger remotely.

Steps to Create a Remote Build Trigger:

1. **Enable Remote Build Trigger in Jenkins Job:**
   - Go to the Jenkins dashboard.
   - Open the job you want to trigger remotely.
   - Click on **Configure**.
   - Scroll down to the **Build Triggers** section.
   - Check the option **"Trigger builds remotely (e.g., from scripts)"**.
   - Enter an **Authentication Token** (a secret string that you will use to trigger the build).
   - Click **Save**.
2. **Create the URL for Remote Triggering:**
   - The remote triggering URL has the following format:

   ```
   http://<JENKINS_URL>/job/<JOB_NAME>/build?token=<YOUR_TOKEN>
   ```

   Replace:

   - `<JENKINS_URL>` with your Jenkins server URL.
   - `<JOB_NAME>` with the name of your Jenkins job.
   - `<YOUR_TOKEN>` with the token you specified in step 1.

   Example:

   ```
   http://localhost:8080/job/MyJob/build?token=mySecretToken
   ```

3. **Trigger the Build Remotely:**
   - You can trigger the build using:
     - A web browser by entering the URL.
4. **Verify the Build Trigger:**

- Check your Jenkins job to see if the build started successfully.
- You should see a new build in the job's build history.

SCM Polling

To set up SCM (Source Code Management) polling in Jenkins, follow these steps:

Steps to Configure SCM Polling:

1. **Open Jenkins and Access the Job:**
   - Go to your Jenkins dashboard.
   - Select the job you want to configure SCM polling for.
   - Click on **Configure**.
2. **Enable Poll SCM Option:**
   - Scroll down to the **Build Triggers** section.
   - Check the option **"Poll SCM"**.
3. **Specify the Polling Schedule:**
   - In the text box that appears, you need to define the polling schedule using cron syntax.
   - The format for cron expressions is:

     ```
     MINUTE HOUR DOM MONTH DOW
     ```

     Example cron schedules:

     - `H/5 * * * *` **- Poll every 5 minutes.**
     - `H 2 * * 1-5` **- Poll at 2 AM on weekdays (Monday to Friday).**

   Important Notes:

   - The `H` (hash) in the cron expression helps to distribute the load across multiple builds by introducing a random offset.
   - Make sure your cron expression is correctly configured to avoid frequent or unnecessary builds.
4. **Save the Configuration:**
   - After setting the polling schedule, click **Save**.

How SCM Polling Works:

- Jenkins will automatically check the configured SCM (like Git, SVN, etc.) based on the schedule you set.
- If any changes are detected in the repository, Jenkins will trigger a build.

Example Use Cases:

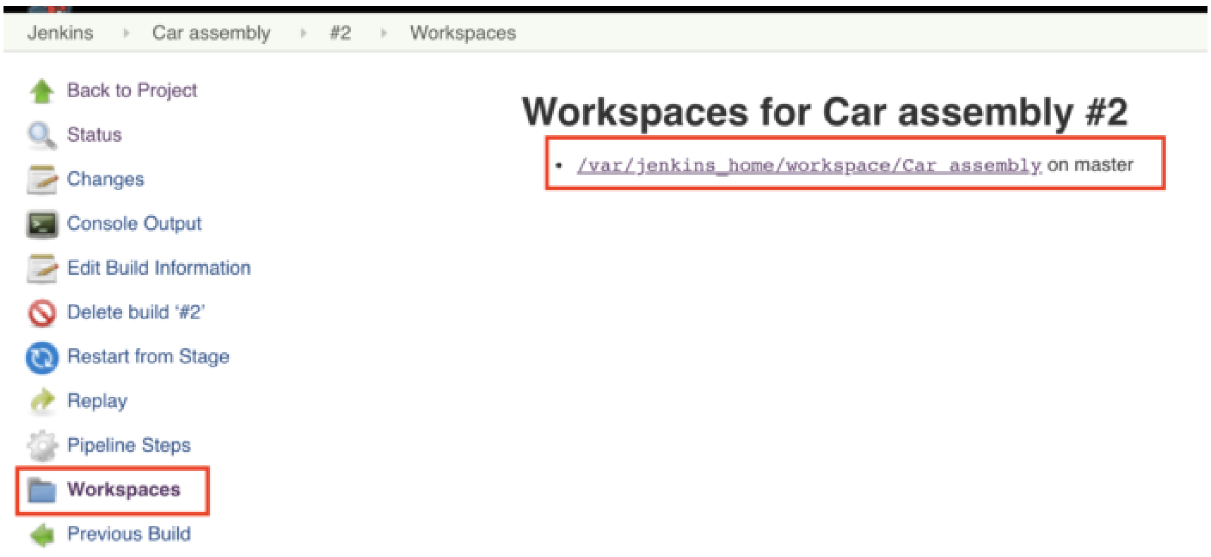
- Regularly checking for updates in a Git repository and triggering a build whenever new commits are pushed.
- Automating builds for projects that frequently receive updates.

Common Pitfalls:

- **Over-Polling:** Polling too frequently can put unnecessary load on your Jenkins server and the SCM system.
- **Correct Cron Syntax:** Ensure the cron syntax is correct, as any mistakes might result in missed builds or overly frequent polling.

By following these steps, Jenkins will automatically monitor your repository and trigger builds whenever there are code changes, making it a useful feature for continuous integration workflows.

Working with Jenkins Workspaces:



Workspaces for Car assembly #2

- /var/jenkins_home/workspace/Car assembly on master

1. **What is a Jenkins Workspace?**
   - A Jenkins workspace is the directory on the Jenkins server or agent where the source code is checked out and the build process takes place.
   - It contains all the files and artifacts related to a specific build, such as the source code, compiled binaries, logs, and test results.
2. **Workspace Usage:**
   - Jenkins uses the workspace to clone the code from SCM, perform builds, run tests, and generate outputs like packages or Docker images.
   - The workspace is unique to each job or pipeline execution, preventing conflicts between different builds.
3. **Managing Workspaces:**
   - Jenkins administrators can set policies for cleaning up workspaces, particularly after builds, to free up disk space.
   - Pipelines can use the `workspace` variable to refer to the current workspace or navigate to specific directories using the `dir` step.
4. **Working with Workspaces on Agents:**

- In a distributed Jenkins setup, where builds are performed on multiple agents, each agent has its own workspace for the job. This allows parallel execution and better resource management.