Experiment 04

Aim:

Integrate a working repository with Jenkins & implementing its scripted pipeline.

Theory:

In modern software development, speed, stability, and collaboration are crucial. This is where **DevOps** practices like **Continuous Integration (CI)** come into play.

Continuous Integration (CI) is a software development practice where developers frequently merge their code changes into a central repository. Each integration is verified by an automated build and test process, allowing teams to detect problems early and improve the software's quality and delivery speed.

Jenkins is an open-source CI tool written in Java. It acts as an automation server that supports building, testing, and deploying software projects. Jenkins supports plugins that integrate with virtually every tool in the DevOps toolchain, from version control systems like Git to build tools like Maven and Gradle.

Maven is a build automation and dependency management tool for Java projects. It uses a file named pom.xml (Project Object Model) to manage project dependencies, build configuration, testing, and packaging.

GitHub is a popular web-based hosting service for version control using Git. By integrating GitHub with Jenkins, every code push or pull request can automatically trigger Jenkins to:

- Pull the updated source code
- Build the application using Maven
- Run test cases
- Generate reports
- Deploy artifacts

This experiment explores how Jenkins connects with GitHub and Maven to automate the software development lifecycle. It eliminates manual processes, reduces integration issues, and improves code stability.

Advantages of Jenkins CI Integration:

- Early detection of bugs and integration conflicts
- Reduced manual intervention and errors
- Real-time feedback on code quality

- Faster development and deployment cycles
- Supports scalable and repeatable processes across teams

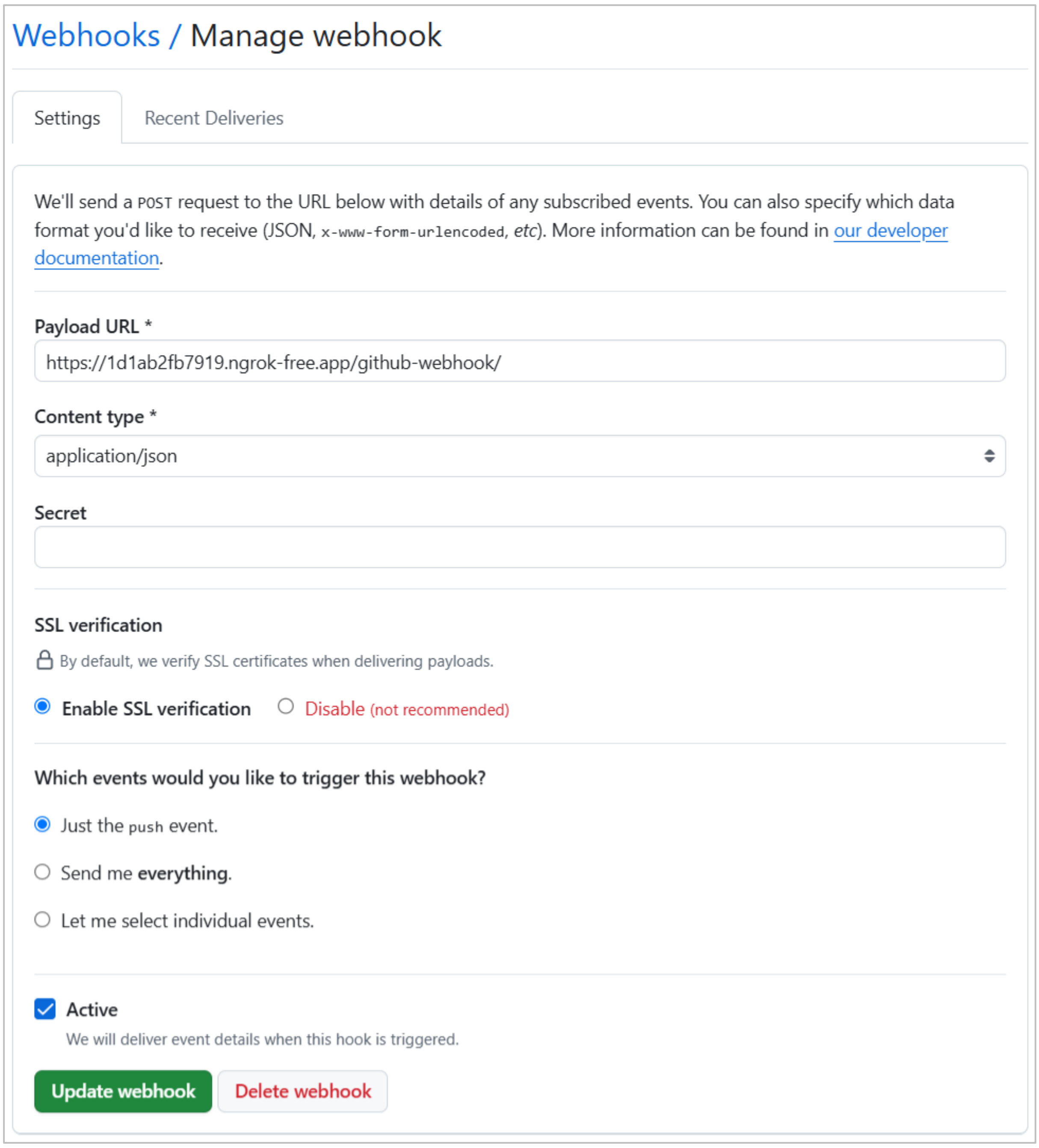
REQUIREMENTS

- Java JDK (version 11 or above)
- Apache Maven
- Jenkins WAR package
- Git and GitHub repository
- GitHub Personal Access Token (for private repositories)
- Web browser
- Optional: ngrok (to expose localhost for webhook)

Part C: Connect Jenkins to GitHub

1. **Fork or Create Maven Project**
    - Use a sample project like Spring PetClinic or create your own Java project with:
        - pom.xml
        - Source and test files
2. **Set Up GitHub Webhook**
    - GitHub → Repository Settings → Webhooks → Add Webhook
        - Payload URL: http://<your-ip>:8080/github-webhook/
        - Content type: application/json
        - Event: Push events
    - If Jenkins is hosted locally, use ngrok to expose:

      ```
      >> ngrok http 8080
      ```



Webhooks / Manage webhook

Settings | Recent Deliveries

We'll send a `POST` request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, `x-www-form-urlencoded`, *etc*). More information can be found in our developer documentation.

Payload URL *: https://1d1ab2fb7919.ngrok-free.app/github-webhook/

Content type *: application/json

Secret:

SSL verification

By default, we verify SSL certificates when delivering payloads.

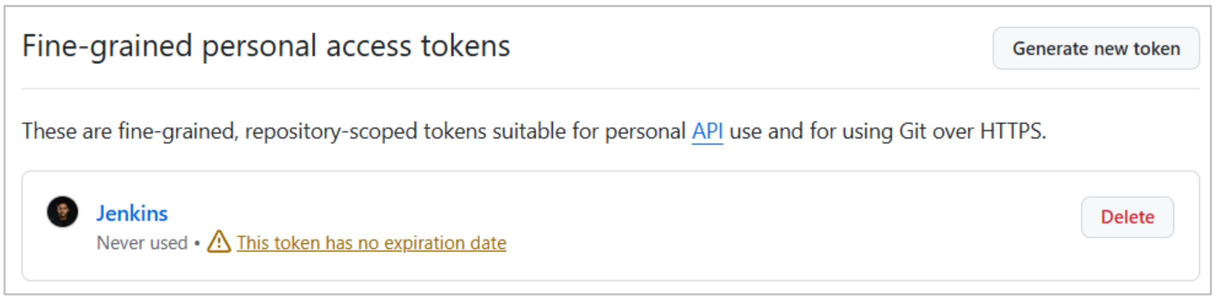
[✓ Enable SSL verification] Disable (not recommended)

Which events would you like to trigger this webhook?

[✓ Just the `push` event.]
Send me **everything**.
Let me select individual events.

[✓ Active]
We will deliver event details when this hook is triggered.

3. **Add GitHub Credentials in Jenkins**
   - Jenkins Dashboard → Manage Jenkins → Credentials → (Global) → Add Credentials
   - Type: Username with Password
     - Username: GitHub username
     - Password: GitHub Personal Access Token
   - Generate a token from:
     - GitHub → Settings → Developer Settings → Personal Access Tokens
     - Scope: repo, workflow



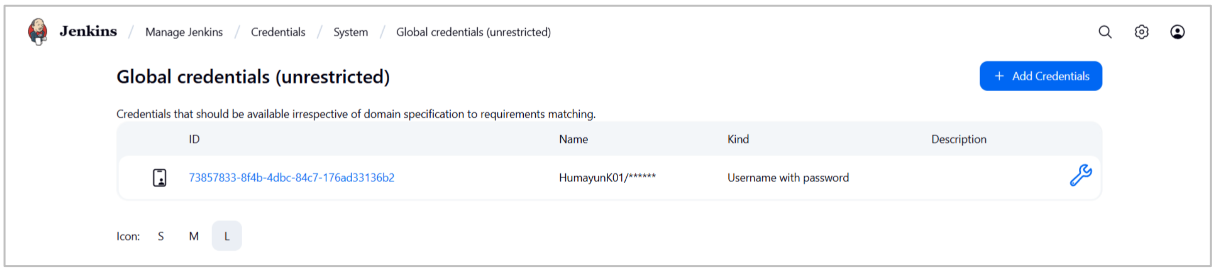
Fine-grained personal access tokens

Generate new token

These are fine-grained, repository-scoped tokens suitable for personal API use and for using Git over HTTPS.

Jenkins
Never used • ⚠ This token has no expiration date

Delete



Jenkins / Manage Jenkins / Credentials / System / Global credentials (unrestricted)

Global credentials (unrestricted)

+ Add Credentials

Credentials that should be available irrespective of domain specification to requirements matching.

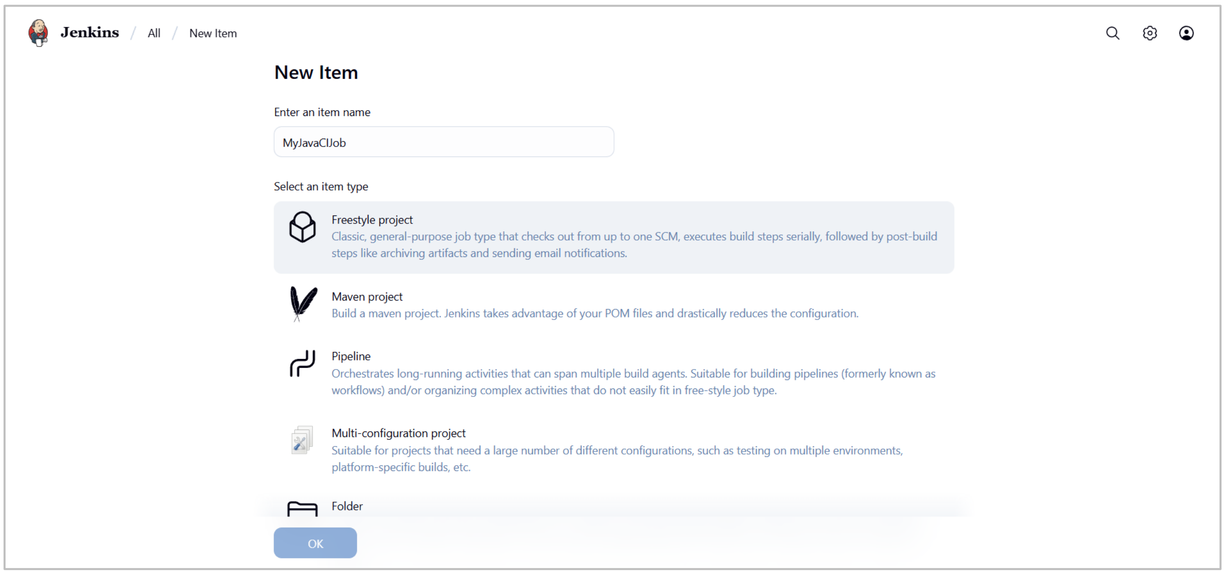
ID	Name	Kind	Description
73857833-8f4b-4dbc-84c7-176ad33136b2	HumayunK01/******	Username with password	

Icon: S M L

Part D: Create Jenkins Build Job

1. Create Freestyle Project

- Jenkins Dashboard → New Item → Name: MyJavaCIJob
- Select: Freestyle Project → OK



Jenkins / All / New Item

New Item

Enter an item name

MyJavaCIJob

Select an item type

Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

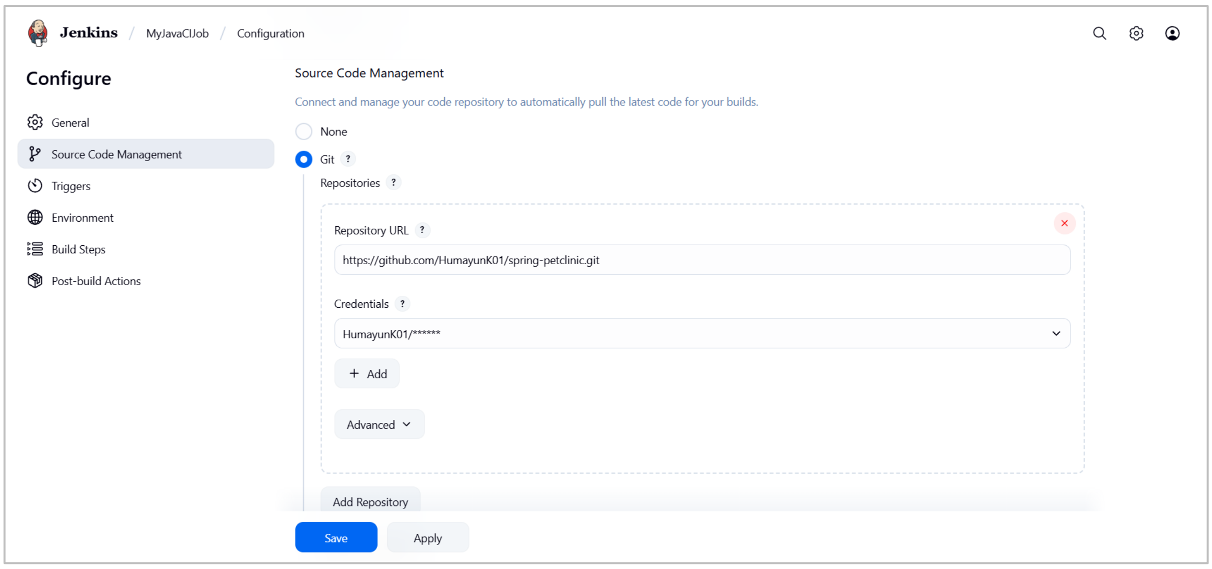
Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder

OK

2. Configure Source Code Management

- Select Git
- Provide GitHub repository URL
- https://github.com/HumayunK01/spring-petclinic.git
- Add credentials (for private repos)


Jenkins / MyJavaCIJob / Configuration

Configure

Source Code Management

Connect and manage your code repository to automatically pull the latest code for your builds.

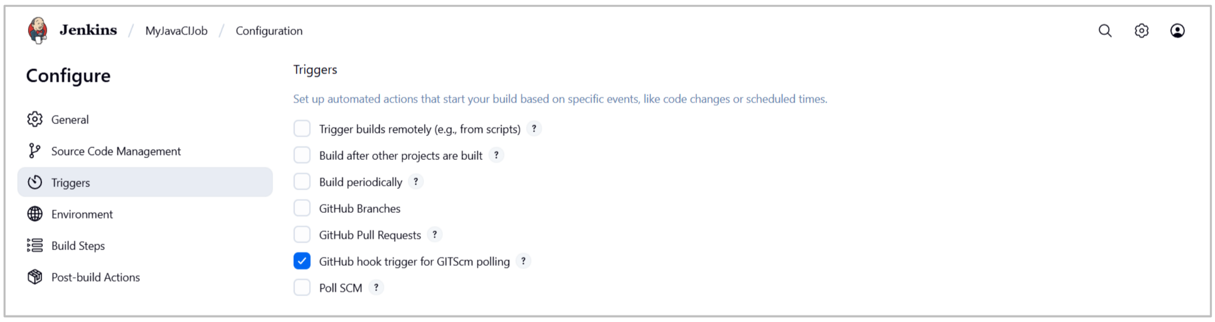
( ) None
(●) Git

Repositories

Repository URL: https://github.com/HumayunK01/spring-petclinic.git

Credentials: HumayunK01/******

3. **Set Build Trigger**
   - Enable: GitHub hook trigger for GITScm polling


Jenkins / MyJavaCIJob / Configuration

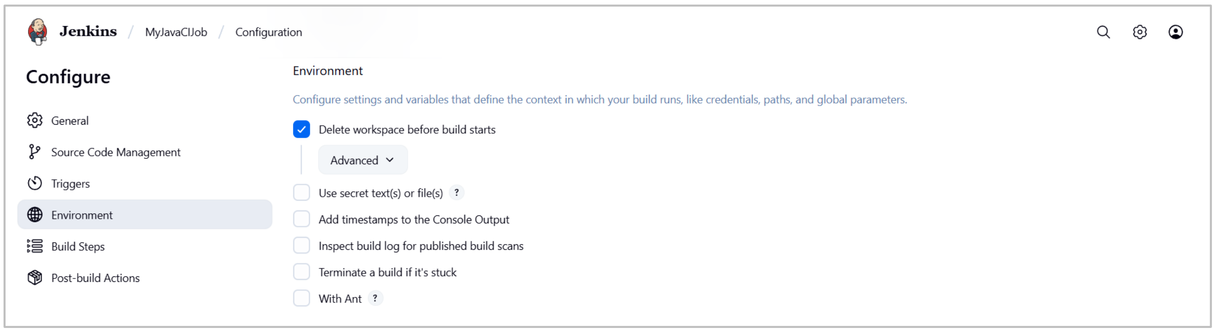
Configure

Triggers

Set up automated actions that start your build based on specific events, like code changes or scheduled times.

- [ ] Trigger builds remotely (e.g., from scripts)
- [ ] Build after other projects are built
- [ ] Build periodically
- [ ] GitHub Branches
- [ ] GitHub Pull Requests
- [x] GitHub hook trigger for GITScm polling
- [ ] Poll SCM

4. **Configure Build Environment**
   - Optionally enable: Delete workspace before build starts


Jenkins / MyJavaCIJob / Configuration

Configure

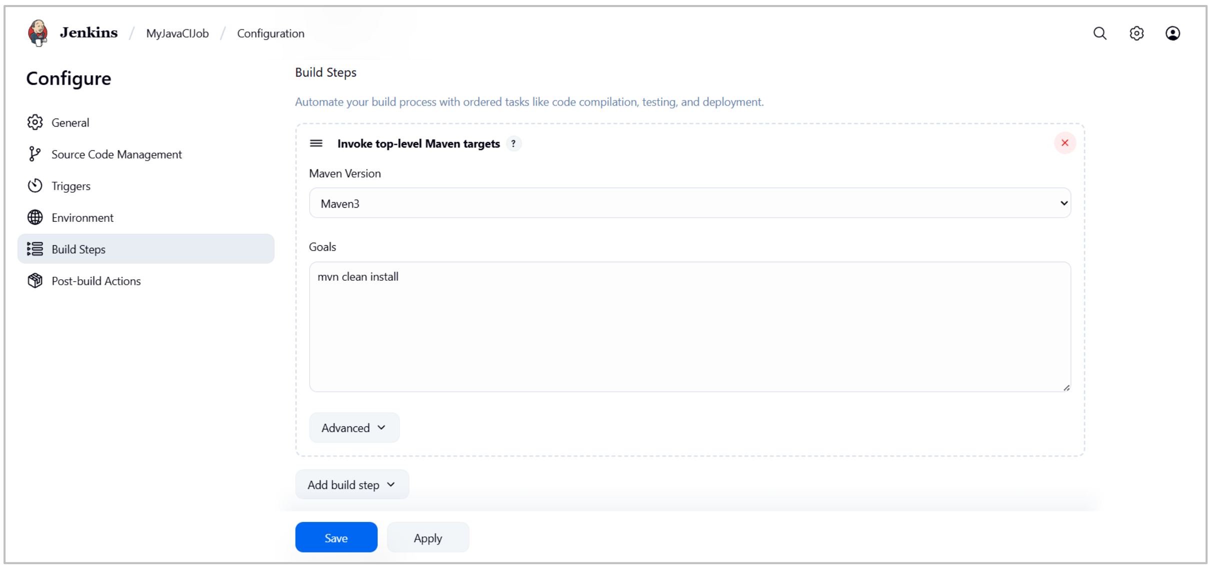
Environment

Configure settings and variables that define the context in which your build runs, like credentials, paths, and global parameters.

- [x] Delete workspace before build starts
- [ ] Use secret text(s) or file(s)
- [ ] Add timestamps to the Console Output
- [ ] Inspect build log for published build scans
- [ ] Terminate a build if it's stuck
- [ ] With Ant

5. **Add Build Steps**
   - Build Step: Invoke Maven Targets
     - Command:

```
>> mvn clean install
```



Build Steps

Automate your build process with ordered tasks like code compilation, testing, and deployment.

Invoke top-level Maven targets

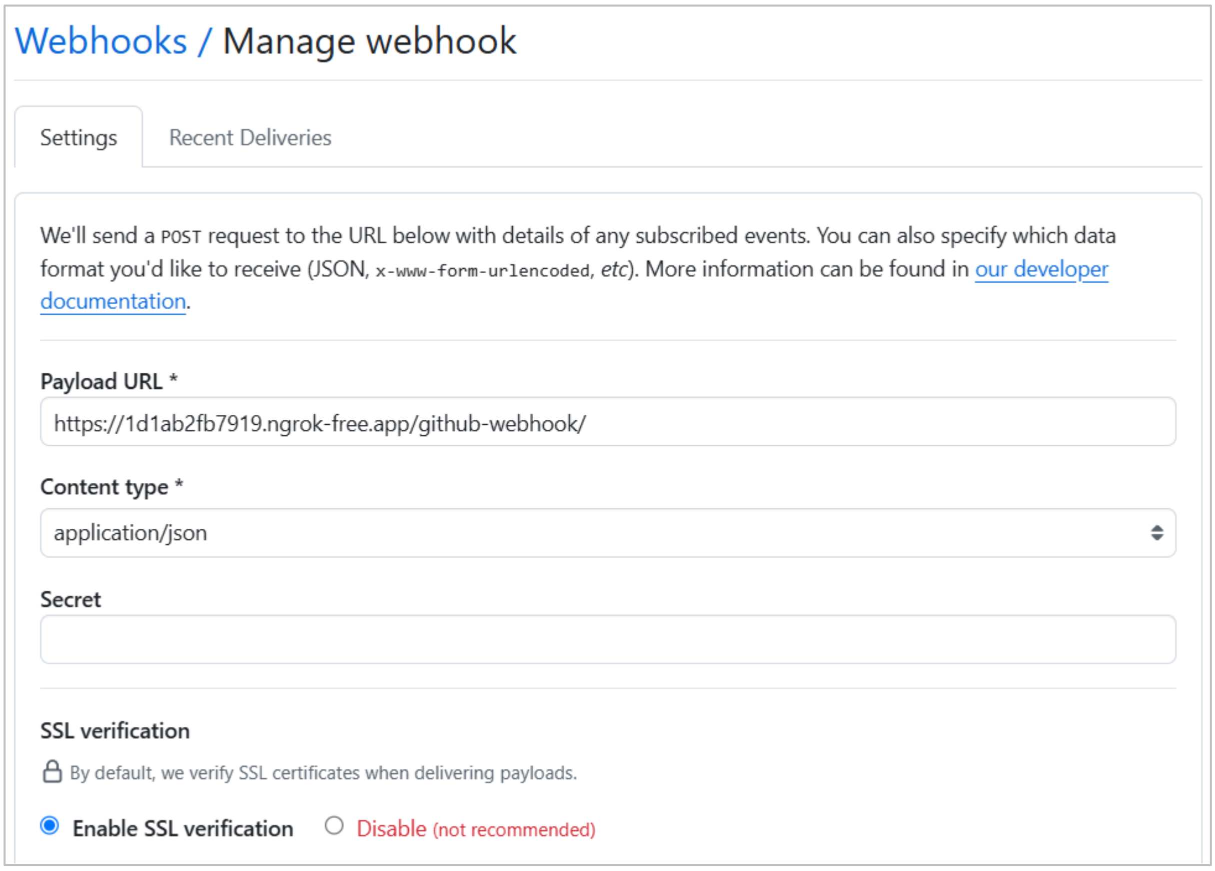
Maven Version: Maven3

Goals:

```
mvn clean install
```

Output:

- GitHub webhook settings



We'll send a `POST` request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, `x-www-form-urlencoded`, *etc*). More information can be found in our developer documentation.

Payload URL *: https://1d1ab2fb7919.ngrok-free.app/github-webhook/

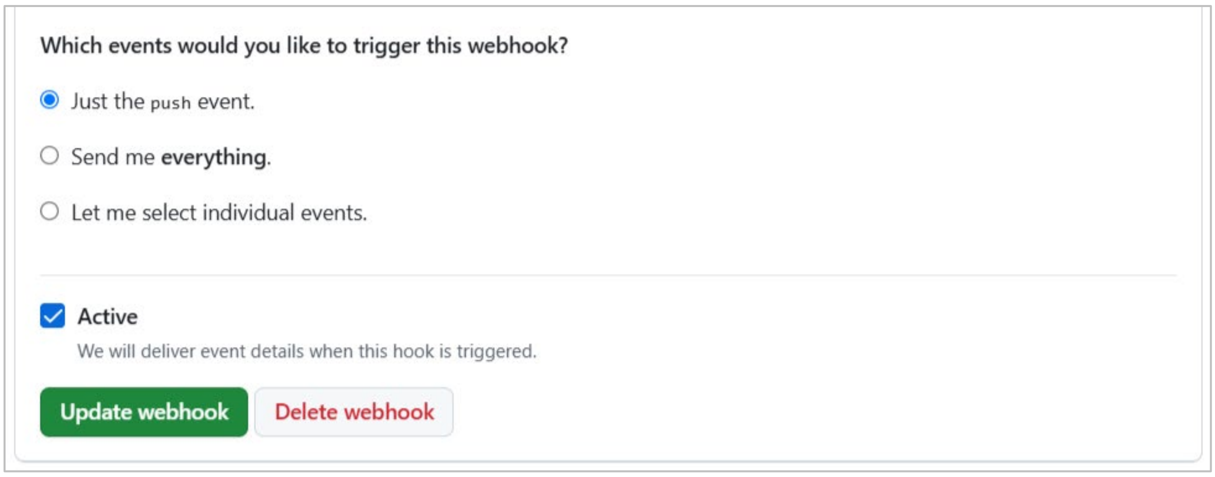
Content type *: application/json

Secret:

SSL verification

By default, we verify SSL certificates when delivering payloads.

[✓ Enable SSL verification] Disable (not recommended)



Which events would you like to trigger this webhook?

[✓ Just the push event.]

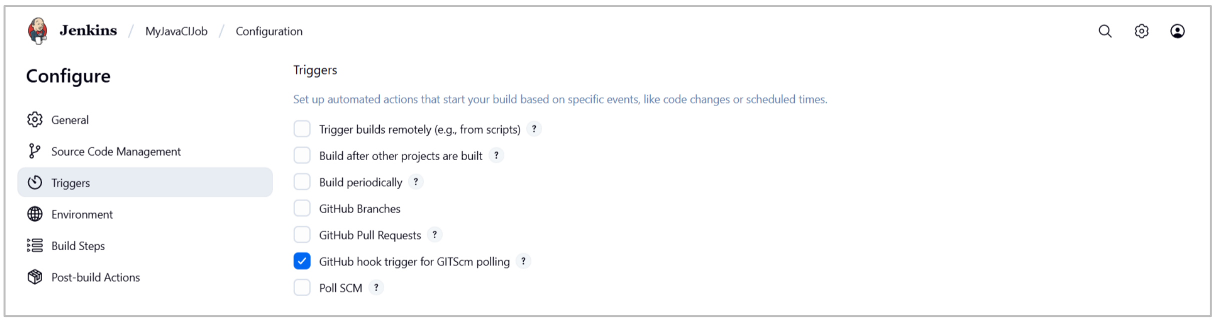
Send me **everything**.

Let me select individual events.

☑ Active

We will deliver event details when this hook is triggered.

- Build trigger and output logs



Triggers

Set up automated actions that start your build based on specific events, like code changes or scheduled times.

☐ Trigger builds remotely (e.g., from scripts)
☐ Build after other projects are built
☐ Build periodically
☐ GitHub Branches
☐ GitHub Pull Requests
☑ GitHub hook trigger for GITScm polling
☐ Poll SCM

Conclusion:

The integration of a working repository with Jenkins and the implementation of its scripted pipeline were successfully achieved. This configuration enables automated execution of build, test, and deployment processes directly from the source code repository, ensuring seamless CI/CD workflows. The scripted pipeline offers greater flexibility and control over each stage, facilitating customization to project-specific needs. As a result, the development process becomes more efficient, reliable, and consistent, reducing manual intervention and potential errors.