jenkinsv4.md

Introduction to Jenkins

Jenkins is an open-source automation server that helps developers build, test, and deploy their applications efficiently. It enables Continuous Integration (CI) and Continuous Deployment (CD), automating the software development lifecycle.

Why Jenkins?

- **Automation**: Reduces manual effort in building, testing, and deploying applications.
- **Scalability**: Supports distributed builds across multiple machines.
- **Plugins**: A rich ecosystem of plugins extends Jenkins' functionality.
- **Easy Integration**: Works well with GitHub, Docker, Kubernetes, and cloud services.

How Jenkins Works

Jenkins follows a **pipeline-based** approach:

1. **Developers push code** to a repository (e.g., GitHub, GitLab).
2. **Jenkins pulls the code**, builds it, and runs tests.
3. **If successful**, Jenkins deploys the application to a staging or production environment.

Deploying Jenkins on Kubernetes

To deploy Jenkins on a Kubernetes cluster, follow these steps:

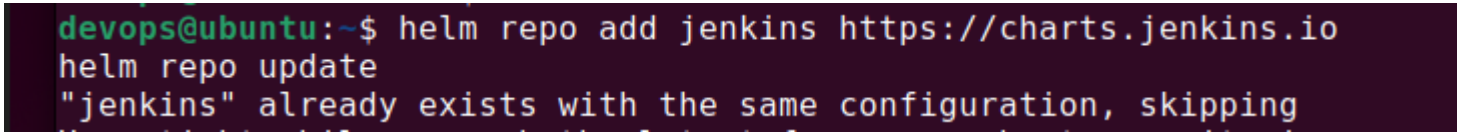
Prerequisites

- A running Kubernetes cluster
- Helm installed
- kubectl configured to interact with the cluster

Step 1: Add Jenkins Helm Repository

```
helm repo add jenkins https://charts.jenkins.io
helm repo update
```

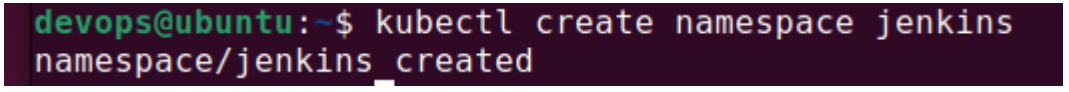
(Screenshot: Adding Helm repository)



Step 2: Create a Namespace for Jenkins

```
kubectl create namespace jenkins
```

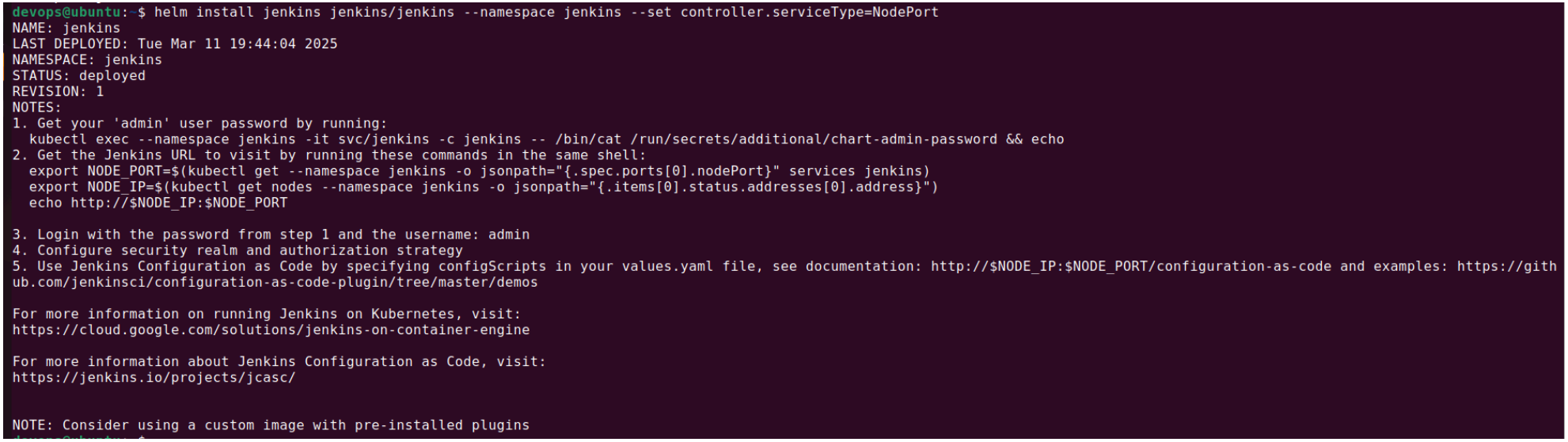
(Screenshot: Creating Kubernetes namespace)



Step 3: Install Jenkins Using Helm

```
helm install jenkins jenkins/jenkins --namespace jenkins --set controller.serviceType=NodePort
```

(Screenshot: Installing Jenkins with Helm)

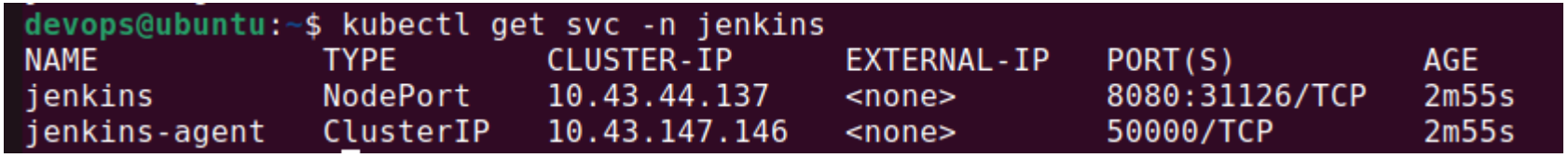


Step 4: Access Jenkins

Find the Jenkins service port:

```
kubectl get svc -n jenkins
```

(Screenshot: Checking Jenkins service details)



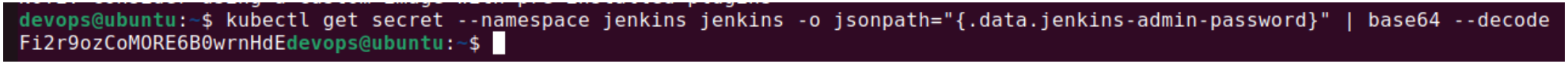
You should see an output like this:

```
NAME      TYPE       CLUSTER-IP      EXTERNAL-IP  PORT(S)          AGE
jenkins   NodePort   10.43.150.139   <none>       8080:32000/TCP   5m
```

Step 5: Get Admin Password

```
kubectl get secret --namespace jenkins jenkins -o jsonpath="{.data.jenkins-admin-password}" | base64 --decode
```

(Screenshot: Retrieving admin password)

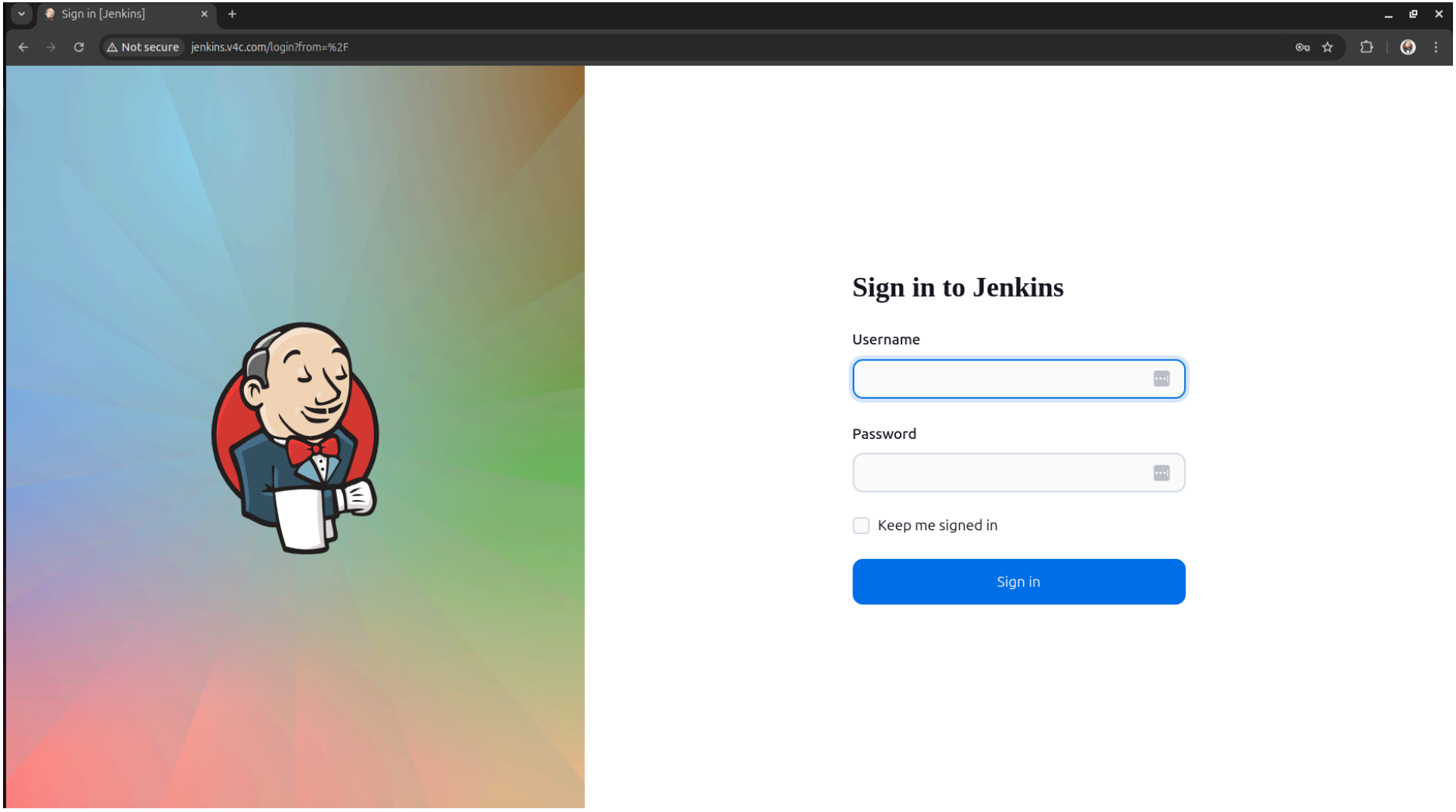


Log in to Jenkins as a Admin User

1. Open your web browser and go to your Jenkins URL.

   `http://<your-node-ip>:32000`
2. Enter your **Admin Username** and **Password**, then click **Sign in**.

   The default username is usually **admin**.



Add Plugins in Jenkins

Jenkins allows you to **extend its functionality** using plugins. Adding plugins can help with **CI/CD, integrations, reporting, notifications, security, and more**.

Installing Required Plugins for a Maven Project in Jenkins

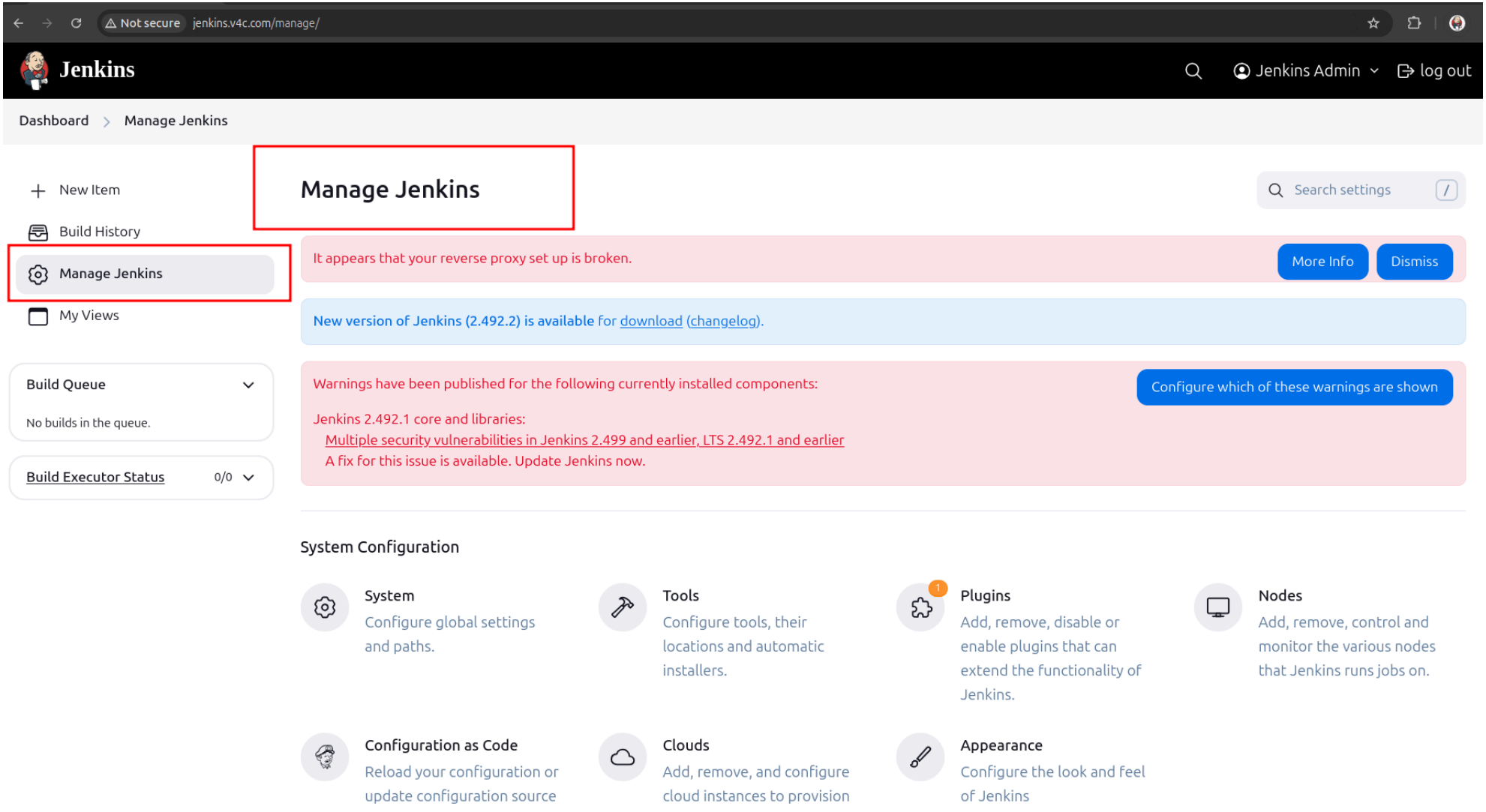
We are deploying a **Maven application** and require three essential plugins:

1. **GitHub Plugin**
   - Enables **integration between Jenkins and GitHub**.
   - Supports **automated builds** triggered by:
     - Repository changes
     - Pull requests
     - Webhook events.
2. **Maven Plugin**
   - Allows **building, testing, and packaging** Maven-based projects in Jenkins.
   - Ensures smooth execution of the **Maven lifecycle** (`clean`, `package`, `test`, `install`).
3. **Matrix Authorization Strategy Plugin**
   - Enables **fine-grained access control** in Jenkins.
   - Lets administrators define **user-specific permissions** for jobs, builds, and configurations.
   - Useful for restricting access based on **roles (e.g., Developer, DevOps)**.

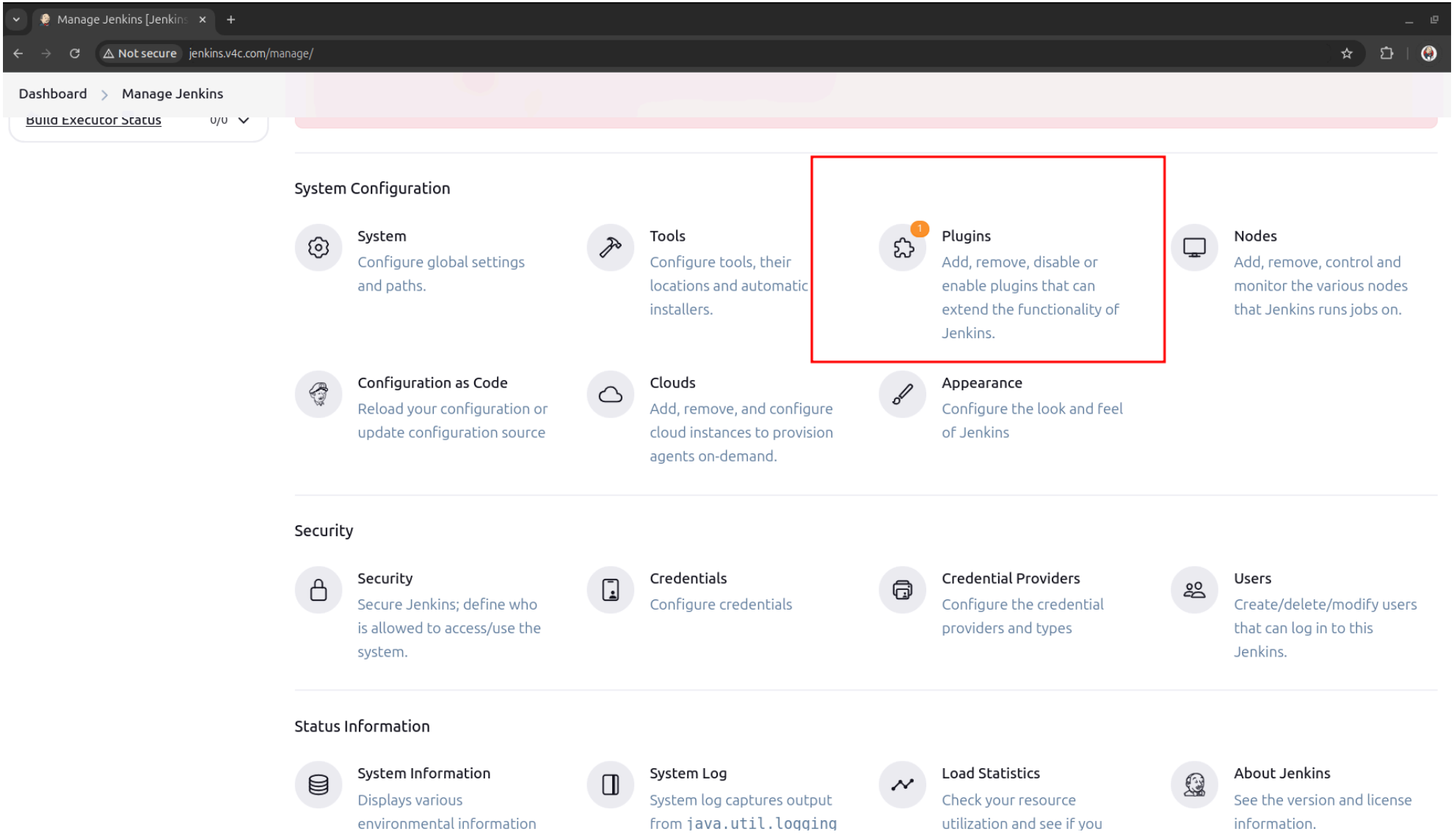
Here’s a simple step-by-step guide for install plugins:

Go to Manage Plugins

In the left-hand menu, click **"Manage Jenkins"**.

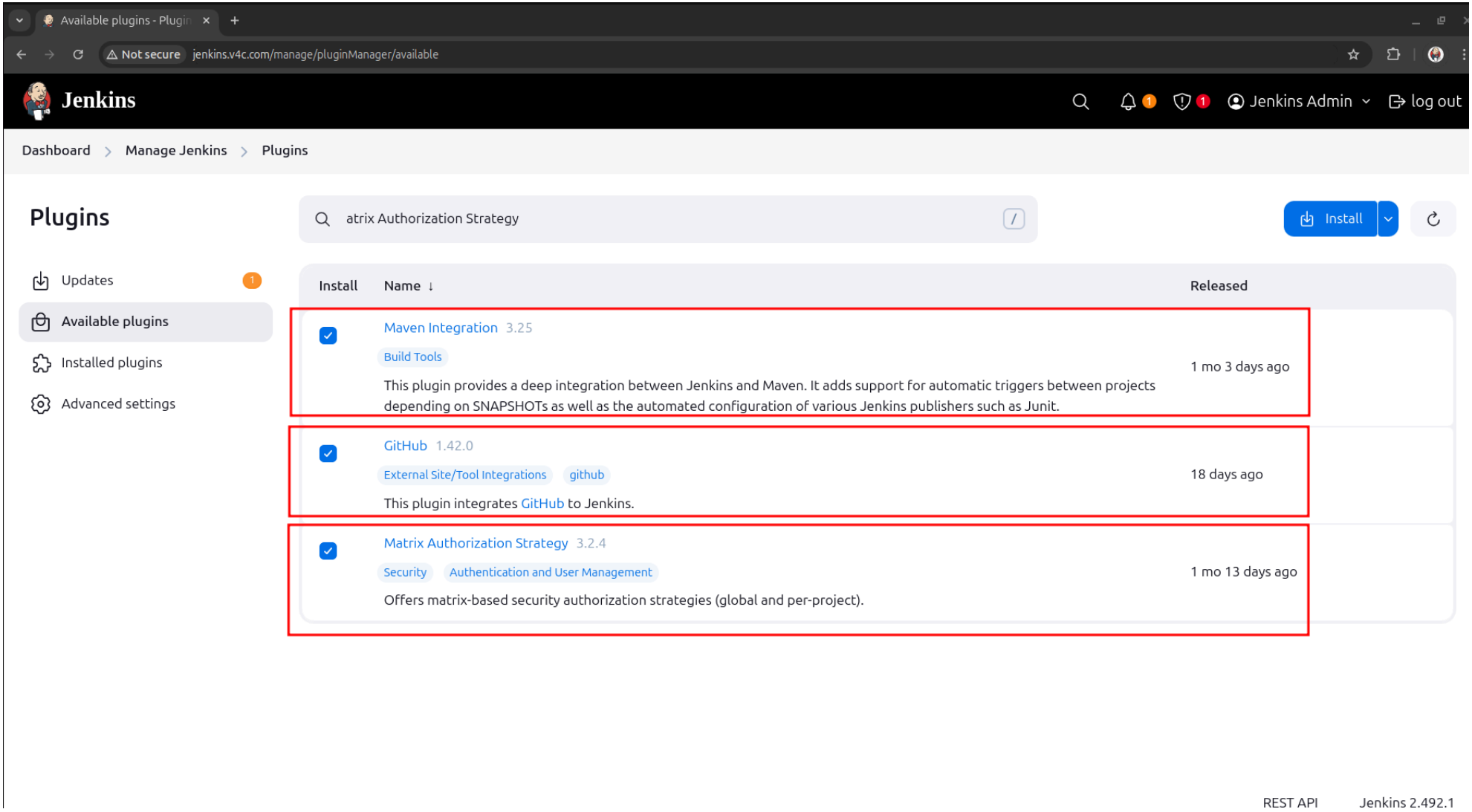


Scroll down and click **"Manage Plugins"**.

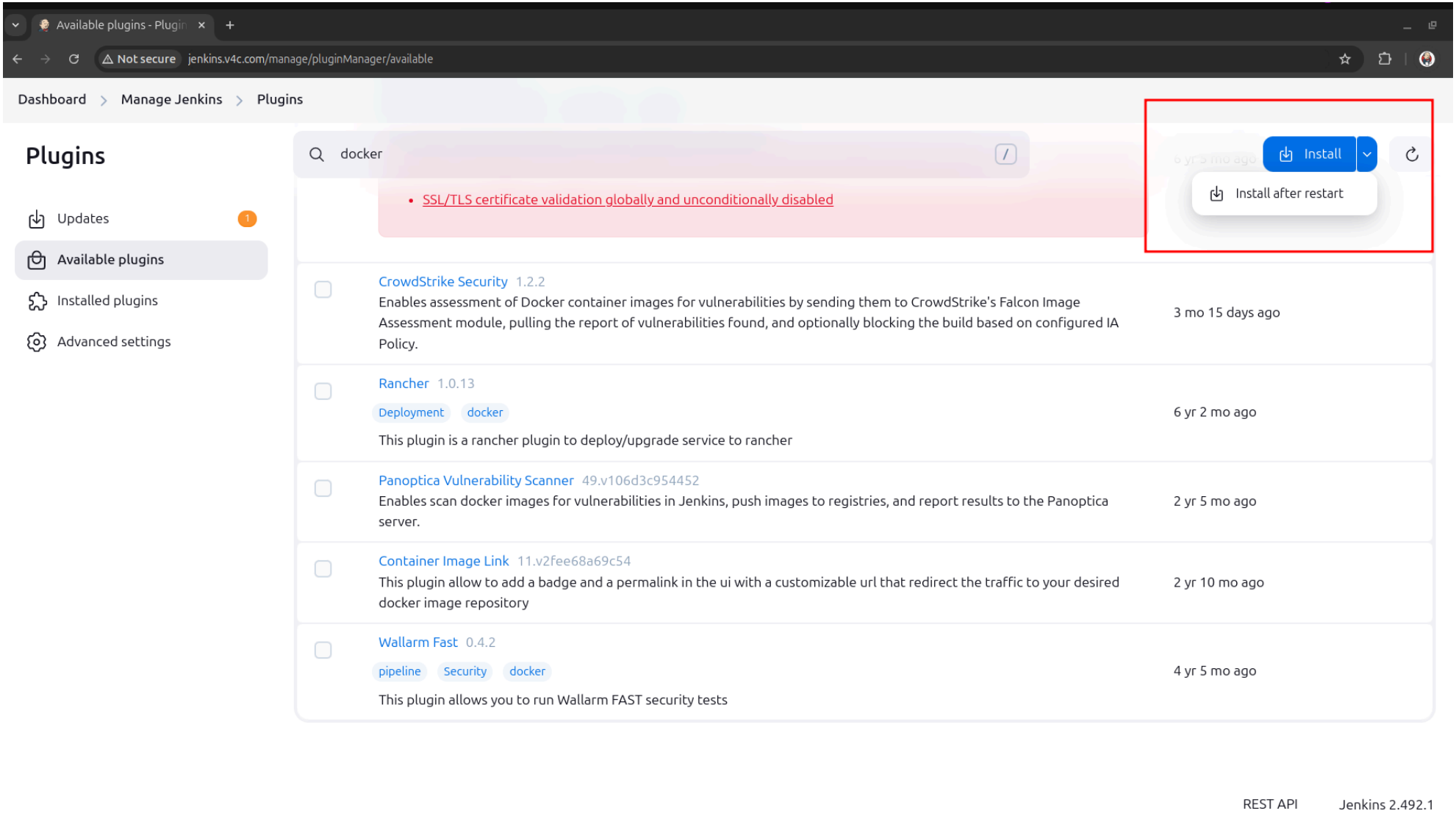


Install a Plugin

Click the **"Available"** tab (this shows plugins you can install).
Use the **search bar** to find the plugin you want (e.g., "GitHub", "Maven", "Matrix Authorization Strategy Plugin").
Check the box next to the plugin name.



Click **"Install without restart"** (or "Download now and install after restart").



Restart Jenkins (If Needed)

- Some plugins require a restart. If prompted, **click "Restart Jenkins when installation is complete"**.
- Alternatively, you can manually restart Jenkins on **Linux** by running: `sudo systemctl restart jenkins`

This will restart the Jenkins service and apply any newly installed plugins.

User Management in Jenkins

What Are We Doing in This Section?

In this section, we will **create two users** in Jenkins and assign them different roles using **Matrix Authorization Strategy**. This is important for **role-based access control (RBAC)**, ensuring that users have the right level of access based on their responsibilities.

By the end of this section, we will have:

- ✅ A **Developer (`dev_user`)** who can **view and trigger builds** but **cannot modify configurations**.
- ✅ A **DevOps Engineer (`deploy_user`)** who can **configure jobs, approve deployments, and troubleshoot issues**.

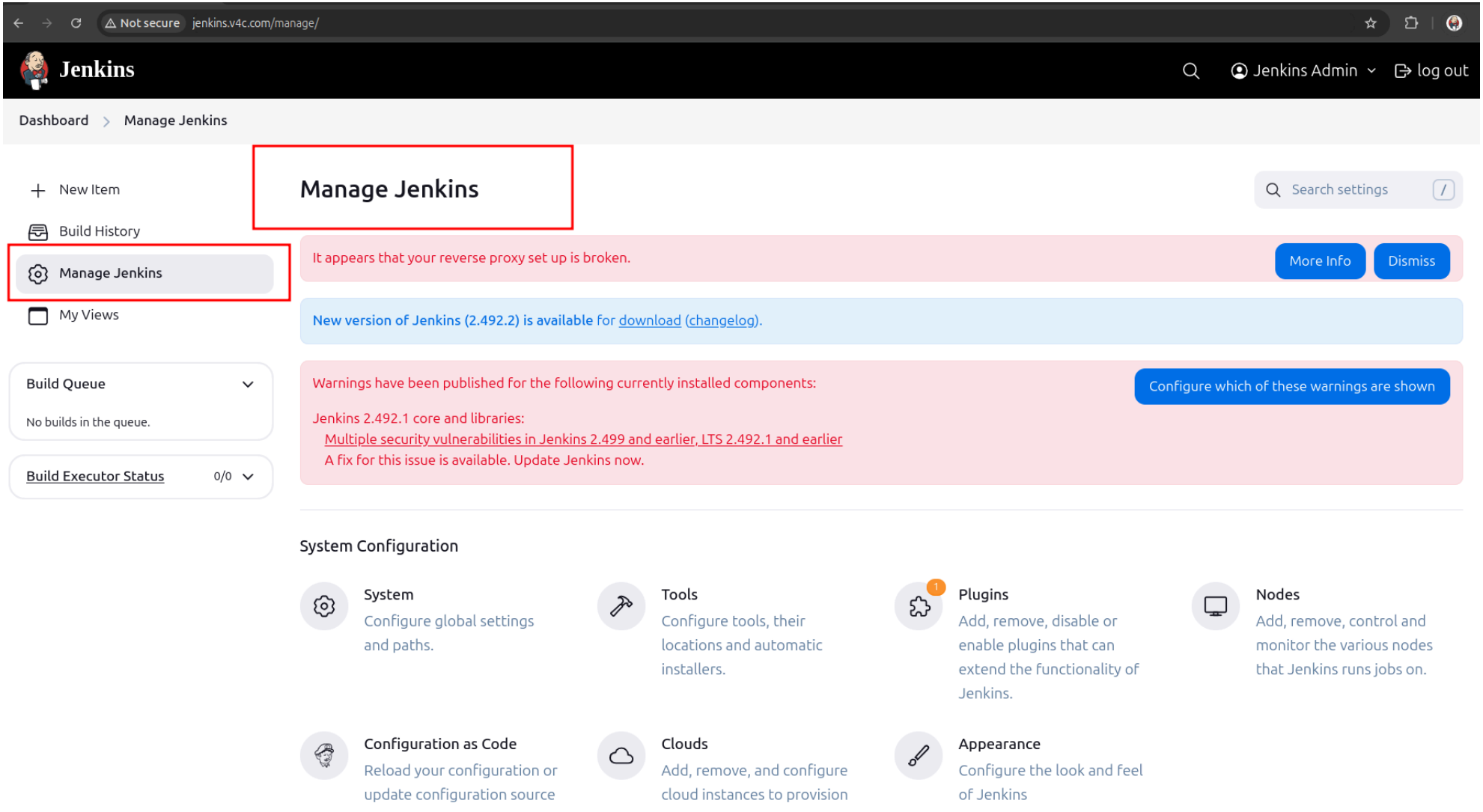
Why Is This Important?

- **Security:** Prevents unauthorized changes to Jenkins jobs.
- **Accountability:** Ensures each action is logged under the correct user.
- **Controlled CI/CD Workflow:** Separates development work from deployment management.

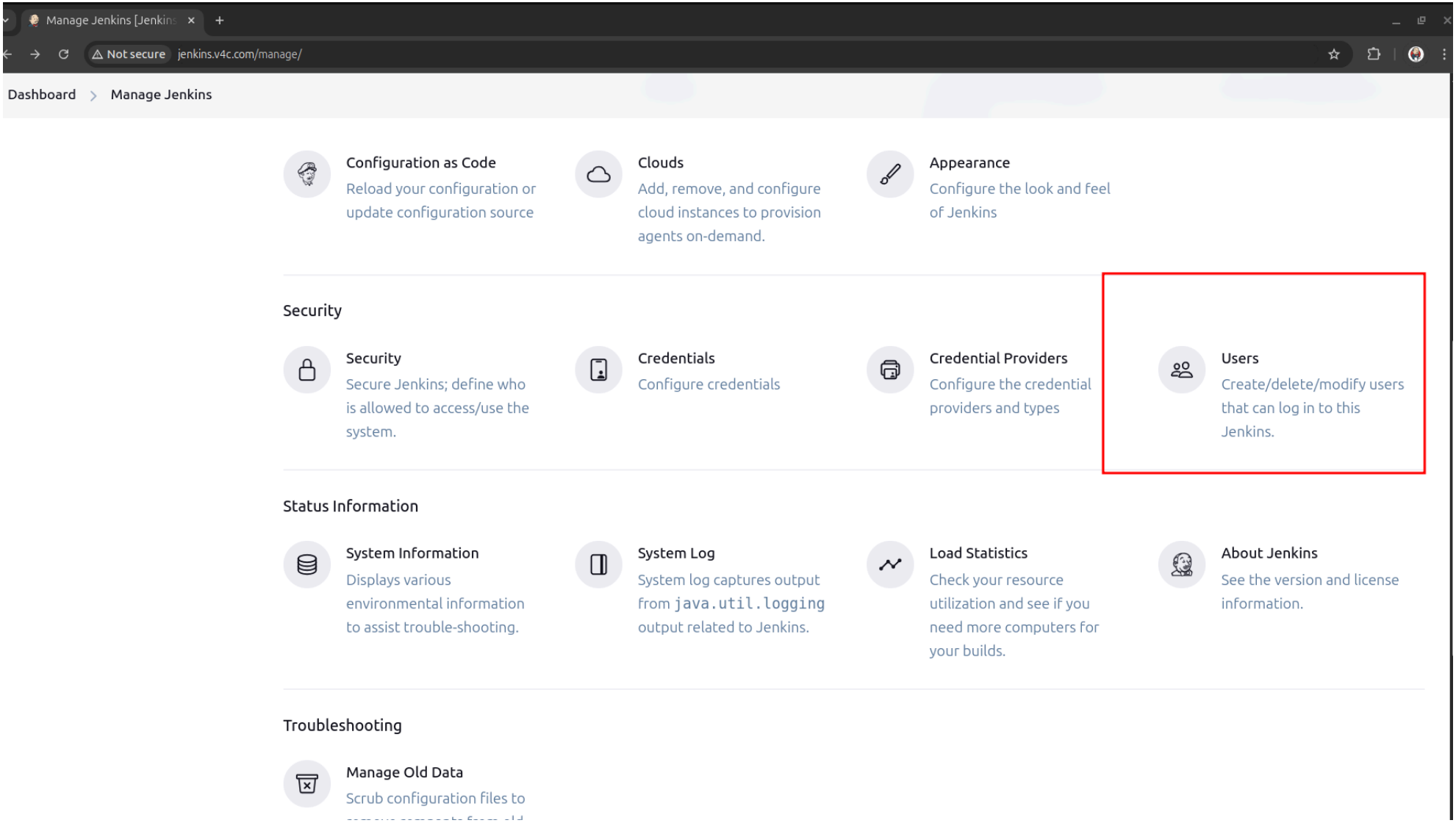
Create Two Users

Navigate to User Management

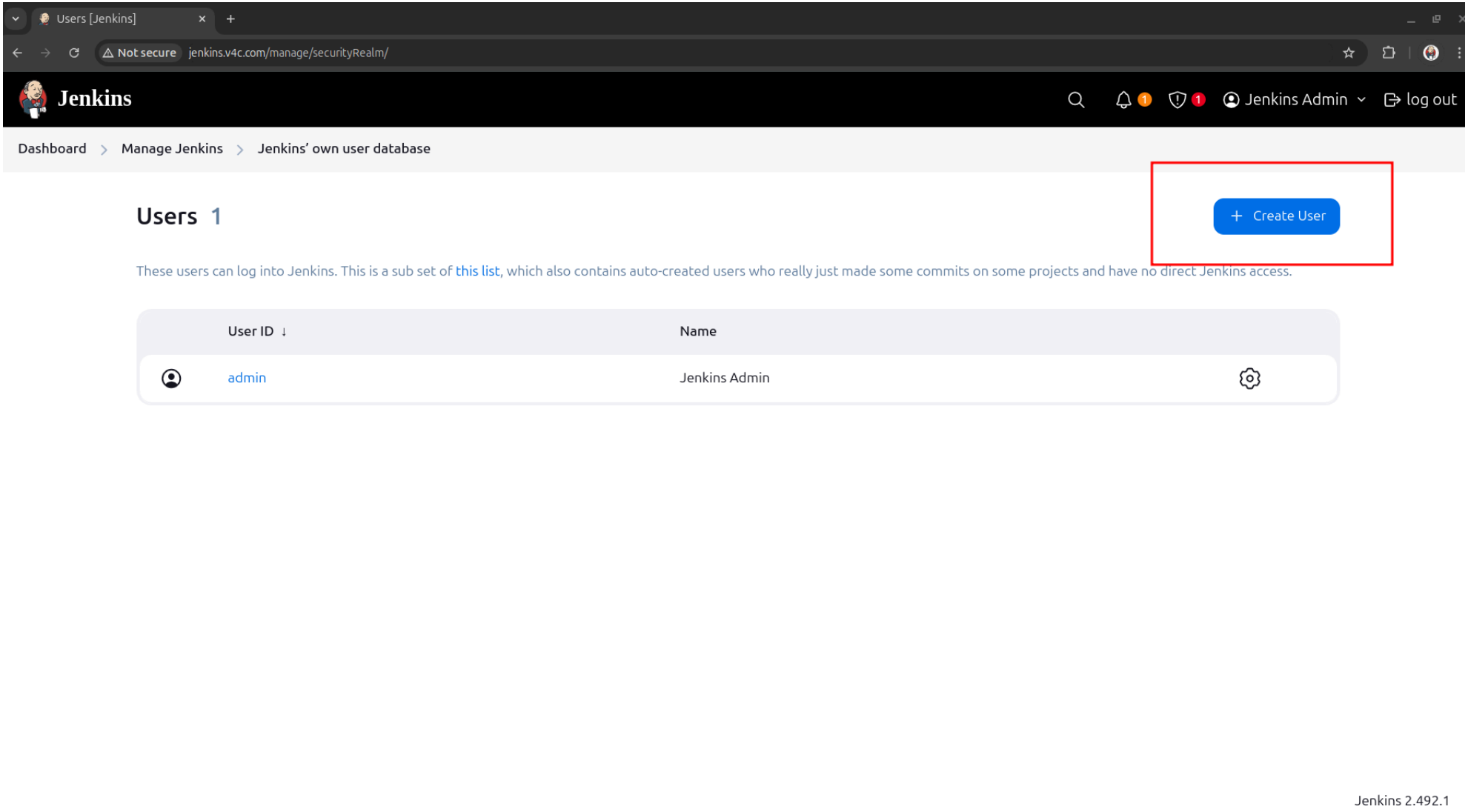
click on **"Manage Jenkins"** from the left sidebar.



Scroll down and click on **"Manage Users"** under the **Security** section.



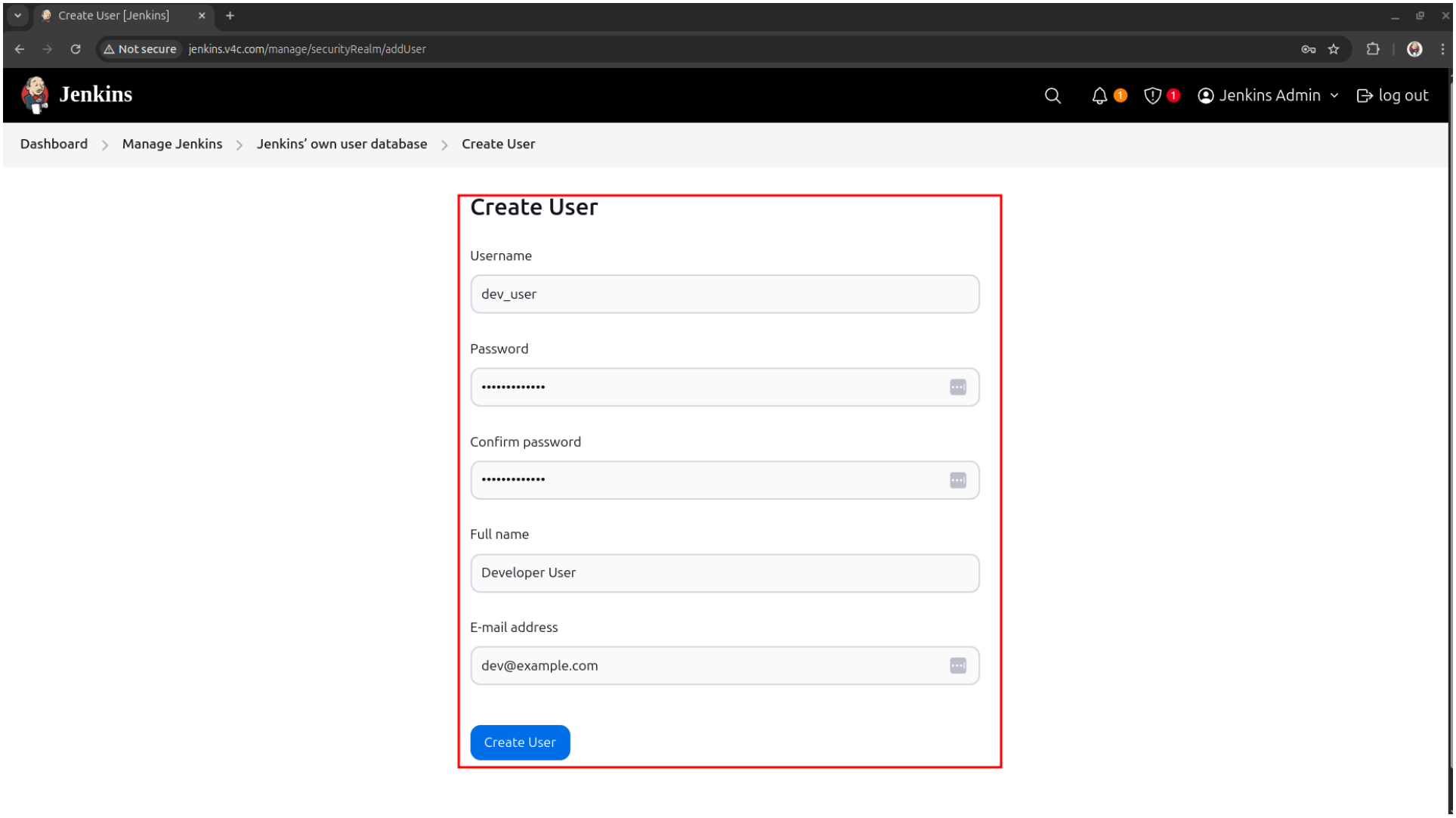
Click on **"Create User"** on the left side.



Create Developer User

1. In the **Manage Jenkins** section, navigate to **Manage Users**.
2. Click **Create User** on the left panel.

3. Fill in the following details:
   - **Username:** `dev_user`
   - **Password:** Set a secure password (e.g., `mypassword123`).
   - **Confirm Password:** Re-enter the same password.
   - **Full Name:** `Developer User`
   - **Email Address:** `dev@example.com`



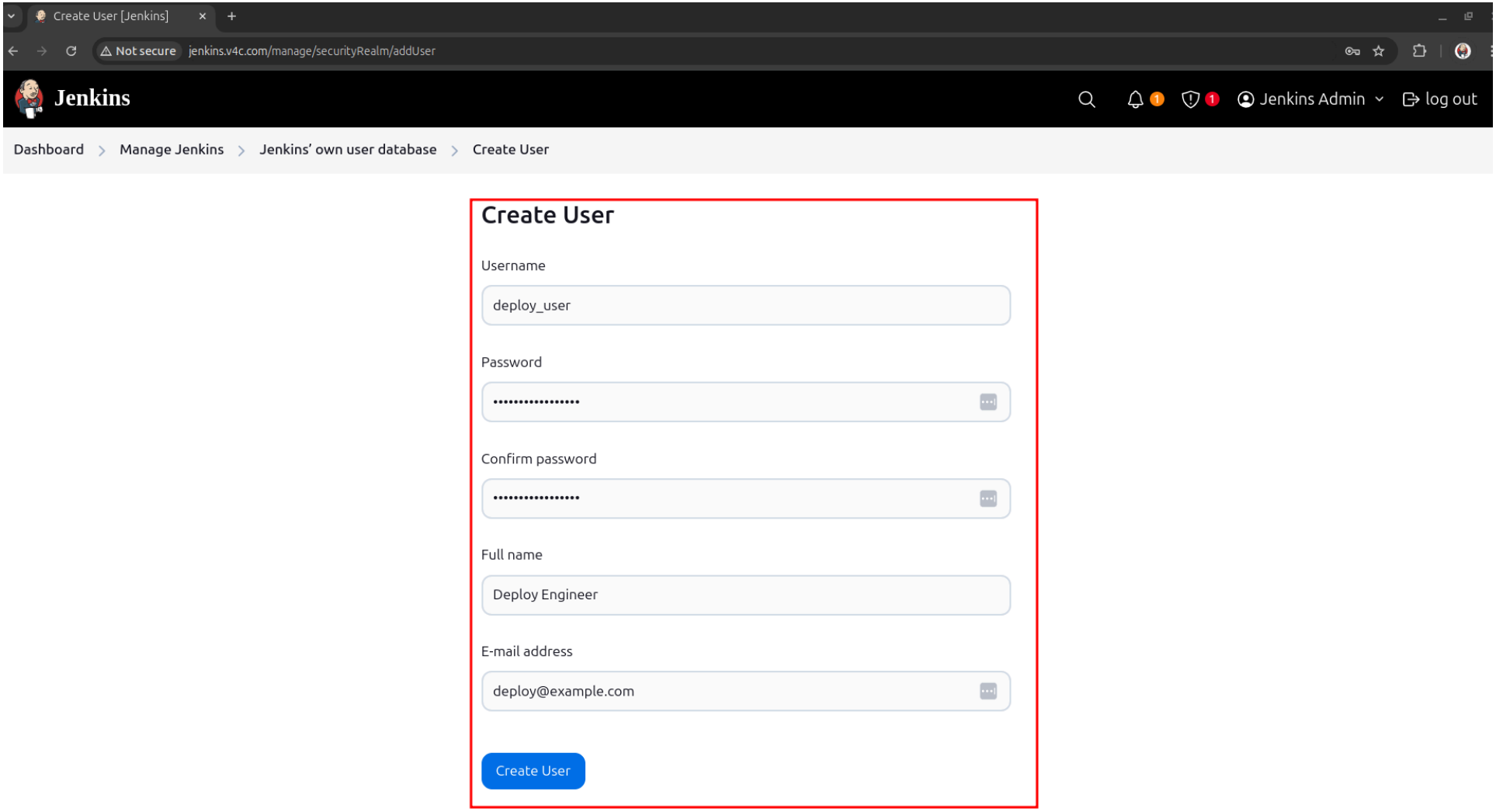
4. Click **Create User** to save the new user.

This user will have **limited permissions**, allowing them only to **view and trigger builds** but not modify configurations.

Create DevOps Engineer User

1. Go to **Manage Jenkins > Manage Users**.
2. Click **Create User**.
3. Enter the following details:
   - **Username:** `deploy_user`
   - **Password:** Set a secure password (e.g., `deploypassword123`).
   - **Confirm Password:** Re-enter the password.
   - **Full Name:** `Deploy Engineer`

- **Email Address:** `deploy@example.com`



4. Click **Create User** to save the new user.

This user will have **more privileges**, including the ability to **configure jobs, approve deployments, and troubleshoot issues**.

Now that both users are created, we will proceed to assign them **role-based access** using **Matrix Authorization Strategy**.

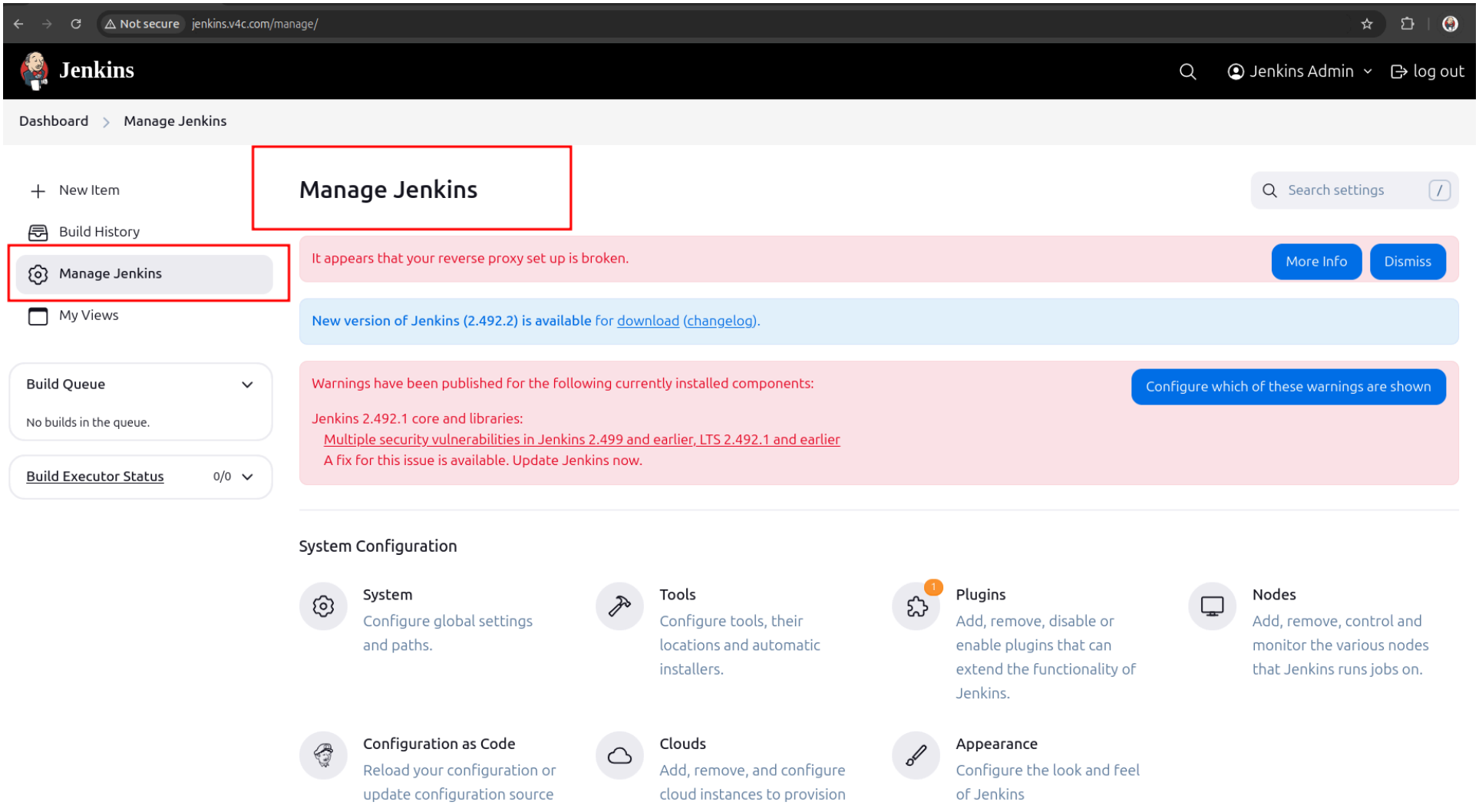
Configure User Permissions

Now that we have installed the required **plugins** and created **two users** (`dev_user` and `deploy_user`), the next step is to **configure role-based access control** using the **Matrix Authorization Strategy Plugin**.

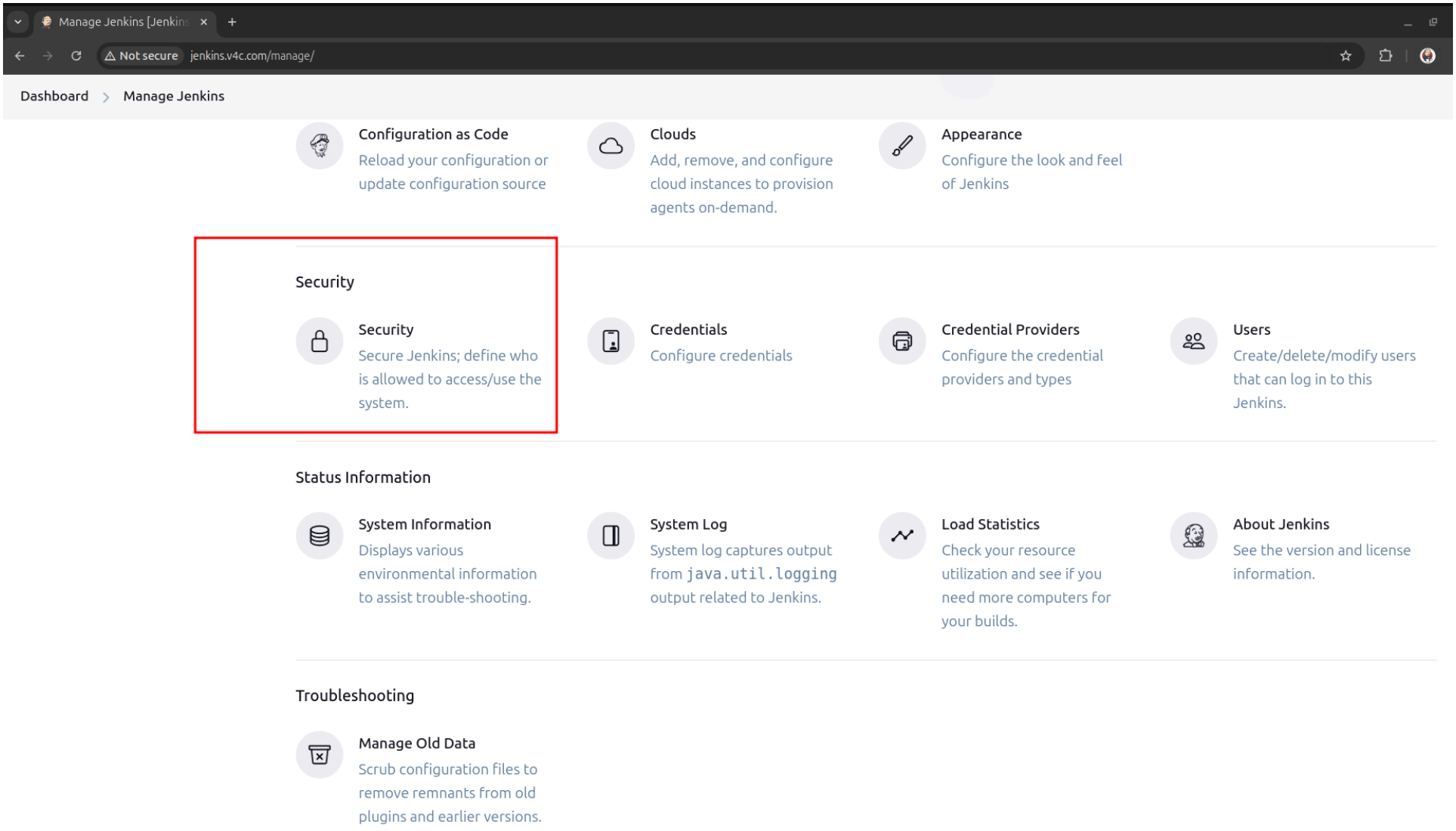
We will now:
✅ Enable **Matrix Authorization Strategy** in Jenkins.
✅ Assign **limited permissions** to `dev_user` (Developer).
✅ Assign **elevated permissions** to `deploy_user` (DevOps Engineer).

Step 1: Enable Matrix Authorization Strategy

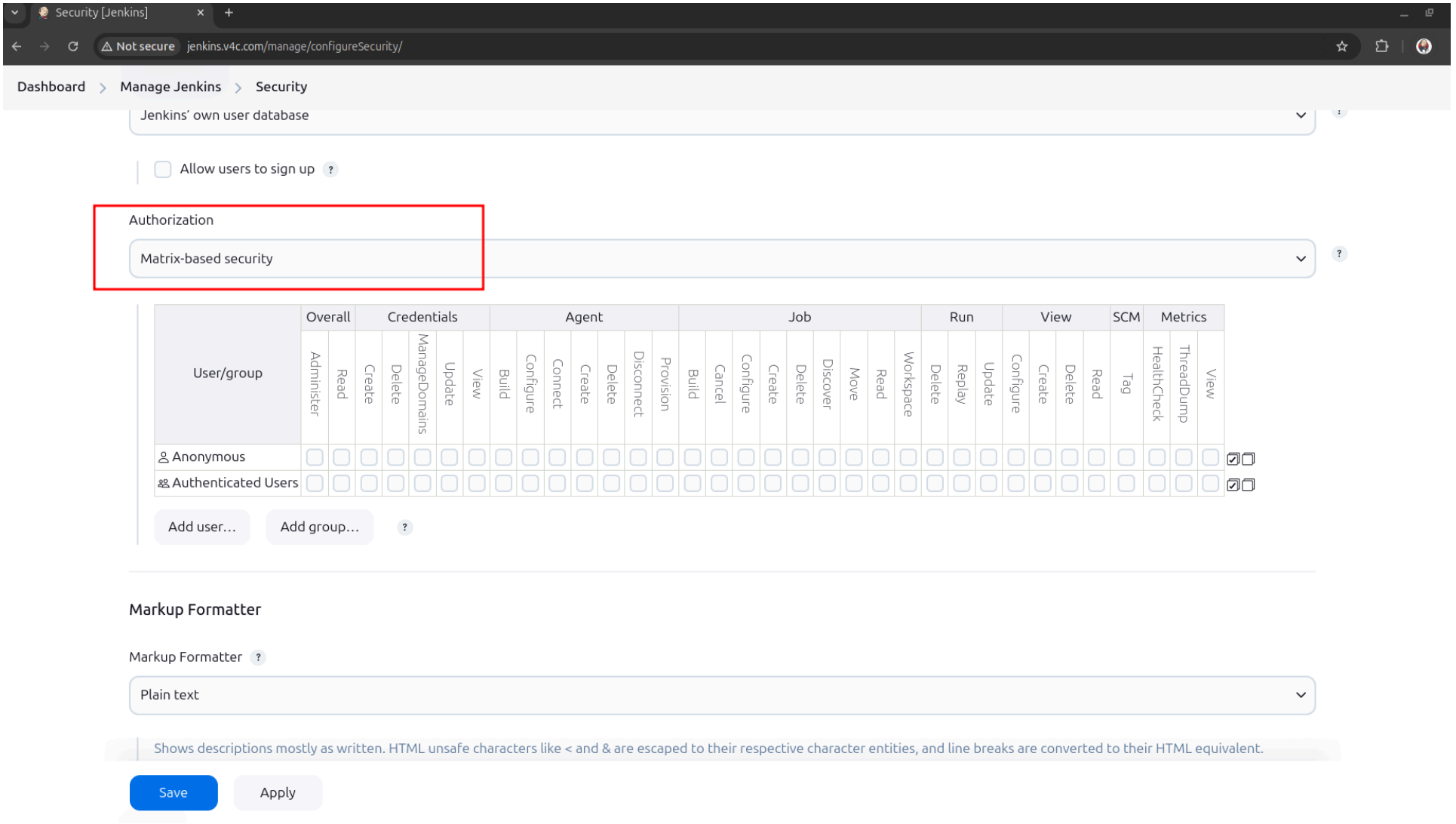
1. **Go to** Jenkins Dashboard → Click **Manage Jenkins**.



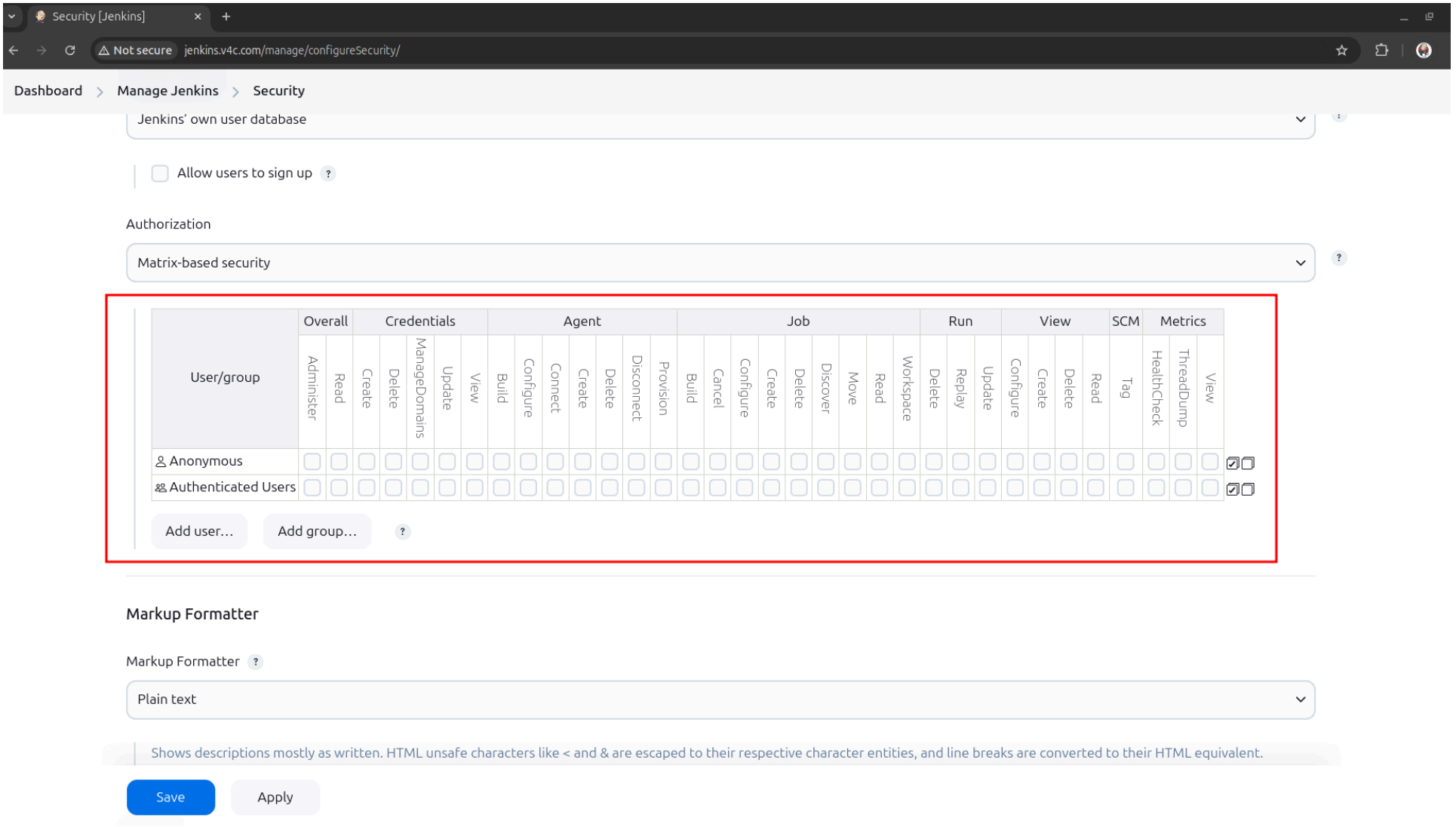
2. Scroll down and click **Configure Global Security**.



3. Under the **Authorization** section, select **Matrix-based security**.



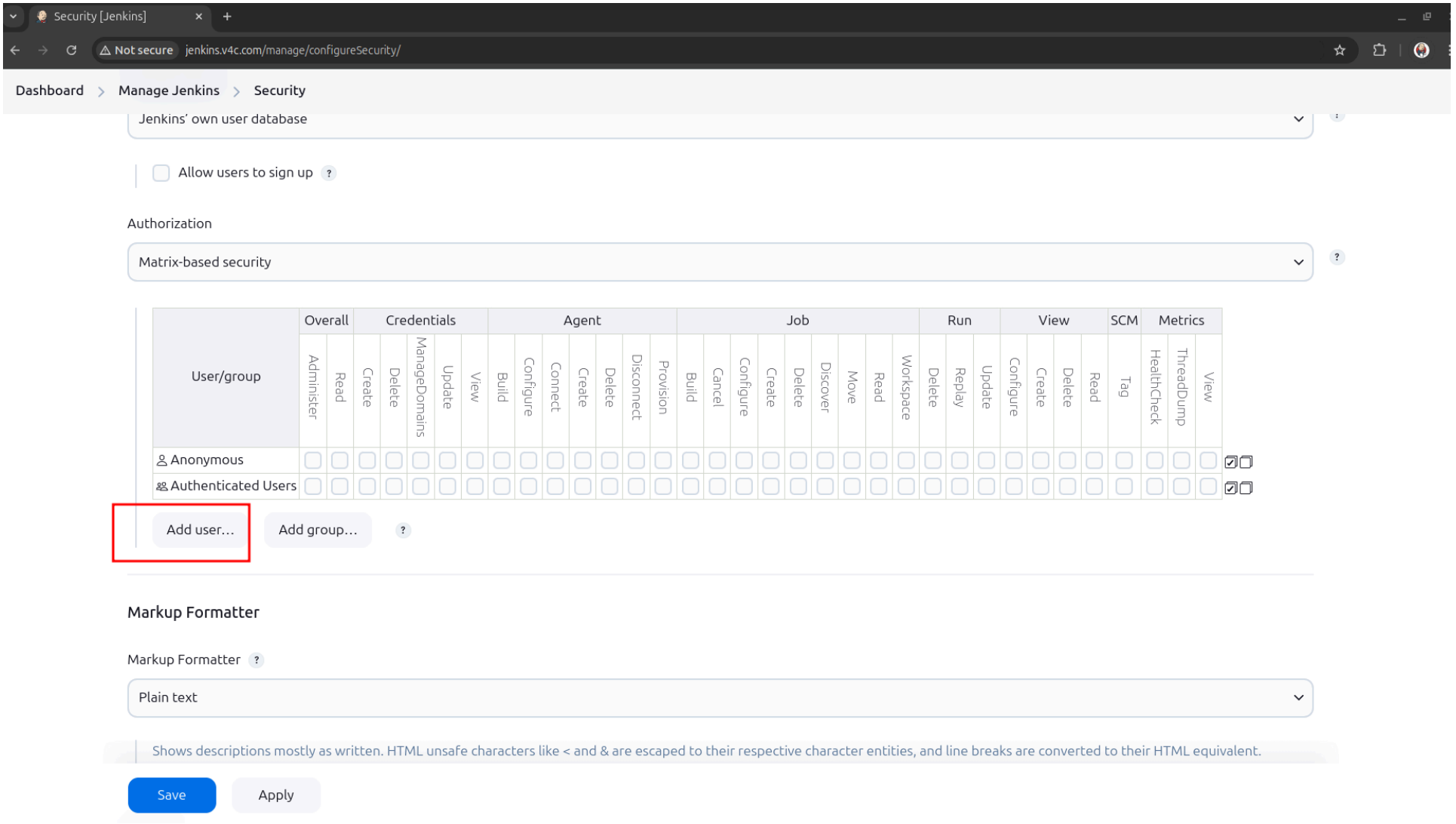
4. A table will appear where we can define permissions for each user.

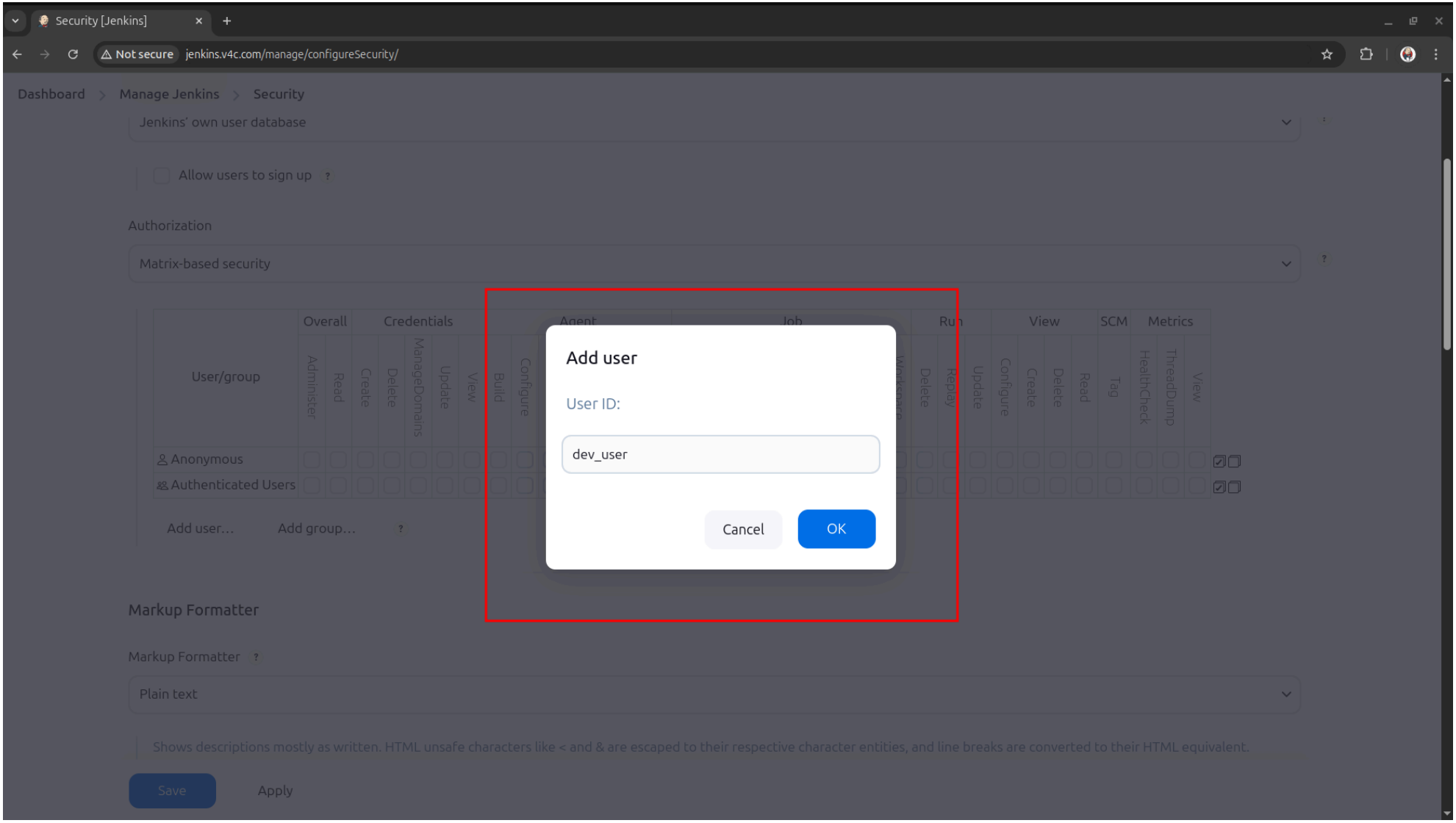


Step 2: Assign Permissions

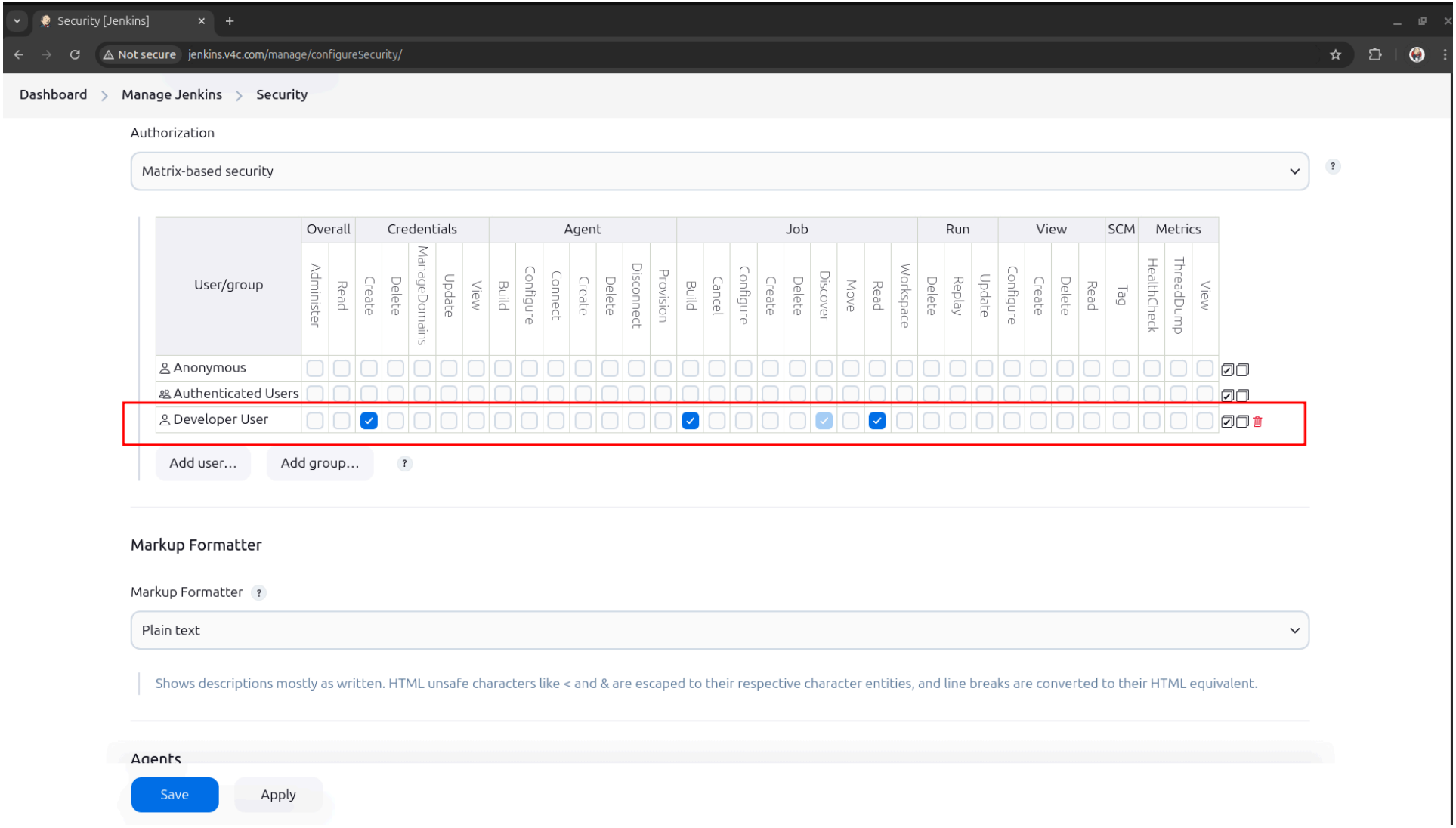
Assign Developer (`dev_user`) Permissions

- Click **Add user…** → Type `dev_user` → Click OK.





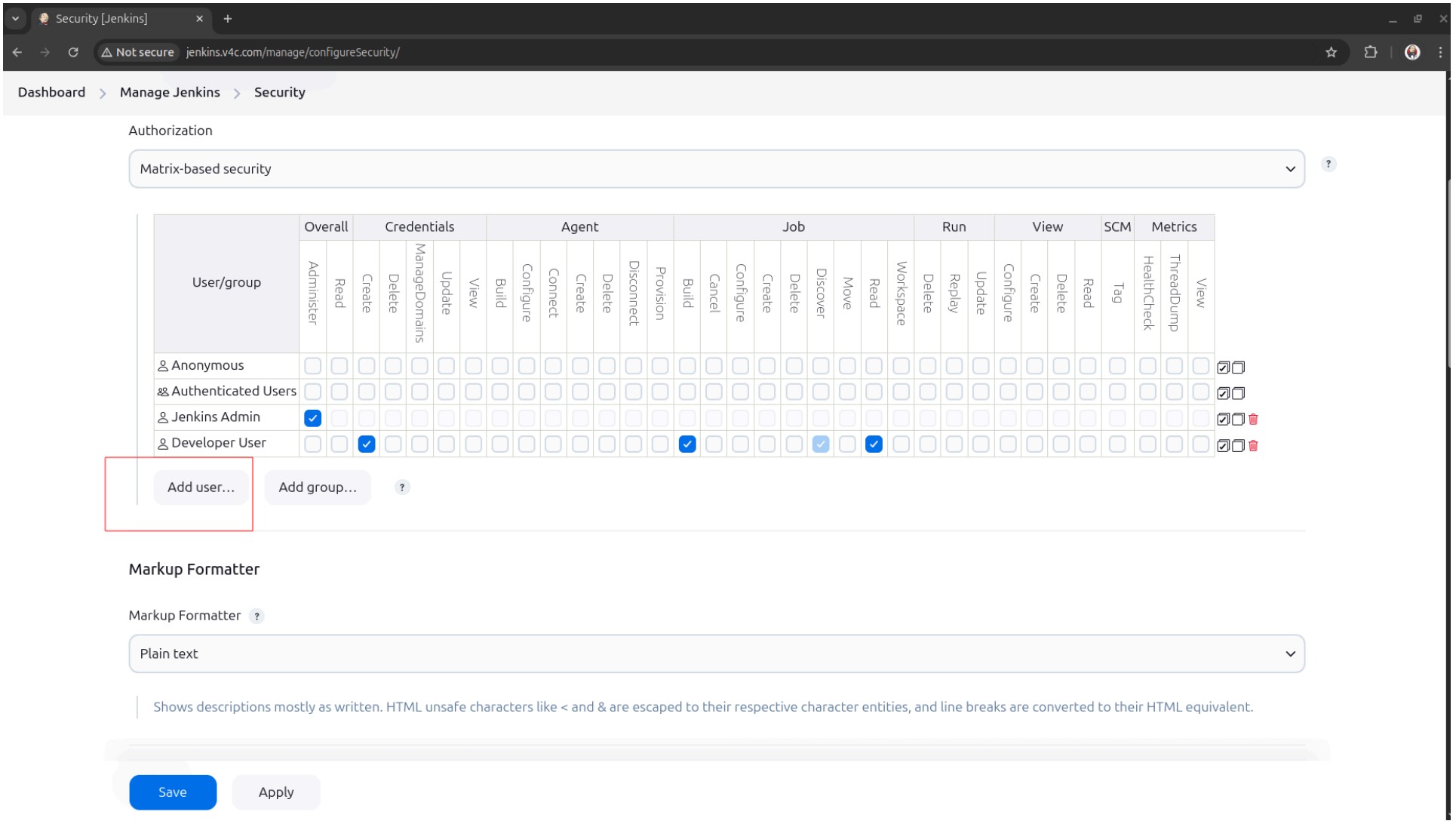
- Assign **basic permissions**:
  - ✅ Read
  - ✅ Job: **Build**
  - ✅ Job: **Discover**
  - ✅ Job: **Read**

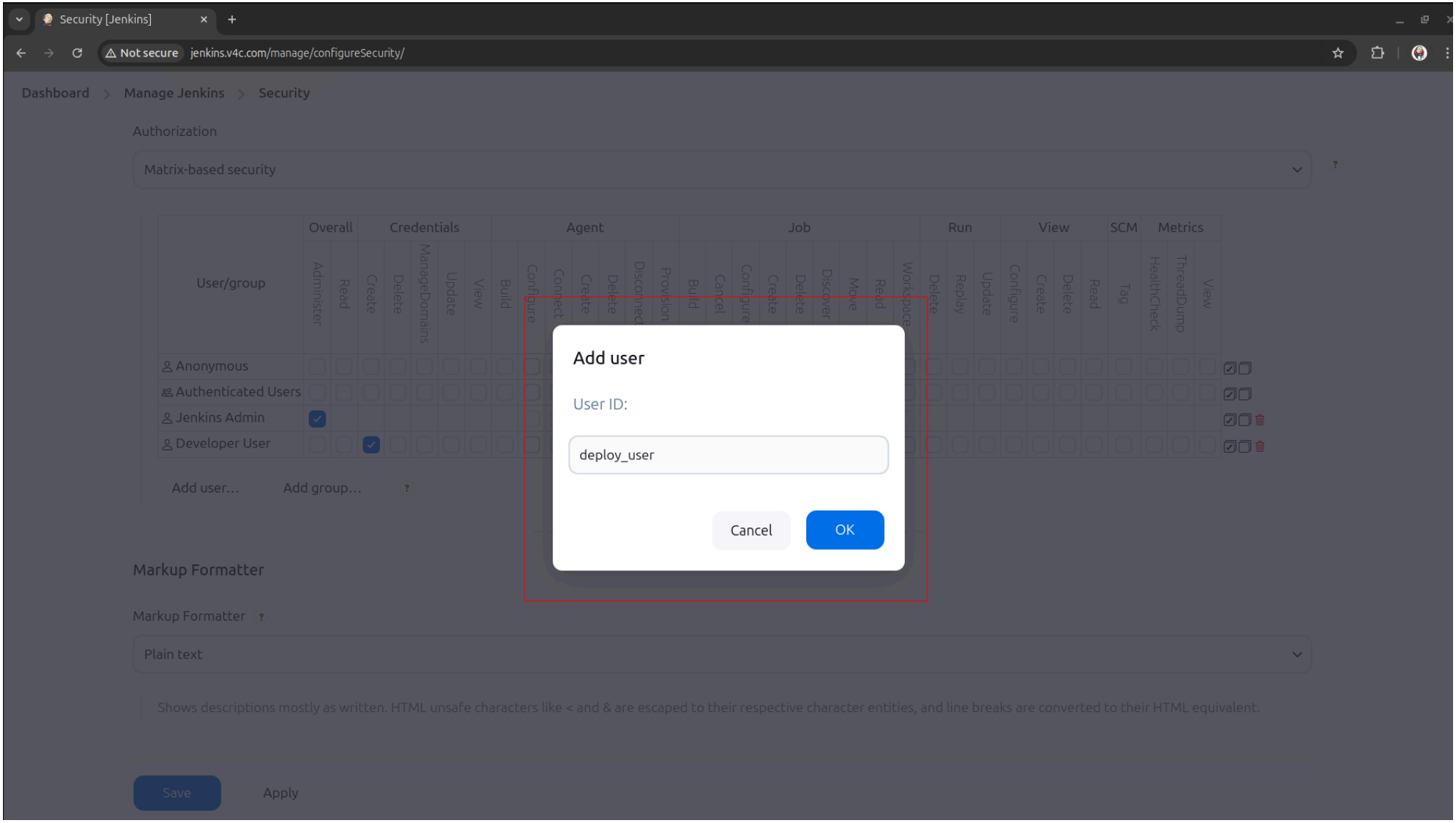


This ensures the **developer can trigger builds but cannot modify configurations**.

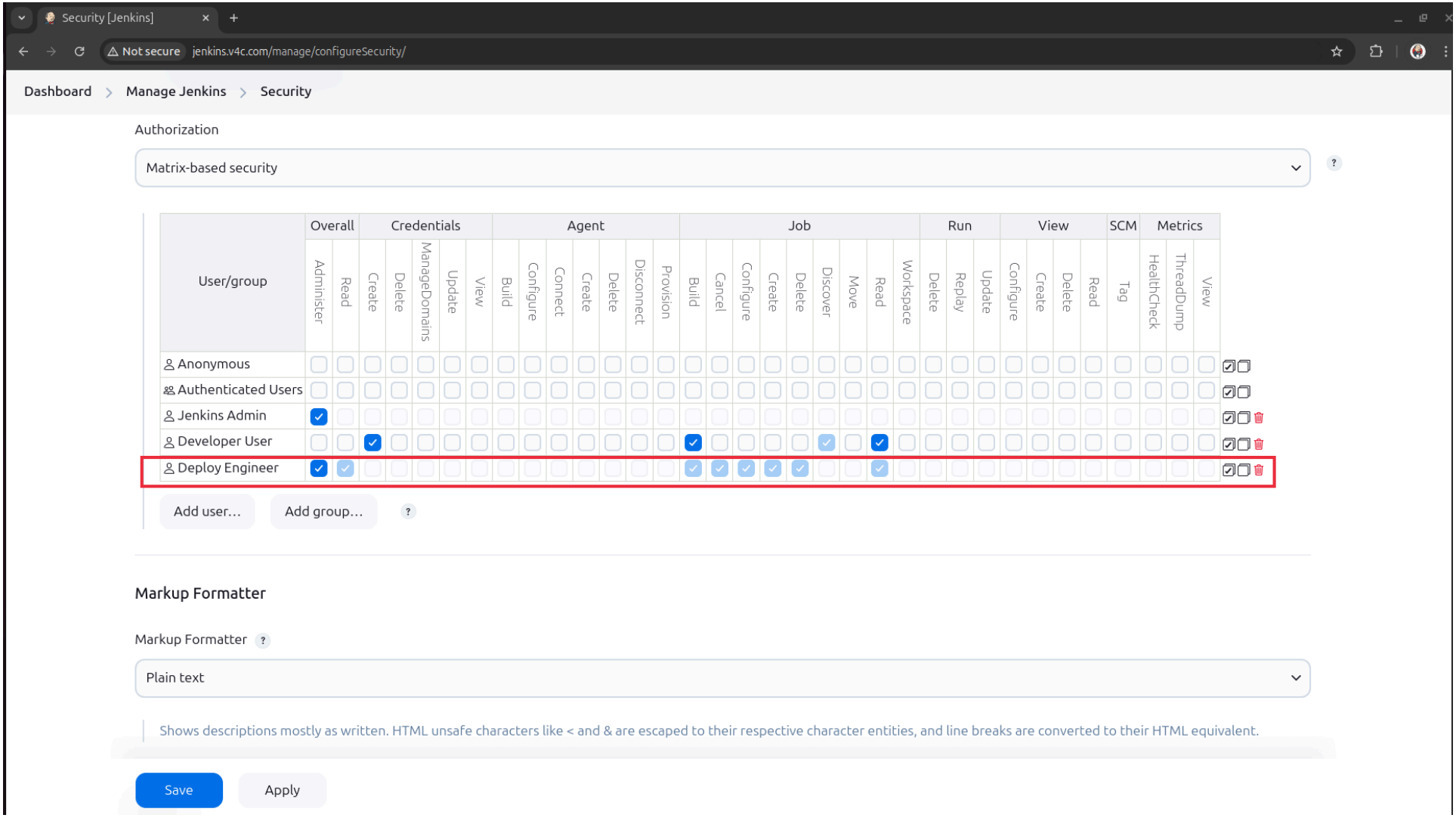
Assign DevOps Engineer (`deploy_user`) Permissions

- Click **Add user…** → Type `deploy_user` → Click OK.





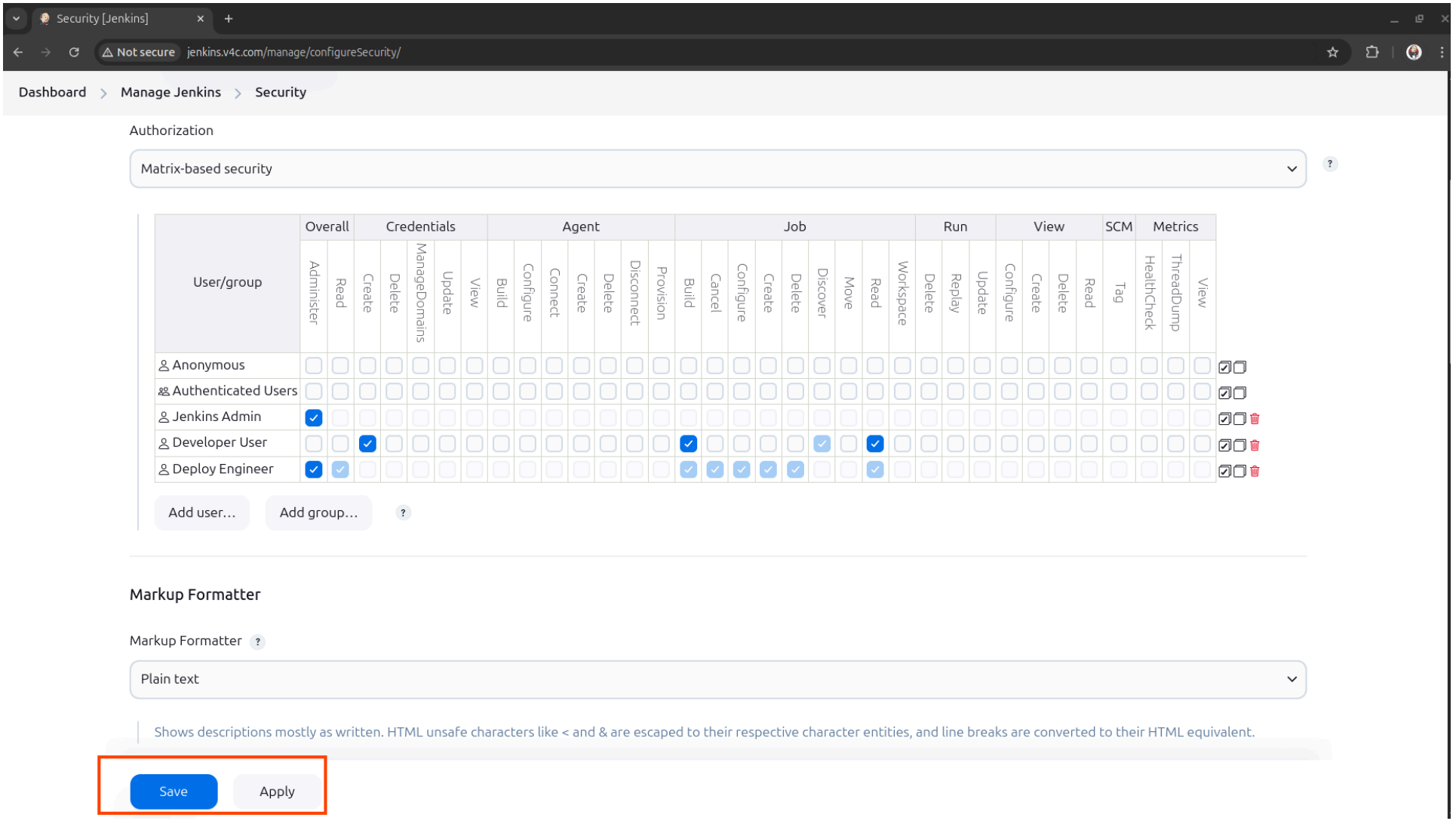
- Assign **advanced permissions**:
  - ✅ Read
  - ✅ Job: **Build, Configure, Delete**
  - ✅ Job: **Read, Create, Cancel**
  - ✅ Administer (**optional, if full access is needed**)



This allows the **DevOps Engineer to manage builds, edit jobs, and approve deployments**.

Save & Apply Changes

1. **Scroll down** and click **Save**.



2. Log out and test the permissions by logging in as `dev_user` and `deploy_user`.

Once this is done, we will proceed to **create a Jenkins pipeline for the Maven project.**

Create a Jenkins Pipeline for the Maven Project

We will now:
✅ Create a **new pipeline job** in Jenkins.
✅ Configure it to **fetch the Maven project from GitHub**.
✅ Set up **Maven build steps** to compile, test, and package the project.

Create a New Pipeline

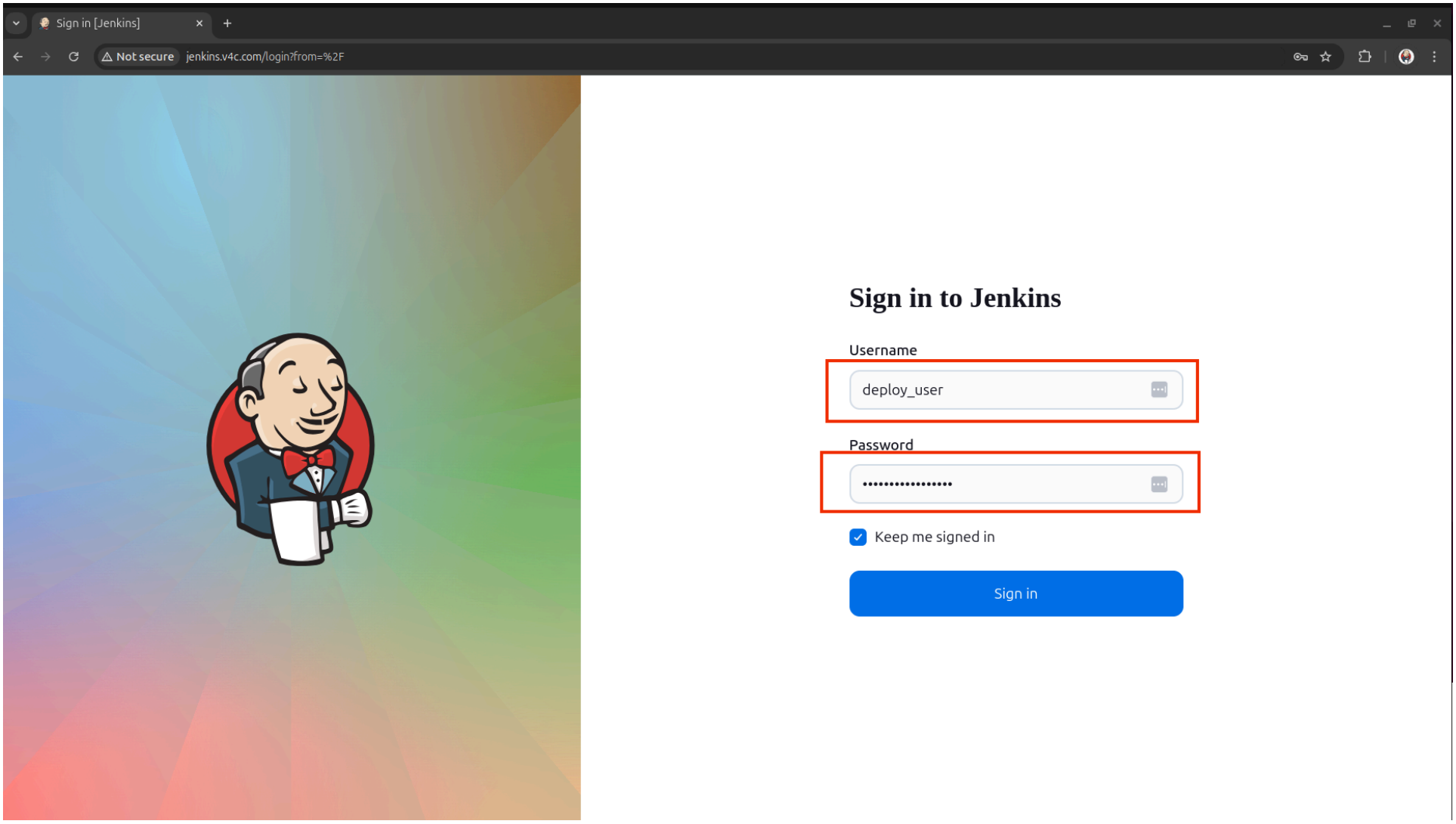
Now that we have installed the necessary plugins and created users, it's time to **set up a Jenkins Pipeline** to build and test the Maven project. Follow the steps below:

**Log in to Jenkins again

1. Open a web browser and navigate to your Jenkins URL:

   ```
   http://<your-jenkins-server>:8080
   ```

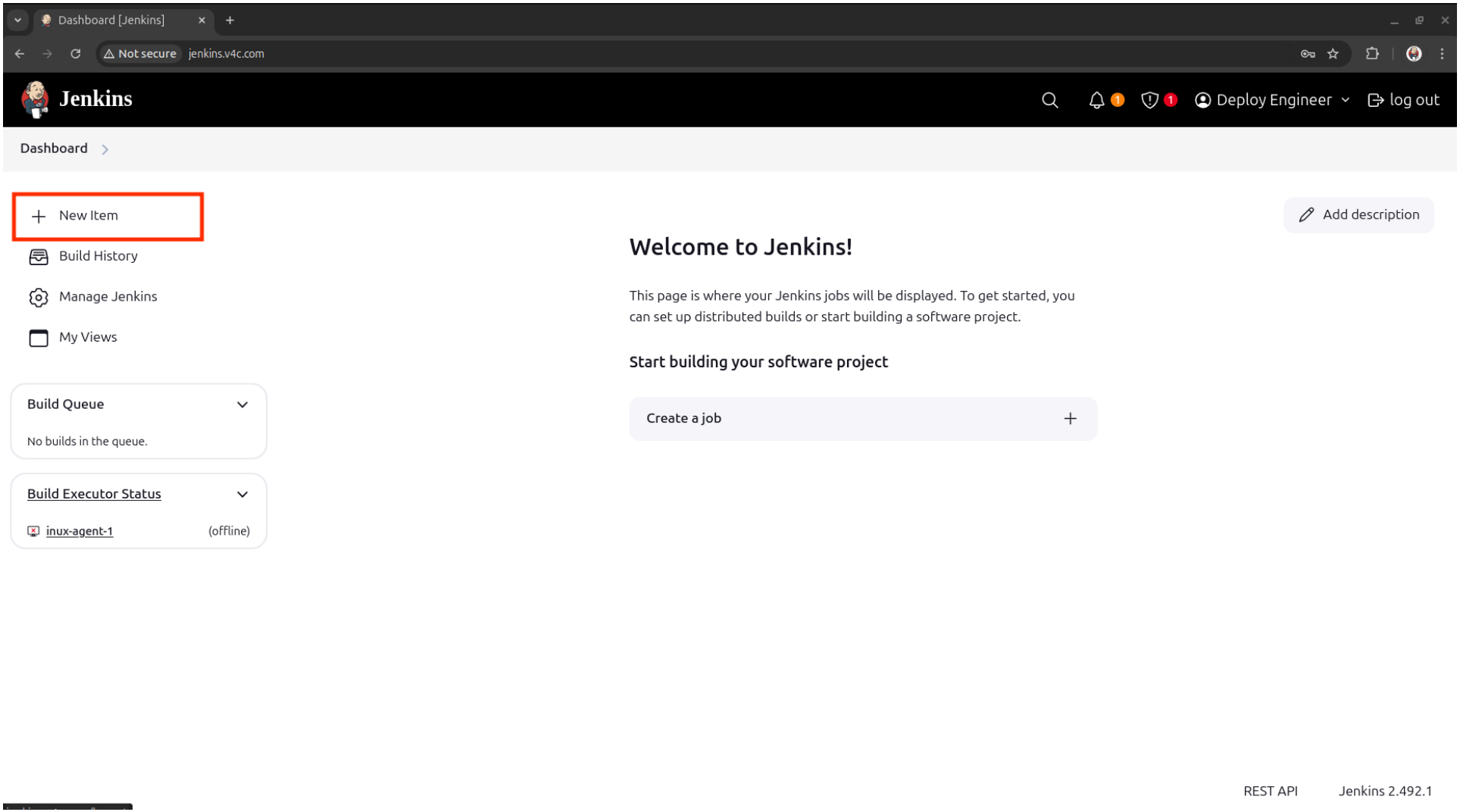
2. Enter the **deploy_user** credentials (Username: `deploy_user`, Password: `deploypassword123`).



3. Click **Sign in**.

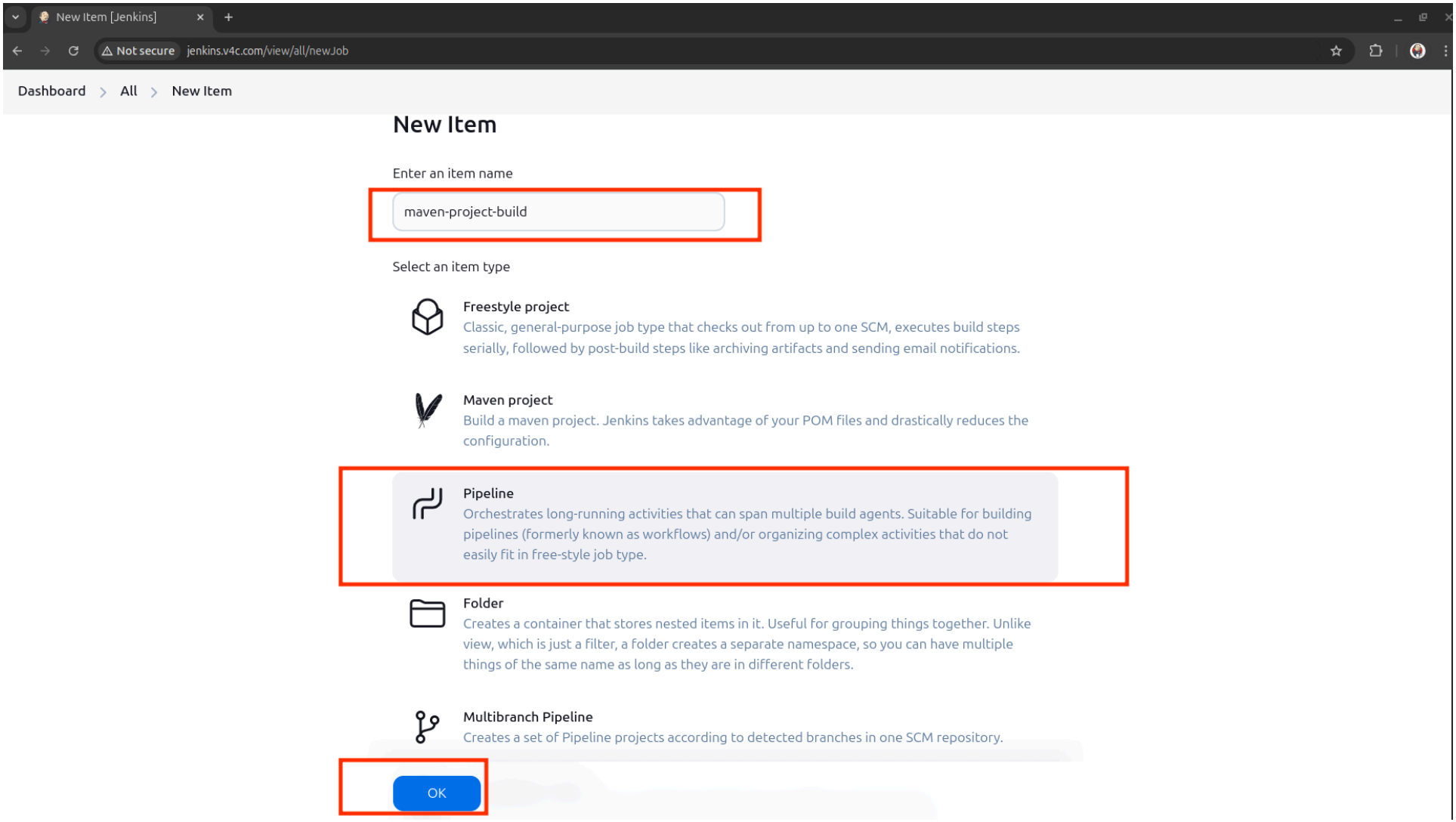
Create a New Pipeline Job

1. After logging in, click on **"New Item"** from the Jenkins dashboard.



2. In the **Enter an item name** field, provide a descriptive name for the pipeline.
   - Example: `maven-project-build`.

3. Under **"Type"**, select **Pipeline** (since we are setting up a CI/CD pipeline for a Maven project).



4. Click **OK** to proceed.

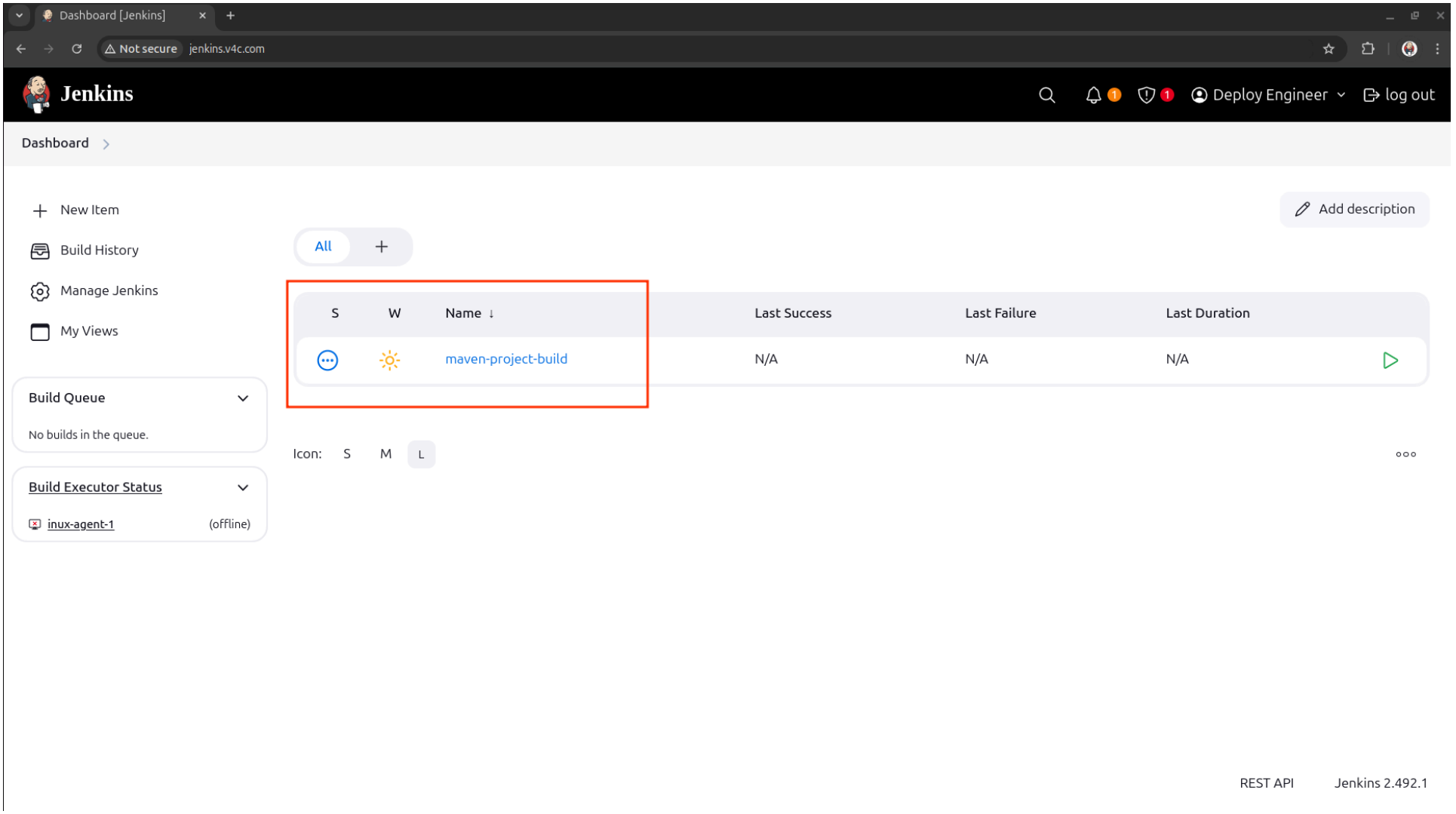
Link GitHub Repository to Jenkins Pipeline

To fetch the Maven project from GitHub, we need to configure **Source Code Management (SCM)** in Jenkins. This allows Jenkins to automatically pull the latest code before building the project.

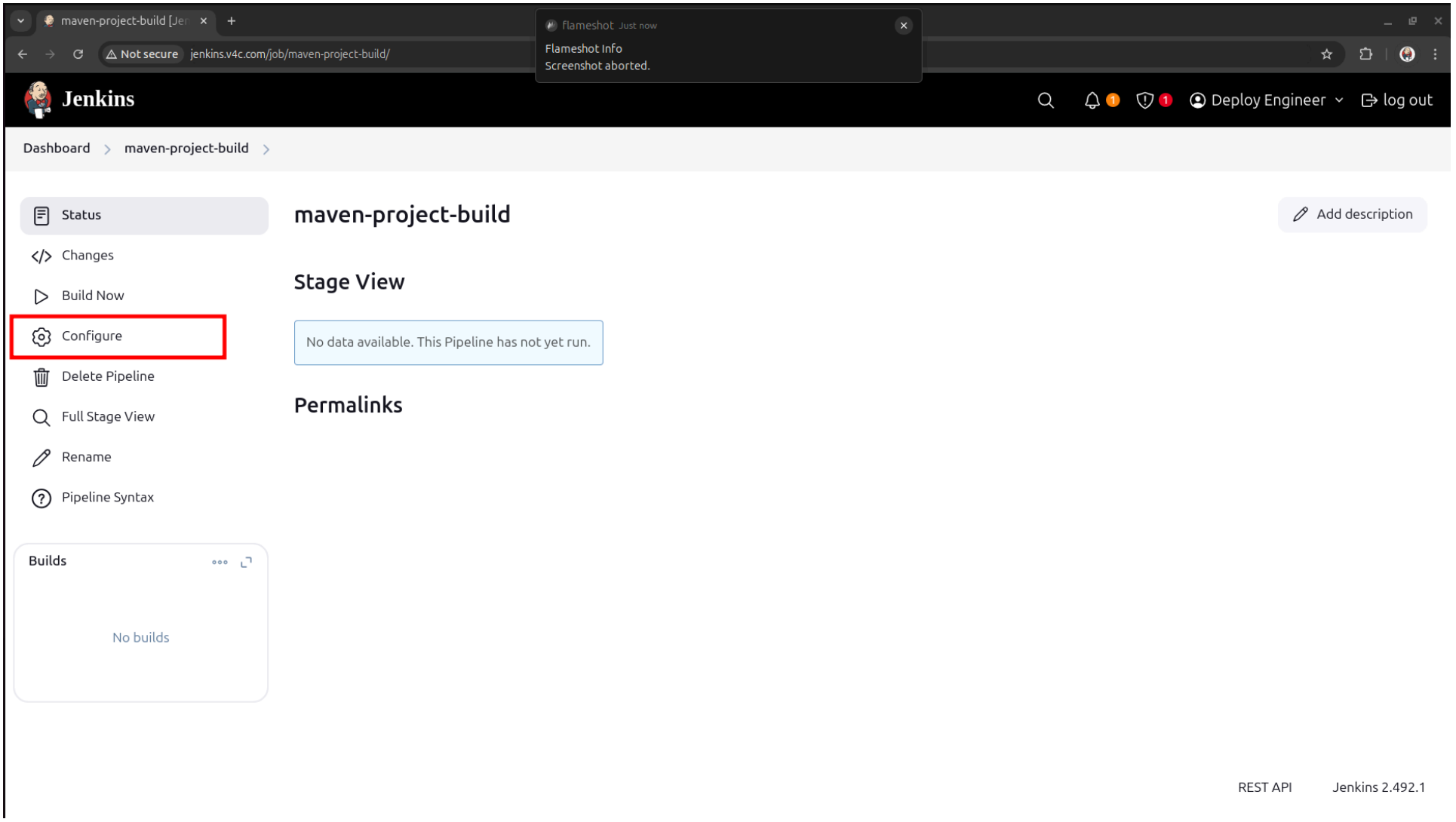
Steps to Link GitHub Repository

Open the Pipeline Configuration Page

- From the **Jenkins Dashboard**, click on your **Pipeline job** (`maven-project-build`).

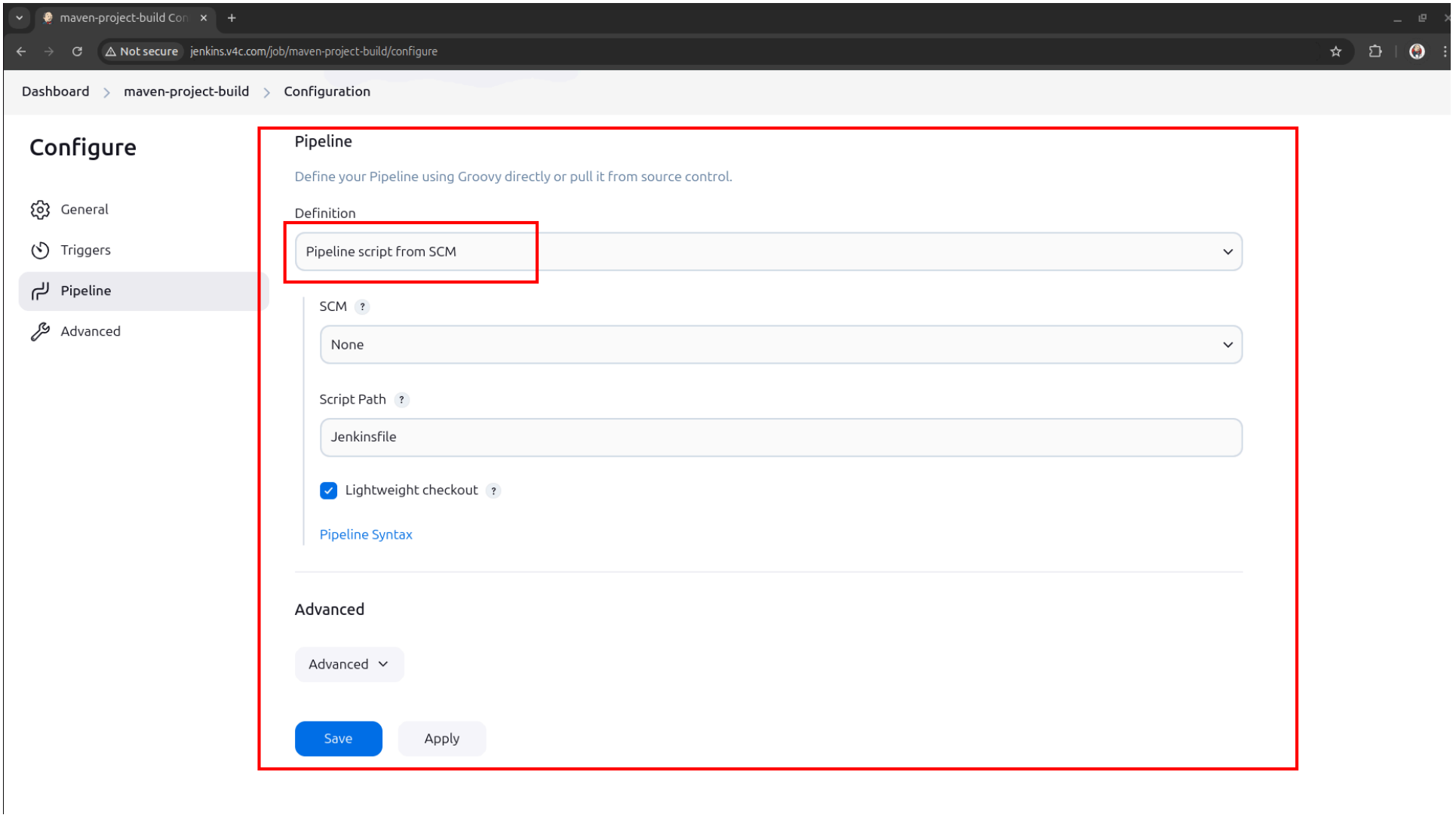


- Click **Configure** from the left sidebar.

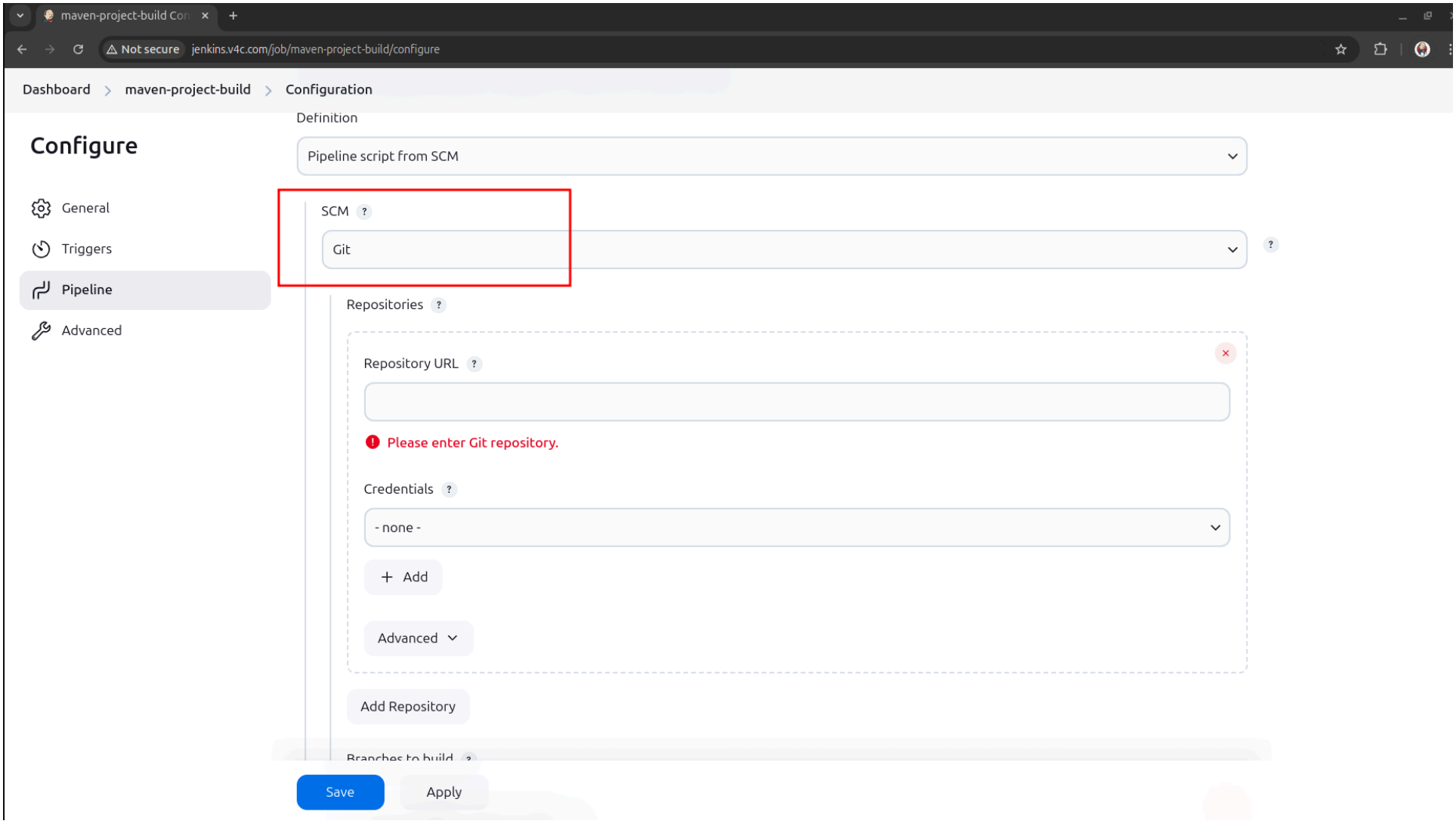


Enable Source Code Management (SCM)

- Scroll down to the **Source Code Management** section.



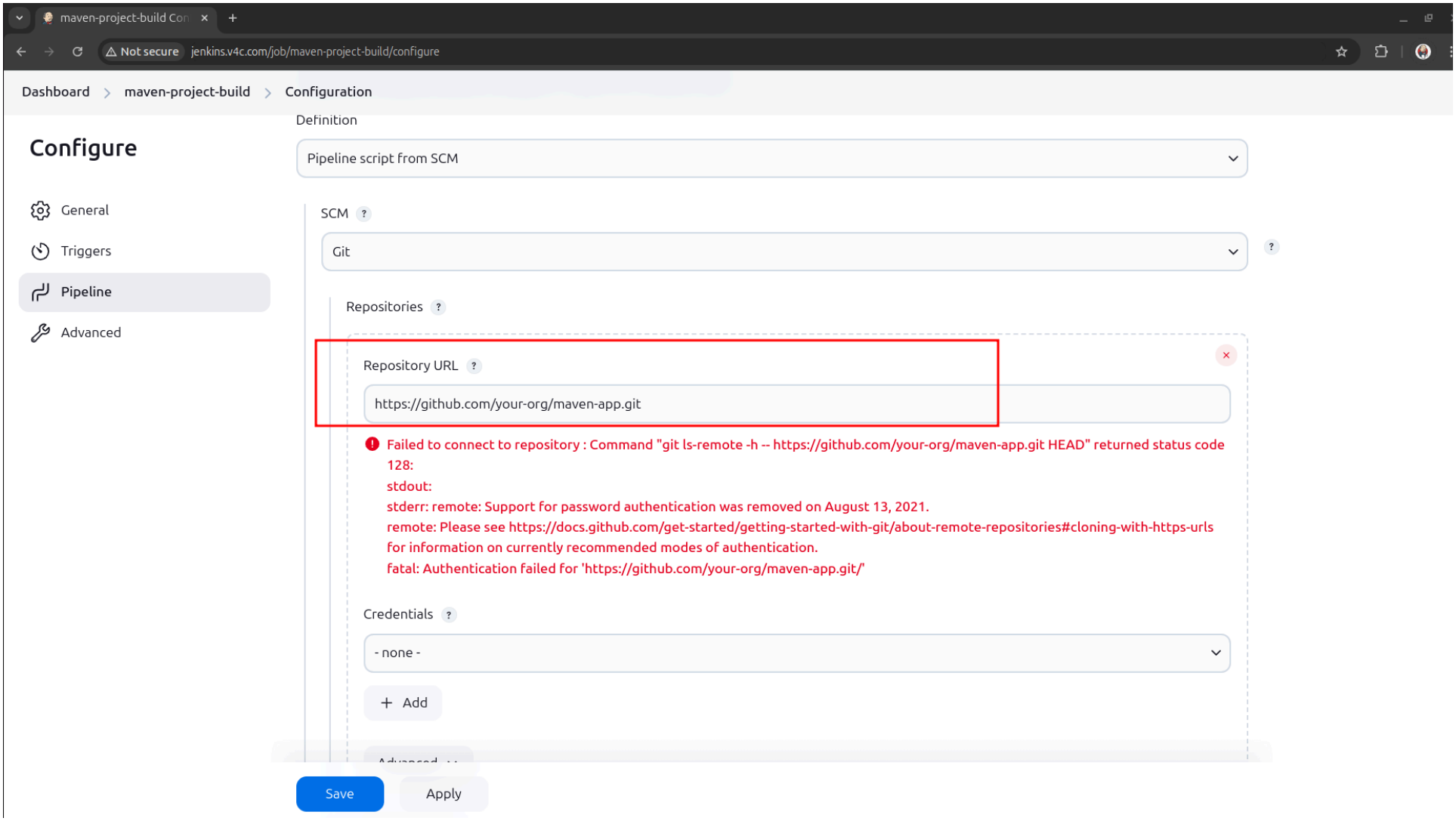
- Select **Git** (this option appears if the Git plugin is installed).



Enter the GitHub Repository URL

- In the **Repository URL** field, enter the GitHub repository link. **Example:** `https://github.com/your-org/maven-app.git`
- If the repository is **public**, Jenkins can access it without authentication.

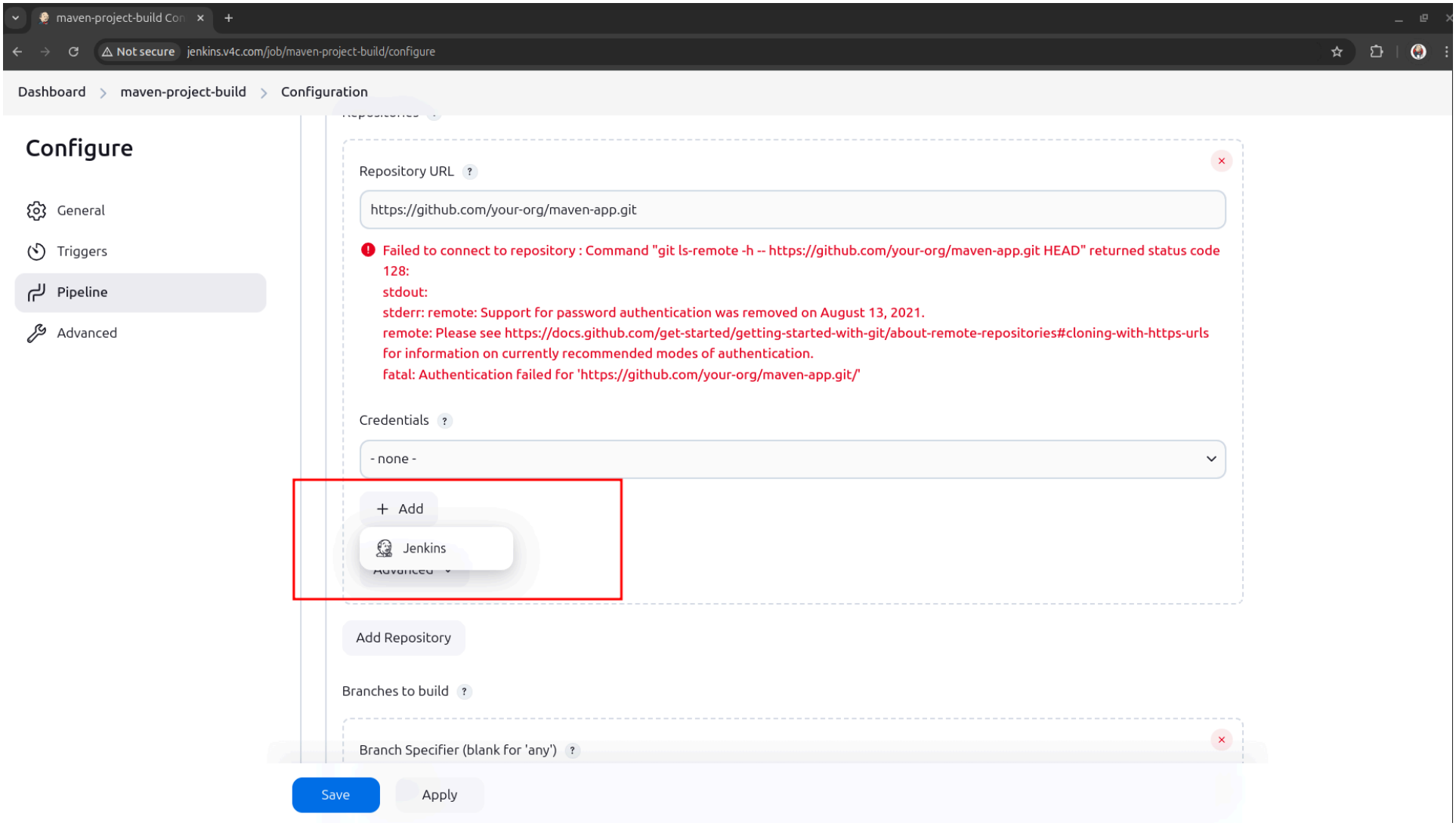
- If the repository is **private**, Jenkins needs credentials to access it.



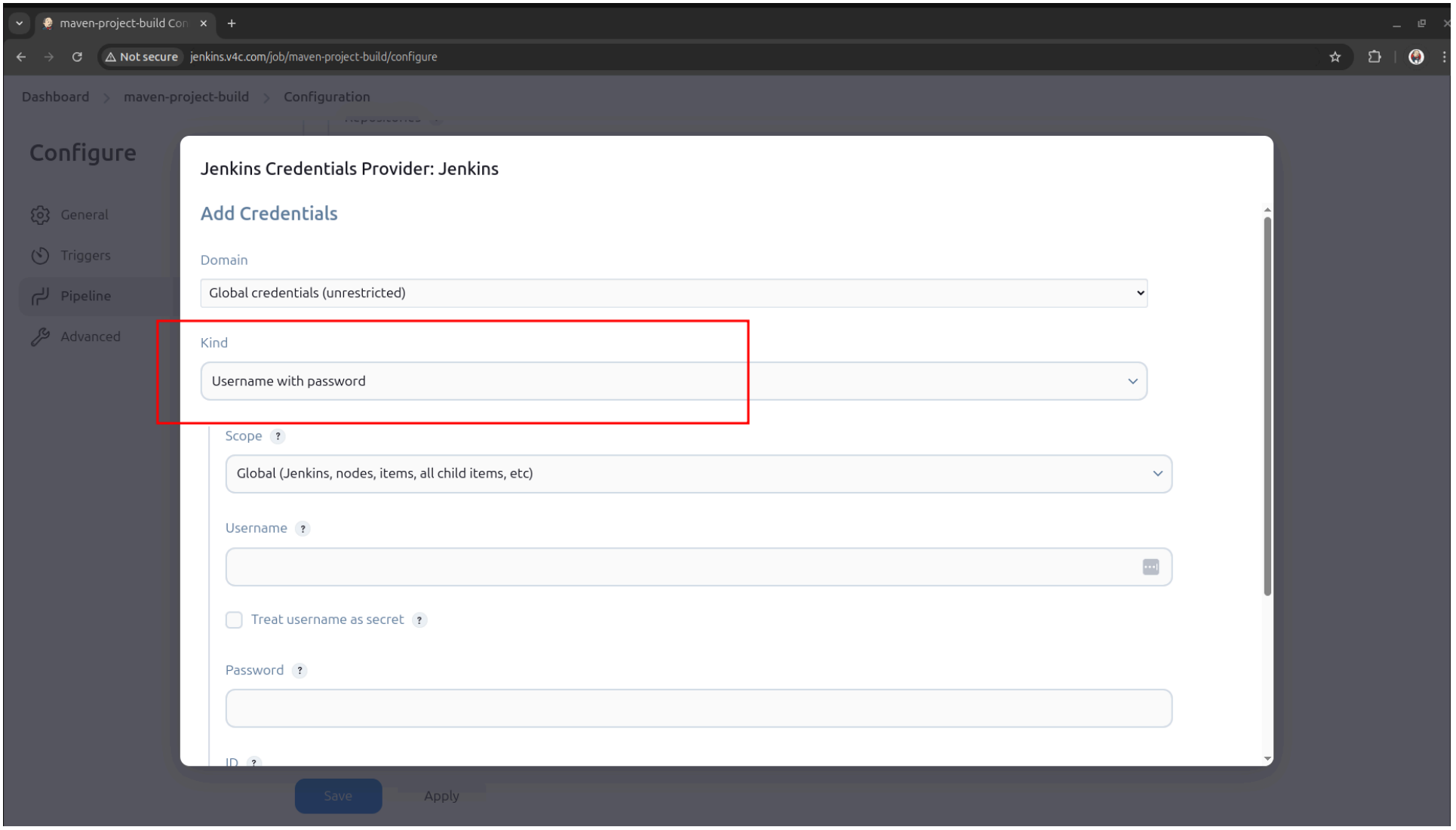
Add GitHub Credentials (For Private Repositories)

If your GitHub repository is **private**, Jenkins needs authentication to pull the code. Follow these steps:

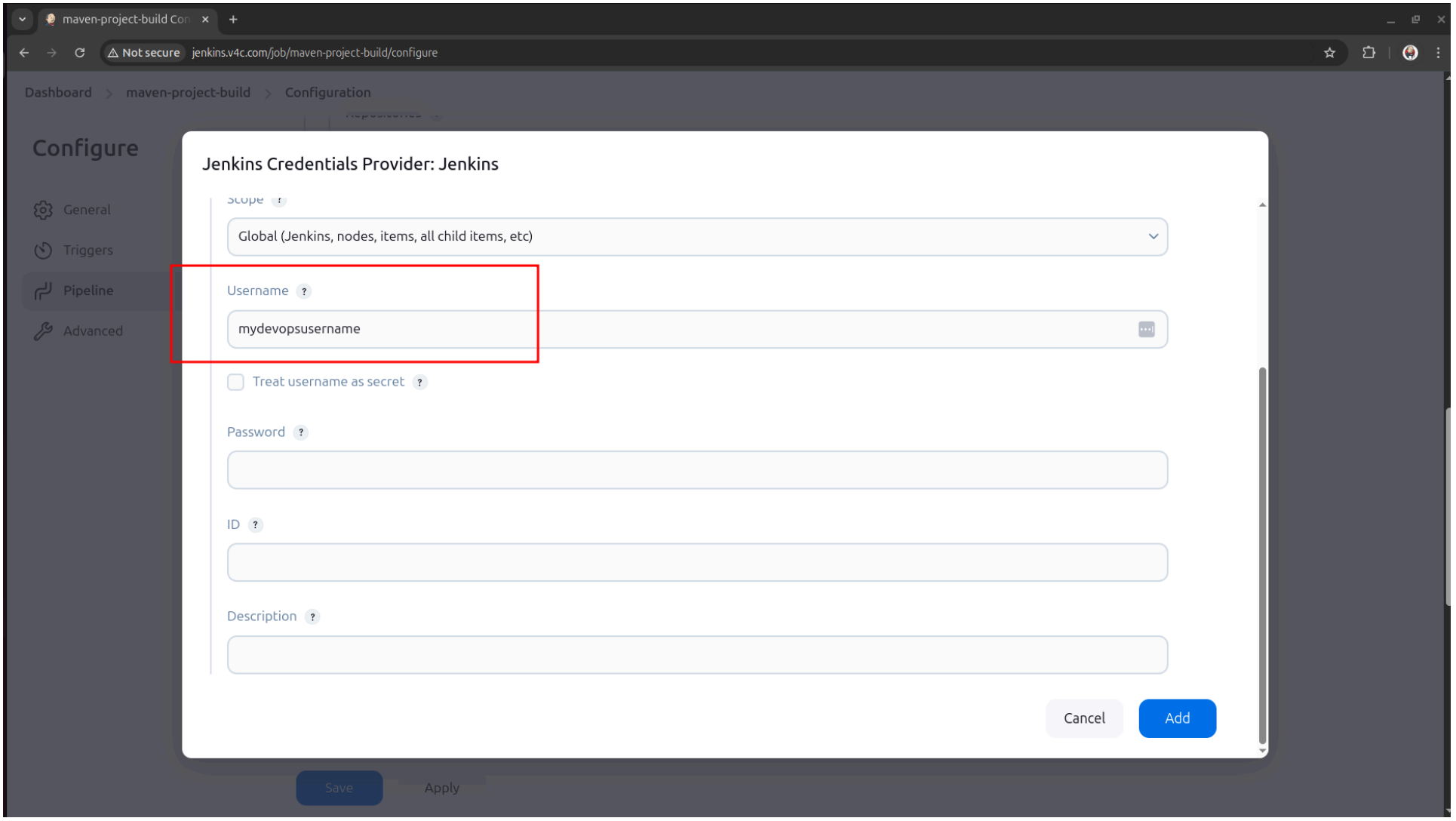
1. Click **Add → Jenkins** (next to the "Credentials" dropdown).



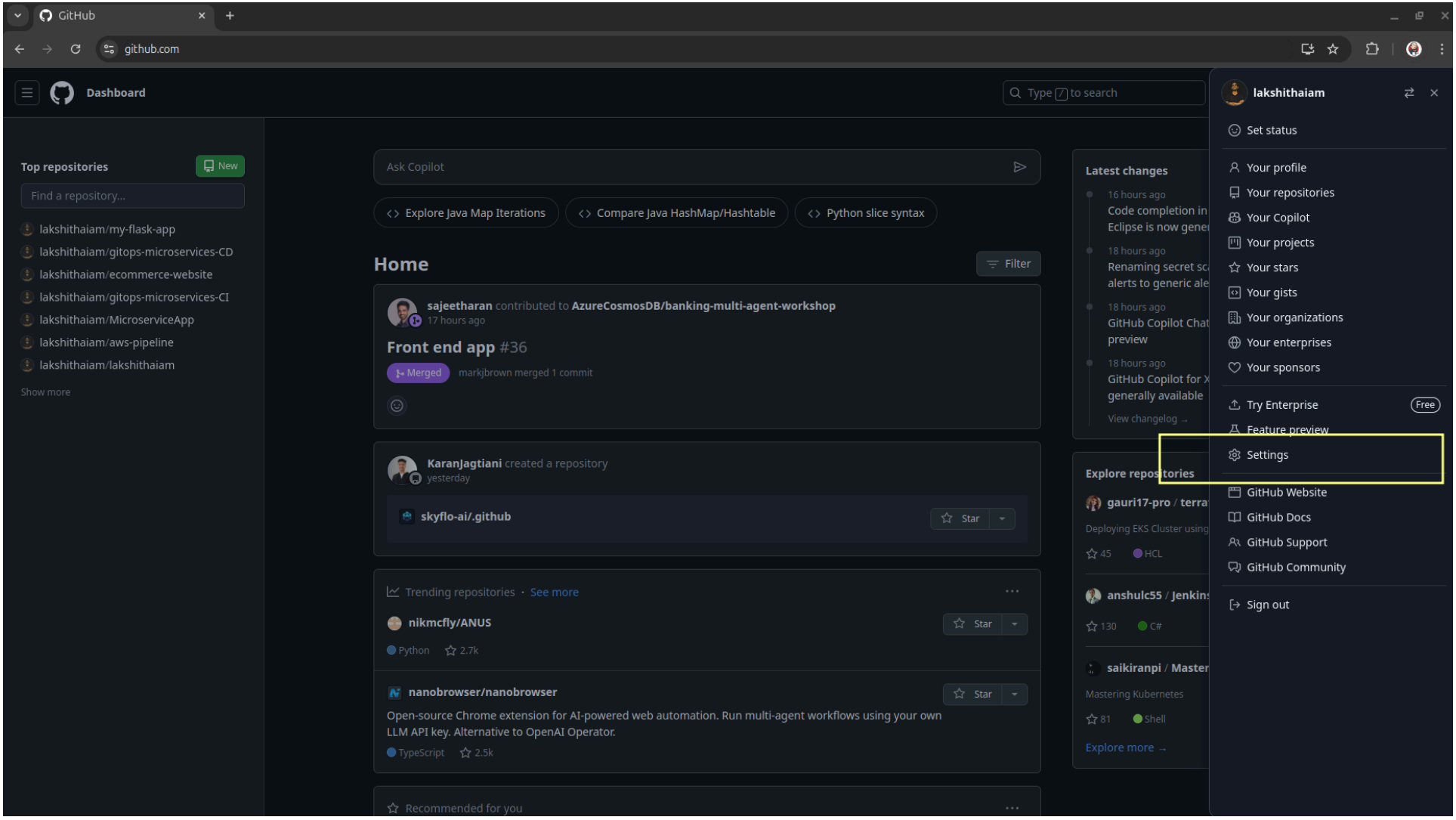
2. Select **Kind: Username with password**.

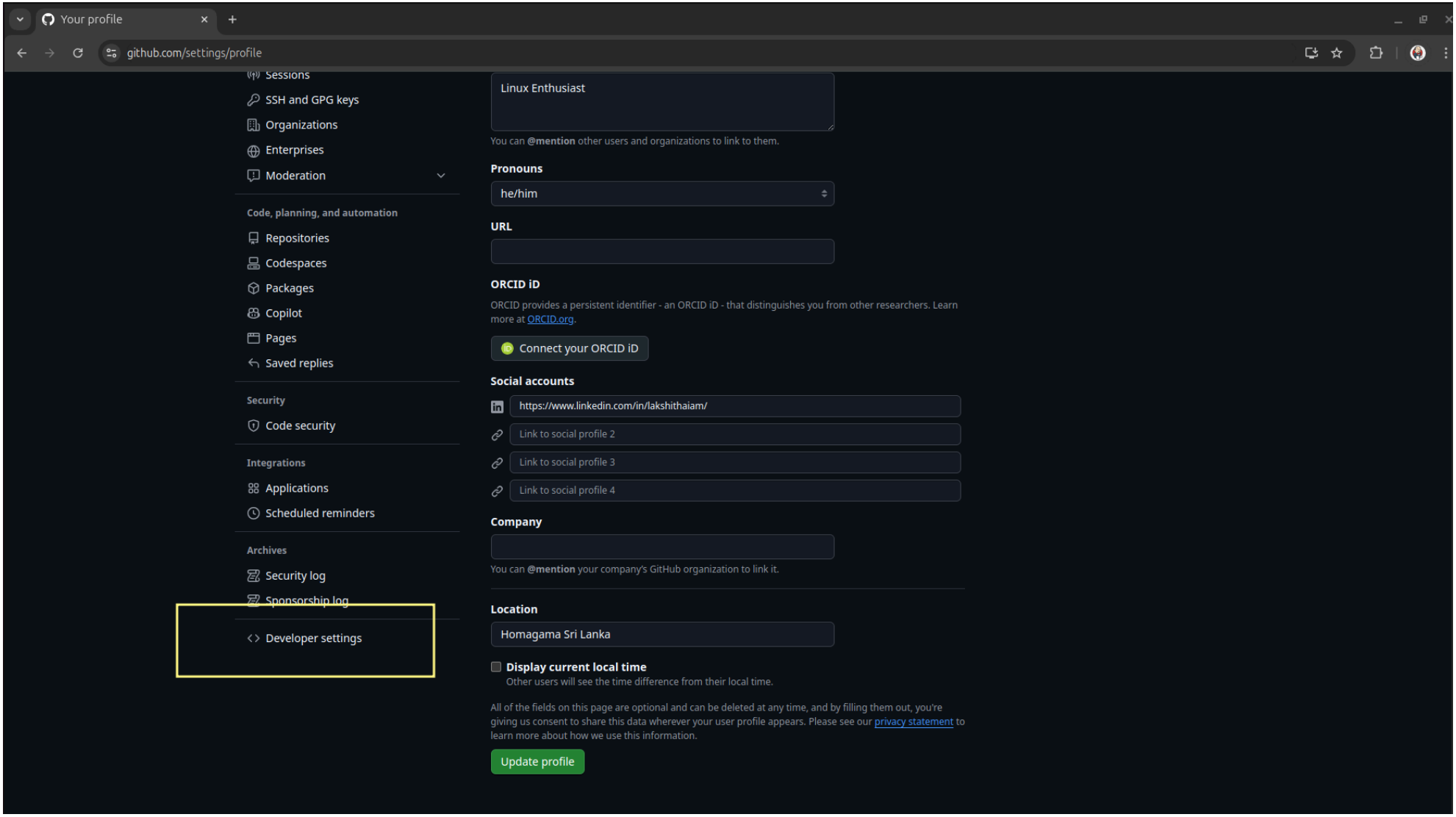


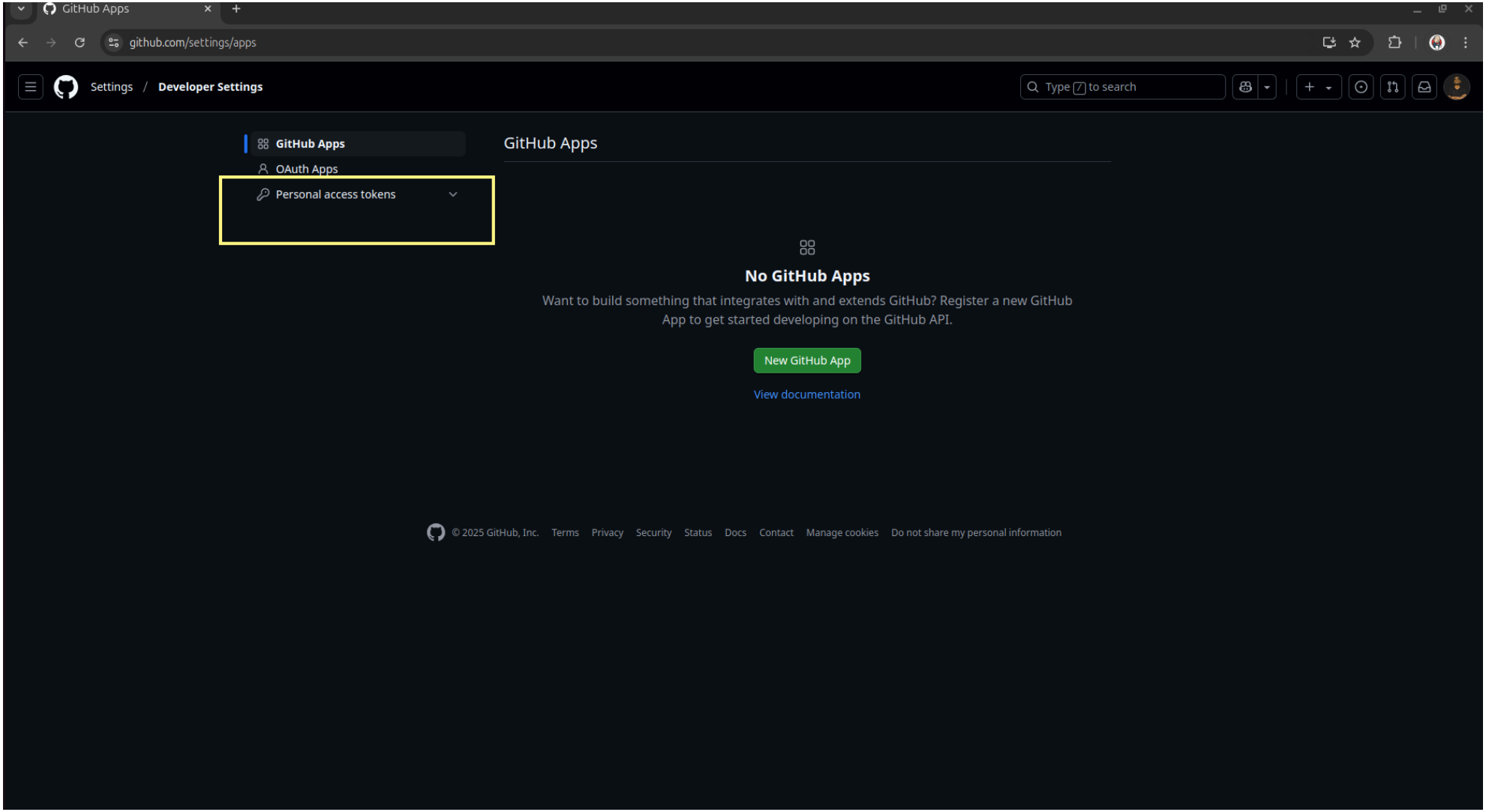
3. Enter your **GitHub username**.

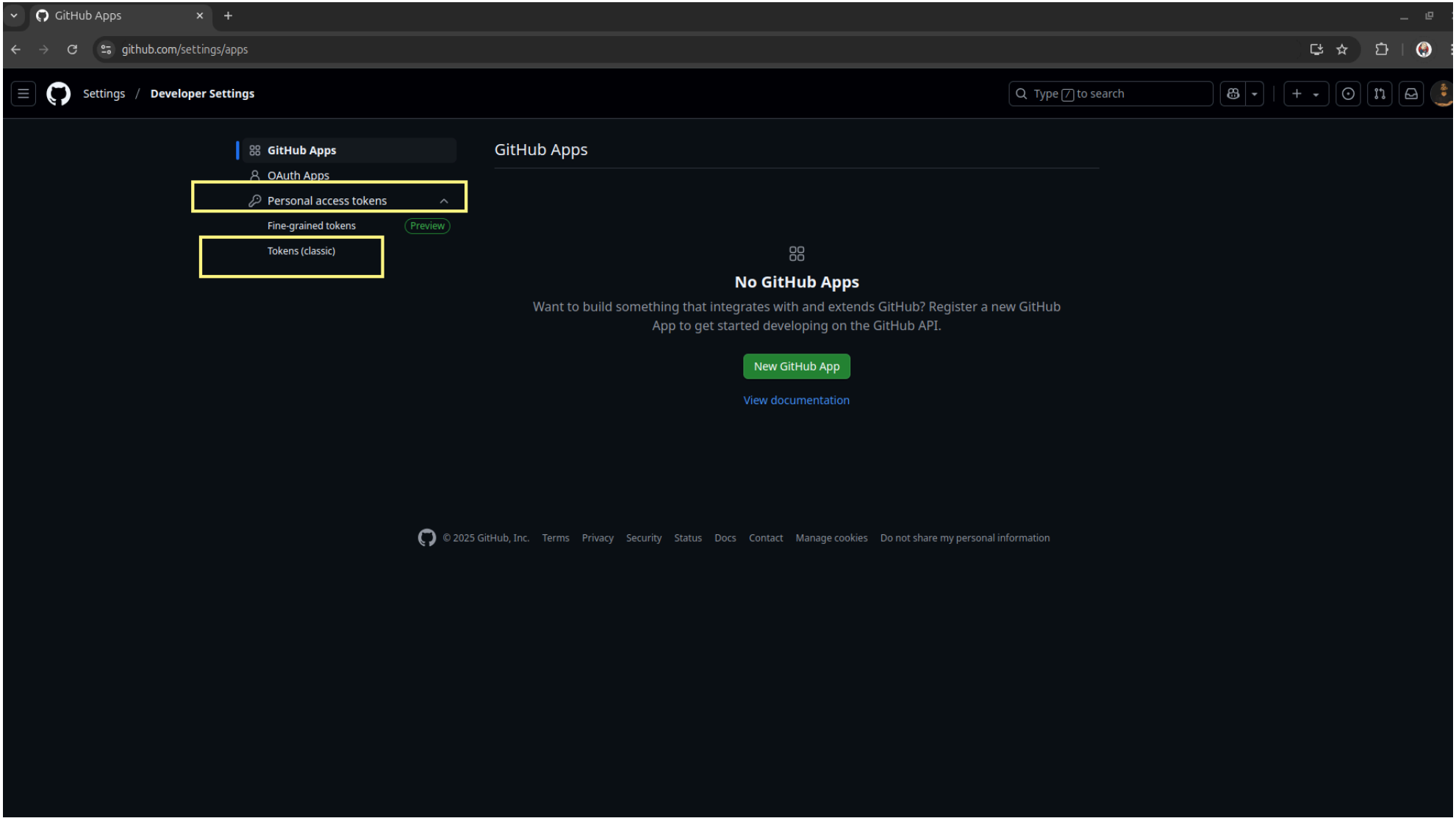


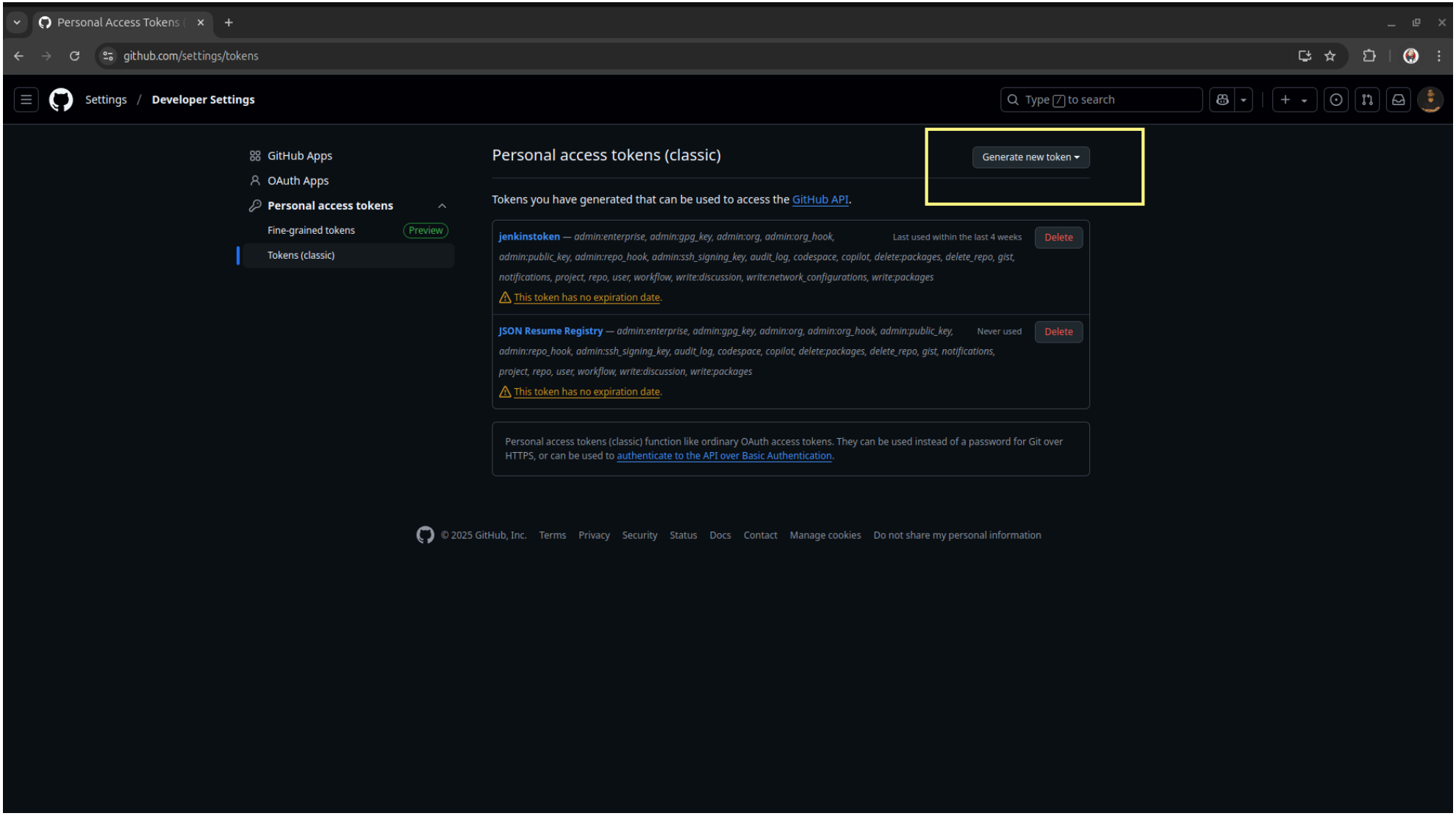
4. Enter a **Personal Access Token (PAT)** instead of a password
   To generate a PAT, go to **GitHub → Settings → Developer Settings → Personal Access Tokens**.


GitHub
github.com
Dashboard
Type / to search
lakshithaiam
Set status
Your profile
Your repositories
Your Copilot
Your projects
Your stars
Your gists
Your organizations
Your enterprises
Your sponsors
Try Enterprise
Free
Feature preview
Settings
GitHub Website
GitHub Docs
GitHub Support
GitHub Community
Sign out
Top repositories
New
Find a repository...
lakshithaiam/my-flask-app
lakshithaiam/gitops-microservices-CD
lakshithaiam/ecommerce-website
lakshithaiam/gitops-microservices-CI
lakshithaiam/MicroserviceApp
lakshithaiam/aws-pipeline
lakshithaiam/lakshithaiam
Show more
Ask Copilot
Explore Java Map Iterations
Compare Java HashMap/Hashtable
Python slice syntax
Home
Filter
sajeetharan contributed to AzureCosmosDB/banking-multi-agent-workshop
17 hours ago
Front end app #36
Merged
markjbrown merged 1 commit
KaranJagtiani created a repository
yesterday
skyflo-ai/.github
Star
Trending repositories · See more
nikmcfly/ANUS
Star
Python
2.7k
nanobrowser/nanobrowser
Star
Open-source Chrome extension for AI-powered web automation. Run multi-agent workflows using your own LLM API key. Alternative to OpenAI Operator.
TypeScript
2.5k
Recommended for you
Latest changes
16 hours ago
Code completion in Eclipse is now gene
18 hours ago
Renaming secret sc alerts to generic ale
18 hours ago
GitHub Copilot Chat preview
18 hours ago
GitHub Copilot for X generally available
View changelog →
Explore repositories
gauri17-pro / terra
Deploying EKS Cluster using
45
HCL
anshulc55 / Jenkins
130
C#
saikiranpi / Master
Mastering Kubernetes
81
Shell
Explore more →


Your profile
github.com/settings/profile
Sessions
SSH and GPG keys
Organizations
Enterprises
Moderation
Code, planning, and automation
Repositories
Codespaces
Packages
Copilot
Pages
Saved replies
Security
Code security
Integrations
Applications
Scheduled reminders
Archives
Security log
Sponsorship log
Developer settings
Linux Enthusiast
You can @mention other users and organizations to link to them.
Pronouns
he/him
URL
ORCID iD
ORCID provides a persistent identifier - an ORCID iD - that distinguishes you from other researchers. Learn more at ORCID.org.
Connect your ORCID iD
Social accounts
https://www.linkedin.com/in/lakshithaiam/
Link to social profile 2
Link to social profile 3
Link to social profile 4
Company
You can @mention your company's GitHub organization to link it.
Location
Homagama Sri Lanka
Display current local time
Other users will see the time difference from their local time.
All of the fields on this page are optional and can be deleted at any time, and by filling them out, you're giving us consent to share this data wherever your user profile appears. Please see our privacy statement to learn more about how we use this information.
Update profile


GitHub Apps
github.com/settings/apps
Settings / Developer Settings
Type / to search
GitHub Apps
OAuth Apps
Personal access tokens
GitHub Apps
No GitHub Apps
Want to build something that integrates with and extends GitHub? Register a new GitHub App to get started developing on the GitHub API.
New GitHub App
View documentation
© 2025 GitHub, Inc. Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information

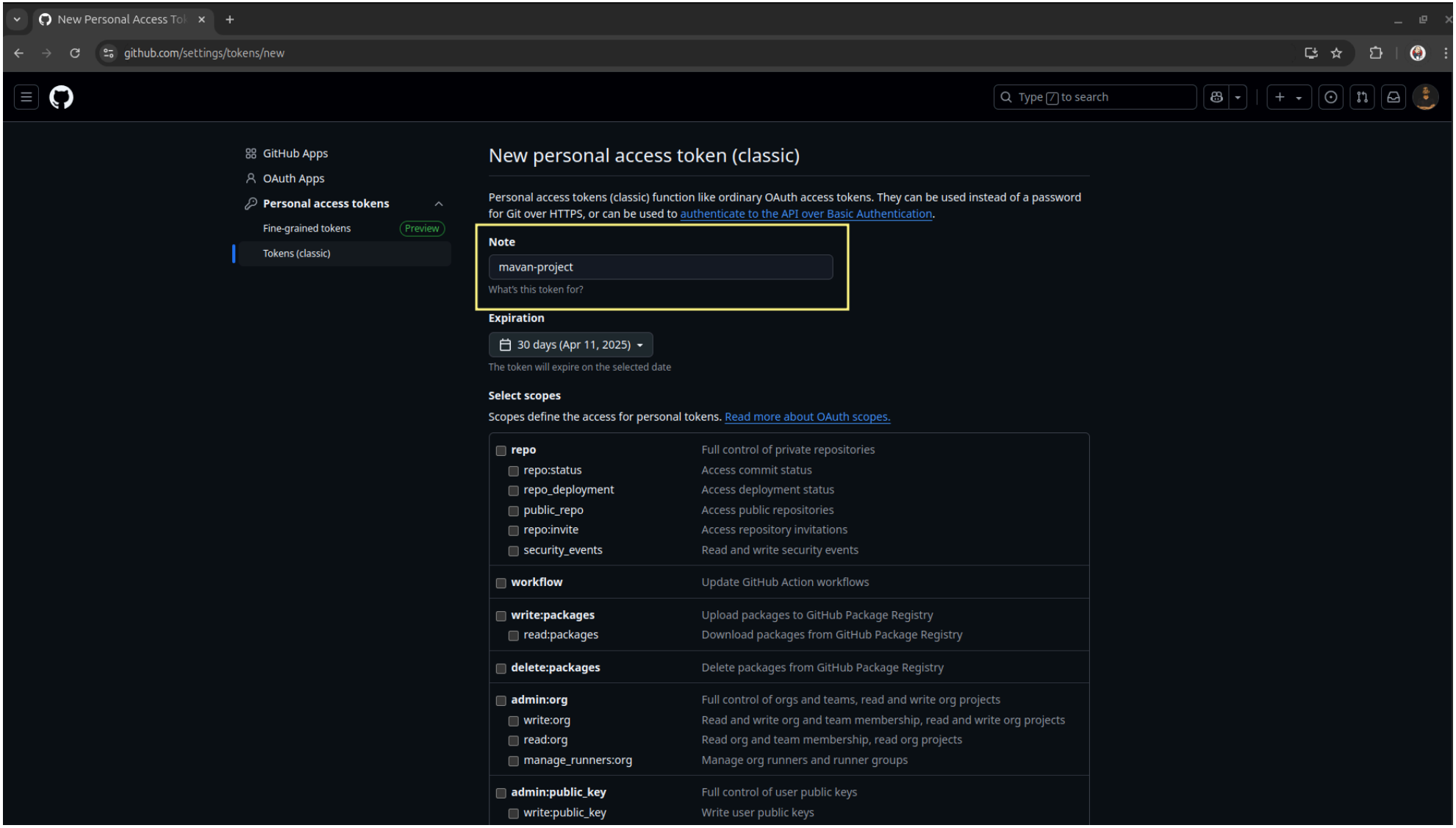

GitHub Apps
github.com/settings/apps
Settings / Developer Settings
Type / to search
GitHub Apps
OAuth Apps
Personal access tokens
Fine-grained tokens
Preview
Tokens (classic)
GitHub Apps
No GitHub Apps
Want to build something that integrates with and extends GitHub? Register a new GitHub App to get started developing on the GitHub API.
New GitHub App
View documentation
© 2025 GitHub, Inc. Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information

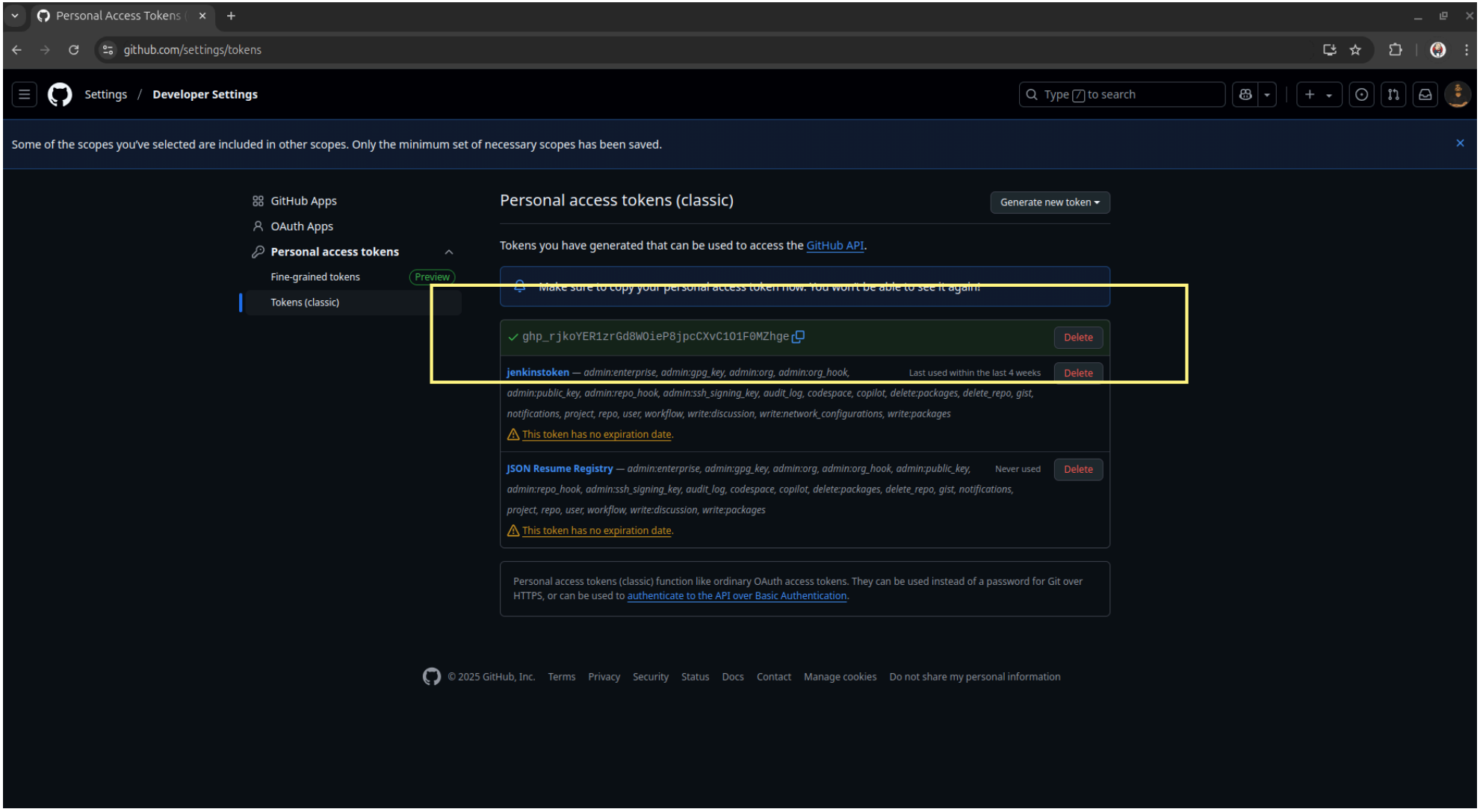


Click **Generate new token** and select scopes like:

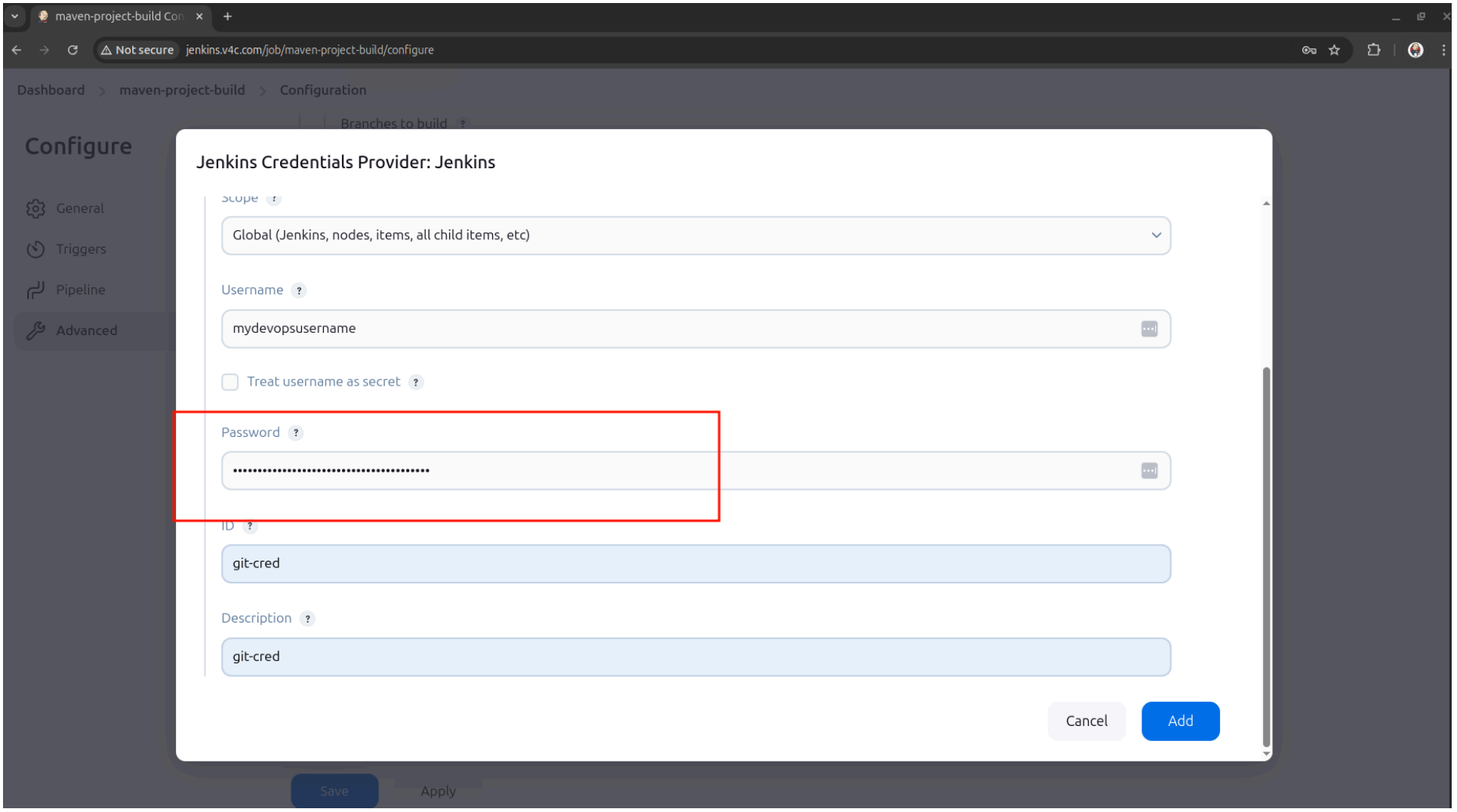
✅ `repo` (Full control of private repositories)

✅ `read:packages` (Read access to GitHub Packages)

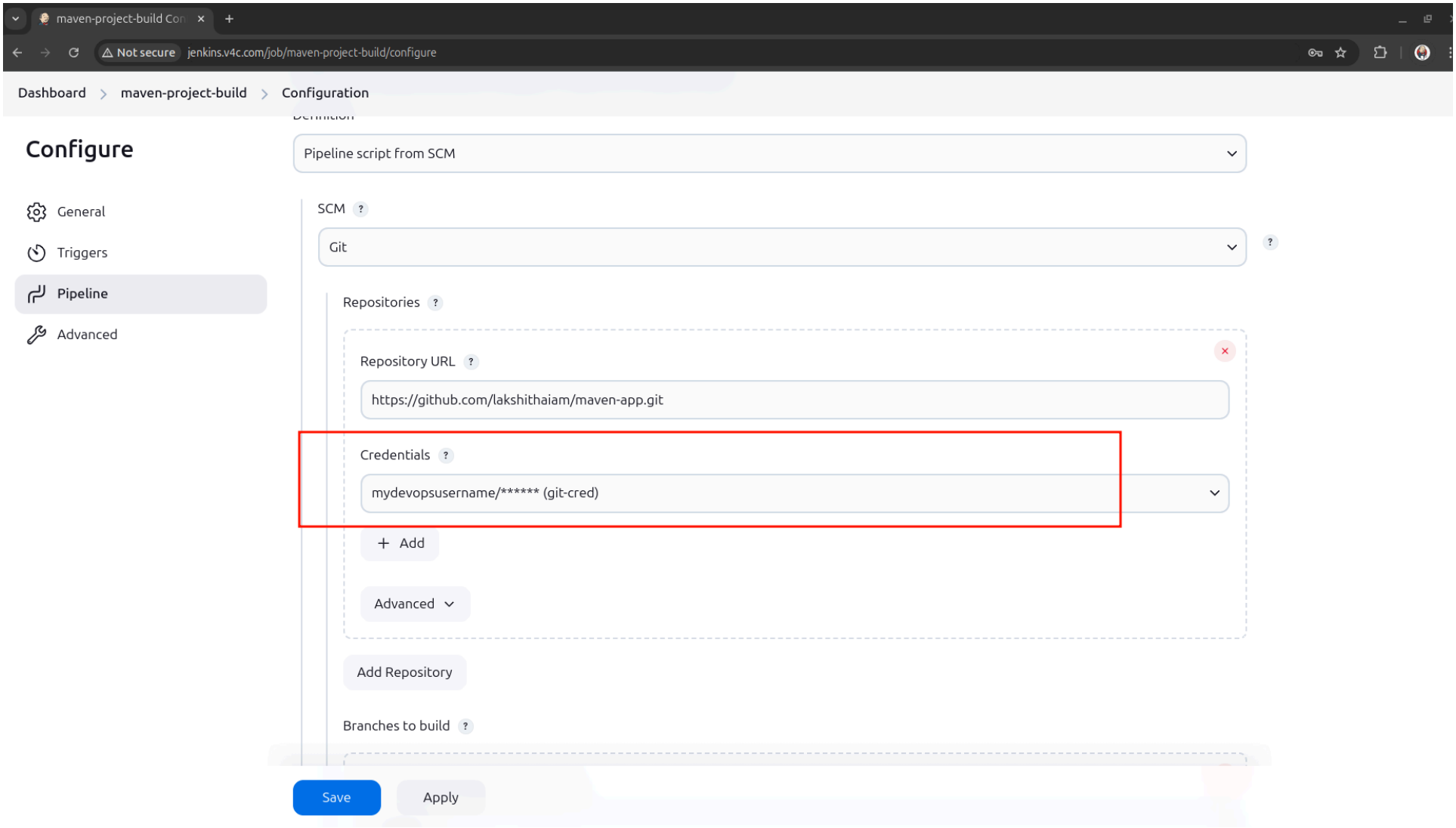

New Personal Access Tok
github.com/settings/tokens/new
Type / to search
GitHub Apps
OAuth Apps
Personal access tokens
Fine-grained tokens
Preview
Tokens (classic)
New personal access token (classic)
Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.
Note
mavan-project
What's this token for?
Expiration
30 days (Apr 11, 2025)
The token will expire on the selected date
Select scopes
Scopes define the access for personal tokens. Read more about OAuth scopes.
repo
Full control of private repositories
repo:status
Access commit status
repo_deployment
Access deployment status
public_repo
Access public repositories
repo:invite
Access repository invitations
security_events
Read and write security events
workflow
Update GitHub Action workflows
write:packages
Upload packages to GitHub Package Registry
read:packages
Download packages from GitHub Package Registry
delete:packages
Delete packages from GitHub Package Registry
admin:org
Full control of orgs and teams, read and write org projects
write:org
Read and write org and team membership, read and write org projects
read:org
Read org and team membership, read org projects
manage_runners:org
Manage org runners and runner groups
admin:public_key
Full control of user public keys
write:public_key
Write user public keys


Personal Access Tokens
github.com/settings/tokens
Settings / Developer Settings
Type / to search
Some of the scopes you've selected are included in other scopes. Only the minimum set of necessary scopes has been saved.
GitHub Apps
OAuth Apps
Personal access tokens
Fine-grained tokens
Preview
Tokens (classic)
Personal access tokens (classic)
Generate new token
Tokens you have generated that can be used to access the GitHub API.
Make sure to copy your personal access token now. You won't be able to see it again!
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
Delete
jenkinstoken — admin:enterprise, admin:gpg_key, admin:org, admin:org_hook, admin:public_key, admin:repo_hook, admin:ssh_signing_key, audit_log, codespace, copilot, delete:packages, delete_repo, gist, notifications, project, repo, user, workflow, write:discussion, write:network_configurations, write:packages
Last used within the last 4 weeks
Delete
This token has no expiration date.
JSON Resume Registry — admin:enterprise, admin:gpg_key, admin:org, admin:org_hook, admin:public_key, admin:repo_hook, admin:ssh_signing_key, audit_log, codespace, copilot, delete:packages, delete_repo, gist, notifications, project, repo, user, workflow, write:discussion, write:packages
Never used
Delete
This token has no expiration date.
Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.
© 2025 GitHub, Inc. Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information

5. Click **Add** to save the credentials.



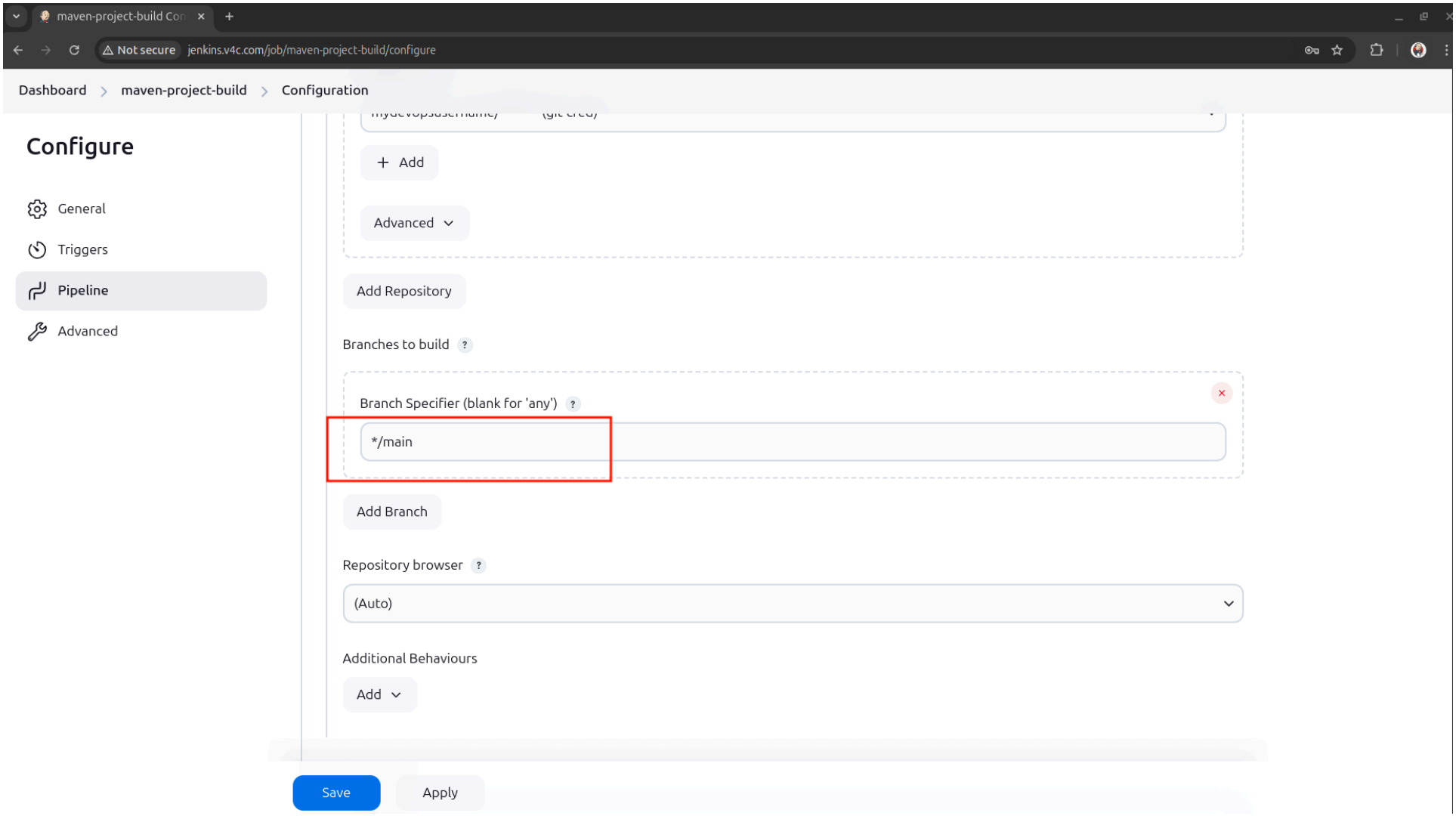
6. Select the saved credentials from the **Credentials** dropdown.



Specify the Branch

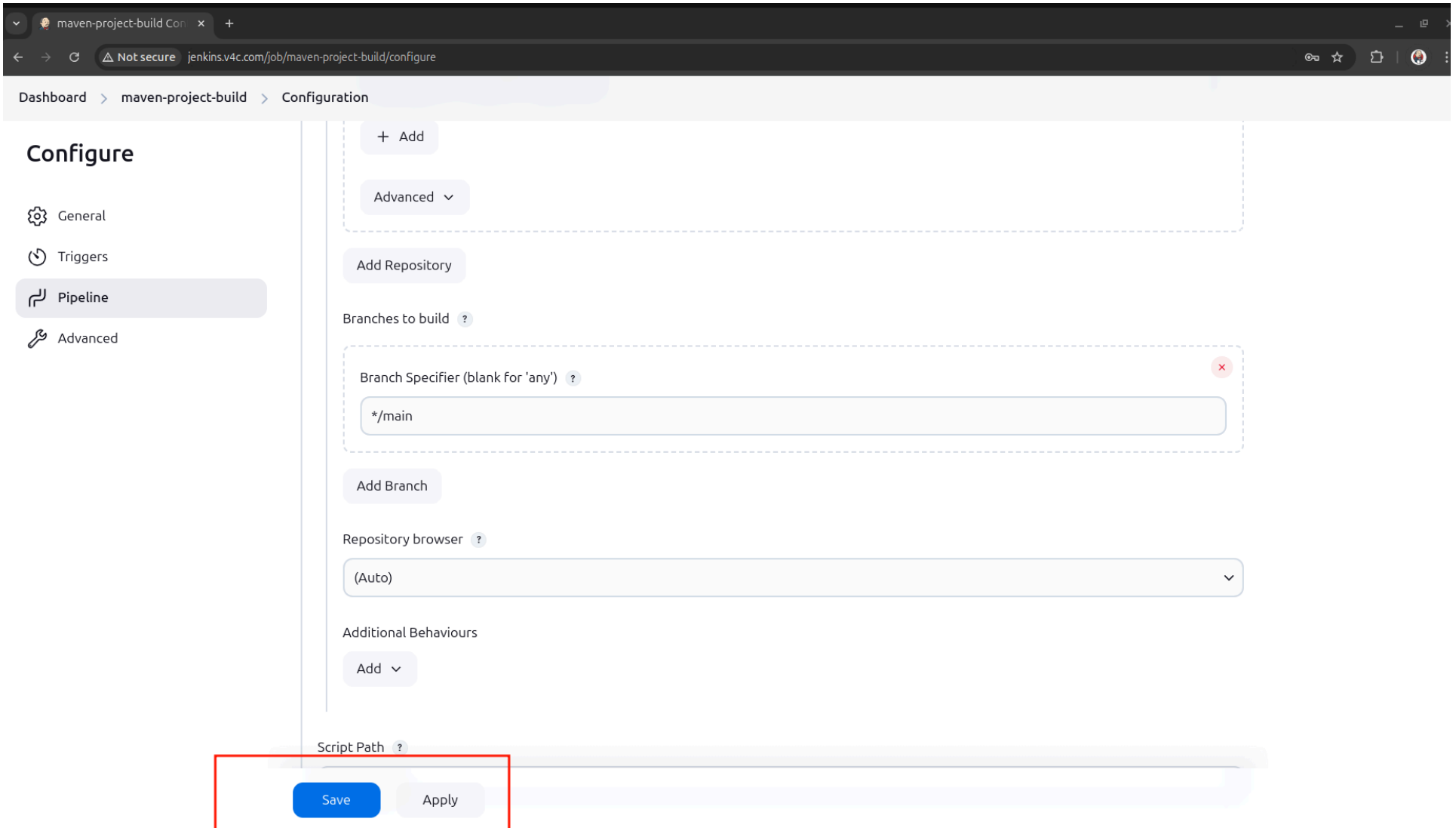
- If you want to build a specific branch (e.g., `main` or `develop`), enter the branch name in the **Branches to build** field.
  Example:
  : main

- If left blank, Jenkins will pull the **default branch** of the repository.



Save the Configuration

- Click **Save** to apply the changes.



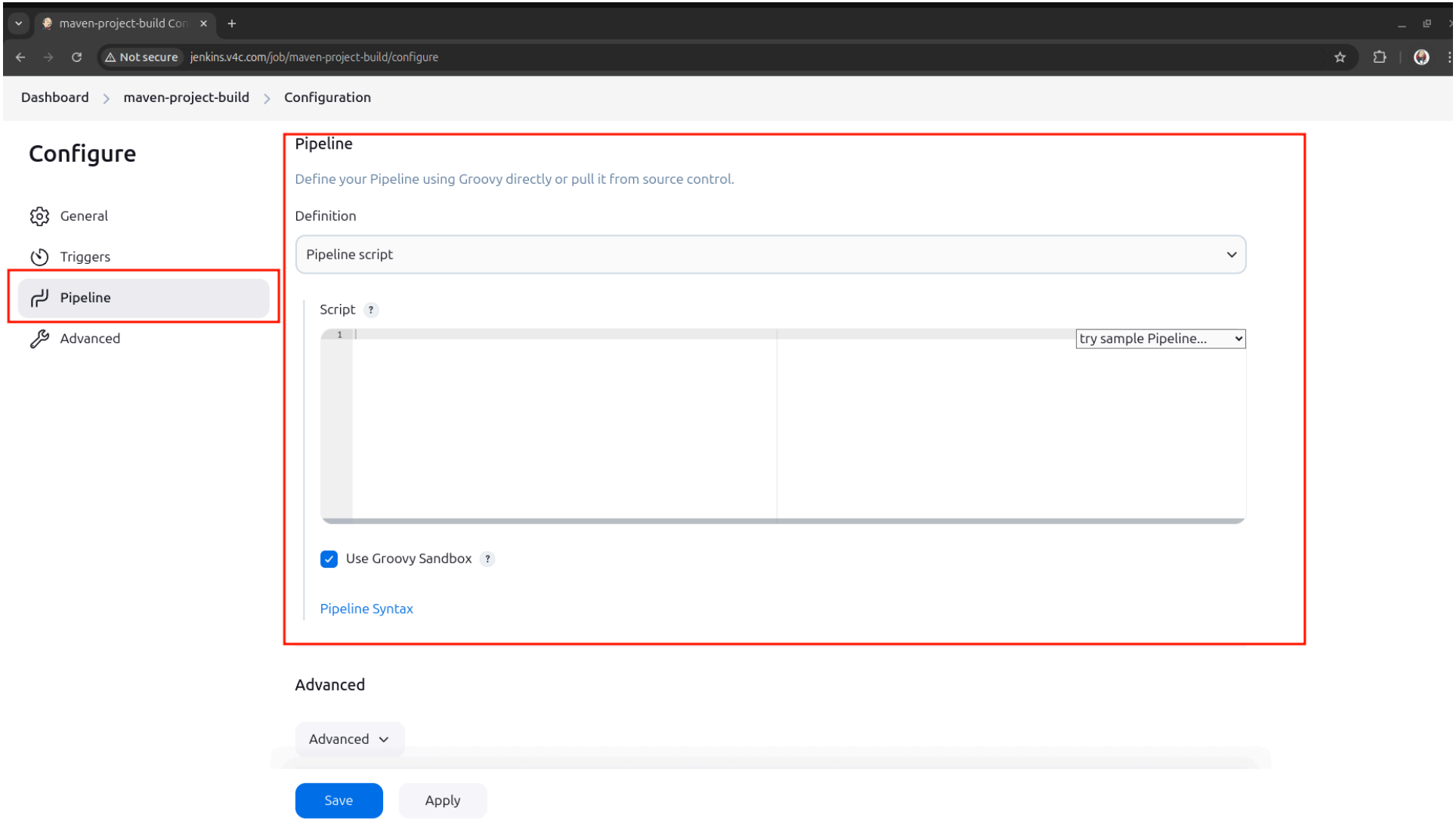
Now, your Jenkins pipeline is successfully linked to the GitHub repository!

Set Up the Pipeline Script

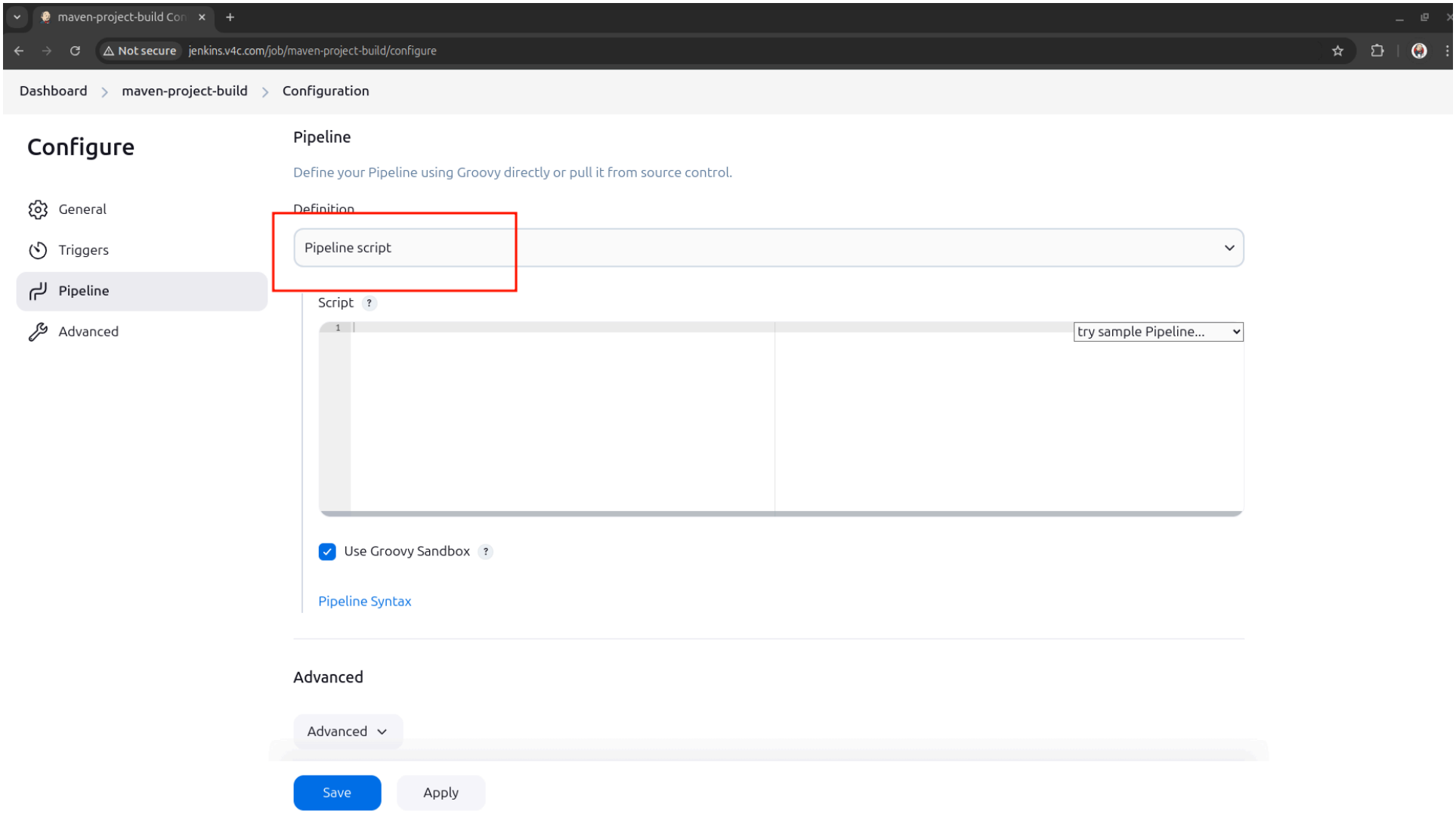
In this step, we will define a **Jenkins pipeline script** to automate the build, test, and package process for our **Maven project**.

Where to Add the Script?

1. Scroll down to the **Pipeline** section in the job configuration page.



2. Select **Pipeline Script** from the dropdown menu.



3. Copy and paste the following script into the text box:

Pipeline Script Breakdown

```
pipeline {
    agent any
```

- `pipeline {}` → Defines the start of the Jenkins pipeline.
- `agent any` → Runs the pipeline on any available Jenkins agent.

Checkout Code from GitHub

```
        stage('Checkout') {
            steps {
                git branch: 'main', url: 'https://github.com/your-org/maven-app.git'
            }
        }
```

- **Purpose:** Retrieves the latest code from the **GitHub repository**.
- **`git branch: 'main'`** → Specifies the branch to clone (change `'main'` if using another branch like `'develop'`).
- **`url: 'https://github.com/your-org/maven-app.git'`** → Replace with your actual repository URL.
- **Outcome:** The repository is cloned onto the Jenkins workspace.

Build the Application

```
        stage('Build') {
            steps {
                sh 'mvn clean install'
            }
        }
```

- **Purpose:** Compiles the Java code and downloads project dependencies.

- **`mvn clean install`** →
  - `clean` → Removes old build artifacts.
  - `install` → Builds the project and installs the generated JAR/WAR file in the local Maven repository.
- **Outcome:** The application is successfully compiled and ready for testing.

Run Unit Tests

```
        stage('Test') {
            steps {
                sh 'mvn test'
            }
        }
```

- **Purpose:** Runs the **unit tests** written for the project.
- **`mvn test`** → Executes all test cases defined in the project.
- **Outcome:** Ensures the code is functional and does not break any existing features.

Package the Application

```
        stage('Package') {
            steps {
                sh 'mvn package'
            }
        }
```

- **Purpose:** Bundles the application into a **deployable artifact** (JAR or WAR).
- **`mvn package`** → Creates a package (`.jar` or `.war`) inside the `target/` directory.

- **Outcome:** The final artifact is built and ready for deployment.

Final Pipeline Script (Full View)

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
                git branch: 'main', url: 'https://github.com/your-org/maven-app.git'
            }
        }

        stage('Build') {
            steps {
                sh 'mvn clean install'
            }
        }

        stage('Test') {
            steps {
                sh 'mvn test'
            }
        }

        stage('Package') {
            steps {
                sh 'mvn package'
```

```
            }
        }
    }
}
```

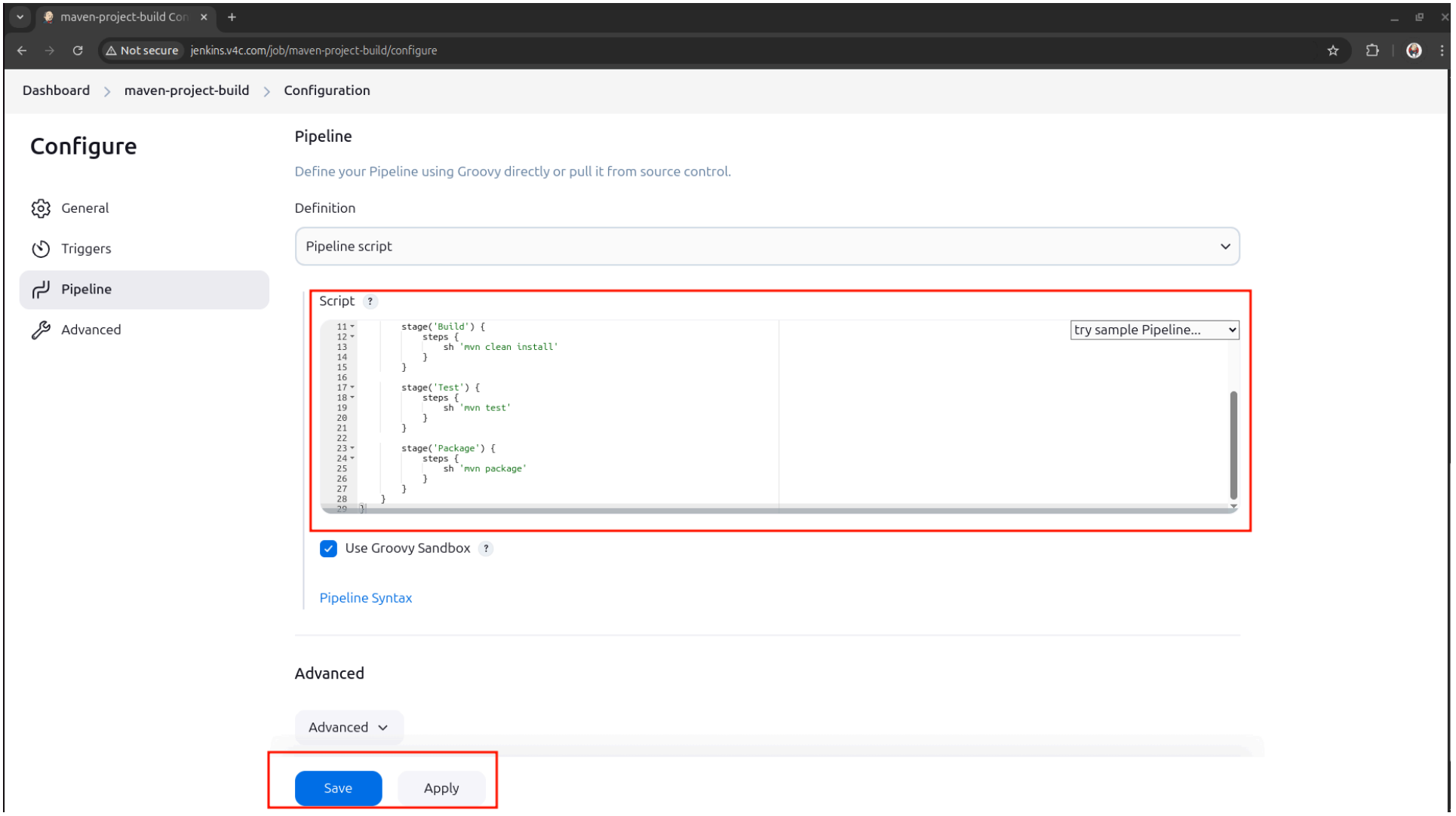
Next Steps After Adding the Pipeline Script

Now that you've copied the pipeline script into Jenkins, follow these steps:

Save and Apply the Pipeline Job

1. Scroll down to the bottom of the pipeline configuration page.

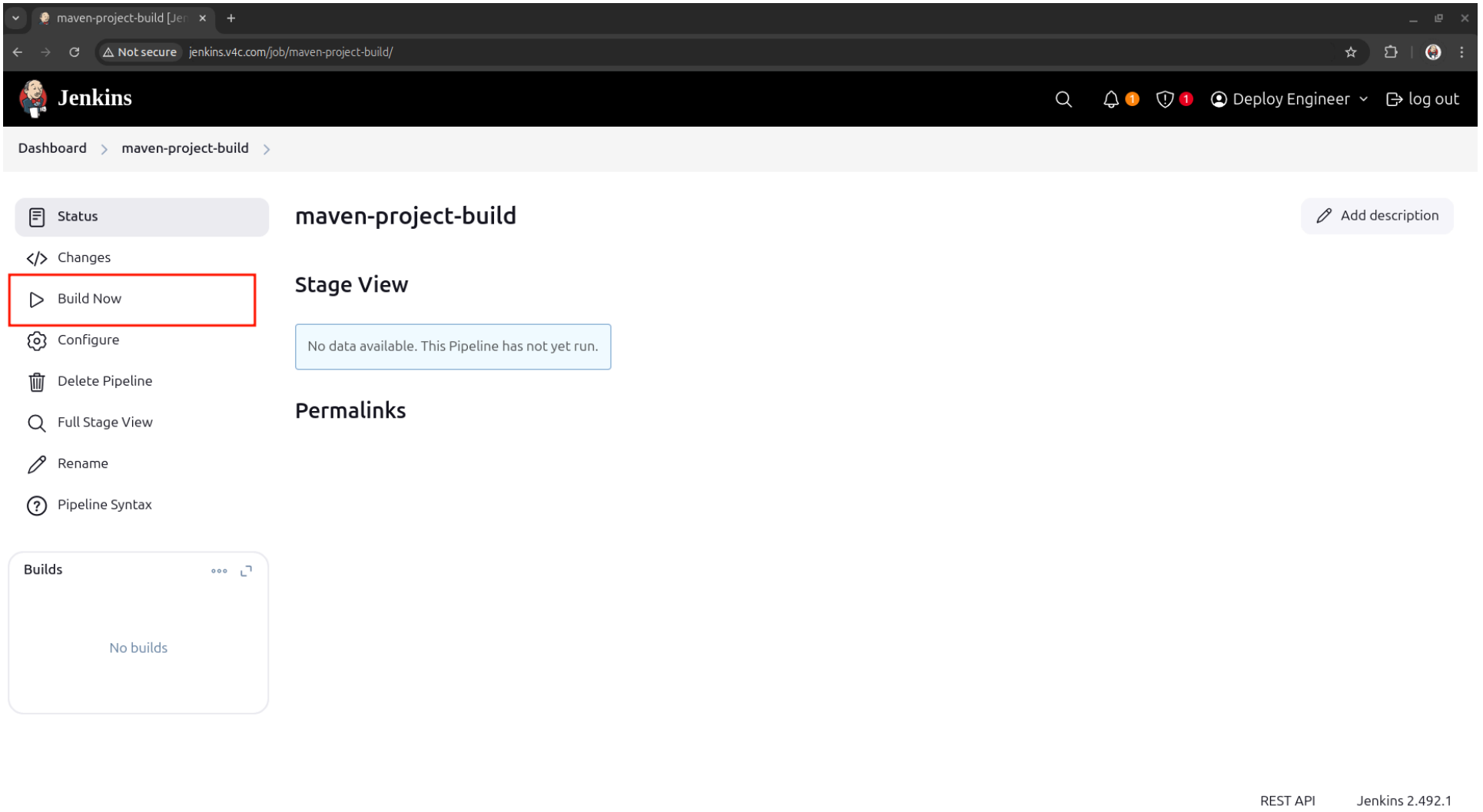
2. Click **Apply** and then **Save**.



Trigger the First Build

1. On the Jenkins job page, click **"Build Now"**.

2. Jenkins will start executing the pipeline.



Conclusion

Jenkins is a powerful tool for automating software builds and deployments. In this guide, we covered: ✅ What Jenkins is and why it is useful ✅ Deploying Jenkins on Kubernetes ✅ Creating a simple Jenkins job ✅ Running a basic Jenkins pipeline

---