# **Building an App as a Freestyle Jenkins Project**

## **Introduction**

Jenkins is a powerful tool for automating continuous integration. One of the simplest ways to implement CI for a software application is to configure a freestyle project in Jenkins. With a freestyle project, you can configure Jenkins to execute a build every time a change is made to the source code.

Learning to configure a freestyle Jenkins project will also prepare you for more advanced ways of utilizing Jenkins. After completing this exercise, you will know how to implement a freestyle Jenkins build for an application.

## **The Scenario**

The team has asked us to configure a Jenkins project to build the train-schedule app. The source code for the application is hosted in the [GitHub repository](https://github.com/ACloudGuru-Resources/cicd-pipeline-train-schedule-pipelines). The app already has build automation set up using gradle wrapper, and can be built with ./gradlew build. The team wants Jenkins to execute this automated build every time changes are pushed to the GitHub repository.

We will need to:

* Configure Jenkins to authenticate with GitHub
* Create a freestyle project in Jenkins
* Configure the project to build the train-schedule app
* Set up a webhook to trigger the build whenever changes are made to the repository in GitHub
* Configure the build to archive *trainSchedule.zip* as a build artifact

## **Logging In**

We're going to be landing on a public GitHub repository (linked to above) and creating a new fork, using our own personal GitHub account. To configure Jenkins though, the use the credentials and IP provided in the hands-on lab overview page to log into our Jenkins server.

## **Fork the Existing GitHub Repository**

Using the [GitHub link](https://github.com/ACloudGuru-Resources/cicd-pipeline-train-schedule-pipelines) provided a little earlier, get into the repository and click on the **Fork** button in the upper right. This will give us our own personal fork. Once that's done, we'll find ourselves sitting in the new fork's GitHub page.

Now we've got to create an API key that will allow Jenkins to interact with GitHub. Let's click on our profile picture (in the way upper right of the screen) and select **Settings** from the menu that drops down. In the next screen, click on **Developer settings**, a button that's on the lower left part of the screen. Now we can click on **Personal access tokens** in the left-hand menu of *this* page.

Click the **Tokens (classic) > Generate new token > Generate new token (classic)** button in the upper-right of the screen. We'll have to fill out some information on the next page:

* *Token description:* Give it something descriptive (**jenkins** is a good idea)
* Check **admin:repo\_hook**
* Click **Generate token** at the bottom of the page.

Now we're taken to a screen with the actual token. Copy this string, because we'll need it later.

## **Jenkins Configuration**

Use the public IP on the hands-on lab Credentials section to load our Jenkins server on your web browser. Remember to append :8080 (port 8080) to the end of the provided Jenkins Server Public IP.

Once you are logged in, click **Manage Jenkins** in the left-hand menu. On this next screen, a little ways down, click on the **Configure System** link (it's got a gear icon next to it) and we'll be able to set up the communication with GitHub.

On the next screen, scroll down a bit to the **GitHub** section. Click on the **Add GitHub Server** dropdown, and select **GitHub Server** from the list. A web form will appear. Give this a *Name* of **github**. We can leave *API URL* alone, but we need to put some credentials in. Click the **Add** dropdown, select **Jenkins**, and we'll see a window pop out with a form we've got to fill in. With this next form section, click the **Kind** dropdown and select **Secret text**. We're going to paste the token we copied earlier (when we generated it in GitHub) into the **Secret** field. Give it an *ID* of **github\_key**, and a *Description* of **GitHub Key**. Click the **Add** button. Back on the **Configure System** screen, we've got to now select that **GitHub Key** we just created, from the *Credentials* dropdown.

Leave *Manage hooks* checked, and click **Test connection** over on the right, to make sure GitHub and Jenkins are communicating. We'll get a "Credentials are verified" message, and click the **Save** button (you may have to scroll down to see it). Now we can move along.

## **Create a New Jenkins Freestyle Project Called train-schedule**

Jenkins is all set for the most part, but now we've got to actually create a project. We need the URL of our personal fork of the *cicd-pipeline-train-schedule-jenkins*, so just flip back over in the web browser and copy it real quick.

In Jenkins, click on **New Item**. Give it an item name of train-schedule, and select **Freestyle project**. Then click the **OK** button.

In this next screen, we've got to check **GitHub project**, and paste the URL we got from GitHub earlier into the *Project url* box. In the **Source Code Management** section, select **Git** and paste that same URL into the *Repository URL* box. We can leave *Credentials* alone.

In the next section, **Build Triggers**, check the **GitHub hook trigger for GITScm polling** box.

Next we've got to set up build steps. Down in the **Build** section, click the *Add build step* dropdown and select **Invoke Gradle script** from the list. Select **Use Gradle Wrapper** from the new list the shows up, and fill in the *Tasks* part of the form to say **build**.

Scroll down a bit and click on the *Add post-build action* dropdown. Select **Archive the artifacts** from the list that pops up. Now we've got a *Files to archive* form field that we need to fill in. Type **dist/trainSchedule.zip** in there.

Click **Save**, and back out at the main project screen click **Build now** to see if everything is working.

In the lower part of the left-hand menu, we can see our build progressing. Click on the **#1** next to our current build, and we can then click on **Console Output** (in this screen's left-hand menu) to watch as things area actually happening during the process.

This first build is going to take a while, because it's having to go out and download things. Subsequent builds should be much quicker.

## **Trigger a Successful train-schedule Build with a Change in GitHub**

Once the train-schedule project is fully configured, let's make a change to the source code in our GitHub fork. This should trigger a successful build of the train-schedule project. Making a change can mean making *any* change to *any* file in the repository.

Let's click on the README.md file, then once it's open click the pencil icon. We can add any text we want, just something down at the end of the file.

Once we click the **Commit Changes** button, it should trigger a new build. If we head back over to the Jenkins page, we should see a **#2** job that fired off. Notice that this one didn't take so long? Most of the heavy lifting was done in the first build, and this one only incorporated the change we made to the README.md file.

## **Conclusion**

Well, we've done it. Now whenever there's a change in our GitHub repository, it will trigger a new build in our Jenkins project. Congratulations!