Triggers y Notificaciones

Raúl Estrada

Octubre 2020

Triggers and notifications

So far, we have always built the pipeline manually by clicking on the **Build Now** button. It works but is not very convenient. All team members would have to remember that after committing to the repository, they need to open Jenkins and start the build. The same works with pipeline monitoring; so far, we manually opened Jenkins and checked the build status. In this section, we will see how to improve the process so that the pipeline would start automatically and, when completed, notify the team members about its status.

Triggers

An automatic action to start the build is called the pipeline trigger. In Jenkins, there are many options to choose from; however, they all boil down to three types:

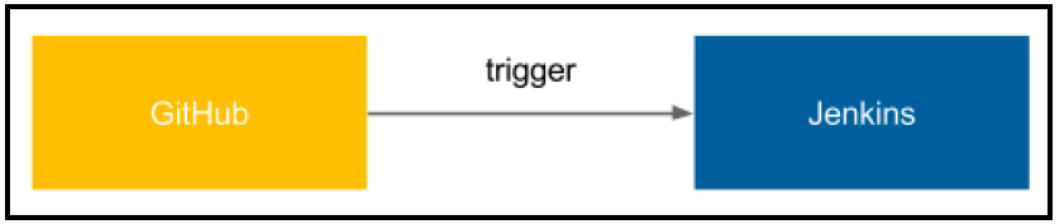
- External
- Polling SCM (Source Control Management)
- Scheduled build

Let's take a look at each of them.

External

External triggers are natural to understand. They mean that Jenkins starts the build after it's called by the notifier, which can be the other pipeline build, the SCM system (for example, GitHub), or any remote script.

The following figure presents the communication:



GitHub triggers Jenkins after a push to the repository and the build is started.

To configure the system this way, we need the following setup steps:

1. Install the GitHub plugin in Jenkins.
2. Generate a secret key for Jenkins.
3. Set the GitHub web hook and specify the Jenkins address and key.

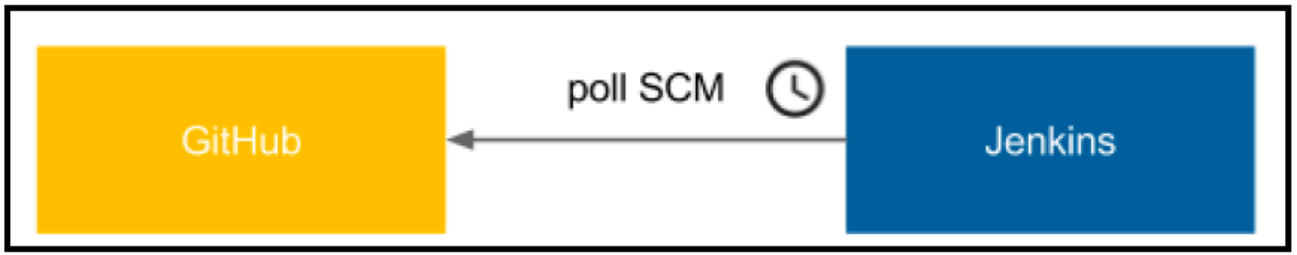
In the case of the most popular SCM providers, dedicated Jenkins plugins are always provided.

There is also a more generic way to trigger Jenkins via the REST call to the endpoint `<jenkins_url>/job/<job_name>/build?token=<token>`. For security reasons, it requires setting `token` in Jenkins and then using it in the remote script.

Jenkins must be accessible from the SCM server. In other words, if we use the public GitHub to trigger Jenkins, then our Jenkins server must be public as well. This also applies to the generic solution; the `<jenkins_url>` address must be accessible.

Polling SCM

Polling SCM trigger is a little less intuitive. The following figure presents the communication:



Jenkins periodically calls GitHub and checks if there was any push to the repository. Then, it starts the build. It may sound counter-intuitive, however, there are at least two good cases for using this method:

- Jenkins is inside the firewalled network (which GitHub does not have access to)
- Commits are frequent and the build takes a long time, so executing a build after every commit would cause an overload

The configuration of **poll SCM** is also somehow simpler because the way to connect from Jenkins to GitHub is already set up (Jenkins checks out the code from GitHub, so it needs to have access). In the case of our calculator project, we can set up an automatic trigger by adding the `triggers` declaration (just after `agent`) to the pipeline:

```
triggers {
     pollSCM('* * * * *')
}
```

After running the pipeline manually for the first time, the automatic trigger is set. Then, it checks GitHub every minute, and for new commits, it starts a build. To test that it works as expected, you can commit and push anything to the GitHub repository and see that the build starts.

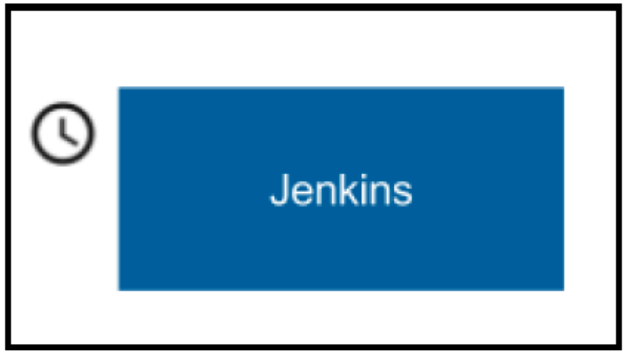
We used the mysterious `* * * * *` as an argument to `pollSCM`. It specifies how often Jenkins should check for new source changes and is expressed in the cron-style string format.

The cron string format is described (together with the cron tool) at `https://en.wikipedia.org/wiki/Cron`.

Scheduled build

Scheduled trigger means that Jenkins runs the build periodically, no matter if there was any commit to the repository or not.

As the following figure presents, there is no communication with any system needed:



The implementation of **Scheduled build** is exactly the same as polling SCM. The only difference is that the keyword `cron` is used instead of `pollSCM`. This trigger method is rarely used for the commit pipeline but applies well to nightly builds (for example, complex integration testing executed at nights).

Notifications

Jenkins provides a lot of ways to announce its build status. What's more, as with everything in Jenkins, new notification types can be added using plugins.

Let's walk through the most popular types so that you can choose the one that fits your needs.

Email

The most classic way to notify about the Jenkins build status is to send emails. The advantage of this solution is that everybody has a mailbox; everybody knows how to use the mailbox; and everybody is used to receiving information by the mailbox. The drawback is that usually there are simply too many emails and the ones from Jenkins quickly become filtered out and never read.

The configuration of the email notification is very simple; it's enough to:

- Have the SMTP server configured
- Set its details in Jenkins (in **Manage Jenkins** | **Configure System**)
- Use `mail to` instruction in the pipeline

The pipeline configuration can be as follows:

```
post {
     always {
          mail to: 'team@company.com',
          subject: "Completed Pipeline: ${currentBuild.fullDisplayName}",
          body: "Your build completed, please check: ${env.BUILD_URL}"
     }
}
```

Note that all notifications are usually called in the `post` section of the pipeline, which is executed after all steps, no matter whether the build succeeded or failed. We used the `always` keyword; however, there are different options:

- **always:** Execute regardless of the completion status
- **changed:** Execute only if the pipeline changed its status
- **failure:** Execute only if the pipeline has the **failed** status
- **success:** Execute only if the pipeline has the **success** status
- **unstable:** Execute only if the pipeline has the **unstable** status (usually caused by test failures or code violations)

Group chat

If group chat (for example, Slack or HipChat) is the first method of communication in your team, then it's worth considering adding the automatic build notifications there. No matter which tool you use, the procedure to configure it is always the same:

1. Find and install plugin for your group chat tool (for example, the **Slack Notification** plugin).
2. Configure the plugin (server URL, channel, authorization token, and so on).
3. Add the sending instruction to the pipeline.

Let's see a sample pipeline configuration for Slack to send notifications after the build fails:

```
post {
     failure {
          slackSend channel: '#dragons-team',
          color: 'danger',
          message: "The pipeline ${currentBuild.fullDisplayName} failed."
     }
}
```