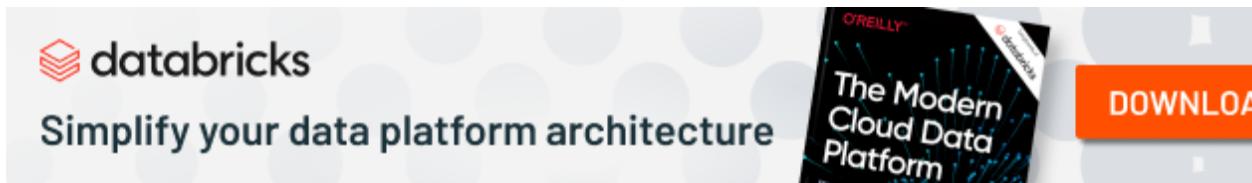




How to Schedule a Job with Jenkins



Alex Chirea



The banner features the Databricks logo and the tagline "Simplify your data platform architecture". To the right is a book cover titled "The Modern Cloud Data Platform" by O'Reilly and Databricks. A large orange "DOWNLOAD" button is on the far right.

Introduction

Scheduling a job in Jenkins is very useful when it comes to developing a great product. Imagine what a great advantage is to transform a time-consuming task into an automated job. It doesn't need any supervision and as long as there are no errors in the workflow, you shouldn't have any worries.

In today's article we'll make a job that will run every five minutes and has the following output:

```
Hello, World!  
Today's:  
<<date>>  
Here's the current time:  
<<time>>
```

Additionally, we'll build a job that watches another project, and depending on its status, prints out an automated statement.

Use-Cases for Automation with Jenkins

But before we dive into the technical part, let's take a moment to consider what can be automated in Jenkins. Of course, with a little imagination, you can tackle most problems and automate them. Let's take a look at an example that could be automated with Jenkins if you have a little (more) patience and you're also eager to deepen your Jenkins skills.

Products like Amazon Echo and Alexa could be configured to call Jenkins jobs that perform various actions. You could relatively easily integrate it into your smart home:

1. Say, "*Alexa, lock the doors*" (that would be the **trigger**, meaning Jenkins will *listen* to that event).
2. Alexa skill activates a Jenkins job.
3. Jenkins' job will use a script to call the smart doors locking system API.
4. Alexa waits for the Jenkins job to finish and tells you how it went, i.e. "*Door successfully locked*", "*The doors are already locked*" or "*The doors could not be locked*".

Of course, that's not a CI/CD job, but is well within the domain of possibilities for existing tech with Jenkins.

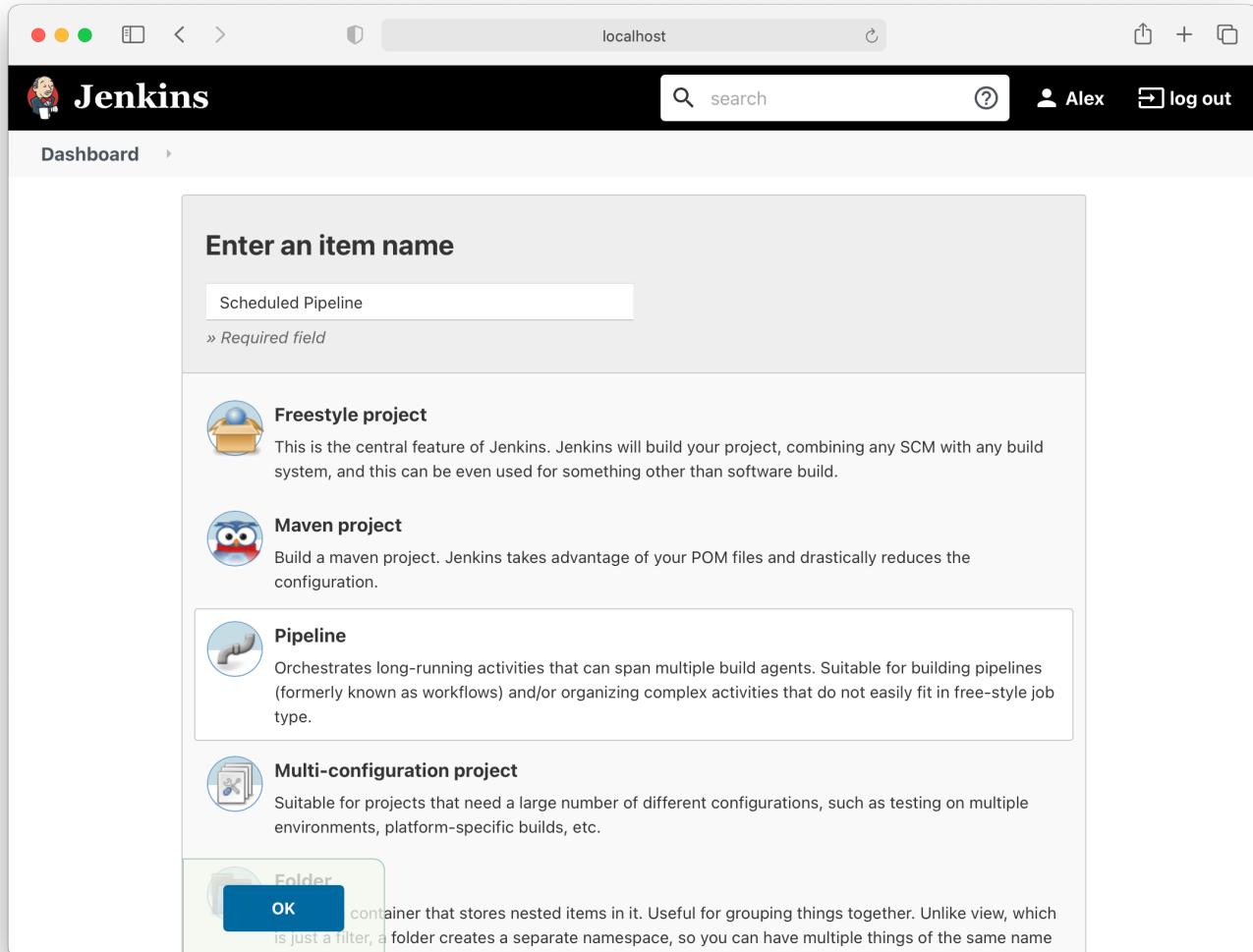
Requirements

- A Jenkins server – for this tutorial, we'll use the official Docker image to set up a Jenkins server
- A job that you'd like automated

Creating a Jenkins Job

Login in to your Jenkins application and click on *Create a Job*. The type of job doesn't matter. It can be *Freestyle*, *Maven*, etc. All that matters is how we configure the scheduler.

For this article, we'll go with *Pipeline*. Choose a name for your project and save it:



Now, before we deal with the scheduling part, let's set up the *Pipeline* Groovy script which will print the message from the introduction.

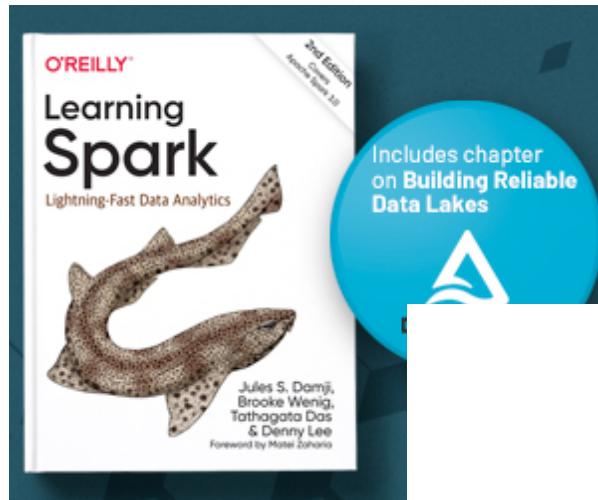
Scroll down to **Pipeline** and add the following script:

```
pipeline {
    agent any

    stages {
        stage('Write Greetings') {
            steps {
                echo 'Hello, World!'
            }
        }
    }
}
```

```
stage('Write Date') {  
    steps {  
        print 'Today\'s '  
        print new Date().format("dd-MM-yyyy", TimeZone.getTimeZone("UTC"))  
    }  
}  
  
stage('Write Hour') {  
    steps {  
        print 'Here\'s the current time '  
        print new Date().format("HH:mm:ss", TimeZone.getTimeZone("UTC"))  
    }  
}  
}
```

The script has three stages: *Write Greetings*, *Write Date*, and *Write Hour*. We use stages in order to better define and structure our script. Also, the stages can be viewed in the Jenkins UI.



In case one of them fails, you'll know which one:

The screenshot shows the Jenkins Pipeline configuration interface. At the top, there are tabs for General, Build Triggers, Advanced Project Options, and Pipeline, with Pipeline selected. Below the tabs is a section titled 'Pipeline' with a sub-section 'Definition'. In the 'Definition' section, there is a code editor containing Groovy pipeline script. The script defines a pipeline with two stages: 'Write Greetings' and 'Write Date'. The 'Write Greetings' stage has one step: 'echo "Hello, World!"'. The 'Write Date' stage has two steps: 'print "Today\\'s "' and 'print new Date().format("dd-MM-yyyy", TimeZone.getTimeZone("UTC"))'. Below the code editor is a checkbox labeled 'Use Groovy Sandbox' which is checked. At the bottom of the 'Definition' section is a 'Pipeline Syntax' link. At the very bottom of the page are 'Save' and 'Apply' buttons.

```
1 pipeline {  
2     agent any  
3  
4     stages {  
5         stage('Write Greetings') {  
6             steps {  
7                 echo 'Hello, World!'  
8             }  
9         }  
10        stage('Write Date') {  
11            steps {  
12                print 'Today\\'s '  
13                print new Date().format("dd-MM-yyyy", TimeZone.getTimeZone("UTC"))  
14            }  
15        }  
16    }  
17}
```

Running the Job

We have our job ready to be triggered! Let's click on **Build Now** and check the output:

The screenshot shows the Jenkins interface for a 'Scheduled Pipeline'. The left sidebar has a 'Build Now' button highlighted with a blue border. The main content area is titled 'Pipeline Scheduled Pipeline' and contains sections for 'Recent Changes', 'Stage View' (which says 'No data available. This Pipeline has not yet run.'), and 'Permalinks'. The bottom right of the screen shows 'REST API' and 'Jenkins 2.272'.

Started by user Alex
Running in Durability level: MAX_SURVIVABILITY
Running on Jenkins in /var/jenkins_home/workspace/Scheduled Pipeline
Hello, World!
Today's:
24-12-2020
Here's the current time:
16:00:38
Finished: SUCCESS

You can also see on the Job's page the average time for every stage:

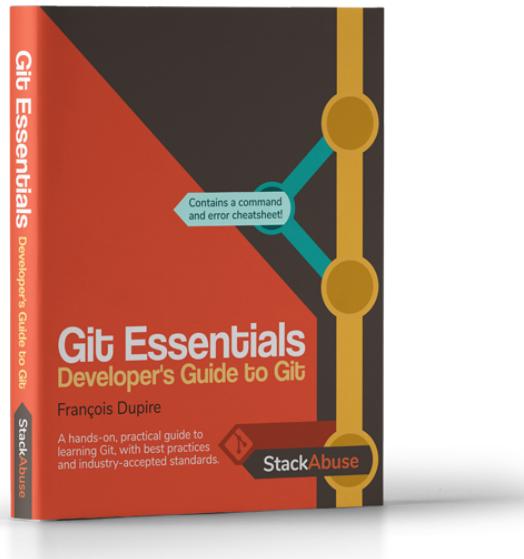
The screenshot shows the Jenkins interface for a 'Scheduled Pipeline' project. On the left, a sidebar lists various options like 'Status', 'Changes', and 'Build Now'. The main content area is titled 'Pipeline Scheduled Pipeline' and contains a 'Recent Changes' section with a note about an amazing example of scheduled jobs. Below it is a 'Stage View' section with a table showing average stage times: Write Greetings (155ms), Write Date (130ms), and Write Hour (213ms). A 'Permalinks' section lists four recent builds. The URL in the browser bar is 'localhost'.

There are many ways to trigger a Jenkins job: manually (*of course*), using a Cron Expression, hooks, watching other projects, etc.

Scheduling Jenkins Jobs with Cron

Let's start with triggering a job via the Cron utility. We'll choose **Build periodically** and for the **Schedule** field use the following expression:

```
H/5 * * * *
```



Free eBook: Git Essentials

Check out our hands-on, practical guide to learning Git, with best-practices, industry-accepted standards, and included cheat sheet. Stop Googling Git commands and actually *learn* it!

[Download the eBook](#)

The screenshot shows the Jenkins 'Scheduled Pipeline' configuration interface. The 'Build Triggers' tab is active. Under 'Build periodically', the cron expression 'H/5 * * * *' is set. A tooltip provides a detailed explanation of cron syntax:

- This field follows the syntax of cron (with minor differences). Specifically, each line consists of 5 fields separated by TAB or whitespace:
- MINUTE HOUR DOM MONTH DOW
- MINUTE Minutes within the hour (0–59)
- HOUR The hour of the day (0–23)
- DOM The day of the month (1–31)
- MONTH The month (1–12)
- DOW The day of the week (0–7) where 0 and 7 are Sunday.

To specify multiple values for one field, the following operators are available. In the order of precedence,

- * specifies all valid values
- M–N specifies a range of values
- M–N/X or */X steps by intervals of X through the specified range or whole valid range
- A,B,...,Z enumerates multiple values

To allow periodically-scheduled tasks to produce even load on the system, the symbol # (for "hash") should be used whenever possible. For example, using 0 0 * * * for a dozen daily jobs will cause a large spike at H * * * * would still execute each job once a day, but not all at the same time, better using limited resources.

At the bottom, there are 'Save' and 'Apply' buttons.

This expression means the job will be executed every 5 minutes. A few useful examples of other patterns could be:

- 0 0 13 1/1 * ? * – every hour, starting with 13:00;
- 0 0 17 ? * MON–FRI * – every working day, at 5 PM;

If you don't want to learn this type of formatting in order to schedule your job, you can use a *Cron Expression* generator like [CronMaker](#).

Let's see the result. For this, we'll have to wait. After 10 minutes you should see that 2 jobs were triggered automatically:

The screenshot shows the Jenkins Pipeline interface. On the left, there's a sidebar with various project management options like Status, Changes, Build Now, Configure, Delete Pipeline, Full Stage View, Rename, and Pipeline Syntax. Below that is the Build History section, which lists five builds: #5 (Dec 24, 2020 9:22 PM), #4 (Dec 24, 2020 9:17 PM), #3 (Dec 24, 2020 9:12 PM), #1 (Dec 24, 2020 4:00 PM), and #2 (Dec 24, 2020 3:57 PM). The build at 9:12 PM is highlighted with a blue box. To the right of the sidebar is the main content area titled "Pipeline SCHEDULED PIPELINE". It contains a "Recent Changes" section with a small icon and the text "Recent Changes". Below it is the "Stage View" section, which is highlighted with a red box. The Stage View shows average stage times and a detailed table for each build. The table has columns for "Write Greetings", "Write Date", and "Write Hour". The data for the builds is as follows:

| Build | Write Greetings | Write Date | Write Hour |
|-------|-----------------|------------|------------|
| #5 | 354ms | 170ms | 206ms |
| #4 | 243ms | 144ms | 140ms |
| #3 | 133ms | 120ms | 114ms |
| #1 | 886ms | 288ms | 357ms |
| #2 | 155ms | 130ms | 213ms |

Notice the first job was executed at 9:12 PM, then at 9:17 PM, and the last at 9:22 PM. **So the rule worked!** We have a job which runs every 5 minutes.

Scheduling a Jenkins Job by Watching Other Projects

Now that we've seen how we can schedule a job periodically, let's create another one that triggers itself every time the first one was successful. Create another Pipeline project, and add the following code:

```
pipeline {
    agent any

    stages {
        stage('It worked') {
            steps {
                echo 'The other job worked!'
            }
        }
    }
}
```

}

}

We have our pipeline script, let's schedule it. In the **Build Triggers** tab, select ***Build after other projects are built.***



Enter the project which needs to be watched and select the cases when this one should be triggered:

Save it and let's wait for a little. You should see the new job trigger itself every time the first one succeeds. You can see the details of the first run of this second job that was triggered by the job created before:

The screenshot shows the Jenkins dashboard for the 'The other job worked' project. The build number is #1, and it was triggered on Dec 24, 2020, at 9:32:25. A red box highlights the 'Started by upstream project' message, which states: 'Started by upstream project [Scheduled Pipeline](#) build number 7 originally caused by: • Started by timer'. The Jenkins version is 2.272.

Here's the output:

```
Started by upstream project "Scheduled Pipeline" build number 7
originally caused by:
  Started by timer
Running in Durability level: MAX_SURVIVABILITY
Running on Jenkins in /var/jenkins_home/workspace/The other job worked
The other job worked!
Finished: SUCCESS
```

Conclusion

Automation is very useful and one of the key things technology brings us, regardless of the process you want to automate.

In this article, we've gone over how to schedule a job in Jenkins to run periodically using a Cron pattern, as well as to run when other projects are built.

#tool #devops #jenkins #groovy

Last Updated: January 25th, 2021

Was this article helpful? 



Improve your dev skills!

Get tutorials, guides, and dev jobs in your inbox.

Enter your email

Sign Up

No spam ever. Unsubscribe at any time. Read our [Privacy Policy](#).



Alex Chirea *Author*



ALSO ON STACKABUSE

JavaScript: Generate Random Number in ...

9 months ago • 2 comments

In this tutorial, we'll go over how to generate a random number in range using ...

Borůvka's Algorithm in Python

4 months ago • 2 comments

In this guide, we'll take a look at the theory and implementation of ...

Python: How Remove a K

6 months ago • 1

Let's learn three Python to remove pairs from a dic

[1 Comment](#)[StackAbuse](#)[Privacy Policy](#)[Login](#)[Recommend](#)[Tweet](#)[Share](#)[Sort by Best](#)

Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS [?](#)

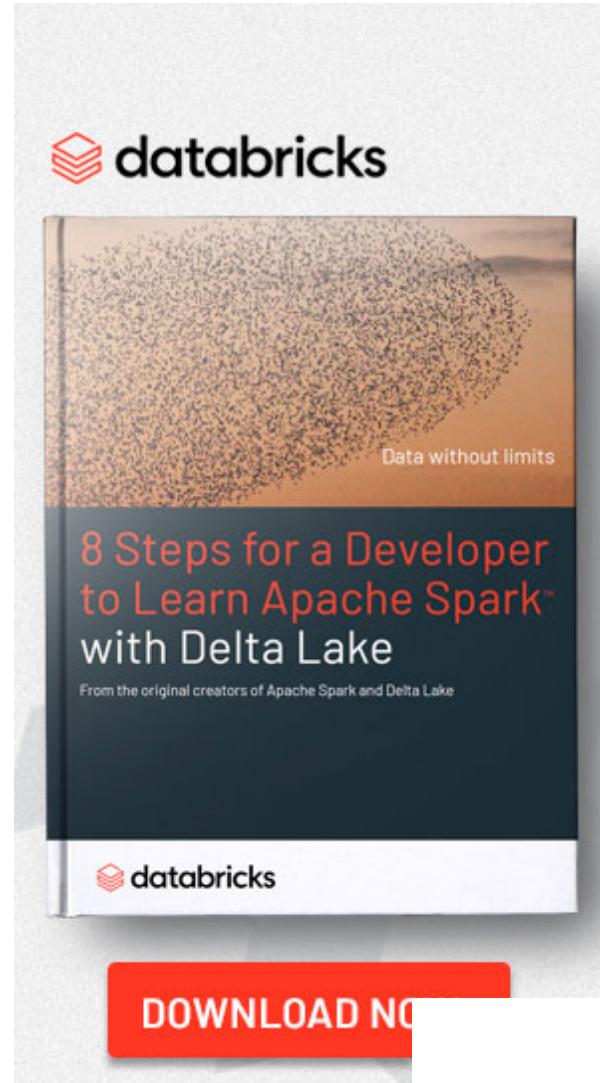
Name

**Rick Kasten** • 3 months ago

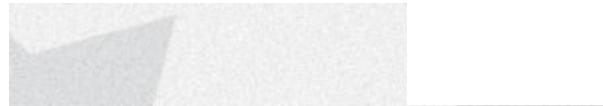
The cron examples you provided

```
0 0 13 1/1 * ? * – every hour, starting with 13:00;  
0 0 17 ? * MON-FRI * – every working day, at 5 PM;
```

do not work. Jenkins requires 5 fields for periodic triggers, not 6.
Jenkins is not as versatile as cron.

[^](#) | [▼](#) • Reply • Share •

The advertisement features the Databricks logo at the top left. Below it is a large image of a starling murmuration against a sunset sky, with the text "Data without limits" overlaid. The main headline reads "8 Steps for a Developer to Learn Apache Spark™ with Delta Lake". A smaller subtext states "From the original creators of Apache Spark and Delta Lake". At the bottom, there's another Databricks logo and a large red button with the text "DOWNLOAD NOW".



Want a remote job?

Senior Go Developer

Toptal 8 days ago

Senior DevOps Engineer (m/f/x)

refurbed 9 days ago

Full Stack Engineer

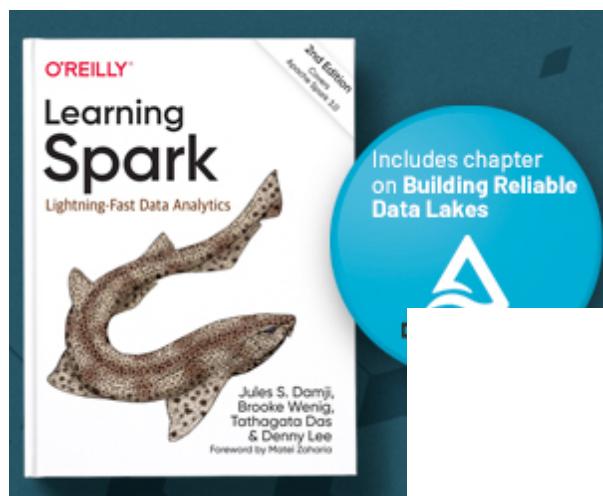
Refinable a month ago

Senior React Developer - Remote

Toptal 2 months ago

More Jobs

Jobs by [HireRemote.io](#)



Prepping for an interview?

Improve your skills by solving one coding problem every day

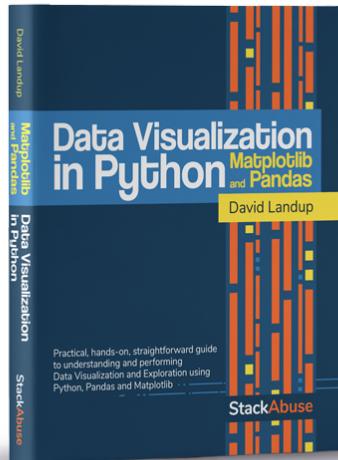
Get the solutions the next morning via email

Practice on **actual problems** asked by top companies, like:



Daily Coding Problem

Better understand your data with visualizations



- ✓ 30-day no-questions refunds

- ✓ Beginner to Advanced
- ✓ Updated regularly (update June 2021)
- ✓ New bonus resources and guides

[Learn more](#)

DELL

It's worth another look.

4.3 (303 Reviews)

Inspiron 15 5000
₹64,990.02

McAfee™
Protect what matters.

Stay protected online for 12 months across all devices, with pre-installed McAfee® LiveSafe™.



