


Jenkins Tutorial

Jenkins is an open source continuous integration tool used to shorten the development lifecycle by automating the test and build process. We built Jenkins into Project Diaper's infrastructure to automate away some of the tedious tests previously given to the infrastructure team when new builds were being deployed.

Current Jenkins server:

Hosted at <https://jenkins.diaper.cf/> with the help of [nginx](#). Please see the nginx config file at the following location for more information.

 /etc/nginx/nginx.conf

Administrators login using the username and password stored in the login secrets folder on Box.

Current CI/CD Architecture

The current Jenkins configuration consists of 3 pipelines: one for dashboard-frontend, one for dashboard-backend, and one for mobile-backend. The pipelines are configured to automatically build the docker image after a push to the master branch (Github uses [webhook](#) to notify Jenkins that a new push is made). It then **docker push** the image to our private docker registry, called nexus, which is hosted on our K8s Cluster. Finally, an email with all output from each pipeline job run and the results of the job are sent to the diaperjenkins@gmail.com email address (password set to the password stored in the login secrets file on Box).

Each pipeline is currently hosted here [DIAPER-project/jenkins-pipelines](#) in the form of three separate Jenkinsfiles which specify the main behavior of the pipelines.

How to add on

If you want to build a new Jenkins pipeline, you start by navigating to the [Jenkins dashboard](#), login with the username and password.

Click "New Item," then enter a name and click "Pipeline," followed by "Ok." Then configure the pipeline with the following options:

General

Enabled



Description

CI pipeline for dashboard-frontend.
1. Trigger by Github WebHook. Used GenericTrigger plugin
2. Clone the code.
3. Docker build and push the image to nexus
4. Modify k8s configs. (ArgoCD will pickup the change)

Plain text [Preview](#)

☐ Discard old builds [?](#)

☒ Do not allow concurrent builds

☒ Abort previous builds [?](#)

☒ Do not allow the pipeline to resume if the controller restarts

☒ GitHub project

Project url [?](#)

<https://github.com/DIAPER-Project/dashboard-frontend.git/>

Advanced [?](#)

☐ Pipeline speed/durability override [?](#)

☐ Preserve stashes from completed builds [?](#)

☐ This project is parameterized [?](#)

☐ Throttle builds [?](#)

Build Triggers

☐ Build after other projects are built [?](#)

☐ Build periodically [?](#)

☒ Generic Webhook Trigger [?](#)

Is triggered by HTTP requests to http://JENKINS_URL/generic-webhook-trigger/invoke

There are example configurations in [the Git repository](#).

You can fiddle with JSONPath [here](#). You may also want to checkout the syntax [here](#).

You can fiddle with XPath [here](#). You may also want to checkout the syntax [here](#).

You can fiddle with regular expressions [here](#). You may also want to checkout the syntax [here](#).

If your job is **not parameterized**, then the resolved variables will just be contributed to the build. If your job is **parameterized**, and you resolve variables that have the same name as those parameters, then the plugin will populate the parameters when triggering job. That means you can, for example, use the parameters in combination with an SCM plugin, like GIT Plugin, to pick a branch.

Post content parameters

Variable

Name of variable

ref

Expression

\$.ref

☒ JSONPath

☐ XPath

Expression to evaluate in POST content. Use [JSONPath](#) for JSON or [XPath](#) for XML.

Value filter

Optional. Anything in the evaluated value, matching this [regular expression](#), will be removed. Having `[^0-9]` would only allow numbers. The regexp syntax is documented [here](#).

Default value

Optional. This value will be used if expression does not match anything.

Add

If you want value of **param1** from post content { "param1": "value1" } to be contributed, you need to add **\$.param1** here.

Header parameters

Add

If you want value of header **param1** to be contributed, you need to add "param1" here.

Request parameters

Add

If you want value of query parameter **param1** to be contributed, you need to add "param1" here.

Token

dashboard-frontend

Optional token. If it is specified then this job can only be triggered if that token is supplied when invoking **http://JENKINS_URL/generic-webhook-trigger/invoke**. It can be supplied as a:

- Query parameter **/invoke?token=TOKEN_HERE**
- A token header **token: TOKEN_HERE**
- A Authorization: Bearer header **Authorization: Bearer TOKEN_HERE**

Token Credential

- none -

+ Add

Same as **token** above, but configured with a *secret text* credential.

Cause

Triggered by \$ref

This will be displayed in any triggered job. You can use the variables here to create a custom cause like "*\$name committed to \$branch*", if you have configured variables named **name** and **branch**.

☐

Override Quiet Period

Allow the trigger to override this job's quiet period. If selected you can provide a quiet period (an integer number of seconds) and the build will use that quiet period instead of its default. If this is selected and no quiet period is given the job's quiet period will still be used. It can be supplied as a:

- Query Parameter **/invoke?jobQuietPeriod=QUIET_PERIOD_HERE**
- A quiet period header **jobQuietPeriod: QUIET_PERIOD_HERE**

☐

Allow several triggers per build

If checked, the plugin will allow one build to have several triggers. If not checked, the plugin will trigger exactly one job when invoked.

☐

Silent response

Avoid responding with information about triggered jobs.

☐

Avoid flattening branches

Avoid flattening any selected branch. If the selected node is a branch, not a leaf, the plugin will, by default, flatten its content and create variables for each leaf on that branch.

☒

Print post content

Print post content in job log.

☒

Print contributed variables

Print contributed variables in job log.

Optional filter

Expression

Regular expression to test on the evaluated text specified below. The regexp syntax is documented [here](#).

Text

Text to test for the given [expression](#). You can use any combination of the variables you configured above.

This is an optional feature. If specified, this job will only trigger when given expression matches given text.

- ☐ GitHub hook trigger for GITScm polling ?
- ☐ Poll SCM ?
- ☐ Quiet period ?
- ☐ Trigger builds remotely (e.g., from scripts) ?

Advanced Project Options

Advanced ▾

Pipeline

Definition

Pipeline script from SCM ▾

SCM ?

Git ▾ ?

Repositories ?

Repository URL ?

Credentials ?

+ Add ▾

Advanced ▾

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

*/main

Add Branch

Repository browser ?

(Auto) ▾

Additional Behaviours

Add ▾

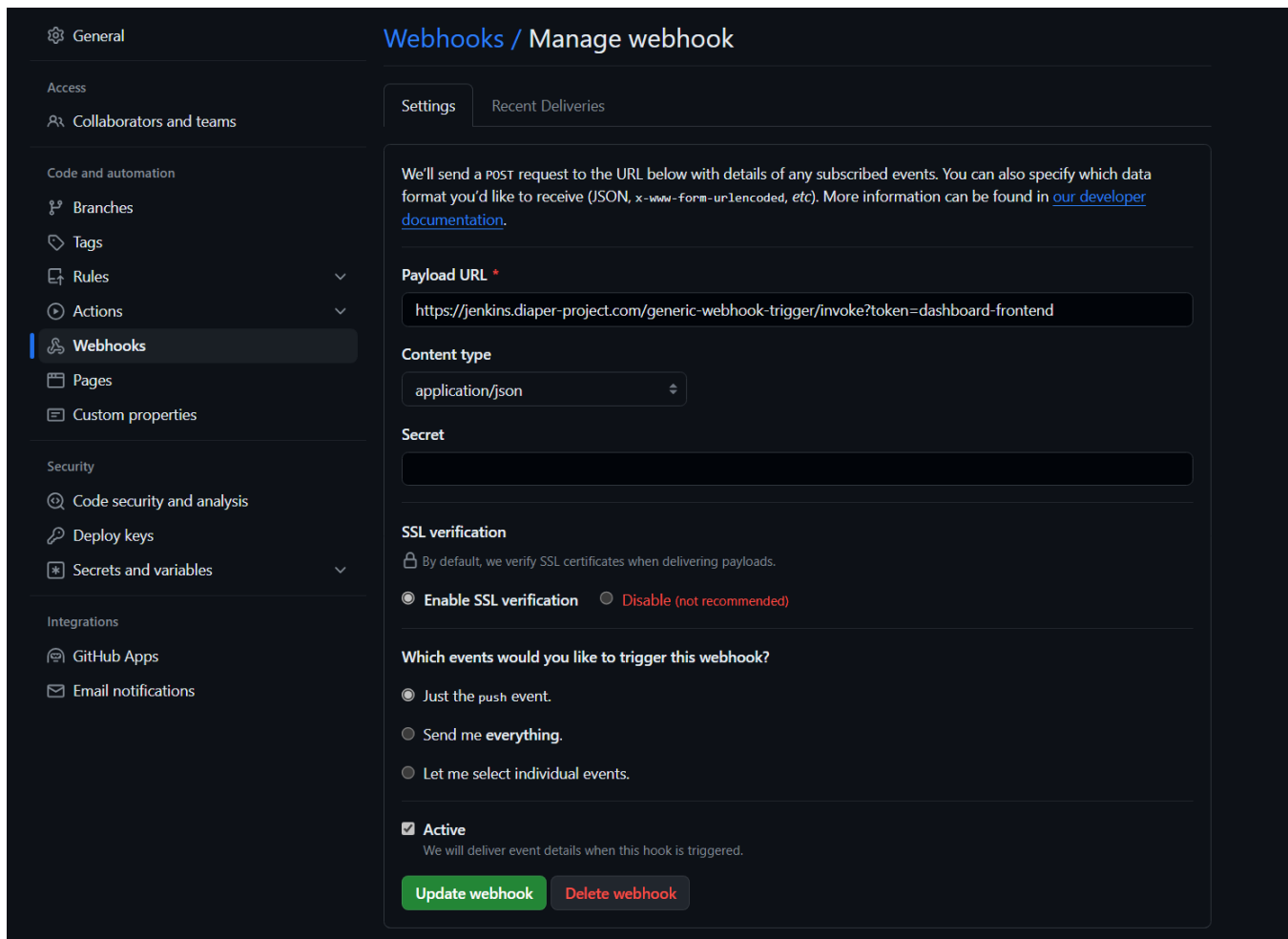
Script Path ?

JenkinsFileDashboardFrontend

☒ Lightweight checkout ?

[Pipeline Syntax](#)

Finally, you need to add web hooks to that very repository with the following configuration:



Troubleshooting

The first step to diagnose issues with the Jenkins server itself is to ssh into the Jenkins EC2 instance and run the following command:

```
$ sudo systemctl status jenkins
```

It should say Active: active (running). If you see some other status besides that you need to restart the server with the following command:

```
$ sudo systemctl restart jenkins
```

For other issues with the Jenkins server itself consider making sure its SSL cert is not expired or checking the status of the nginx server.

If Jenkins is up and running and anything goes wrong with a build you should consult the errors either sent to the email or stored in the failed job report in Jenkins. You can find this by navigating to jenkins.diaper-project.com and signing in with the username and password and clicking on the relevant job.

Restoring the Jenkins Server:

In the incredibly unlikely event that the Jenkins server fails, we backed up our EC2 instance to avoid repeating the many hours of work that went into the current configuration. All you need to do is navigate to the EC2 management console, click "Launch an instance", then click on "My AMIs" and select the most recently stored "Jenkins Server" Linux AMI. Next select T3 micro as the instance type. Then click the configure instance header at the top of the screen and then set the following settings:

Resource Groups & Tag Editor

1. Choose AMI
2. Choose Instance Type
3. Configure Instance
4. Add Storage
5. Add Tags
6. Configure Security Group
7. Review

Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances ⓘ

1

Launch into Auto Scaling Group ⓘ

Purchasing option ⓘ

☐ Request Spot instances

Network ⓘ

vpc-03d003c370593e7f4 | cu-cals-johnson-lab-vpc ⓘ
Create new VPC ⓘ

Subnet ⓘ

subnet-08d780a85643d1a14 | cu-cals-johnson-lab ⓘ
Create new subnet ⓘ

Auto-assign Public IP ⓘ

Enable ⓘ

Finally go into "Configure Security Group" click "Select an existing security group" and then check the box next to Production SG. Finally click "Review and Launch" followed by "Launch." Next select to use an existing key pair from the dropdown menu that appears, select DIAPER-production-key as the key, and click "Launch Instances." A new AWS EC2 server should now begin running with all configurations as of the end of Fall 2021 stored. I recommend backing up the server after any significant changes to the Jenkins EC2 instance configuration.

Installing Jenkins:

While you will likely never need to setup Jenkins yourself, as in the case of a failure you should just restore the EC2 instance. In case that is not possible we created this tutorial to make it easier to reconfigure Jenkins on a new EC2 instance if the need were to arise so you can reuse the pipeline code stored in GitHub and avoid starting a new pipeline from scratch. Before beginning this tutorial make sure you are comfortable interacting with our AWS EC2 instances via SSH.

Generally you just need to follow the tutorial here, and troubleshooting info for issues I ran into during the initial installation are provided below:

<https://www.jenkins.io/doc/tutorials/tutorial-for-installing-jenkins-on-AWS/>

Errors

```
curl: (60) SSL certificate problem: certificate has expired
More details here: https://curl.haxx.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.
error: https://pkg.jenkins.io/redhat-stable/jenkins.io.key: import read failed(2).
```

Try running

```
$ yum upgrade ca-certificates
```

and if that doesn't work try

```
$ sudo yum upgrade ca-certificates --disablerepo=jenkins
```

If you get the following error:


```
Error: Package: jenkins-2.319.1-1.1.noarch (jenkins)
      Requires: daemonize
You could try using --skip-broken to work around the problem
You could try running: rpm -Va --nofiles --nodigest
```

run

```
$ sudo amazon-linux-extras install epel -y
$ sudo yum install daemonize -y
$ sudo yum install jenkins java-1.8.0-openjdk-devel -y
```

Additionally following the installation you need to change the default user Jenkins runs as to "ec2-user". You can follow instructions given here: <http://blog.manula.org/2013/03/running-jenkins-under-different-user-in.html>

You will also need to install the Publish Over SSH plugin and add the private key of the Jenkins server to the Publish Over SSH options under the "Configure System" subpage of "Manage Jenkins." You'll also need to configure a key pair for accessing Github using "Manage Credentials" under "Manage Jenkins." Information on how to configure these plugins is widely available on StackOverflow and Jenkins own docs page.