

03 CI/CD Jenkins

Objective

Part 1

Install Jenkins on a new virtual machine.

Part 2

Have a go at entering some simple configurations into your first job:

- Choose any of your projects from [GitHub](#)
- Enter some information about the project into the `Project url` field, for the `GitHub project` option
- Set a `String Parameter` for the job:
 - Set the `NAME` field to be `VERSION`
 - Set the `Default Value` field to be `0.1.0`
 - Set the `Description` field to be `Version of the application to build`
- Select the `Save` button (you will be redirected to the job's page)
- Select `Build with Parameters`. You should then see the parameter that you configured
- Select `Delete Project` and confirm the deletion

Part 3

We'll have a look at creating some builds here, focusing on build step configuration, statuses and storing artifacts.

Create a Job

First, we will need a new Jenkins job to work on, so create one called anything that you like.

Add a Build Step

Let's add a build step that we know will succeed. Select the `Add build step` button and add an `Execute shell` build step.

Add the following script into the `Command` field:

```
#!/usr/bin/env bash
echo "Hello from the Jenkins job named: ${JOB_NAME}"
```

Run the Job

Save the job and then build it; you should then have one successful build in your history for that job.

Once you navigate to the console output, you should see an output like this:

Console Output

```
Started by user admin
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/My Project
[WS-CLEANUP] Deleting project workspace...
[WS-CLEANUP] Deferred wipeout is used...
[WS-CLEANUP] Done
[My Project] $ /bin/sh -xe /tmp/jenkins2828915287112120083.sh
+ echo 'Hello from the Jenkins job named: My Project'
Hello from the Jenkins job named: My Project
Finished: SUCCESS
```

Make the Build Fail

Now that the last build succeeded, let's see what a failed build looks like!

All we have to do is make it so that the script we added to the command box "fails".

Jenkins will treat any script or application exiting with a non-zero status as a failure.

So to create a failed build, let's add `exit 1` to the script box, which will make the script exit with a code of 1:

```
#!/usr/bin/env bash
echo "Hello from the Jenkins job named: ${JOB_NAME}"
exit 1
```

Console Output

```
Started by user admin
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/My Project
[WS-CLEANUP] Deleting project workspace...
[WS-CLEANUP] Deferred wipeout is used...
[WS-CLEANUP] Done
[My Project] $ /bin/sh -xe /tmp/jenkins8077683492031581351.sh
+ echo 'Hello from the Jenkins job named: My Project'
Hello from the Jenkins job named: My Project
+ exit 1
Build step 'Execute shell' marked build as failure
Finished: FAILURE
```

Fix the Build and Create Artifacts

We can, of course, remove the `exit 1` from the build step to fix it.

After that, let's change the script to create several files, and then put them in a zip archive called `archive.zip`:

```
#!/usr/bin/env bash
echo "Hello from the Jenkins job named: ${JOB_NAME}"
touch 1.txt 2.txt 3.txt 4.txt 5.txt
zip archive.zip *.txt
```

Next, a **Post-build Action** must be configured to archive the zip files. Please refer to the [Artifacts](#) section for configuring this.

Finish Up

Now try to run the job. You should see artifacts on the project dashboard. If they don't show up, try refreshing the page.

Part 4

This task will take you through configuring a very simple Freestyle Project, which will download a script and run it.

Prerequisites

- GitHub Account
- Jenkins Installed

Create a New GitHub Repository

GitHub is of course where our code will be, we'll need to prove that Jenkins can download that code and access in the Job.

Setup a repository so we can configure a Jenkins job to access and use it in later steps:

1. Create a *public* GitHub repository for this exercise, you can call it `jenkins-freestyle-project`.
2. Add a script to the repository called `run.sh`, with the contents:
3. `echo 'Hello from run.sh!'`

Create a Jenkins Job

The Jenkins job is going to be able to:

- download the repository that we created
 - run the `run.sh` script
1. Create a new Freestyle Project on Jenkins, you can this job whatever you would like.
 2. Configure the Job to download the repository
 - Under *Source Code Management*, select *Git*
 - Enter `https://github.com/[YOUR_USERNAME]/jenkins-freestyle-project`, replacing `[YOUR_USERNAME]` with your GitHub username.
 3. In the *Build* section, create a build step to *Execute shell* and enter the following:
 4. `sh run.sh`

Run the Job

Now everything is setup, *Save* the changes that were made and the *Build* the job.

Go and check the console output of the build to see that the job has executed, the end of the output should show that the script on the repository has run correctly:

```
+ sh run.sh
+ Hello from run.sh!
+ Finished: SUCCESS
```

Clean Up

Feel free to now delete the created resources:

- Jenkins job

- **GitHub Repository**

Part 5

Create a pipeline job that has 3 stages.

1. Clones down a repository, making sure it doesn't already exist.
2. Creates a script in that repository
3. Pushes the new script to Github

Part 6

In this demo we'll be automating the deployment of a simple website.

There are 3 main components to this application:

- **Frontend**
A HTML and JavaScript website that makes calls to the Backend service.
This website is hosted with NGINX as a webserver
- **Backend**
A Python Flask server which can connect to a database
- **Database**
MySQL Database.

Prerequisites

- Jenkins installed on a Ubuntu/Debian machine, ideally a virtual machine so that it can be deleted afterwards.
- Full `sudo` access for the `jenkins` user with no password required.
- Jenkins must not be installed in a Docker container for this demo
- Ports `8080` and `80` accessible to the machine

Setup Git Repositories

For this example we are going to need couple of Git repositories.

Go ahead and create these repositories on GitHub:

- `jenkins-scm-frontend`
- `jenkins-scm-backend`
- `jenkins-scm-database`

Upload the Provided Code

Upload the code found in the `frontend`, `backend` and `database` projects available at <https://gitlab.com/qacdevops/jenkins-scm> to these repositories that you created.

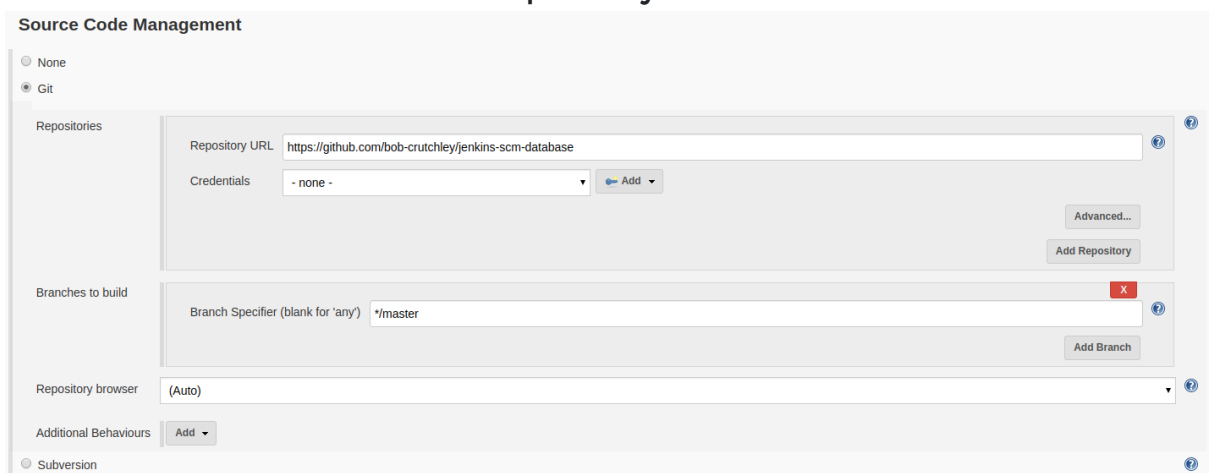
You can do this by initializing the folders as Git repositories, copying in the code and then configuring your GitHub repositories as remotes.

Create the Jenkins Jobs

You will need a Jenkins job for each GitHub repository; `frontend`, `backend` and `database`.

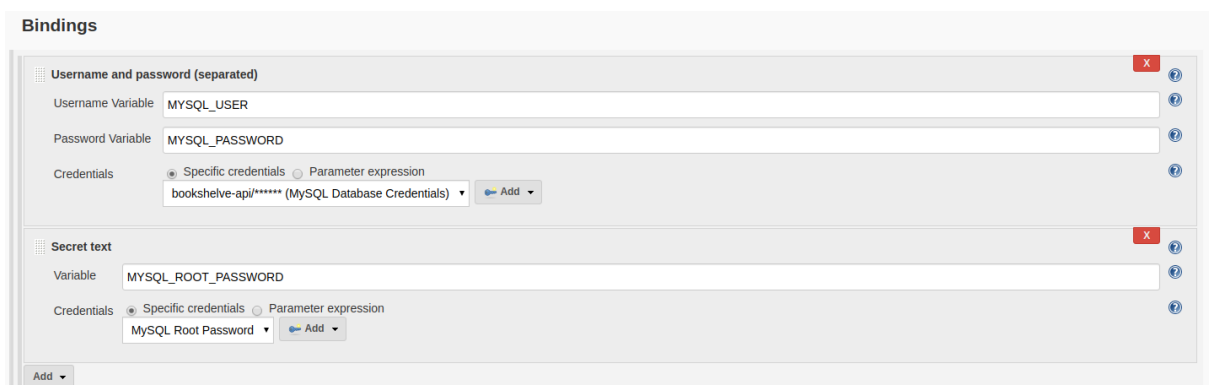
Lets create the database job first.

- Configure the source code management section to download the master branch from database repository.



The screenshot shows the 'Source Code Management' configuration page in Jenkins. The 'Git' radio button is selected. Under 'Repositories', the 'Repository URL' is set to 'https://github.com/bob-crutchley/jenkins-scm-database' and 'Credentials' is set to '- none -'. There is an 'Add Repository' button. Under 'Branches to build', the 'Branch Specifier (blank for \'any\')' is set to '*/master' and there is an 'Add Branch' button. The 'Repository browser' is set to '(Auto)'. At the bottom, there is an 'Additional Behaviours' section with an 'Add' button and a 'Subversion' radio button.

- Check the `GitHub hook trigger for GITScm polling` option under the `Build Triggers` section.
- Under `Build Environment`, select `Use secret text(s) or file(s)`. Add a `Username and Password (separated)` and a `Secret text` binding. Use the add button to create credentials for these and name the variables as shown below:



The screenshot shows the 'Bindings' configuration page in Jenkins. It has two sections: 'Username and password (separated)' and 'Secret text'. In the 'Username and password (separated)' section, 'Username Variable' is 'MYSQL_USER' and 'Password Variable' is 'MYSQL_PASSWORD'. 'Credentials' is set to 'Specific credentials' with a dropdown showing 'bookshelve-api/***** (MySQL Database Credentials)' and an 'Add' button. In the 'Secret text' section, 'Variable' is 'MYSQL_ROOT_PASSWORD' and 'Credentials' is set to 'Specific credentials' with a dropdown showing 'MySQL Root Password' and an 'Add' button. There is an 'Add' button at the bottom left.

The actual value of the credentials (password contents etc) can be whatever you like.

- Add an `Execute shell` build step with the following configured as the command:

```
export MYSQL_USER
export MYSQL_PASSWORD
export MYSQL_ROOT_PASSWORD
export MYSQL_HOST="localhost"
export MYSQL_DATABASE="bookshelve"
./setup.sh
```

- Build the job to make sure that it succeeds

Now that you have a working Job for the database, complete the same steps as above for the, backend and frontend.

Make sure to reuse any credentials that you made previously, don't create new ones.

See the Application Working

You should now be able to navigate to your machine's address on port `80` to see the application working.

Web hook Configuration

We have already configured the Jenkins jobs to accept web hooks as triggers so now we just need to setup the web hook on a Git service provider like GitHub etc.

For this example we will be discussing how you can use GitHub to send Web Hooks to your instance of Jenkins, do the following for each of the GitHub projects that you have created:

1. On your GitHub project navigate to the `Settings` tab.
2. Click on `Webhooks`
3. Set the `Payload URL` to be `[JENKINS_ADDRESS]/github-webhook/`; Don't miss the trailing `/` on the end of the Payload URL!
4. Set the `Content type` to be `application/json`
5. Select `Add Webhook`

Push a New Change

We can now have a look at making some changes on the remote repositories to trigger automatic deployments of our new changes.

Frontend

Add `<h1>UPDATE</h1>` after line `9` in the `frontend/index.html` file.

Push the new changes to the GitHub repository, then then you should see the frontend Job automatically build.

Now try navigating to the site to see your changes.

Database

We can add a new item into the database to see the changes.

In the `database/setup.sql` append the statement shown below, then push the changes to the database repository:

```
INSERT INTO Books (  
    Name, Author, Image  
) VALUES (  
    "Harry Potter and the Philosopher's Stone",  
    "J.K. Rowling",  
    "https://books.google.com/books/content/images/frontcover/39iYWTb6n6cC?fife=w200-h300"  
);
```

The changes that you made should then be reflected on the running site.