Lab 5 - Actual CI/CD pipeline

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 an Ubuntu machine, ideally in EC2.
* 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 from the Jenkins server.

### **1. Fork and clone the Git repositories**

For these three GitHub repositories:

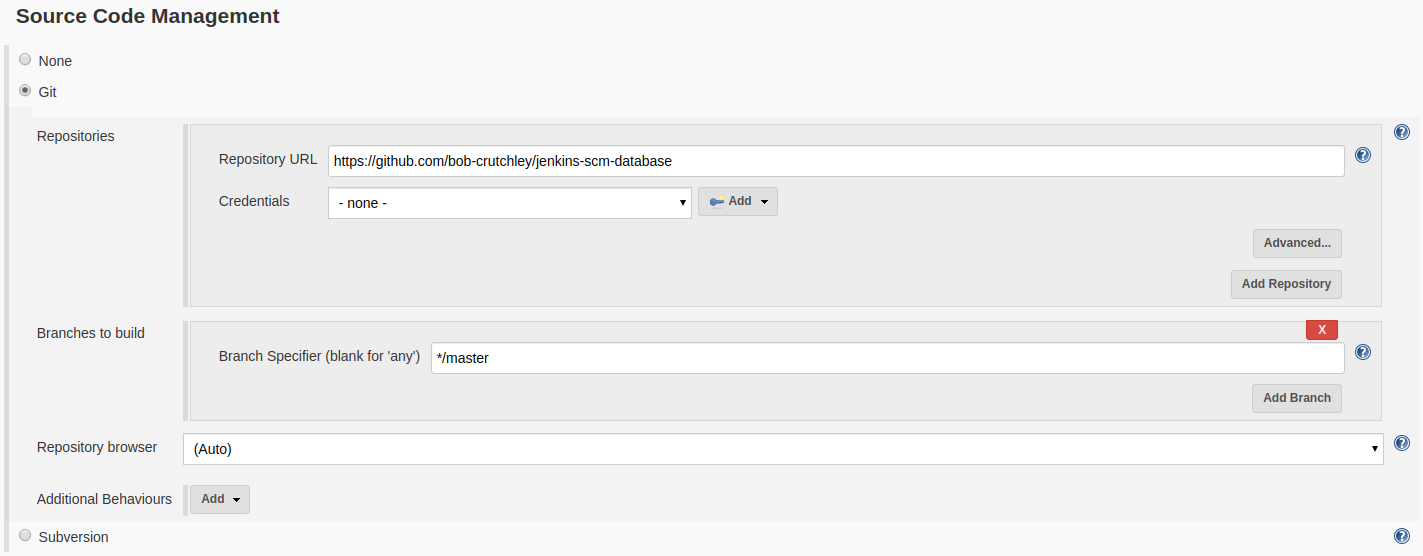
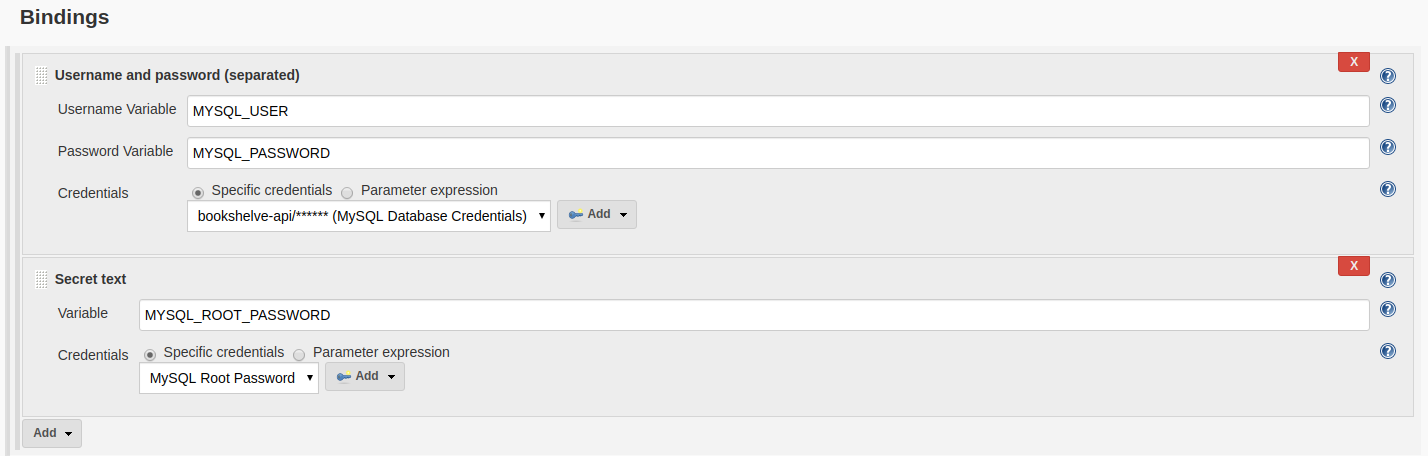
* https://github.com/qa-apprenticeships/devops-backend
* https://github.com/qa-apprenticeships/devops-database
* https://github.com/qa-apprenticeships/devops-frontend

Then clone each of them onto your local PC.

### **2. 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 devops-database GitHub repository.  
  
* 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 actual value of the credentials (password contents etc) can be whatever you like -- the Jenkins job will (shortly) use the username and password you enter to setup a MySQL database with that username and password as its "root" user account. If you're stuck, try using "bookshelve-api" and "Abcd1234". For the root password, again, you're sprecifying what you want it to be setup as, try "Defg5678".
* 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"

chmod +x setup.sh

./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 projects in GitHub.

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

Note: if during the setup of the backend service, your EC2 crashes during the [Install] stage, you may need to do this:

* stop and restart your EC2 instance (note it will be given a new IP address)
* restart your Jenkins service, via PuTTY/SSH:
  + sudo systemctl restart jenkins
* Run the Jenkins job again

Note: to test the backend (API) project is working, use PuTTY/SSH to run the following command:

* curl localhost:8000/api/books

Note: if you get an error to do with database access, make sure the mysql docker container is running, using the PuTTY/SSH command:

* sudo docker container start mysql

### **3. See the Application Working**

You should now be able to navigate to your EC2 machine's address on port 80 to see the application working.

If you can't access the website remotely, try accessing it locally from within PuTTY/SSH, using:

* curl localhost/index.html

### Then check your inbound IP rules on the EC2 instance if you still have issues accessing the website from outside.

### **4. 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, then Add webhook
3. Set the Payload URL to be http://[JENKINS\_IP]:8080/github-webhook/  
   Don't forget the trailing / on the end of the Payload URL!
4. Set the Content type to be application/json
5. Select Add Webhook

### **5. 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.