Notes on Jenkins Labs

# Adding projects to git

This assumes you are creating your repository from scratch, as opposed to forking the projects from the qa-apprenticeships GitHub projects. It also assumes you're using SSH credentials to authenticate (rather than username + password) - if not, change the URL to be in HTTPS format.

cd database

git init

git add .

git commit -m "first commit"

git remote add origin git@github.com:fredsmith1970/jenkins-scm-database.git

git push -u origin master

# Setting up GitHub App security

This allows us to authenticate from Jenkins to GitHub - without needing to use username and password (which is being deprecated and won't work soon).

In GitHub, logged in as your repository owner, do this:

* Go to **Settings**, then **Developer Settings**
* Click **New GitHub App**
* For name, specify "<your-initials> Jenkins"
* For Webhook URL, use http://<your-jenkins-server-domain-name>:8080/github-webhook/
* For permissions, pick:
  + Commit statuses - read and write
  + Contents: read and write
  + Metadata: read only
  + Pull requests: read-only
* For subscribe to events, pick all events
* Click Create GitHub app
* Click generate a private key - private key is generated and downloaded
* Convert the private key to a format suitable for Jenkins

openssl pkcs8 -topk8 -inform PEM -outform PEM -in key-in-your-downloads-folder.pem -out converted-github-app.pem -nocrypt

* In GitHub, install the App you created into your repository account
  + Limit the repository access to just the repositories you want access to (or all)
* In Jenkins, under username, click Credentials
* Click on the global credential store
* Click Add credentials
* For Kind, pick GitHub App
* For ID, use <your-initials>-jenkins
* For App ID, use the app ID from the info page in GitHub
* For the secret key, paste the contents of the converted PEM file that you downloaded and then converted
* Click Test Connection - should work!

NB: at one point, this all went wrong because the GitHub App didn't have any permissions for Content, and after changing that I had to go through an approvals process before it took effect.

# Preparing the virtual machine

The EC2 virtual machine will have a Jenkins account, which is the account that the Jenkins service runs as. This accounts must be a member of the sudo group (so it can run commands with elevated privileges) and also must be able to do this without being prompted for a password (even though the account itself still has a password) - this is what the notes mean in the lab itself.

To make the jenkins user a member of the sudo group:

sudo usermod +aG sudo jenkins

To allow the jenkins user to perform sudo without being prompted for a password:

sudo visudo

Append the following line to the bottom of the file:

jenkins ALL=(ALL) NOPASSWD: ALL

sudo -k

# The database project build steps

This process failed because the setup.sh file couldn't be executed, because it didn't have execute permissions. I think this is a consequence of it having being downloaded from version control - I presume such flags don't travel (it would be different for Windows and Linux anyway).

So, I needed to add this extra step:

export MYSQL\_USER

export MYSQL\_PASSWORD

export MYSQL\_ROOT\_PASSWORD

export MYSQL\_HOST="localhost"

export MYSQL\_DATABASE="bookshelve"

chmod +x setup.sh

./setup.sh

# Docker container issues

I originally ran into a problem where the mysql docker container wasn't set up right (I'd been trying to get round the execute permissions using sh, before realising that meant that setup.sh would be executed using a different kind of shell, and [[ ]] conditions would fail.

So, I ended up in a state where the container was created, but the SQL user accounts hadn't been set up. After that, even if the container was started again, you couldn't connect to it because the user accounts weren't there. To get the script to fix this, I had to delete the non-running container.

sudo docker container list --all

sudo docker container rm <mysql-container-id>

After that, when the script ran in the context of the Jenkins build, it was all OK again because now the container got created again (with the right password details passed in via the Jenkins build environment).

Another part of the problem was I'd tried to re-run the script but outside the context of the jenkins build - this had created the container but hadn't been able to pass in the environment variables from the build configuration - so after that, it would never work.

# Building the backend

Similar to the database. Don't need to export MYSQL\_ROOT\_PASSWORD this time.

First time I tried this, the VM died part way through, after [INSTALL]. Had to stop and restart server.

To test the backend from within the VM, do this via PUTTY:

curl localhost:8000/api/books

I got 404 errors for other URLs (this shows that the flask server is working, though!)

For the API url, I initially got error 500 - I think because after the reboot, the mysql container wasn't running. To restart the mysql container (and check first):

sudo docker container list -all

sudo docker container start mysql

After that, the curl command worked.

Note that you can't check the API from your actual PC, because it runs on port 8000 which our AWS security group doesn't expose (nor do we want it to!)

# Building the frontend

Similar to the backend. Don;t need any environment variables for this one (it communicates with an API via port 8000 within localhost - not exposed.

The way the API call works is that the webpage uses axios to call the web service, but it does it via a URL within the website itself (/api/...). This URL fragment is mapped by nginx to be proxied to the URL localhost:8000/<previous-url> URL, so the actual API call is repeated by the nginx server to the backend server, but the web browser thinks its just making an API call to the same domain as the web page was on!

# Choice of AWS sever capacity

Originally I chose a free-tier server (t2.micro) for this work. But, I think it's too underpowered for this kind of workload (running Jenkins + docker + 3 x application parts). So, I changed the instance type to be t2.medium (2 core CPU, 4GB RAM) and it seems much smoother now!

# Backing up projects and pipelines from Jenkins

Initially the only copy of all your Jenkins data will be on the AWS server - if you terminate it, or need to rebuild it, you will have lost all your work.

sudo mkdir /var/backups/jenkins

sudo chown jenkins:jenkins /var/backups/jenkins

Install the "backup" plug-in.

Configure with backup directory = /var/backups/jenkins (zip format)

Select everything to be backed up

After running the backup, zip file will be in /var/backups/jenkins

To get the file off, you need to be able to use the utility scp. Fo rthat you need a PEM format key pair file to authenticate with. So, make a new key pair via AWS. Extract the public part of the PEM key via this command in Git Bash:

ssh-keygen -y -f mykey.pem > mykey.public

Connect to the remote AWS instance using PUTTY, then edit the .ssh/authorized\_keys file:

sudo vi .ssh/authorized\_keys

Append to this file the contents of the mykey.public file (it should start ssh-rsa).

Perform the file copy using scp within Git Bash (replace the EC2 domain and key file name as appropriate):

scp -i mykey.pem ubuntu@ec2-52-56-163-198.eu-west-2.compute.amazonaws.com:/var/backups/jenkins/\*.zip .

Incidentally, by swapping the source and destination around, you can also use this approach to transfer files *to* the remote server, if you need to.

# Restarting Jenkins

If necessary, restart Jenkins via PUTTY

sudo service jenkins restart

-or- sudo /etc/init.d/jenkins start|stop|restart|status

Problem seems to be I can't access Jenkins from outside, via port 8080. Also, can't access the book app on port 80.

But, from within PUTTY via CURL I can see the site!

Actually, the problem could be that the public domain of the EC2 instance keeps changing?!

Maybe I should install a load balancer / static address?