Last week we looked at manually deploying our application, using our AWS EC2 instance this week we will be looking at integrating our AWS EC2 instance with CircleCI so when our the main branch on our GitHub repository is updated our application will automatically be deployed.

When complete upload a screenshot of your completed deployment job in CircleCI.

Graphical user interface, application

Description automatically generated

Navigate to <https://app.circleci.com/> and select your built application then click project settings.

Graphical user interface, text, application, email

Description automatically generated

Now click on the environment variables tab.

Graphical user interface, text, application, email

Description automatically generated

Now select the Add environment variable button. An environment variable is a variable that can be accessed by a process that is running and can be specified for a particular environment for example in this lab we are going to be using an environment variables to log into our AWS instance from Circle CI.

Graphical user interface, text, application

Description automatically generated

Specify the name of the variable to be EC2\_USERNAME and the value (the value used by the process running) to the username of our AWS EC2 instance by default for ubuntu machines this will be ubuntu.

Graphical user interface, text, application

Description automatically generated



Now add another environment variable this will be the DNS of our EC2 instance to allow use to log into the server, this can be found in your instance summary for your AWS EC2 instance, the page you would have been initially shown upon launching the instance, this is where you can find details about your launched instance.

Graphical user interface, application

Description automatically generated

In AWS click on your account name in the top right hand corner and select "Security credentials".

Graphical user interface, text, application, Teams

Description automatically generated

Now select the Access keys dropdown menu.

Graphical user interface, text, application, email

Description automatically generated

Click the Create New Access Key button.

Graphical user interface, text, application, email

Description automatically generated

You will be then shown a message with your Access Key ID and your Secret Access Key these are very important save these in values a text file or somewhere safe before clicking the close button as we will need these in the coming steps.

Graphical user interface, application

Description automatically generated

Now back in CircleCI again create another environment variable called AWS\_ACCESS\_KEY\_ID and specify the value for this access key to be the value AWS gave you when you generated your access key.

Graphical user interface, text, application

Description automatically generated

Now create another environment variable called AWS\_SECRET\_ACCESS\_KEY and specify the value for this access key to be the value AWS gave you when you generated your access key.

Graphical user interface, text, application, email

Description automatically generated

Now click on your SSH Keys in the side bar and click the Add SSH key button.

Graphical user interface, text, application

Description automatically generated

Open your .pem file in Atom or Visual Studio Code and copy all of the entire contents of it into the private key field and specify the hostname to be the public IPV4 value of your EC2 instance (this can be found in the details page in your launched instance).

Graphical user interface, text, application

Description automatically generated

Your newly created SSH key should look like the above

Text

Description automatically generated

Update your .circleci/config.yml file to now contain a job for deploying our application via CircleCI, note on line 17 and 18 we have requires: -build this means if our build job does not pass our deploy job will not be run.

Line 45 means that we will SSH into our AWS instance from circle CI using our EC2\_USERNAME and EC2\_PUBLIC\_DNS environment variables (the $EC2\_USERNAME is how we use these in bash. The "rm -rf …" is what will be executed when we ssh into our EC2 instance so we remove our old ExampleApplication folder which had our application code we then clone down the most up to date from github.com and then we run our deploy.sh script in this repository that we will make in the next step.

**Note: the ExampleApplication/ in line 45 should be replaced with your repository name and https://github.com/joshuacassidy/ExampleApplication.git should be replace with your application's GitHub repository.**

A screenshot of a computer

Description automatically generated with medium confidence

Now let's make the deployment job from the previous step first in the root directory of your application called deploy.sh and copy the above contents into deploy.sh this will be the script responsible for relaunching our application.

Text

Description automatically generated

Now open up your terminal or GitBash using the cd command to navigate to the directory of your application and run the following commands:

* git status
* git add .
* git commit -m"added continuous delivery"
* git push

These commands will push the changes to the main branch of your git repository causing the example application to be redeployed.

Graphical user interface, application

Description automatically generated

When you navigate to the pipeline now for the Example Application you should now see the build and deploy jobs pass with your application automatically being redeployed to AWS

Graphical user interface, text, application

Description automatically generated

If you click into the pipleline you will be able to see details about the jobs with the build job running first and the deploy job running second with the deploy job being dependant on the build job.

Graphical user interface, text

Description automatically generated

We could also inspect the console output of the deploy job to see the output when the job was running this is excellent for debugging purposes.

Graphical user interface

Description automatically generated with medium confidence

If you navigate to the URL of your application you should see your application being deployed so now let's update the html of our application so we can see our continuous delivery pipeline in action.



So let's make a new branch to update some of our HTML so run the commands:

* git branch add\_home\_link
* git checkout add\_home\_link

This will make a new branch for our changes to our HTML to be made.

Text

Description automatically generated

Now let's add a link to the home page by adding the line of code on line 9.

Text

Description automatically generated

Now that we have made our updates run the following commands:

* git status
* git add .
* git commit -m"added home link"
* git push origin add\_home\_link

This will push to our add\_home\_link branch which means our application will not be deployed.

Graphical user interface, text, application, email

Description automatically generated

Now navigate to your github repository for the application and click pull requests.

Graphical user interface, text, application, email, Teams

Description automatically generated

Click create pull request

Graphical user interface, text, application

Description automatically generated

Now specify that we want to merge our add\_home\_link branch into our main branch and click create pull request.

Graphical user interface, text, application, email

Description automatically generated

Click create pull request

Graphical user interface, text, application

Description automatically generated

Now click merge pull request this will merge our branch into main thus deploying our application to CircleCI, this merge pull request in a software development environment will typically occur when the changes have passed a code review by another developer.

Graphical user interface, application

Description automatically generated

We can now see our newly added link in the deployed version of our application.

In your terminal / GitBash you can switch back to your main branch by running:

* git checkout main

You can update your local version of your main branch to be the same as the GitHub repository by running the command (note we merged in the cloud not on our local machine):

* git pull