National College of Ireland

BSc in Computing

2014/2015

David O’Brien

X12112275

X12112275@student.ncirl.ie

Jenkins Continuous Integration with Docker based agents

Project Report

Table of Contents

Executive Summary

The following document will outline the advantages of using container software such as Docker in a continuous integration environment. Using container software can increase the efficiency of existing cloud based virtual machines and also help reduce operational costs as this type of configuration would require less machines.

The solution to this is to run multiple containers on a dedicated machine. Each of these containers will exist completely isolated from the rest of the containers. I will show how in continuous integration this is a great tool for running multiple builds in isolated containers, using different runtimes and completely isolated from each other.

Introduction

Background

The purpose of this project is to demonstrate the usefulness of container software such as Docker on cloud based machines and in particular in a continuous integration environment. When using virtual machines on a cloud provider such as Amazon there can be a large degree of wasted compute power.

Using continuous integration as the example for this project, this can be caused by a number of different reasons which are most notably the need for build isolation and varied build environments. Having a build that requires isolation essentially dedicates an entire virtual machine to that build but may only need to be ran once or twice a day. This leads to a situation where that machine is now a waste of available resources and costing money at the same time.

Aims

The primary aim of this project is to demonstrate how to use a virtual machine to its maximum potential and without wasting the processing power. The other primary benefit is to show that container software reduces the need for more virtual machines and therefore can help reduce cost and redundant servers.

Using Docker it is possible create a container on the virtual machine that will namespace areas such as the process tree and file system. This means many multiple containers can co-exist on one machine but exist in isolation as the name spaced environments do not interact with each other.

Another solution this allows is the ability to customize the run-times and software that exists inside each container. Allowing this customization means we can keep the environment inside each container exactly suited to particular build.

Technologies

Docker:
Docker is container software that is used to add namespaces to certain areas of a Linux operating system such as the file system and process tree. When you run a Docker image you are running an encapsulated computer environment that is isolated from other Docker containers.

The only thing Docker containers share is that they all use the host machines Linux kernel and because of this the underlying operating system must always be Linux.

Jenkins:

Jenkins is a continuous integration platform that has been chosen due to it being open source and yet still suitable for enterprise solutions. This project will be using Jenkins to launch custom Docker containers that will contain a Jenkins slave-agent and any runtimes needed to complete a build. Different Jenkins builds launch different Docker containers depending on what is required for each build.

Gradle / Groovy:
Gradle is one of the languages that a project will be using. Using Gradle it is possible to set build to completely control the build process and using Groovy it will be possible to run tests that can generate output to the Jenkins builds once completed.

Amazon Web Services:
The project will be built on two Red Hat Enterprise Linux machines hosted on Amazons Elastic Cloud Compute. This provides us with scalable, on demand and pay per use virtual machines. One machine will contain the Jenkins software and operate as the main landing page for the project. The second machine simply contains an installation of Docker. All other software required for builds to run will be contained within the Docker containers themselves.

GIT:
GIT is version control software which will be used by Jenkins to check out code that will be compiled and tested inside the Docker containers. Using GIT allows builds to be run on code change or at periodic times to ensure that the latest version of the code is valid and compiles.

System

Requirements

Functional requirements

Data requirements

For this project the main Data requirements will include the code base that will be stored on GIT and checked out where appropriate to specific Jenkins builds. The code base will include separate repositories for different builds such as a Gradle repository that will contain Gradle build information and tests.

Tests that are ran in these builds will have the results saved to an xml file which can then be parsed post build and be displayed in Jenkins post build. This will catalogue the results of each build and provides a clear indication of any problems that might exist in the repository code.

Docker images will be stored to Docker Hub should they need to be redeployed to any new or existing servers. This allows the creation of a new machine to handle increased load quickly as the only pre-requisite it will need is to have Docker installed.

User requirements

Users looking to use the system will have to register with the Jenkins server and approved by the admin. Once the user is verified permissions can be given to either be allowed run builds or to edit and create new project plans.

With this system a developer would be for example be able to manually run a build after committing new code to GIT if the particular build did not have repository polling enabled. The developer would not however have the rights to alter or delete anything that could have a knock on effect in other areas.

Environmental requirements

Given how the system is being set up each build will be provisioned its own Docker container containing a Linux environment designed to suit that build exactly. This keeps containers small and rapidly deployable and eliminates any possible software conflicts that could interfere with the build.

This means each build should have a Docker container that suits its needs. Aside from some more specialised containers a generic Java 8 with GIT and Maven container for example could be used for builds that do not require any special software to run.

This may seem excessive at first but considering the benefits of being able to deploy multiple environments onto one machine it is worth it. The process of creating a new Docker image is relatively straight forward once you know the process so a new image can be created and deployed to Docker Hub or your own private repo in a short space of time, very similar to setting up a new Linux machine.

An example of a Dockerfile that was used to create the Gradle environment with a Jenkins-slave:

```
FROM ubuntu:trusty
MAINTAINER David OBrien <davidobrien212@gmail.com>

# Make sure the package repository is up to date.
RUN apt-get update && \
        apt-get -y upgrade && \
        apt-get install -y curl && \
        apt-get install -y git && \
        apt-get install -y openssh-server && \
        sed -i 's|session    required    pam_loginuid.so|session    optional
        pam_loginuid.so|g' /etc/pam.d/sshd && \
        mkdir -p /var/run/sshd && \
        apt-get install -y openjdk-7-jdk && \
        adduser --quiet jenkins && \
        echo "jenkins:jenkins" | chpasswd && \
        adduser jenkins sudo && \
        mkdir -p /opt && \
        curl -o /opt/gradle-2.9-bin.zip https://downloads.gradle.org/distributions/gradle-
2.9-bin.zip && \
        apt-get install -y unzip && \
        unzip /opt/gradle-2.9-bin.zip -d /opt && mv /opt/gradle-2.9 /opt/gradle && \
        rm -f /opt/gradle-2.9-bin.zip && \
        mkdir -p /var/lib/jenkins/.ssh

ADD .ssh/id_rsa /var/lib/jenkins/.ssh/
ADD .ssh/id_rsa.pub /var/lib/jenkins/.ssh/
ADD .ssh/known_hosts /var/lib/jenkins/.ssh/

RUN echo "jenkins" | sudo -S chown -R jenkins:jenkins /opt/gradle && \
  echo "jenkins" | sudo -S chown -R jenkins:jenkins /home/jenkins

# Standard SSH port
EXPOSE 22

CMD ["/usr/sbin/sshd", "-D"]
```

In the example above we could easily modify the RUN command to include new or additional runtimes that are needed.

Usability requirements

Once all the processes are complete the system will work simply by queuing a build in Jenkins. All of the implementation for the project involves automating the build plans with particular focus on using Docker based environments.

Once the processes are complete a user can click a build to run, if a build executable for that particular build/environment is not already available Jenkins will launch a build remotely on a separate server dedicated to running Docker environments. All other aspects such as checking out code will be preconfigured to point at the desired repository.

Once implemented correctly simply started the build will be all the interaction that is required for the average user. For an admin or the user that manages the environments knowledge of creating and deploying the new environments is a must and the nature of this project assumes that anyone with those elevated rights will have the appropriate knowledge to complete these tasks.

It is important to remember this is not a project that is intended for someone with no technical computing skill to use and is intended as a solution for continuous integration to be used by skilled workers in IT.

Implementation

Describe the main classes/functions used in the code. Consider to show and explain interesting code snippets where appropriate.

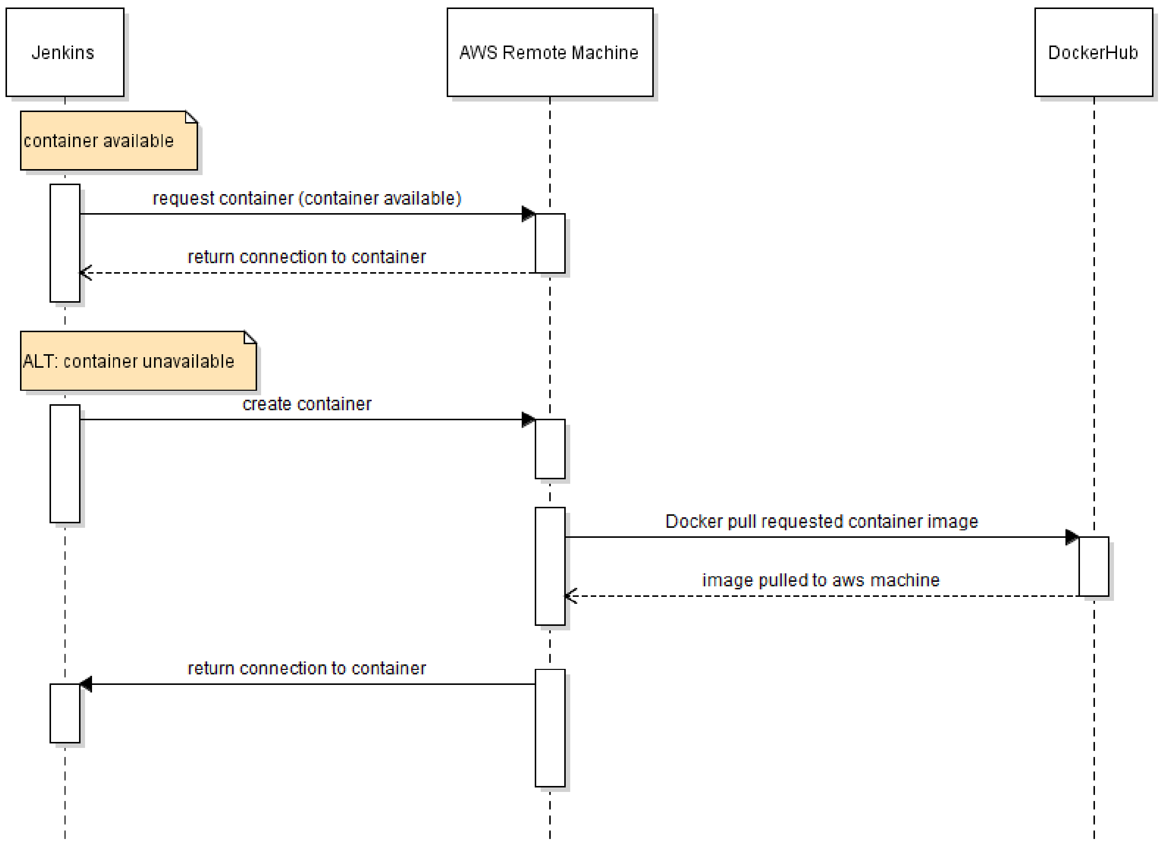
The project consists of the architectural back end providing the continuous integration software Jenkins that has been installed on an Amazon server. The Jenkins platform runs build plans that get run inside Docker containers on another dedicated server. These containers are created using a Dockerfile, an example of which can be seen above, and contain everything required for a given build.

Builds are tagged with labels specifying which container should be used for that given build. If a container is available then it will be used however if not Jenkins will start another container remotely on the build machine.

Due to the nature of the project so far the majority of the work has been carried out using the Linux terminal or the Amazon / Jenkins control panels. Getting the network communication in place and in particular getting the Jenkins build agents to be remotely created in Docker containers was the biggest implementation to date. Now that the process is complete it can be replicated for a more diverse array of containers.

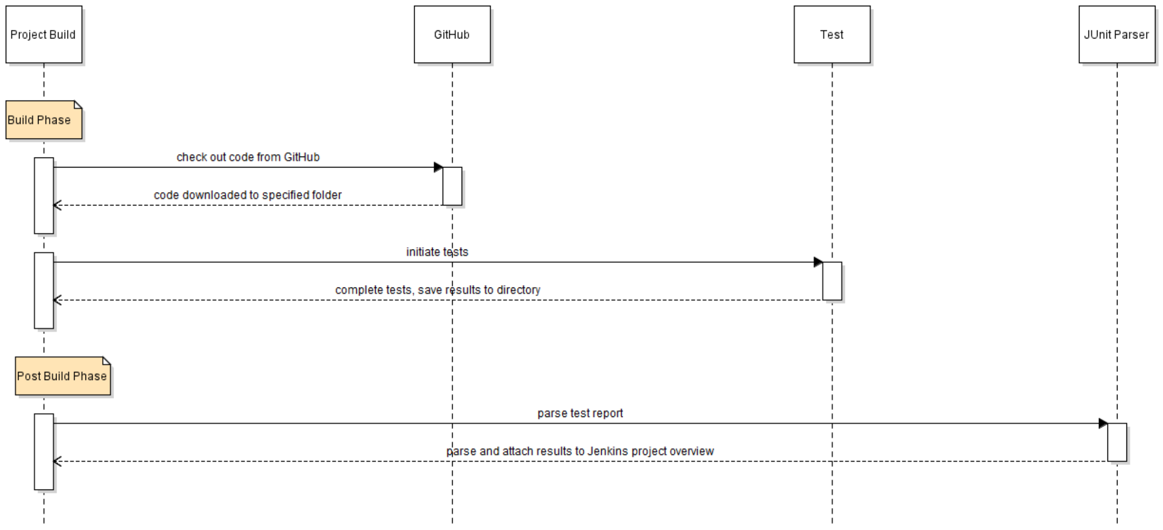
The following is a sequence diagram representing the process of obtaining a container for Jenkins builds. Jenkins has been preconfigured to connect to a specific AWS remote machine that handles all of the explicit building of projects, the machine containing Jenkins does not run any builds to allow maximum performance with multiple users.

When a build starts it looks for a specific active container with a custom label denoted to that type of build, to allow for many different containers but ensures only the appropriate is used. If no such container is currently active Jenkins sends a request to the dedicated AWS machine asking for a container suited to the build. If the image used to create the container is not currently on the AWS machine, this can depend on different policy settings configured in Jenkins, the image is then pulled from the configured Docker Hub account. Once downloaded to the build machine a container is launched using the image and connection info is returned to Jenkins. From here the container is ready to accept builds. Multiple containers can be launched from the Dockerfile image.


Jenkins
AWS Remote Machine
DockerHub
container available
request container (container available)
return connection to container
ALT: container unavailable
create container
Docker pull requested container image
image pulled to aws machine
return connection to container

The following is the process the runs within each container on each build. Firstly now that a build container is available the code is checked out into the working directory of the container. This can be done using plugins but to ensure all information is contained within the container a git clone bash command is used as a build step that is ran from within the container. Once the code to test is downloaded the ./gradlew test command is called, this example uses gradle but use whatever is appropriate to the build. The gradlew command is a wrapper for the gradle command however it is runnable on machines, or containers, that have no preinstalled software. This command checks for the existence of gradle in the host environment and if not present downloads and configures the software for use. This step allows for easy distribution of the code into multiple environments.

Once gradle is configured the contained tests are ran and a report is generated in the build sub directory of the current folder. As a post build step these results can then be parsed using a JUnit parser which provides an easy to read display of exactly what tests have passed, if any have failed or which were skipped, useful for quarantining non critical or outdated tests. Another advantage of the parser is it provides an easy to read graphical history of tests allowing for fast diagnoses of potential problems. This output allows developers to quickly implement fixes for the software that can be committed to Git and upon successful commit re-run almost immediately by building out the project again as the fresh up to date code is always cloned from Git.


Project Build
GitHub
Test
JUnit Parser
Build Phase
check out code from GitHub
code downloaded to specified folder
initiate tests
complete tests, save results to directory
Post Build Phase
parse test report
parse and attach results to Jenkins project overview

When combined all of the above provides the foundation of the closed loop development that is key to a test driven environment. Having the ability to quickly identify problems, submit fixes and retest increases productivity. Developers themselves never need to have direct control of the build environment in order to analyse the problem and provide a solution.

Testing

Testing the software within the containers is handled using JUnit. Tests should be written using tools related to the language being used to develop the software in the container. In each Jenkins project the directory where the test outputs are located after testing should be added to the project configuration. These results can then be parsed in a post build step and be included in the Jenkins project overview screen.

In the example being used simple tests are written using Groovy and use JUnit. As a build step in the project the command:

```
./gradlew test
```

Is called which runs all tests specified in the gradle test package. Once complete the results are output to the build directory in the docker container under the specified build folder:

```
“${JOB_NAME}_${BUILD_NUMBER}”/build/test-results/TEST-classname.xml
```

These results are then parsed using a Jenkins JUnit plugin that shows test results for each build as well as a history of passing / failing tests.

Since results are contained within the Docker container for persistence it is advisable to mount a network drive to the containers on launch and thus allowing the results to be copied to more persistent storage to maintain the testing history for each project. See GUI section for test results interface.

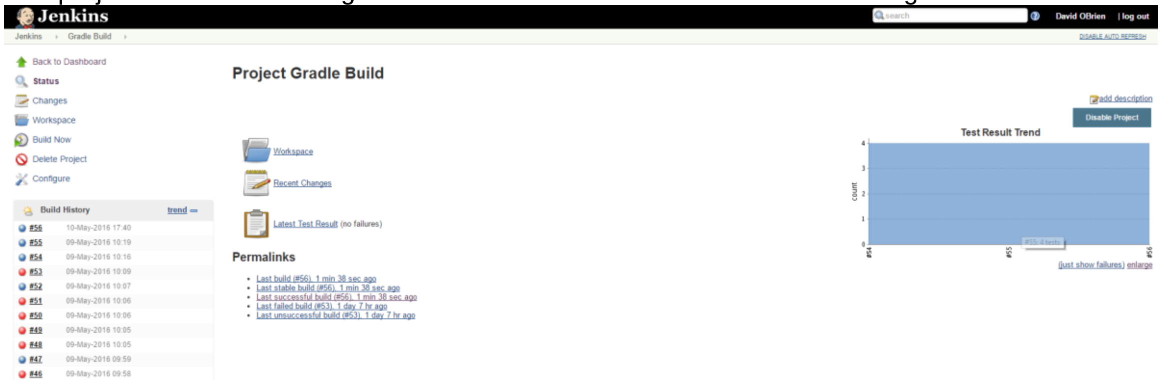
Graphical User Interface (GUI) Layout

There will be no interface created by me for this project. The only GUI to be used will be provided by the Jenkins server and the AWS console for managing your virtual machines. Aside from that all work will be carried out using the terminal where creation of the Dockerfile’s will be completed. SSH access to the remote Linux machines will also be all terminal based.

Setup and communication between virtual machines and Docker containers is all handled from the Jenkins control panel. Now that the communications process and configurations on the remote machine for Docker Hub and Git are in place there is no need for trivial user control on the build machine. Any further Docker images pushed to Docker Hub or code contained on Git can be specified using the Jenkins control panel. Jenkins also provides the display of test results through the use of a JUnit parser which provides a graphical breakdown of overall test history per project and displays the results of each build.

Note: for a clearer view of the images zoom the page.

The project overview showing the recent test results as a blue chart on the right of screen:



Project Gradle Build

Build History	trend
#56	10-May-2016 17:40
#55	09-May-2016 10:19
#54	09-May-2016 10:16
#53	09-May-2016 10:09
#52	09-May-2016 10:07
#51	09-May-2016 10:06
#50	09-May-2016 10:06
#49	09-May-2016 10:05
#48	09-May-2016 10:05
#47	09-May-2016 09:59
#46	09-May-2016 09:58

Workspace

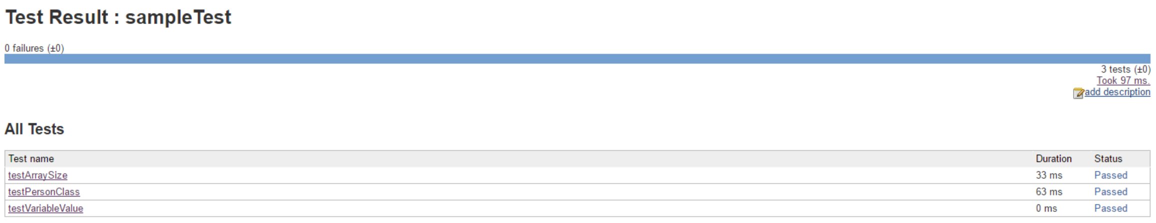
Recent Changes

Latest Test Result (no failures)

Permalinks

- Last build (#56), 1 min 38 sec ago
- Last stable build (#56), 1 min 38 sec ago
- Last successful build (#56), 1 min 38 sec ago
- Last failed build (#53), 1 day 7 hr ago
- Last unsuccessful build (#53), 1 day 7 hr ago

Test results from an individual build expanded:



Test Result : sampleTest

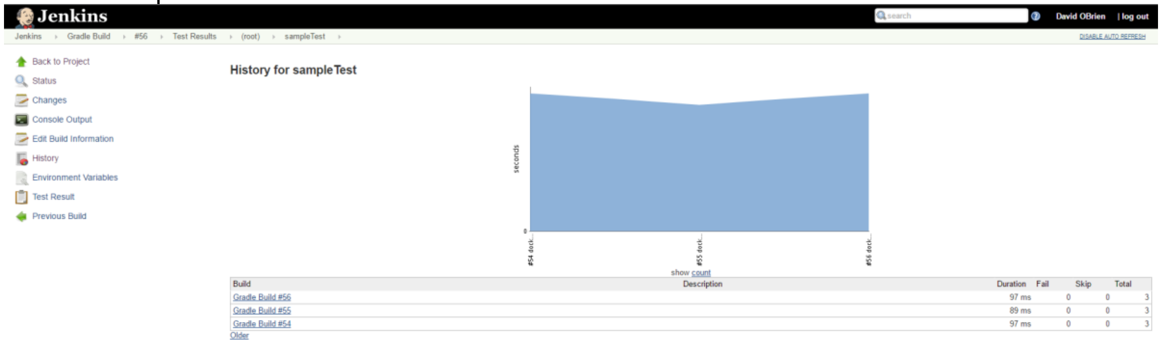
0 failures (±0)

3 tests (±0)
Took 97 ms.
add description

All Tests

Test name	Duration	Status
testArraySize	33 ms	Passed
testPersonClass	63 ms	Passed
testVariableValue	0 ms	Passed

A more in depth view which shows the individual tests as well as test duration:

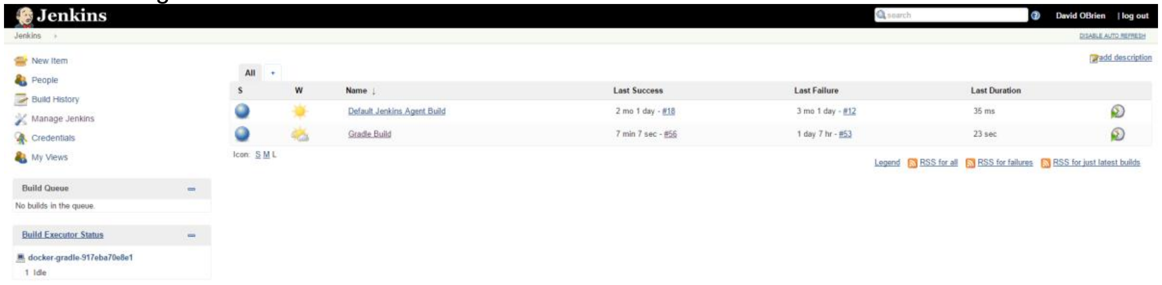


History for sampleTest

Build	Description	Duration	Fail	Skip	Total
Gradle Build #56		97 ms	0	0	3
Gradle Build #55		89 ms	0	0	3
Gradle Build #54		97 ms	0	0	3

Older

The Jenkins landing page which shows available projects to build, an active container in lower left corner waiting for builds:



S	W	Name ↓	Last Success	Last Failure	Last Duration	
		Default Jenkins Agent Build	2 mo 1 day - #18	3 mo 1 day - #12	35 ms	
		Gradle Build	7 min 7 sec - #56	1 day 7 hr - #53	23 sec	

Build Queue

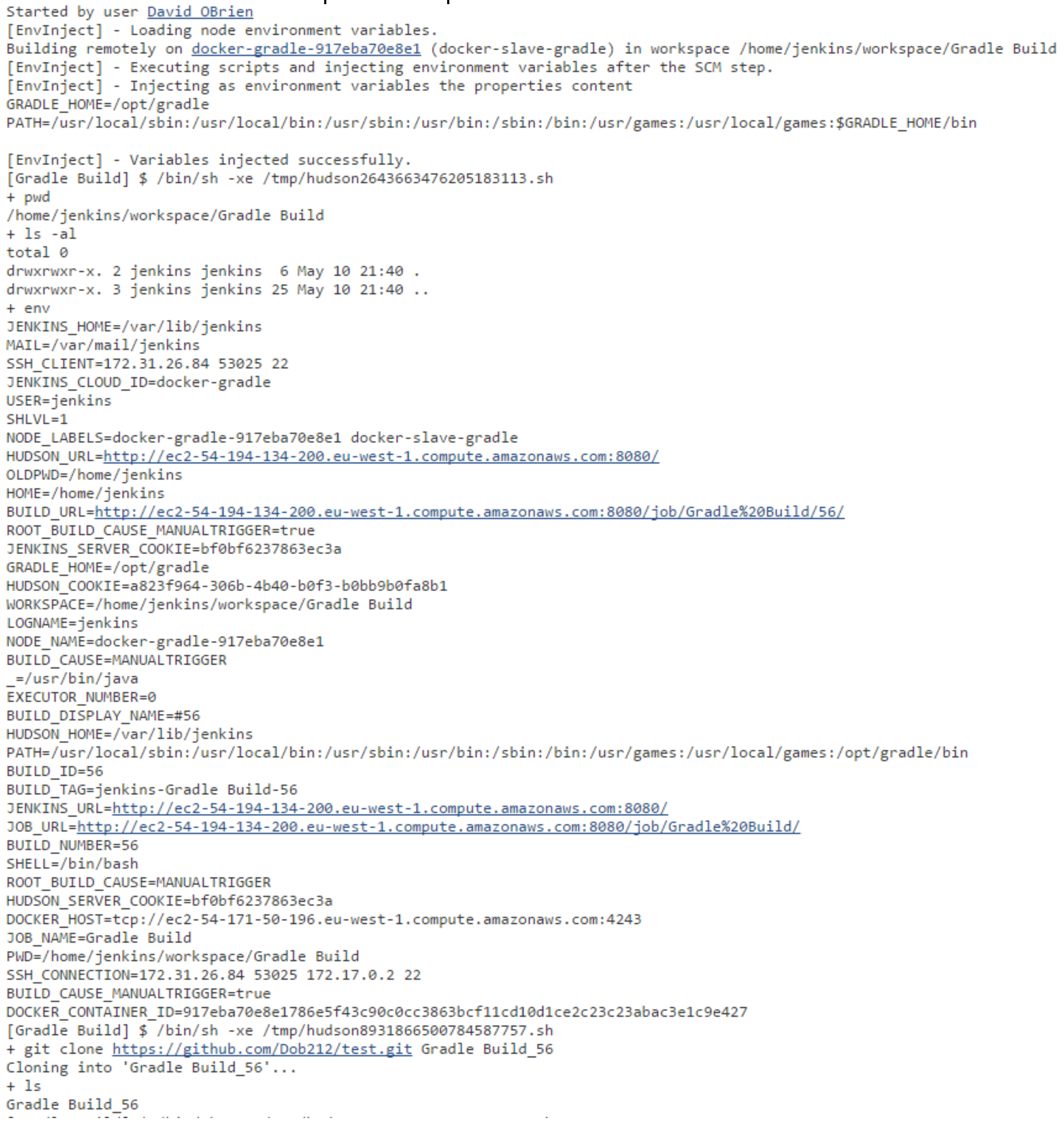
No builds in the queue.

Build Executor Status

docker-gradle-917eba70e8e1

1 Idle

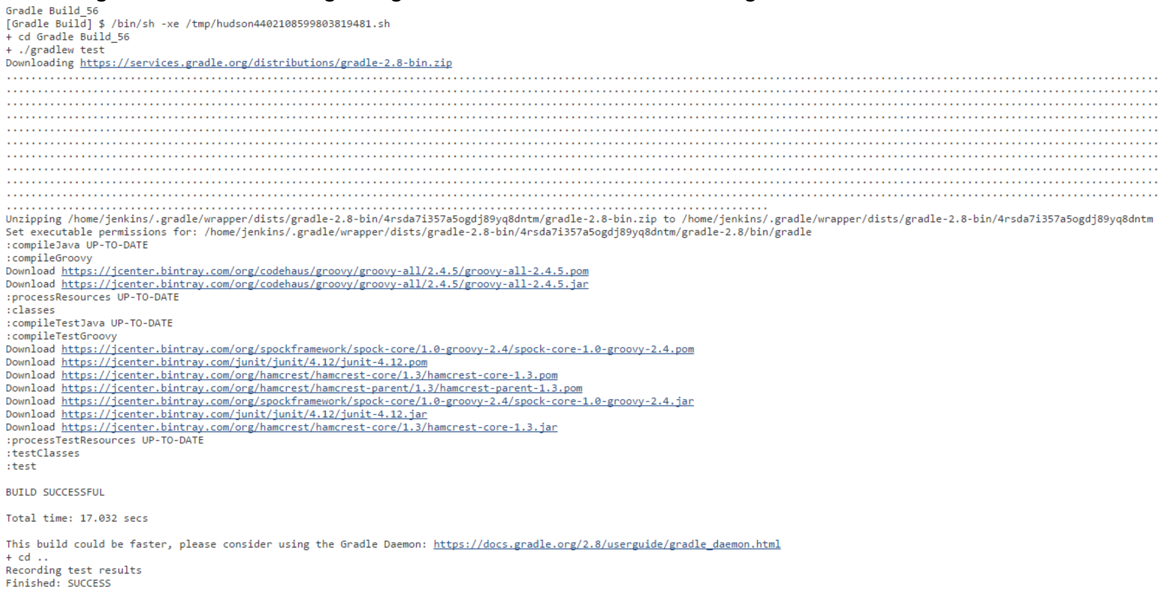
The following is an example of console output from the Jenkins build, it shows the name of the container that the job is currently running on at the top, lists environment variables and at the bottom shows the Git clone operation to pull code into the container:



```
Started by user David OBrien
[EnvInject] - Loading node environment variables.
Building remotely on docker-gradle-917eba70e8e1 (docker-slave-gradle) in workspace /home/jenkins/workspace/Gradle Build
[EnvInject] - Executing scripts and injecting environment variables after the SCM step.
[EnvInject] - Injecting as environment variables the properties content
GRADLE_HOME=/opt/gradle
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:$GRADLE_HOME/bin

[EnvInject] - Variables injected successfully.
[Gradle Build] $ /bin/sh -xe /tmp/hudson2643663476205183113.sh
+ pwd
/home/jenkins/workspace/Gradle Build
+ ls -al
total 0
drwxrwxr-x. 2 jenkins jenkins  6 May 10 21:40 .
drwxrwxr-x. 3 jenkins jenkins 25 May 10 21:40 ..
+ env
JENKINS_HOME=/var/lib/jenkins
MAIL=/var/mail/jenkins
SSH_CLIENT=172.31.26.84 53025 22
JENKINS_CLOUD_ID=docker-gradle
USER=jenkins
SHLVL=1
NODE_LABELS=docker-gradle-917eba70e8e1 docker-slave-gradle
HUDSON_URL=http://ec2-54-194-134-200.eu-west-1.compute.amazonaws.com:8080/
OLDPWD=/home/jenkins
HOME=/home/jenkins
BUILD_URL=http://ec2-54-194-134-200.eu-west-1.compute.amazonaws.com:8080/job/Gradle%20Build/56/
ROOT_BUILD_CAUSE_MANUALTRIGGER=true
JENKINS_SERVER_COOKIE=bf0bf6237863ec3a
GRADLE_HOME=/opt/gradle
HUDSON_COOKIE=a823f964-306b-4b40-b0f3-b0bb9b0fa8b1
WORKSPACE=/home/jenkins/workspace/Gradle Build
LOGNAME=jenkins
NODE_NAME=docker-gradle-917eba70e8e1
BUILD_CAUSE=MANUALTRIGGER
_=/usr/bin/java
EXECUTOR_NUMBER=0
BUILD_DISPLAY_NAME=#56
HUDSON_HOME=/var/lib/jenkins
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/opt/gradle/bin
BUILD_ID=56
BUILD_TAG=jenkins-Gradle Build-56
JENKINS_URL=http://ec2-54-194-134-200.eu-west-1.compute.amazonaws.com:8080/
JOB_URL=http://ec2-54-194-134-200.eu-west-1.compute.amazonaws.com:8080/job/Gradle%20Build/
BUILD_NUMBER=56
SHELL=/bin/bash
ROOT_BUILD_CAUSE=MANUALTRIGGER
HUDSON_SERVER_COOKIE=bf0bf6237863ec3a
DOCKER_HOST=tcp://ec2-54-171-50-196.eu-west-1.compute.amazonaws.com:4243
JOB_NAME=Gradle Build
PWD=/home/jenkins/workspace/Gradle Build
SSH_CONNECTION=172.31.26.84 53025 172.17.0.2 22
BUILD_CAUSE_MANUALTRIGGER=true
DOCKER_CONTAINER_ID=917eba70e8e1786e5f43c90c0cc3863bcf11cd10d1ce2c23c23abac3e1c9e427
[Gradle Build] $ /bin/sh -xe /tmp/hudson8931866500784587757.sh
+ git clone https://github.com/Dob212/test.git Gradle Build_56
Cloning into 'Gradle Build_56'...
+ ls
Gradle Build_56
```

Build log continued showing the gradle download and executing the test classes:



```
Gradle Build_56
[Gradle Build] $ /bin/sh -xe /tmp/hudson4402108599803819481.sh
+ cd Gradle Build_56
+ ./gradlew test
Downloading https://services.gradle.org/distributions/gradle-2.8-bin.zip
..........................................................................................................................................................................................
..........................................................................................................................................................................................
..........................................................................................................................................................................................
..........................................................................................................................................................................................
..........................................................................................................................................................................................
..........................................................................................................................................................................................
..........................................................................................................................................................................................
..........................................................................................................................................................................................
..........................................................................................................................................................................................
..........................................................................................................................................................................................
...........................................................................................................................
Unzipping /home/jenkins/.gradle/wrapper/dists/gradle-2.8-bin/4rsda7i357a5ogdj89yq8dntm/gradle-2.8-bin.zip to /home/jenkins/.gradle/wrapper/dists/gradle-2.8-bin/4rsda7i357a5ogdj89yq8dntm
Set executable permissions for: /home/jenkins/.gradle/wrapper/dists/gradle-2.8-bin/4rsda7i357a5ogdj89yq8dntm/gradle-2.8/bin/gradle
:compileJava UP-TO-DATE
:compileGroovy
Download https://jcenter.bintray.com/org/codehaus/groovy/groovy-all/2.4.5/groovy-all-2.4.5.pom
Download https://jcenter.bintray.com/org/codehaus/groovy/groovy-all/2.4.5/groovy-all-2.4.5.jar
:processResources UP-TO-DATE
:classes
:compileTestJava UP-TO-DATE
:compileTestGroovy
Download https://jcenter.bintray.com/org/spockframework/spock-core/1.0-groovy-2.4/spock-core-1.0-groovy-2.4.pom
Download https://jcenter.bintray.com/junit/junit/4.12/junit-4.12.pom
Download https://jcenter.bintray.com/org/hamcrest/hamcrest-core/1.3/hamcrest-core-1.3.pom
Download https://jcenter.bintray.com/org/hamcrest/hamcrest-parent/1.3/hamcrest-parent-1.3.pom
Download https://jcenter.bintray.com/org/spockframework/spock-core/1.0-groovy-2.4/spock-core-1.0-groovy-2.4.jar
Download https://jcenter.bintray.com/junit/junit/4.12/junit-4.12.jar
Download https://jcenter.bintray.com/org/hamcrest/hamcrest-core/1.3/hamcrest-core-1.3.jar
:processTestResources UP-TO-DATE
:testClasses
:test

BUILD SUCCESSFUL

Total time: 17.032 secs

This build could be faster, please consider using the Gradle Daemon: https://docs.gradle.org/2.8/userguide/gradle_daemon.html
+ cd ..
Recording test results
Finished: SUCCESS
```

Evaluation

Testing for the application will mostly take the form of unit tests. Using the Gradle init Groovy project construct it provides a folder layout for Groovy code as well as Tests. This will allow me to initiate these unit tests during the build and test phase of a Jenkins build.

Any test failures here will be reported using the output log in Jenkins that will identify the failed test as well as the suspected cause. Providing many tests for the methods in the Groovy classes ensures that we have accounted for most of the problems that may occur during an applications run.

Creating a Docker image also has its own checks during the Docker build command. Any instructions provided in the Dockerfile will be executed one by one during the Docker build command, if any part of Docker build fails the image compilation is aborted and diagnosed. Once a custom Docker image has been created it is also possible to enter a running container using software such as nsenter. Once a container has launched there is no way to enter the running container using the standard Docker api, nsenter however can attach itself to the namespace of the process id that is running the Docker container. This allows you to change from the Host system to the container file system.

Tools like nsenter are essential for debugging and analysing live containers as it allows an admin to perform minor maintenance tasks. Anything more than minor changes should be implemented using changes to the Dockerfile and a rebuild of the image.

Conclusions

As a process for building out a continuous integration test driven environment I feel I have shown how the underlying architecture can be set up for be scalable for any size of company or organisation. The use of a closed loop development environment along with the ability to add further virtual machines to increase performance and provide new solutions provides straightforward scalability while still being manageable on any level by a single user for small scale implementations or a small team for large scale.

As the process expands simply by using cloud based virtual machines which are maintained by the provider of the service there is a lot less of an overhead in maintaining even large systems. Using container software has also demonstrated how to get the most out of individual virtual machines so that overall less are needed which saves on the total operational cost of maintaining your infrastructure.

Once this particular process has been put into place overall maintenance by the team in control is also greatly simplified again by the use of containers and also a version control system, in this case Git, as all further software whether it is container images or code can be uploaded to the preconfigured repositories and from there all further control can be handled from the Jenkins control panel.

Overall what I set out to do at the beginning of the project has been achieved. I think I was realistic in my goals and was confident that I could deliver on everything that was planned. My initial fear that the scope of the project would not be clearly understood was quickly put to rest as I was lucky enough that both of my supervisors had a good understanding of what it was I was hoping to achieve.

Further development or research

Further to this project I would have liked to create persistent network storage that could be attached to containers as a shared drive. This drive would allow data from the containers to be copied over to the storage in particular information that would be useful when debugging failed builds such as build logs and test reports.

I have done this previously so I know that technically it is not a major undertaking and requires only an added options when launching the container and choosing a mount point on which to attach it. This could also have been done with simple adding a mounted directory from the amazon host machine but financial limitations restricted my ability to do this as with other projects I am already pushing the limits of my Amazon free tier allowance.

Ideally I would have loved an industry level project to build how to really showcase the build process but having something of that size and complexity would have created an extra burden of time that I did not have due again to other project commitments. While the projects that I am building are only showing simple test examples I wish to stress that almost any project can be built out and tested using this process as long as there is a Docker container containing everything needed for the project to run and due to the nature of containers can be run side by side with completely different containers in complete isolation.

References

Appendix

Project Proposal

Project Proposal

Docker build agents in Continuous Integration

Name: David O'Brien
Student Number: x12112275
Email: x12112275@student.ncirl.ie

BSc (Hons) in Computing

Cloud Computing stream.

Objectives:

The aim of this project is to create Jenkins build agents that will operate from Docker containers that are deployed on Amazon Compute virtual machines. Once complete there will be one virtual machine hosted in AWS that will be running Jenkins and have the capability of running multiple side by side Docker containers that can build separate projects but be completely isolated from each other.

Having secured a position as a Build Engineer in Workday once college is completed I know that this project will directly benefit me in a professional capacity. Knowing also that Workday are looking to integrate Docker into their build environments this project offers me the opportunity to explore these technologies and get a head start on my professional career.

On a more personal level I have found this area of computing in general to be a lot more interesting to me than App development, whether they are general apps or games. Working on the systems side is far more interesting to me. This is a direct influence from the internship I completed in Workday. Before joining the build engineering team I would not have thought it was an area that would appeal to me but maintaining builds, implementing work a round's and finding new ways to speed up the build process are all very interesting to me.

My number one aim throughout this project is to develop skills that will benefit me in the coming years in my career and also give me an advantage over other workers in my field. Having experience with software like Jenkins, Docker, AWS and Groovy / Gradle will give me a direct advantage over other potential candidates in the future.

Background:

From the time I spent during my internship in Workday I was working on a Continuous Integration system called Bamboo from Atlassian. Interacting with this everyday it became apparent that there were certain limitations in how it builds out projects. Also in a company like Workday where there are hundreds of build projects and plans spread out of many servers it became clear that certain machines were being severely under used. This was usually due to the importance of the build it contained or issues arising from conflicts cause with multiple builds on the same machine.

The benefits of configuring a system with separate containers is the ability to scale the power of the build machine using AWS and maximize the power of the provided virtual machine. In some cases it may be necessary to dedicate a machine to building one project or one subset of projects. This approach has the potential to waste resources as the machine that is dedicated to these build may only be using for example 40% of its potential which adds up to wasted infrastructure. This has a larger impact on bare metal servers that cannot be scaled the same way virtual machines can.

Another benefit of using containers as build agents is reduced chance of builds conflicting with each other. On many continuous integration solutions it will be necessary to test your builds in different environments. This could include testing environments for upgrade to a new version of Java or Ant etc. Instead of requiring more variety of virtual machines we can instead use our underutilized one to run parallel jobs.

This becomes possible using container software such as Docker. Docker allows the use of a wide range of Linux images for creating environments. Base images can be customized to the user’s needs and launched in an isolated, name spaced environment on the host machine. As the Docker container uses the Host OS’s kernel it has access to any kernel feature that the Host OS does. The benefit of having multiple containers includes using different containers to test different versions of software (as mentioned above with Java and Ant) or the ability to run two completely different builds on the one machine using separate containers.

Technical Approach:

Beginning this project I will be familiarising myself with the Jenkins Continuous Integration tool. Having previously used Bamboo I have a general understanding of how CI tools function but Jenkins will be new to me. Once comfortable with this I will provision an Amazon virtual machine with which to install it.

The Amazon virtual machine will be the primary operational environment for the project. The virtual machine will need to be configured with Jenkins, Git, Docker and SSH functionality. Once setup and configured custom Docker based slave agents will need to be created to provide the ability to build projects.

The slave agents will be created in a custom image using a Dockerfile to customize all aspects of the image. Using a Dockerfile allows complete control of what software and runtimes will be installed inside the container. This also allows the creation and setup of User accounts. Docker containers default to running as the root user and in many cases this is not the desired as certain files may need to be created under with particular user and group ids. While this can be changed after the fact it is much easier to create a User with defined uid and gid (to suit other stages of your build process) and switch to that user before the container is created. This also allows for custom init scripts using the .bash-profile or .bash_rc to fine tune the container once it becomes active.

Once the environment setup is complete dummy applications written in in Groovy / Gradle will be committed to Git. These applications will feature a wide array of tests that will be ran while building in Jenkins. These builds will be compiling on the runtimes contained within a Docker containers. The Jenkins builds will be configured to run each time a commit is made to Git which allows a complete observable trail that will indicate where a build breaks and the cause.

Special resources required:

Additional training will be required for the project most specifically in the area of Jenkins and AWS. In my previous internship in Workday I was exposed in depth to Docker although there are still many aspects such as networking that I will require more research into.

The main component when it comes to hardware is provisioning configuring the virtual machine from Amazon. This will be the main platform for development and will host all software needed to complete the builds. Using SSH to connect to the Amazon instance allows work to be done from any location and any machine.

Project plan:

Attached with Microsoft Project File

Technical Details:

As my project is not a conventional app like project it is not as easy to define a primary language. Instead I will list the languages and software to be used and what functionality it will provide. At the most base level a lot of interaction with the project will be done with shell commands. The Docker api is accessed through the Terminal and while you can use a Gui solution I personally prefer the terminal interaction as you can issue commands as a part of a separate script.

The Docker api itself is used to apply namespaces to the core linux systems allowing in essence to instantiate the Host machine. When a namespace is applied it allows anything within that namespace to work in isolation to the rest of the Host machine, this include process trees and file systems etc. This is essential in isolating different builds from each other and installing different software into different filesystems with separate runtimes.

Jenkins is an open source continuous integration platform that builds and tests software. When connected to version control software this allows developers to continuously test changes on commit and helps provide the most up to date builds possible.

Gradle and Groovy will be used as the primary application languages. Groovy applications using the Gradle build tool will be used to run unit tests and flesh out basic, separate applications that can be run in separate containers. Gradle allows many build options when it comes to structuring the order in which tasks are carried out. This allows tasks to run as doFirst(), doLast() etc. which do exactly as they sound. A tool like this gives you exact control over how your build is completed. One major advantage is if you needed to download a secure token for credential verification, you could specify that you want the token downloaded from a secure server using curl with credentials, parse that token into variables and then most importantly delete the token before any further steps are completed. This ensures that no sensitive data is left lying around the file system and all functions like this can be completed before the main tasks even take place. Dependency resolution from multiple sources whether they are private of open repositories are also handled in the build.gradle file.

Gradle follows the same syntax as Groovy and therefore allows you to write out your build commands with all the extra functionality that Groovy offers. The sample application itself will be written in Groovy which will be invoked from the build.gradle file. These classes will do a variety of basic tasks that can also be tested using unit tests.

Git will be used as the version control software. Previously I have used Subversion as VCS so this finally gives me the opportunity to get to know Git better.

Amazon Web Services will also play a major part in the project and I will be using a hosted virtual machine from Amazon's EC2 platform. This will allow me to have maximum uptime on the service that is provided. This machine will host the Jenkins build server and also at least 2 Docker containers that contain a Jenkins slave agent for building out projects.

Evaluation:

Testing for the application will mostly take the form of unit tests. Using the Gradle init Groovy project construct it provides a folder layout for Groovy code as well as Tests. This will allow me to initiate these unit tests during the build and test phase of a Jenkins build.

Any test failures here will be reported using the output log in jenkins that will identify the failed test as well as the suspected cause. Providing many tests for the methods in the Groovy classes ensures that we have accounted for most of the problems that may occur during an applications run.

Creating a Docker image also has its own checks during the Docker build command. Any instructions provided in the Dockerfile will be executed one by one during the Docker build command, if any part of Docker build fails the image compilation is aborted and diagnosed. Once a custom Docker image has been created it is also possible to enter a running container using software such as nsenter. Once a container has launched there is no way to enter the running container using the standard Docker api, nsenter however can attach itself to the namespace of the process id that is running the Docker container. This allows you to change from the Host system to the container file system.

Tools like nsenter are essential for debugging and analyzing live containers as it allows an admin to perform minor maintenance tasks. Anything more than minor changes should be implemented using changes to the Dockerfile and a rebuild of the image.

Project Plan

Monthly Journals

Journals:

Student name: David OBrien
Programme: BSc in Computing
Month: September
My Achievements
During the month of September my primary aim in regards to the project was to get a clear idea of what route I would be taking it. I knew from my experiences in Workday that I wanted to use

tools such as Docker and a continuous integration platform. Primarily because these are the tools that I will be using once I join Workday after college so this will be a great way to hit the ground running.

The team I am joining in Workday primarily use Bamboo by Atlassian as their CI tool, too avoid messing around with licences etc. and also because it will be beneficial to know I decided to use Jenkins for my project coupled with Docker and AWS. Getting my project proposal finalised helped me focus on what exactly I needed to do. As I am not doing a traditional App style project the route to completing it required a different train of thought than just backend and ui etc.

My Reflection

Aside from figuring out the path to finishing the project I haven't started anything practical yet as other Reports and CA's took up quite a bit of time.

Intended Changes

Start practically implementing work on AWS using Jenkins. I need to understand the build tool before I start making remote Docker agents to run builds on.

Supervisor Meetings

Date of Meeting: N/A

Items discussed: N/A

Action Items: Get a server, customize it to my needs, install Jenkins and install Docker

Student name: David OBrien

Programme: BSc in Computing

Month: October

My Achievements

This month my main goal was to familiarize myself with AWS and Jenkins. After reading up a lot of the documentation I was able to launch two base Linux linux machines in AWS. During the setup process I had to configure my SSH key for secure access to the instance's through my laptop terminal and configure ports to allow an incoming SSH connection. As I had never used AWS in this context before it took me a little while to get everything working correctly (AWS subnets etc.)

Once SSH access was complete I was able to remotely log into the server using the Amazon CLI on my laptop. From here I was able to configure 1 machine to install JDK 8 and Jenkins. Using an AWS elastic IP I can connect to Jenkins once the AWS machine is running. This allowed me to do some basic test builds using basic Bash commands which are working.

On the second machine I have installed Docker and this is the machine that will be used to launch the remote Jenkins agents on.

My Reflection

I would have liked to get more work done this month but there have been a lot of time consuming reports and labs. I especially would have liked to get a remote agent installed on the Docker machine even if it was not in a Docker container for the moment.

Intended Changes

This month I will be working on getting remote agents running on the Docker machine as opposed to just running local agents on the local Jenkins machine. If all goes well in that regard I will then begin installing the agents into the Docker containers and test some builds on them.

Supervisor Meetings

Date of Meeting:n/a

Items discussed:n/a

Action Items:n/a

Student name: David OBrien

Programme: BSc in Computing

Month: November

My Achievements

During the course of November I worked to familiarise myself with Jenkins and how to set up remote connections to Docker containers. The first steps taken were to do a docker pull evarga/Jenkins-slave command. This downloaded a basic Jenkins slave Docker image which is preconfigured to with the agent, has exposed port 22 for incoming connections and has the SSH Daemon running by default.

This image will be used for initial connectivity testing and once successful the underlying Dockerfile used to create the image will be customised to include relevant runtimes for particular builds such as Gradle, Ruby or Python etc. The basic container includes OpenJDK 7 by default so is limited in its capabilities.

Once the Docker machine was configured it was necessary to include several plugins to Jenkins for streamlining the launching of remote Docker containers. This would allow me to point the Jenkins server at the Docker install on a separate Amazon machine, launch remote containers and maintain a connection.

Being relatively new to AWS and in particular the networking aspects of it this proved more trouble than anticipated as everything I was doing was new to me. After much trial and error I managed to get an agent launched remotely without using Docker to check network configs. Once this worked I knew it was something I was missing in the Docker configurations.

Unrelated to the above I removed the Elastic IP as it was giving me no benefit but was costing 8 euro for the month.

My Reflection

I would have liked to get more work done this month but there have been a lot of time consuming reports and labs. I especially would have liked to get a remote agent installed on the Docker machine even if it was not in a Docker container for the moment.

Intended Changes

In December I intend to continue working on getting the basic Jenkins Slave containers connected to the Jenkins host server machine.

Supervisor Meetings

Date of Meeting:n/a

Items discussed:n/a

Action Items:n/a

Student name: David OBrien

Programme: BSc in Computing

Month: December

My Achievements

Unfortunately I didn't get to focus on my project in December. A close personal friend was deteriorating rapidly from an aggressive form of cancer and college commitments were put in the background for a while. Anytime I did have was spent making other deadlines.

My Reflection

Intended Changes

Catching up on time will be the focus of January.

Supervisor Meetings

Date of Meeting:n/a

Items discussed:n/a

Action Items:n/a

Student name: David OBrien

Programme: BSc in Computing

Month: January

My Achievements

In January I fixed my issue of connected to a remote Jenkins container. I suspect it might have even been working for a while but I forgot that given the nature of how containers are launched I had never actually requested one.

Once I set the configuration of how the containers would communicate and set the limits of how many could be launched etc. I assumed it would launch automatically and wait for builds.

This was not the case and rightly so as the whole purpose is to use the Docker machine as efficiently as possible. Once the build is run it requests the specific Docker Jenkins-Slave by a label specified in the launch options. Then the correct container is launched on the remote Docker machine and the build is processed within the container. Once this was complete I could move on to customising my own container to suit my needs.

Using the evarga/Jenkins-slave Dockerfile as the template for my own Dockerfile I got started on adding curl, unzip and Gradle to the container image (Dockerfile's are used to install software and preconfigure Docker images) and setup the necessary environment variables. To complete this I simply used curl to download the gradle-2.9-bin.zip to /opt/gradle in the root of the image file system. The complete command:

```
RUN apt-get update && \
        apt-get -y upgrade && \
        apt-get install -y curl && \
        apt-get install -y openssh-server && \
        sed -i 's|session required        pam_loginuid.so|session        optional
pam_loginuid.so|g' /etc/pam.d/sshd && \
        mkdir -p /var/run/sshd && \
        apt-get install -y openjdk-7-jdk && \
        adduser --quiet jenkins && \
        echo "jenkins:jenkins" | chpasswd && \
        adduser jenkins sudo && \
        mkdir -p /opt && \
        curl -o /opt/gradle-2.9-bin.zip https://downloads.gradle.org/distributions/gradle-2.9-bin.zip
&& \
        apt-get install -y unzip && \
        unzip /opt/gradle-2.9-bin.zip -d /opt && mv /opt/gradle-2.9 /opt/gradle && \
        rm -f /opt/gradle-2.9-bin.zip
```

While initially it can look messy using one large RUN command it is in fact the best practice as each additional RUN command adds a new layer to the Docker image and therefore increases the file size.

After this I had another problem in that as the User is changed quite often in the background during the launch of the container and I suspect Jenkins is using the root user to run commands inside the container prefaced with su jenkins and as such isn't using the ENV variables ($GRADLE_HOME and $PATH) declared in the root users profile. I tried adding exports to .bashrc and .profile in the root of the Jenkins user but these were never initiated.

To get around this I used the environment injector plugin through the build configuration options to inject GRADLE_HOME and add to the path using:

```
        GRADLE_HOME=/opt/gradle
        PATH=$PATH:$GRADLE_HOME/bin
```

While Jenkins has an in built option to define global environment variables it does not work with customising the path and so the injector was needed.

With this working I can now run Gradle builds inside of the container and now that I understand the process for configuring my own custom containers many different types of builds will now be able to complete in isolation inside containers. Catching up on Documentation is the priority right now and I will continue work on the project once the midway presentations are complete.

My Reflection

Intended Changes

Further development of different containers is needed for the future.

Supervisor Meetings

Date of Meeting: 29/01/2016

Items discussed: Documentation, midpoint presentation and demo and discussion on progress of the project.
Action Items: Focus on finishing all documentation and submitting on time. Prepare for presentation.

Other Material Used

Any other reference material used in the project for example evaluation surveys etc.
CD containing code should be glued to the technical report.
(**Only applicable to the Final Report in May 2016**)