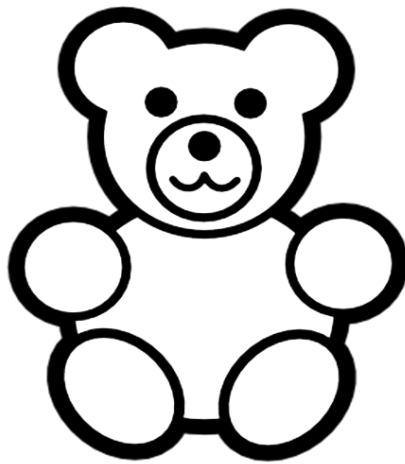


COMPARISON OF CONTINUOUS INTEGRATION SERVICES

Agile Processes



SCRUMMY BEARS

Cian Dowling – R00123948

Joshua Nuttall – R00128796

Eoin Lynch – R00123137

Table of Contents

Travis	2
Setup:	2
Step 1:	2
Step 2:	2
Step 3:	3
Evaluation:	4
Advantages:.....	4
Disadvantages:	5
Azure	5
Set-up Complications	5
Evaluation	6
Advantages:.....	6
Disadvantages:	7
Jenkins.....	7
Setup:	7
Step 1:	7
Step 2:	7
Step 3:	8
Step 4:	10
Step 5:	10
Evaluation:	11
Advantages:.....	12
Disadvantages:	12
Overall Evaluation	13

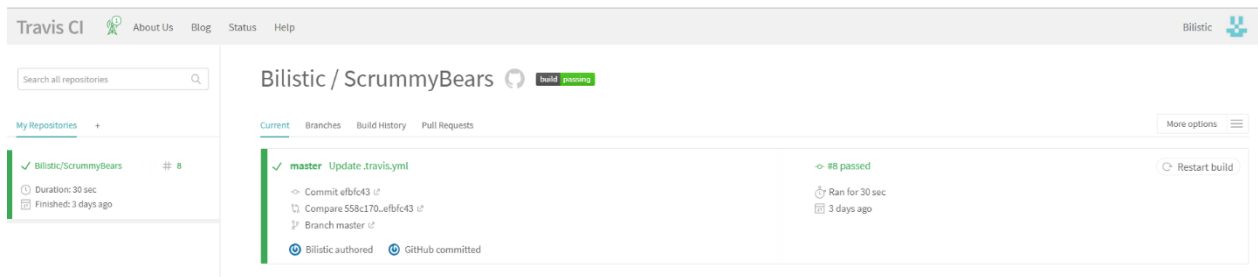
Travis

Setup:

Travis provides an easy interface with few steps to add a git repository and have a continuous integration environment running in a matter of minutes.

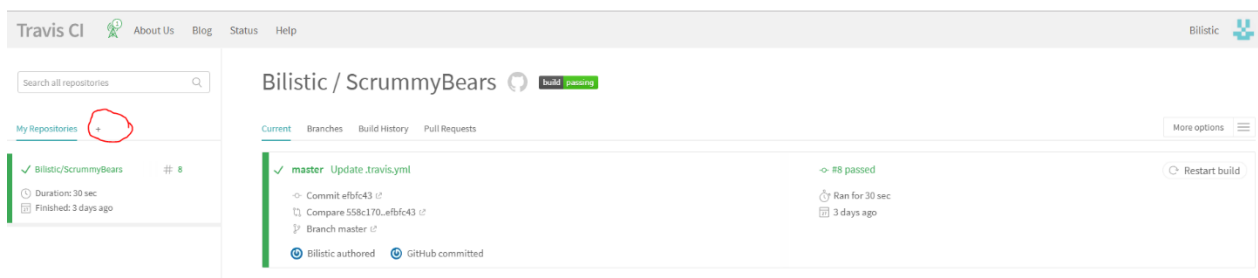
Step 1:

The simplicity of Travis CI starts right at the beginning with sign up. For this we will assume you have a valid GitHub account. Travis CI is an extension for the version control website GitHub as such signing up to GitHub is as easy as pressing the sign in button and logging in using your GitHub credentials. By doing this you give Travis permissions to your GitHub account (presented when signing up). Logging in will present you with the following page (minus my already run tasks):



Step 2:

Next we must configure our GitHub repository to be listened to for updates by Travis. Do this by pressing the indicated button.



This will list all current public GitHub Repositories you own or are a part of, if the repositories shown are not correct simply pressing the sync button should refresh this.

We're only showing your public repositories. You can find your private projects on travis-ci.com.



1
Flick the repository switch on



2
Add .travis.yml file to your repository



3
Trigger your first build with a git push

 Filter repositories



 Bilistic/Comp7039-ASIGN-1



 Bilistic/Phillies-Flowers



 Bilistic/ScrummyBears

Enabling the correct repository is as simple as selecting the listed repository as seen above with the repo "ScrummyBears".

Step 3:

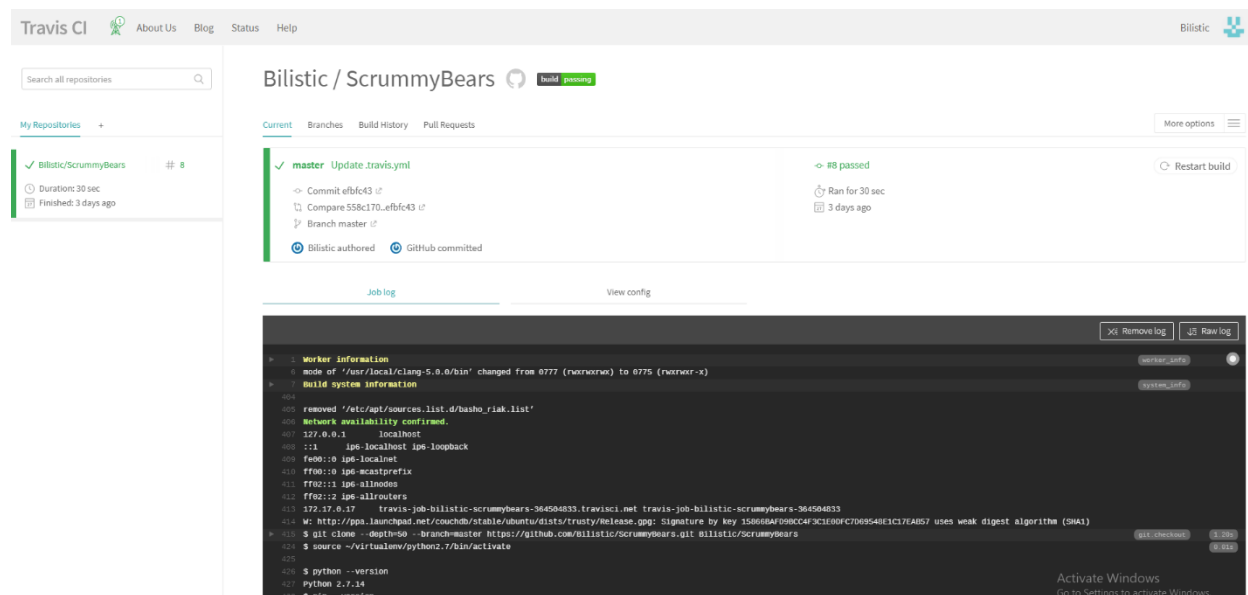
Travis listens for an update on the GitHub repository selected. When an update occurs it attempts to build the files in the repository. To do this your system needs two things, A Yaml file formatted so that Travis can Interpret it and successfully create an environment and a set of tests to run on the newly built system. As such we add a file to our repository called .travis.yml the content of this file is different for each project as it specifies the language, version, environment and test script to run. For example, we have a project test script labeled test_CtoF.py and the language is python 2.7 as such our file is as follows.

```
language: python
python:
  - "2.7"
script:
  - python test_CtoF.py
```

Once complete this is the final step in setting up Travis for continuous integration as such the project will build and test each time the git hum repository is updated.

Evaluation:

As mentioned in the setup numerous times, setup of Travis is extremely easy and is done in two easy steps excluding the sign-up process. Since Travis is presented as an extension of GitHub it is extremely light weight and is presented as a minimalistic Gui upon which someone can easily understand without prior training. Below is an image of the Travis admin panel.



As you can see build success is indicated by colour code green for pass and red for failure as well as the title "passing" by the repo name. For managing multiple repositories, it is as easy as selecting the repo you wish to view on the left-hand column. Build logs are presented in the console area provided. Performance wise Travis operated as expected most of the time. However, when I done two consecutive updates in a row Travis failed to recognize the later and created no build job for this version. I suspect the issue being it was already running a build job for the previous update. As such I had to restart manually the build which resulted in it seeing the new update. This is a minor bug that I created by using GitHub in a manner that would not be the norm as such I don't believe this should be a major problem for someone considering Travis.

Advantages:

- **Ease of Use** – As mentioned throughout the whole application is easy to adapt to and understand with very little learning involved. As simple as deploy and forget.
- **Customization** – The platform offers a range of customizable features for builds supporting up to 32 programming languages and 4 operating systems.
- **Hosting** – Travis is cloud based as it is an extension to GitHub hence it has a consistent presence and removes the need to setup dedicated hosting.

Disadvantages:

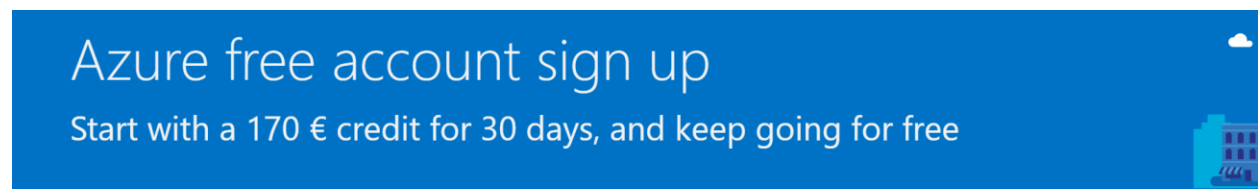
- **GitHub** – User have no choice in what repository system they use for version control due to Travis being an extension of GitHub, whilst a disadvantage this is miniscule and does not really matter as it is the world's most popular form of version control.
- **Testing** – In Travis the inclusion of test files is mandatory to build a system. Whilst this does promote best practices and reduce effort in the long run testing should be an opt in option.

Azure

Set-up Complications

When setting up Microsoft Azure , I began the sign up process, which requested the usual personal information such as name, email, occupation and some company information if applicable. This is to be expected as most, if not all services would request this on sign up.

The problem came when I began the second step of the sign up process. Here I was asked for a phone number to “verify my identity”. I entered my phone number and promptly received a text message containing a code.



1 Identity verification by phone

Country code

Ireland (+353)

Phone number

08 33665963

Text me

Call me

Upon entering the code into the space provided I was taken to a second page. This page requested my Debit/Credit Card details, again for “verification purposes”. Messages on the page informed me that I would not be charged for a free account.

2 Identity verification by card



We keep prices low by verifying that account holders are real people, not bots or anonymous trouble makers. Don't worry, your card will not be charged unless you explicitly convert to a paid offer, though you might see a temporary authorization hold.



Card number

Evaluation

The Azure set up seemed to be extremely easy. The documentation was very clear and concise but was not watered down too much.

The request for card details was a deal breaker in my opinion. I did not feel comfortable giving my personal bank account details to a company in return for access to a so called free service. Nor could I recommend any person to do so.

There has been a lot of controversy recently with regard to the personal information companies are storing about their users. And I believe that we need to be more cautious with the information we provide to these companies. I do not believe that the card information was being stored as a method of verification and instead feel that it was done in order to influence your choice when proposed with the option to upgrade to a premium account by making the payment process quicker. I feel that this is a form of sneaky manipulation.

Overall my experience with using Azure CI was a mixed one. In keeping with most Microsoft products the documentation and sign up ease was second to none. It was quick and easy. However I halted my tests when requested for too much personal data. In a market full of very popular rivals, all of which provide mostly the exact same services, this was enough to disregard Azure. With other options out there, I can not find a reason to risk your personal information to use Microsoft Azure.

My recommendation is to not use Azure CI, or any other CI which requests over the top amounts of personal data, especially for so called free services.

Advantages:

- **Concise Documentation** – The documentation provided for the set up, was very clear and concise.
- **Cost** – Azure provides a completely free service, with options to upgrade for access to premium features.
- **Large Company** – Azure is a service by Microsoft meaning you will have access to some of the best support services and the highest security protocols will be in place to protect your data and your application.

Disadvantages:

- **Personal Information** –Azure requests far too much personal information from users when they sign up to a free account. This includes names, addresses, phone numbers and credit card details.

Jenkins

Setup:

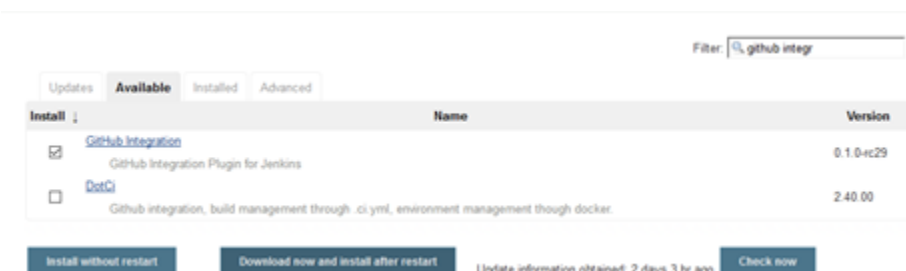
Jenkins provides the ability to automatically initiate a build integration from a GitHub repository. Whenever the repository is updated, Jenkins will start a build of the product to test that the newly integrated code will run successfully.

Step 1:

The first step after Jenkins has been downloaded is to download the GitHub plugin. This is achieved using the 'Manage Plugins' feature on the Jenkins dashboard. This allows GitHub and Jenkins to interact with one another.

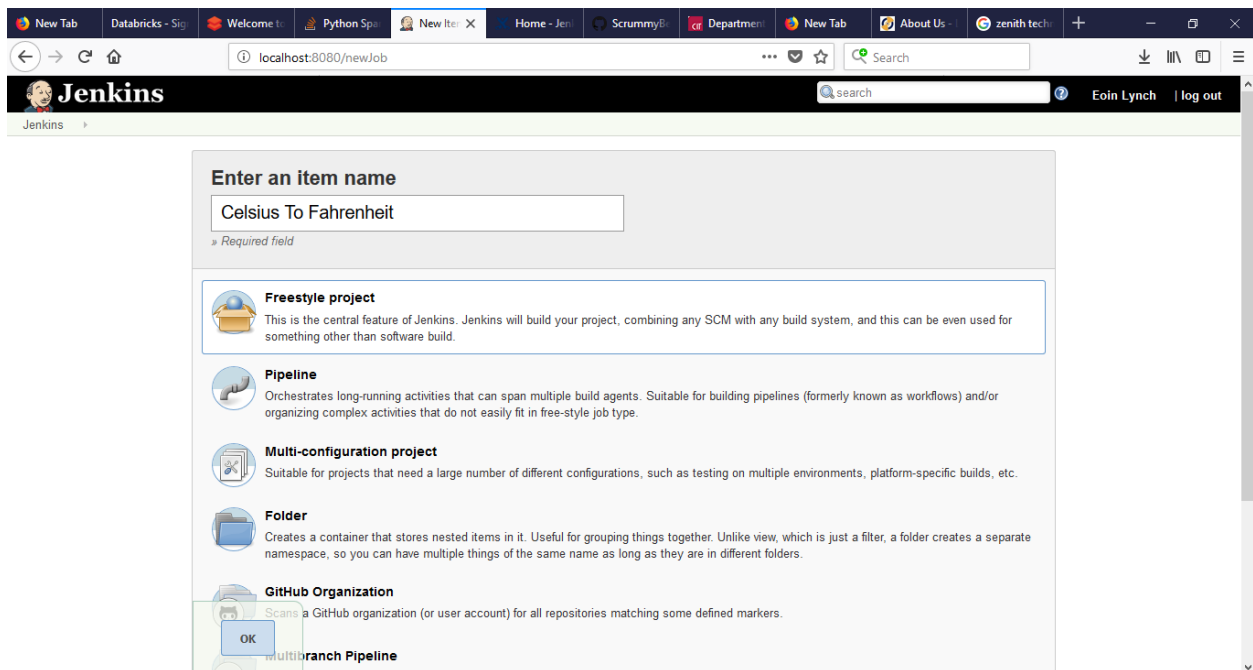


The plugin can be found by typing 'GitHub Integration' into the plugin search bar.



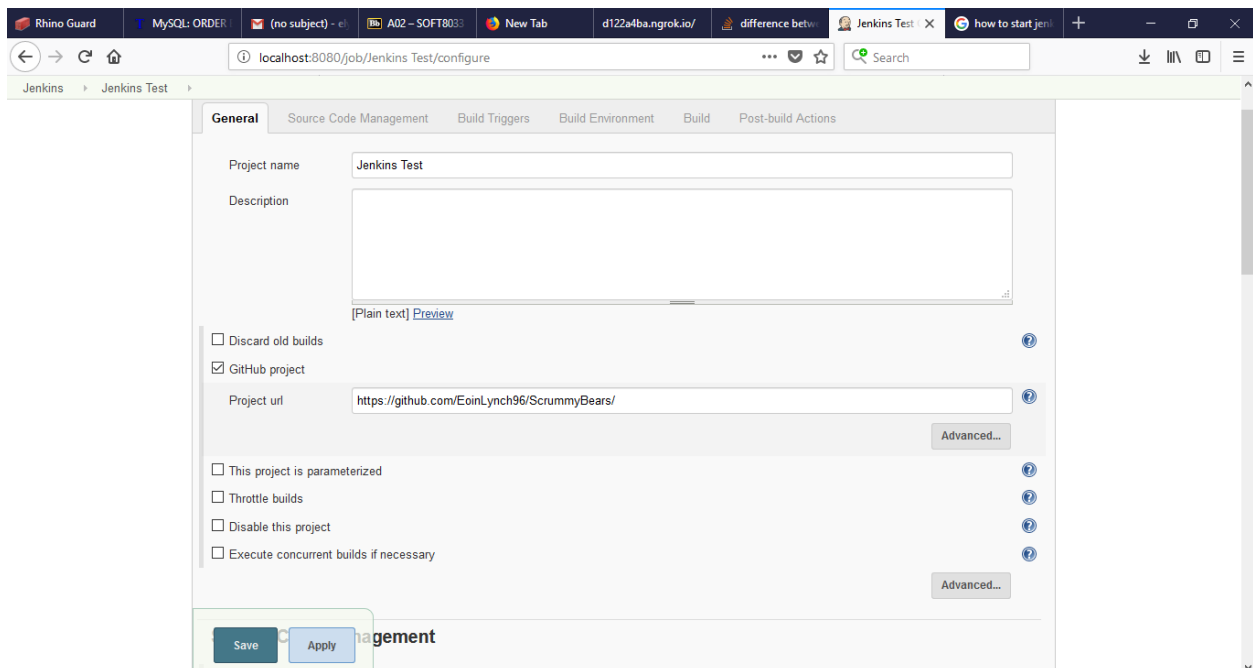
Step 2:

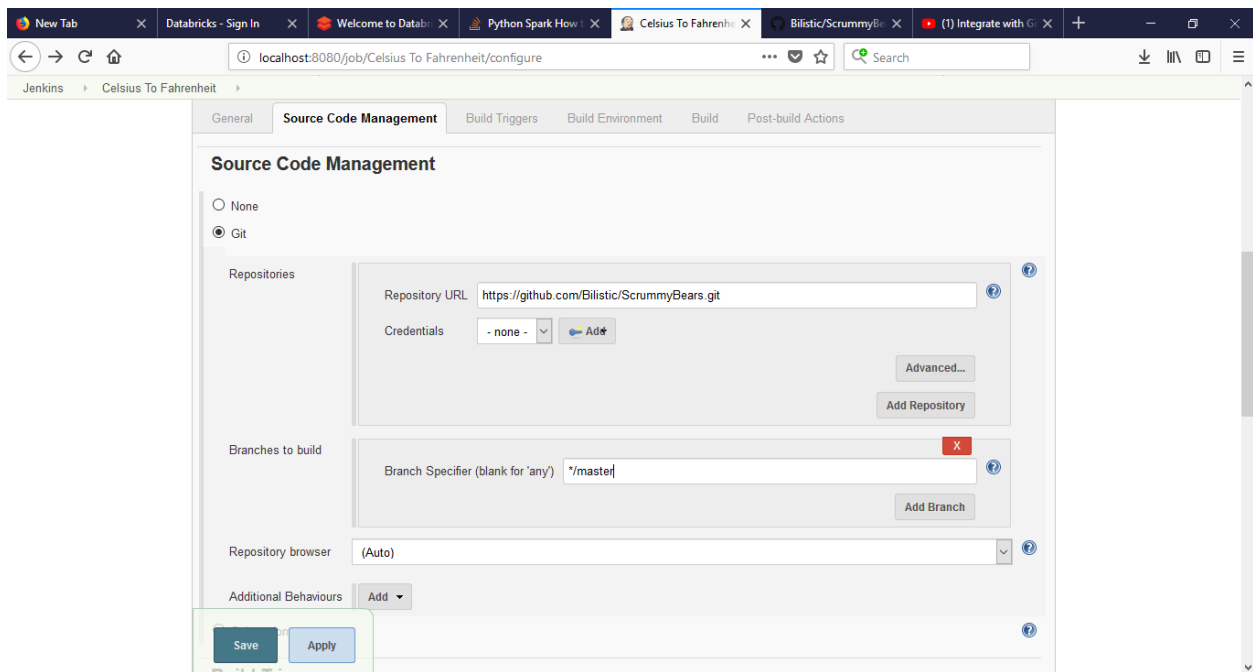
From the Jenkins dashboards, click 'New Item'. We will then give this new job a name. This Jenkins job will be responsible for running the code from our team repository when code is changed or added to it. A 'Freestyle Project' is chosen for this demonstration. Selecting the 'Pipeline' project gives users the ability to break their project into different stages. For larger systems, this can give users and insight into what parts of their projects are breaking (if any).



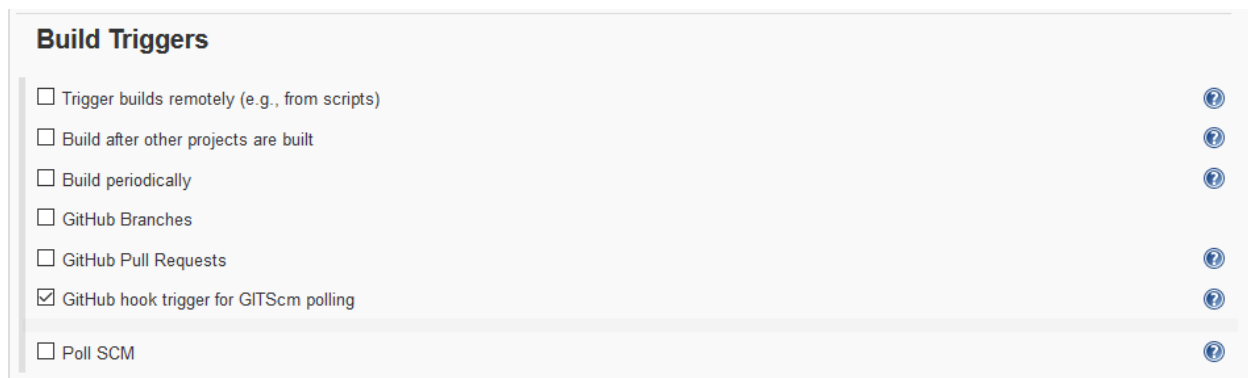
Step 3:

We select 'git' as the source code manager and provide the link to the GitHub repository. The default branch is 'master' but this can be changed if necessary.



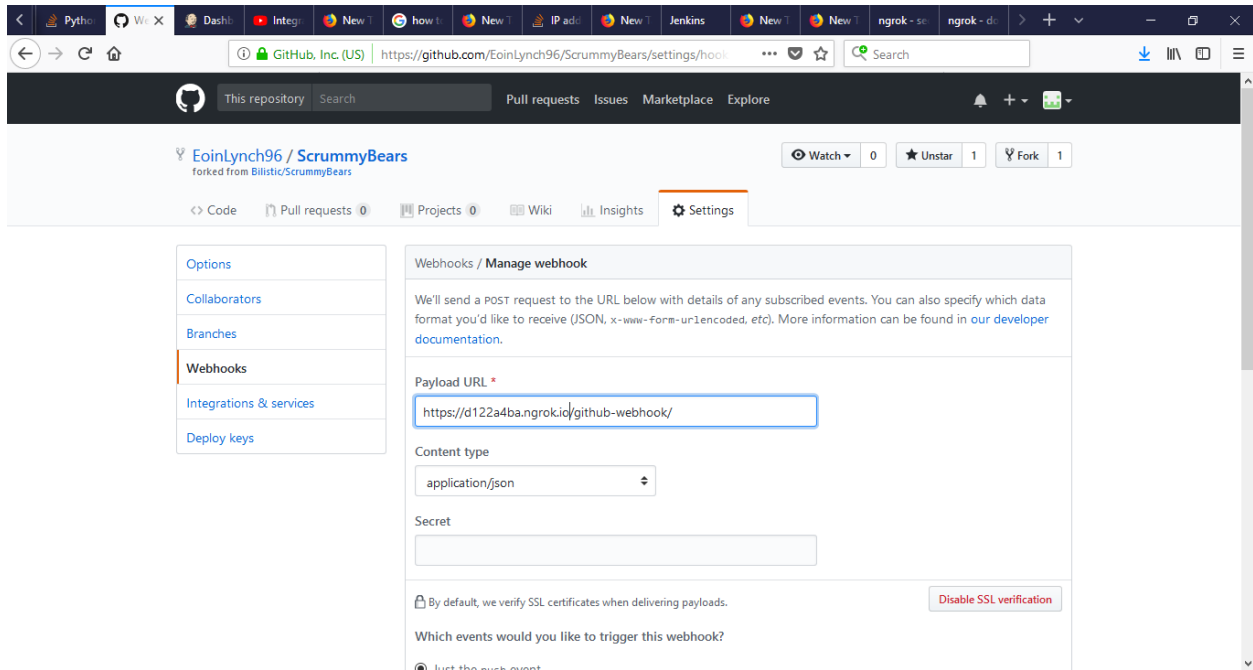


When selecting build triggers, choose 'GitHub hook trigger for GITScm polling'. This tells Jenkins to start building the project when a commit is made to the repo.



Step 4:

Next, we need to setup a 'Webhook'. This allows the GitHub repo to communicate back to the Jenkins service. To do this we first navigate to the 'Settings' tab of the GitHub repository. Next we provide the URL of our Jenkins dashboard to the Webhook payload URL. Be sure to append '/github-webhook/' to the end of this URL.



Step 5:

After following steps 1-4, the Jenkins/GitHub Continuous integration environment will have been set-up. This can be verified by making a change/ submitting a new file to the GitHub repo. After the commit has been made, a display with information about the building project will be available information can be seen in the Jenkins dashboard. If this build is successful, a success message will be displayed once the project has been built.

 [Back to Dashboard](#)

 [Status](#)

 [Changes](#)

 [Workspace](#)



 [Build Now](#)


 [Delete Project](#)


 [Configure](#)

 [GitHub Hook Log](#)


 [GitHub](#)


 **Build History** [trend](#) 





 **#1**

Apr 15, 2018 1:37 PM









 [RSS for all](#)  [RSS for failures](#)

Evaluation:

Overall, the setup of Jenkins was quite straightforward and there is good documentation for using the product. There is very good documentation for the software provided on <https://jenkins.io/doc/> and by other third-party contributors. As well as this, the GitHub integration plugin makes it easy to hook up the GitHub commit trigger. The dashboard GUI makes for a lightweight deployment of the service and is intuitive to use.

[Export as plain XML \(Recurse in subfolders\)](#)

	Build	Time Since ↑	Status	
	Jenkins Test #2	3 min 23 sec	stable	
	Jenkins Test #1	1 day 7 hr	stable	

Success of the builds can be seen in the 'Build History'. This allows users to see exactly what build caused the errors.

Although there weren't many issues during the setup of Jenkins CI with GitHub, a few workarounds were required to get the system up and running. The biggest problem was providing the webhook URL to GitHub from a locally hosted Jenkins service. Unfortunately, GitHub would not accept a URL that reads 'localhost:8080'. A non-local IP address had to be assigned using a tool called Ngrok to tunnel the

localhost to the public internet through a secure tunnel. This then gives an IP address such as <https://daf5384d.ngrok.io> instead of localhost:8080. This address can then be pasted into the Webhooks URL on GitHub. Once completed, the continuous integration should work.

Another minor disadvantage that had to be overcome when setting up the Jenkins CI with the group GitHub. Since only members of the highest permissions can edit the 'Settings' to add a webhook, non-owners of the repo will either need to have increased permissions or can 'fork' the repo themselves to create their own clone of the repo and can then add the necessary Webhook.

Advantages:

- **Good Documentation** – The documentation provided by the Jenkins is clear and easy to follow.
- **Community Support** – Jenkins is open source and is itself a GitHub project with many contributors. This lends itself to an ever-improving project with feedback from users of the system.
- **Future Automation** – Once a project has been configured, all future builds can be run automatically.
- **Supported Repositories** – GitHub is a well-supported source code manager, but Jenkins also allows for integration with other version control software such as SVN.

Disadvantages:

- **Localhost issue with GitHub Webhooks** – Had to use third party software to setup a Webhook on GitHub as the localhost Jenkins was not accepted.
- **Permissions Issues** – The highest privileges are required on GitHub to edit the Webhooks.
- **Not made for microservices** – Jenkins was not designed for continuous integration on multiple smaller services as one would have in a microservices architecture. For this reason there may be some limitations when piping information from one service to another with Jenkins.

Overall Evaluation

All of the reviewed Continuous Integration services have unique features that would make them best suited for certain projects.

For this project we have determined that we can eliminate Azure. It's request for so much personal information sets it far behind both Travis and Jenkins.

Between Travis and Jenkins we have concluded that Travis would be the best fit for this project. Although vastly different, they both offer the same basic services, Travis is a lightweight CI that requires minimum effort on behalf of the user to set up and run. This enhances the experiences of less tech savvy people, allowing for team members of different skill sets and abilities to use and contribute to the project. While Travis is only an extension of GitHub, the functionality it provides is on par with much larger fully fledged services. The free version of Travis offers full functionality and access to all features, but it is limited to public repositories only. Making it a great choice for open source projects.

Jenkins would be a far better alternative for larger corporate development environments. This is due to Jenkins providing a more secure experience by locally hosting the Jenkins Server, removing the need to pass private code to a third party. Development teams are not limited to the use of GitHub like they are on Travis, this allows teams to use source code management systems which provide free or cheaper private repositories. This in turn reduces the overall project costs furthermore this allows for more secure projects as the build and source code can be hosted offline.

From the above we can conclude that Travis is best suited for smaller development teams, who are more concerned with cost, resources and speed of deployment over project privatisation. For larger organisations we believe that Jenkins would be the best option. Due to these differences it is impractical to recommend the perfect solution for all project types.