Development Operations

James Mahoney T00145442

# Introduction

For this Continuous Assessment I have been asked to Build an automated deployment pipeline for an application that I have built or sourced.

I have chosen to use a fork of the spring-projects/spring-petclinic repository from GitHub. This will be the code used throughout. The Petclinic application is a spring boot application. A jar file can be built using maven and can be run from Windows CMD or Linux terminal using the command “java -jar \*\*filename\*\*.jar”.

The tools and technologies used in this report are listed as follows.

* Jenkins
* Maven
* Junit
* SonarQube
* Docker
* Docker Hub
* AWS Cloud Compute EC2
* ssh-agent plugin
* with credentials plugin
* Monitor view plugin

**Pipeline flow diagram**

Diagram

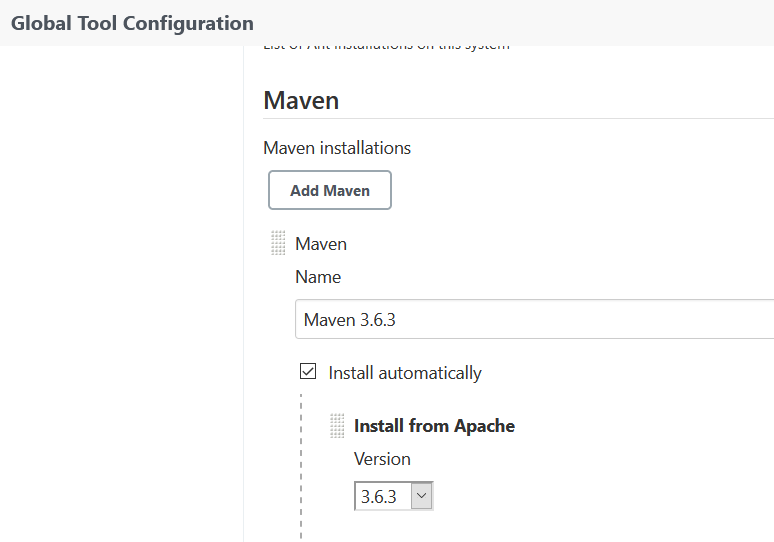
Description automatically generated

# Continuous Integration

Tools and plugins had to be first configured and installed.

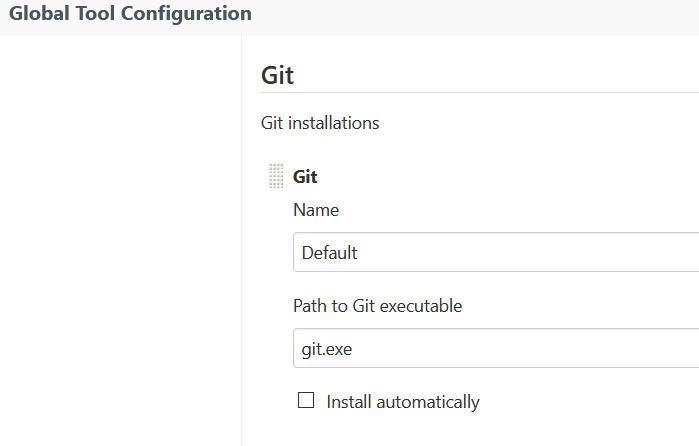
**Maven**

Maven had to be configured in global tool settings as shown below.



**Git**

Git must be set up in global tool configuration.

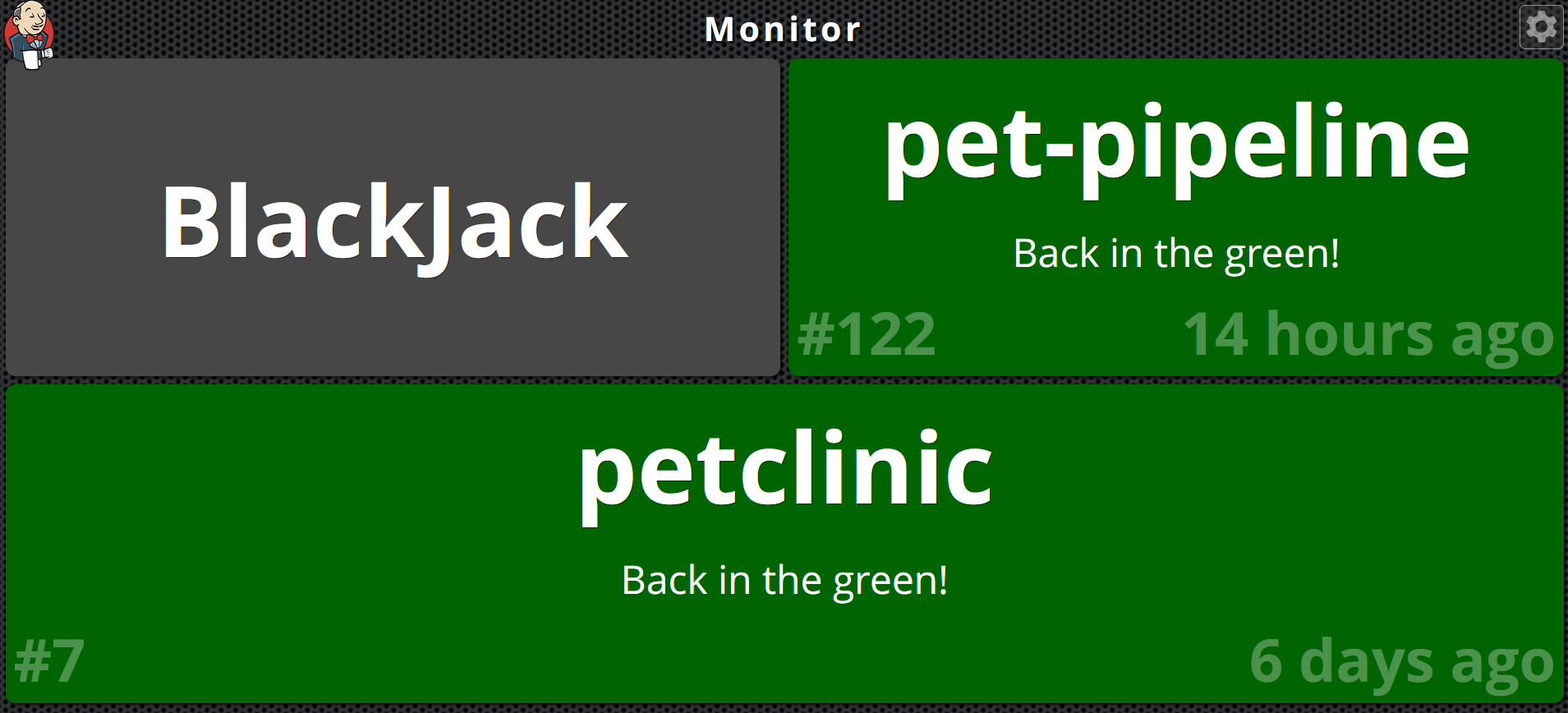


**Junit**

Maven will run all unit tests and report them using surefire.

**Monitor**

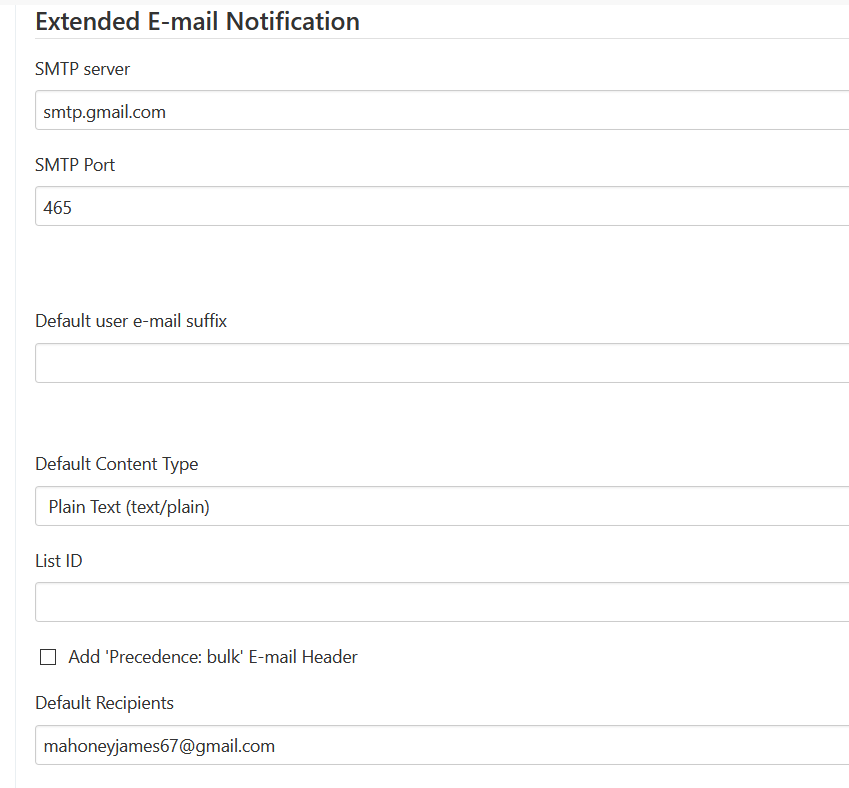
A monitor view was set up using the build monitor view plugin. An example can be seen below.



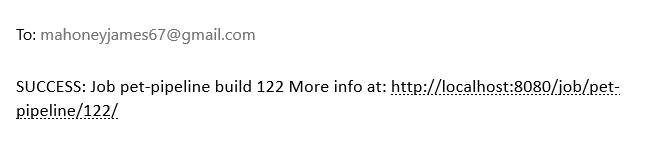
This view shows build success, fails and abortions.

**Email Notifications**

Email was set up by first allowing google account to interact with less secure apps and then setting up the email setting in configuration.

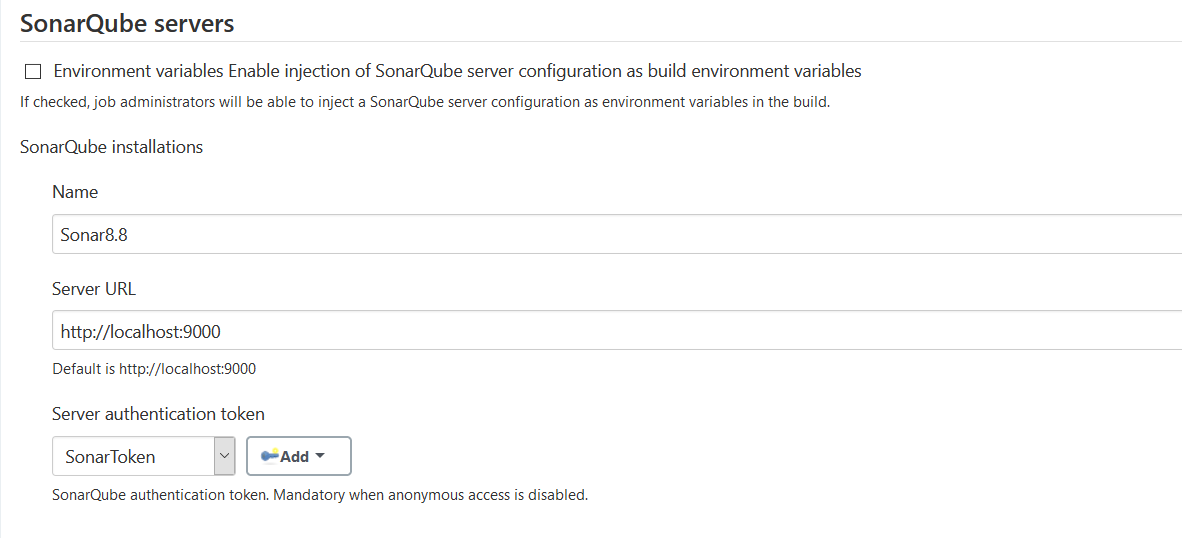


Example of a sent email

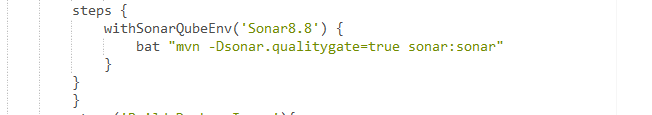


**SonarQube**

SonarQube was set up by first installing the sonar scanner plugin and configuring the server address and access token in Jenkins/configuration/SonarQube servers.

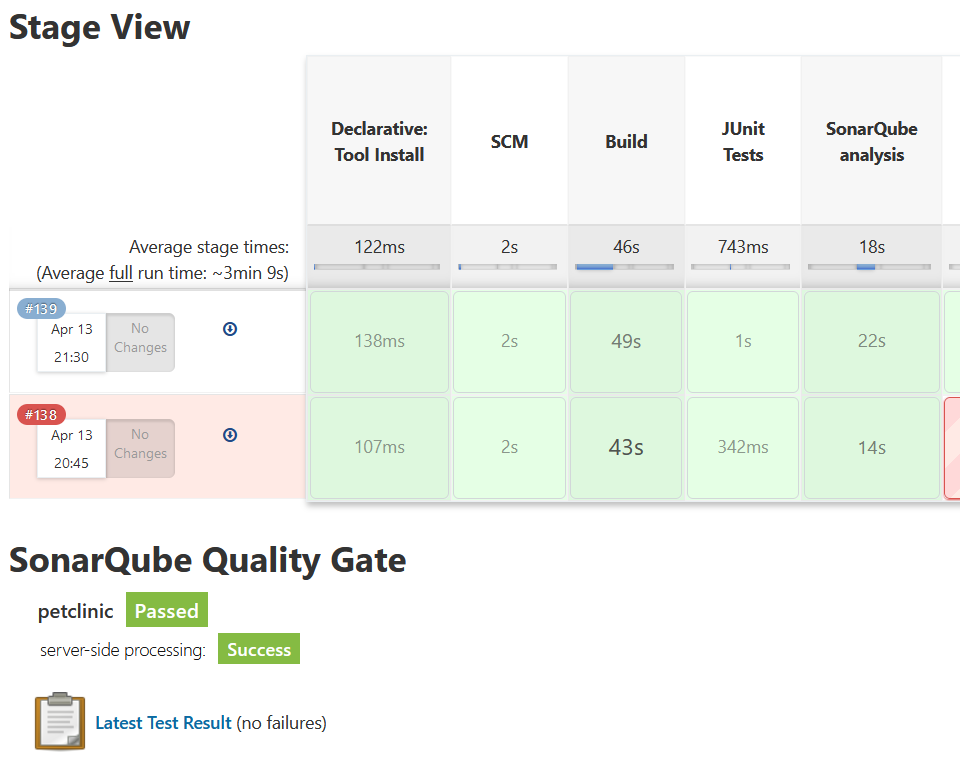


Pipeline code to use SonarQube can be seen below.



The code “-Dsonar.qualitygate-true” will stall the pipeline for the quality gate check to pass.

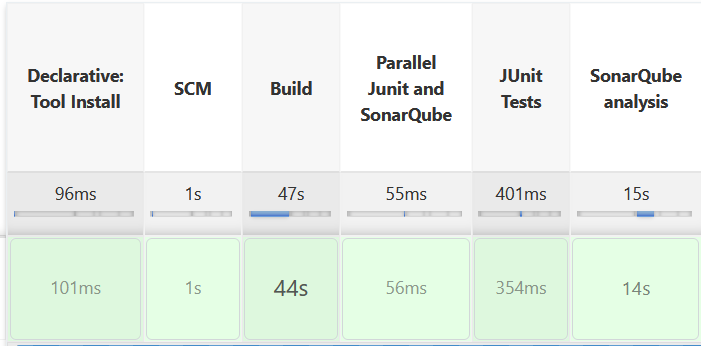
The pipeline view can be seen below along with the quality gate status. Once these steps are confirmed to run, The Junit step and the SonarQube analysis will be run parallel.



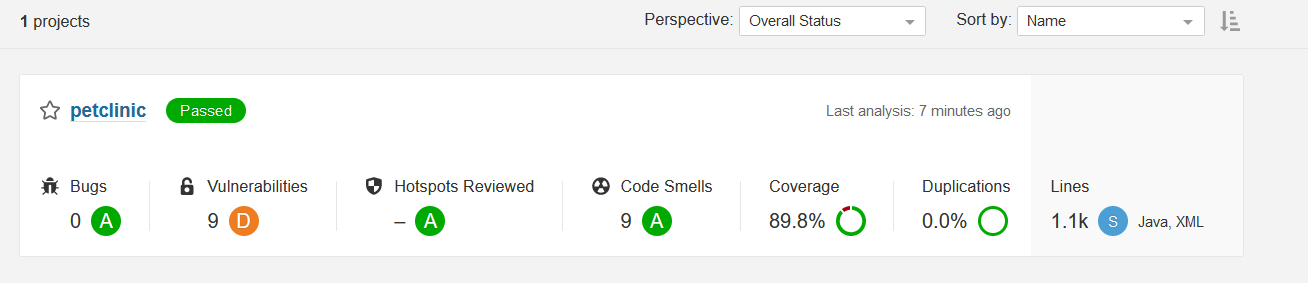
# Building Code

The pipeline code to use install Maven, Git repo, build using Maven and run JUnit tests as well and archive the build artifact and run SonarQube analysis in parallel.





Below is a screengrab of the SonarQube analysis.



As seen above the run was successful. This build can be triggered manually or set to build whenever need. Here is an example of automating to build every day.

Graphical user interface, text, application

Description automatically generated

# End to End Automation

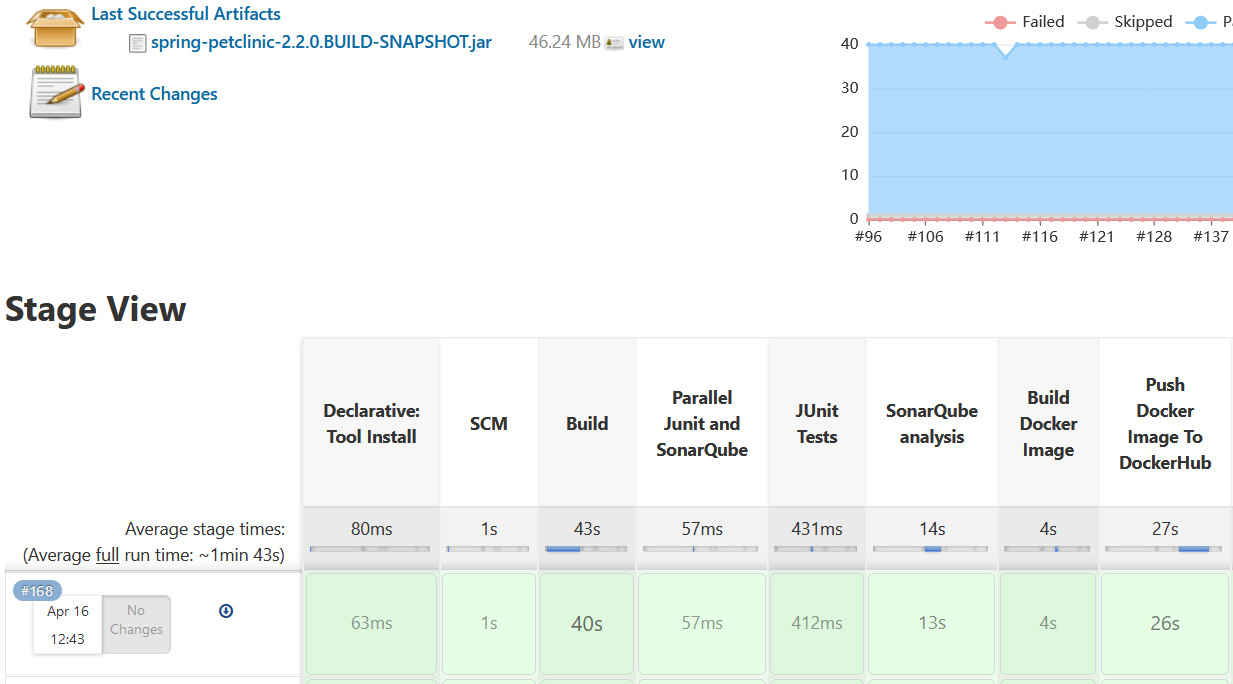
The next addition to the pipeline is a stage to build a docker image and to push container to Docker Hub. The first step is to run a batch command to build the Docker image. This builds the docker image from the Dockerfile.

Next the image is pushed to a dockerhub account once logged in using the with credentials plugin. This plugin lets you set up a variable that is not exposed in code and use the variable in the block.

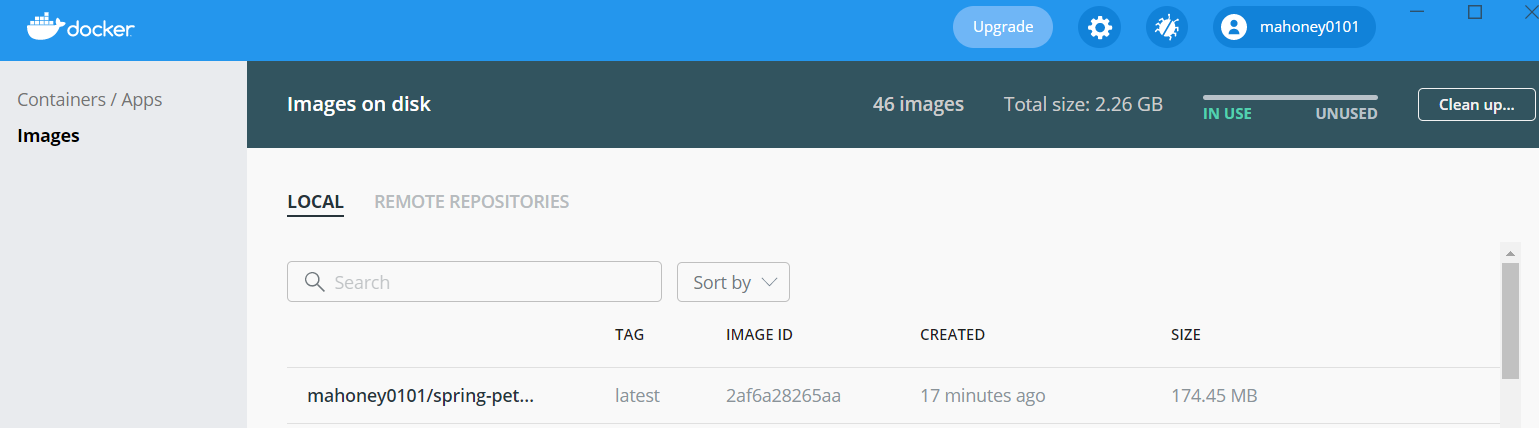
**Pipeline code**



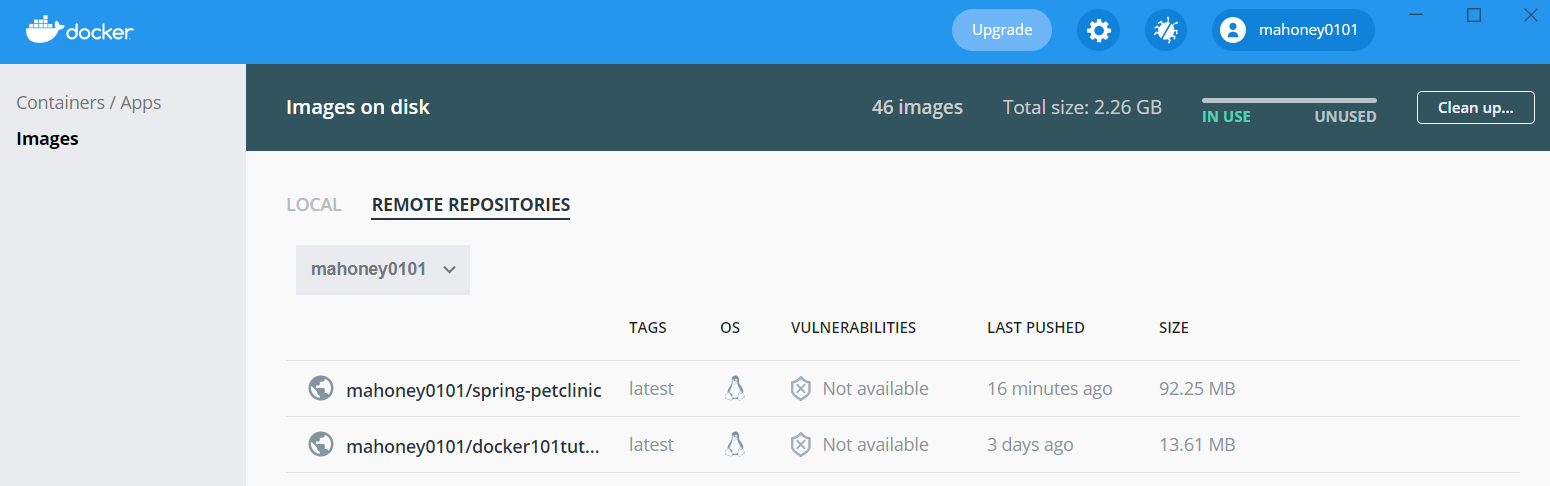
**Successful pipeline**



**Local image**



**Docker Hub image**

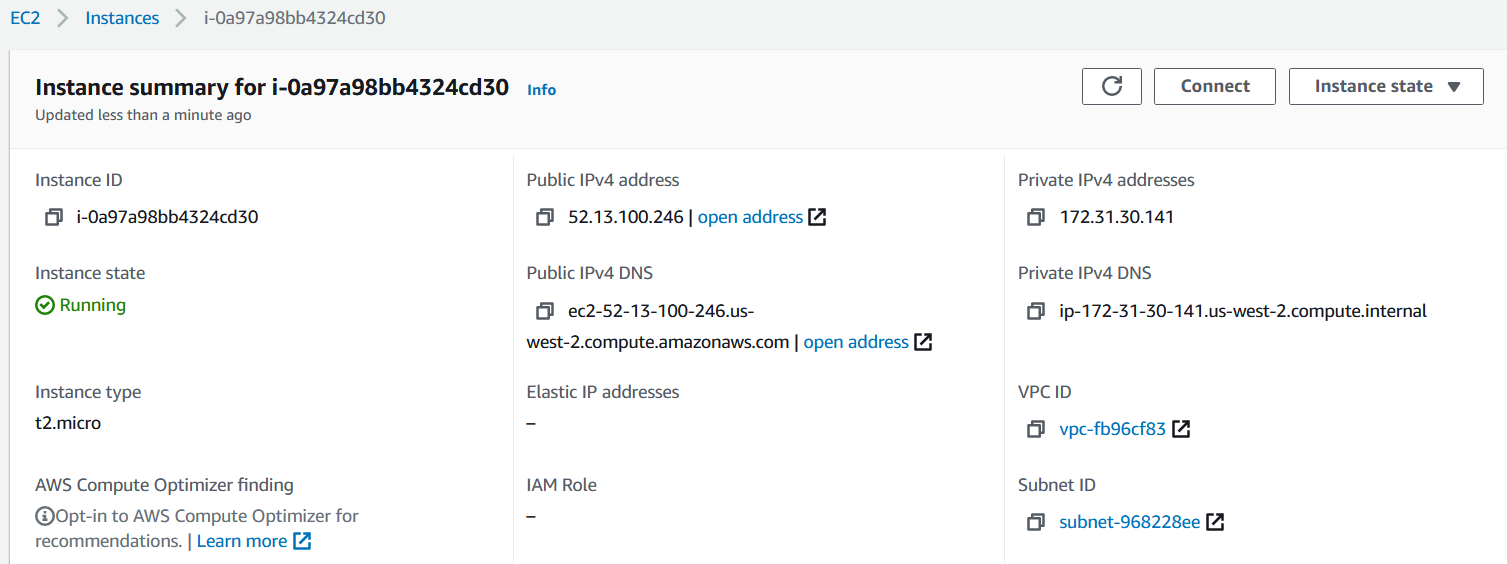


At this point continuous delivery is set up.

# Provisioning Cloud resources

An EC2 image is used to deploy the application to. The machine runs amazons Linux 2 OS.

**Instance Summary**



Docker was installed on the machine and started. The machine was set up so Docker could be called without the sudo command using the following commands.

sudo yum install docker

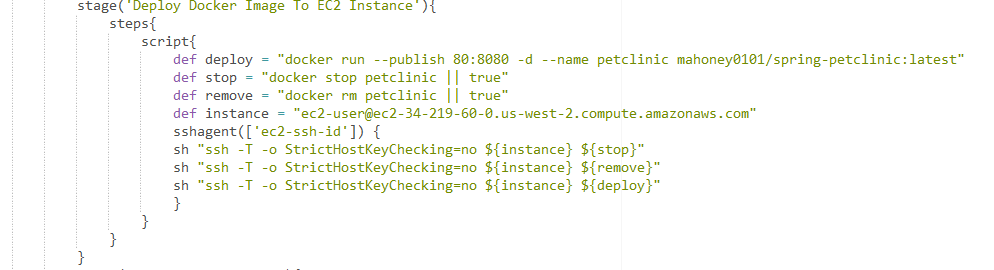
sudo service docker start

sudo usermod -a -G docker ec2-user

# Deploying application

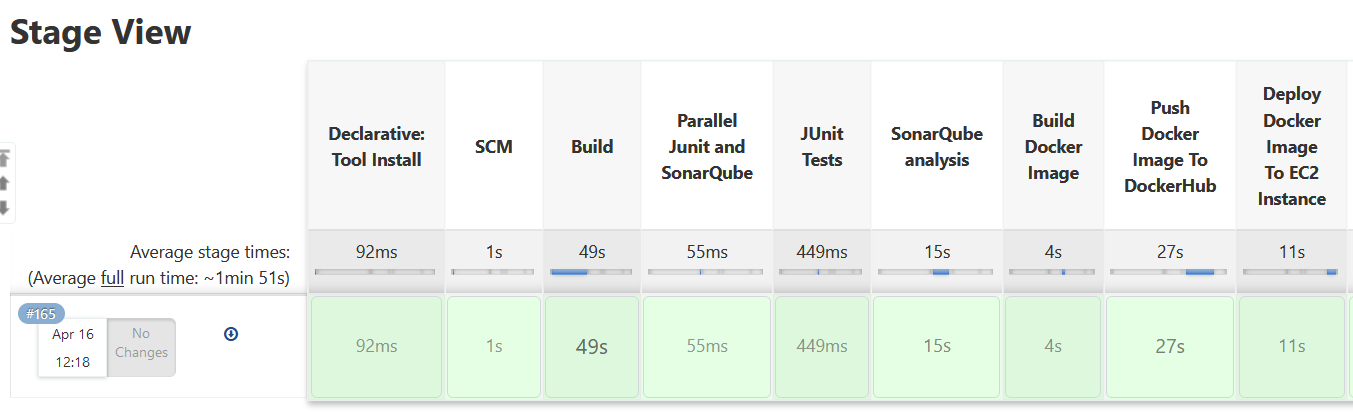
To deploy the application the the EC2 instance the sshagent was used. This plugin allows the user to store a private key and use the key in a ssh command. The pipeline code can be seen below.

**Pipeline code**



In this step, three commands are issued to the EC2 instance. The first command will stop the running instance with the name “petclinic”. The “|| true” will stop jenkins failing if the docker image doesn’t exist. This is mainly for the first time deploying the image. The second command removes the image and the third command deploys the latest image from dockerhub.

**Successful pipeline**



In the docker run command above the port 8080 that the tomcat server uses in the docker image is mapped to port 80. A security rule was set to allow all http traffic for the EC2 instance. http uses port 80 so the application will be viewable when accessing the EC2 instance using http. If you want to use https just change the docker run command in the pipeline to “docker run –publish 443:8080 \*\*imagename\*\*:\*\*tag\*\*”.

Once deployed the application can be seen as below.

**Live application**

Graphical user interface, text, application, website

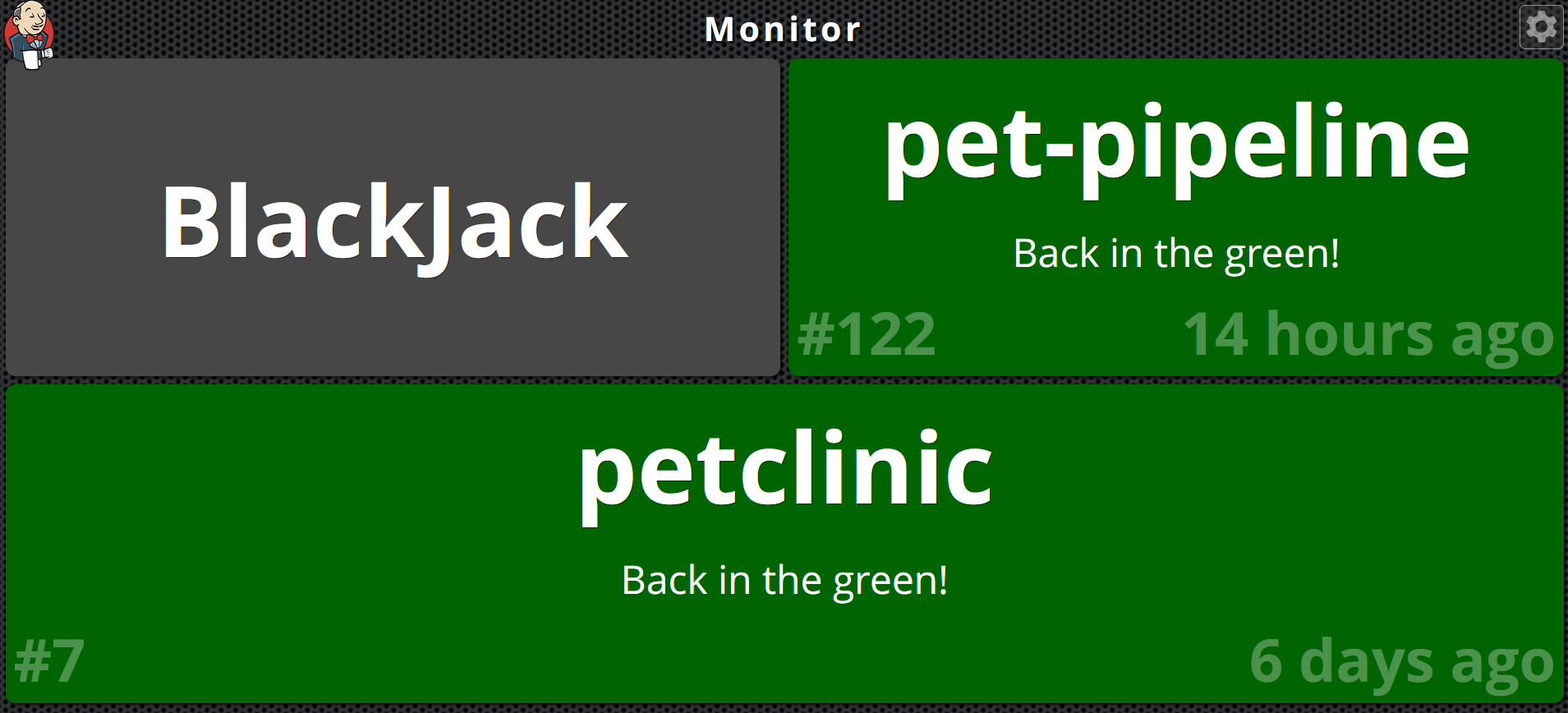
Description automatically generated

# Monitoring Infrastructure and applications

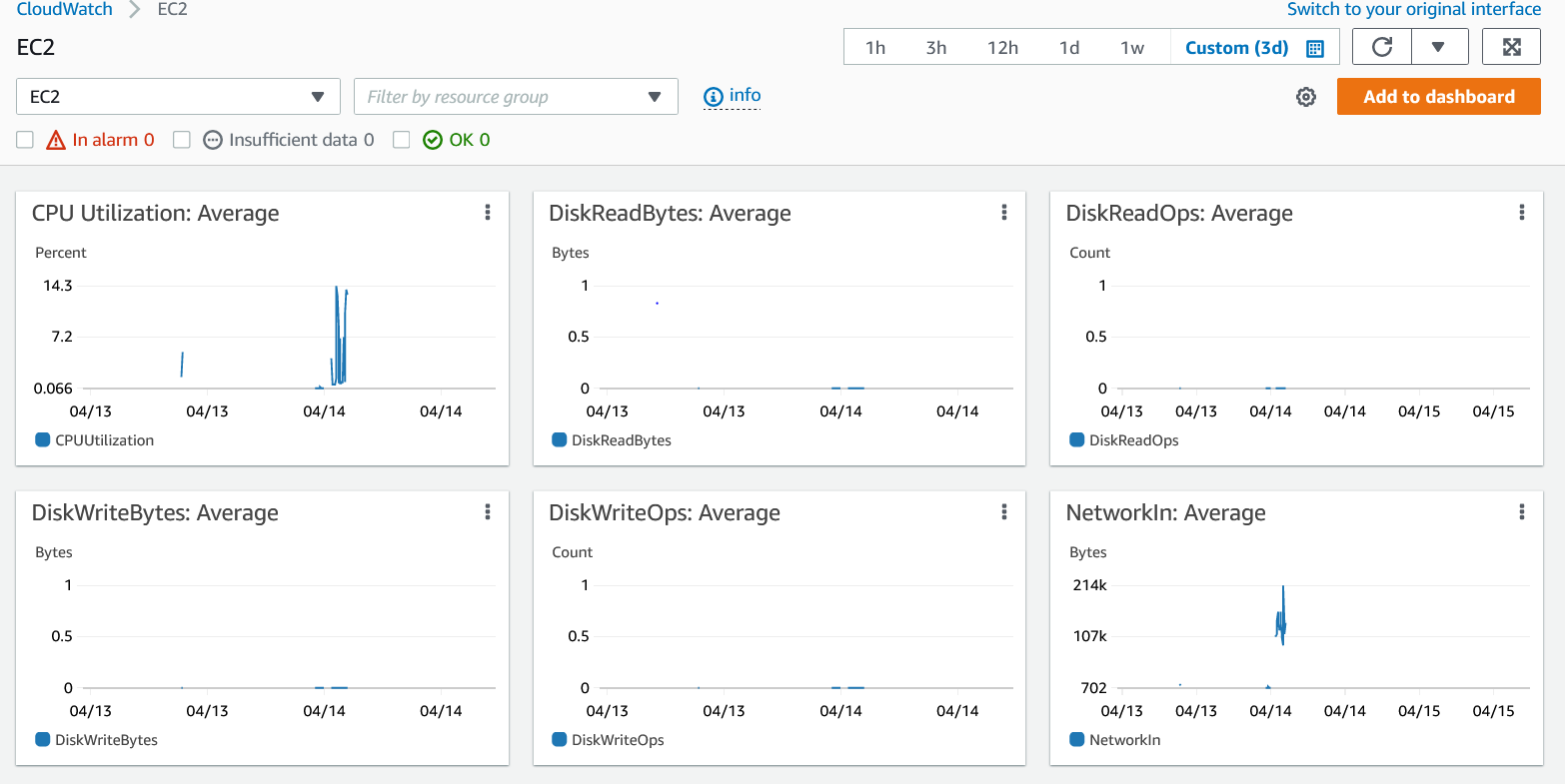
Monitoring infrastructure and applications and resource is an important part of the CI/CD lifecycle as it helps to ensure everything working as planned and allows for a DevOps engineer to stop any failures or performance issues. It gives the DevOps engineers and idea of the general health of the setup. This is important as it allows problems to be caught early.

For this pipeline, the monitor view was used, and this could be displayed in an office allowing members to see if their changes to the codebase did or did not break the current build.

**Monitor**



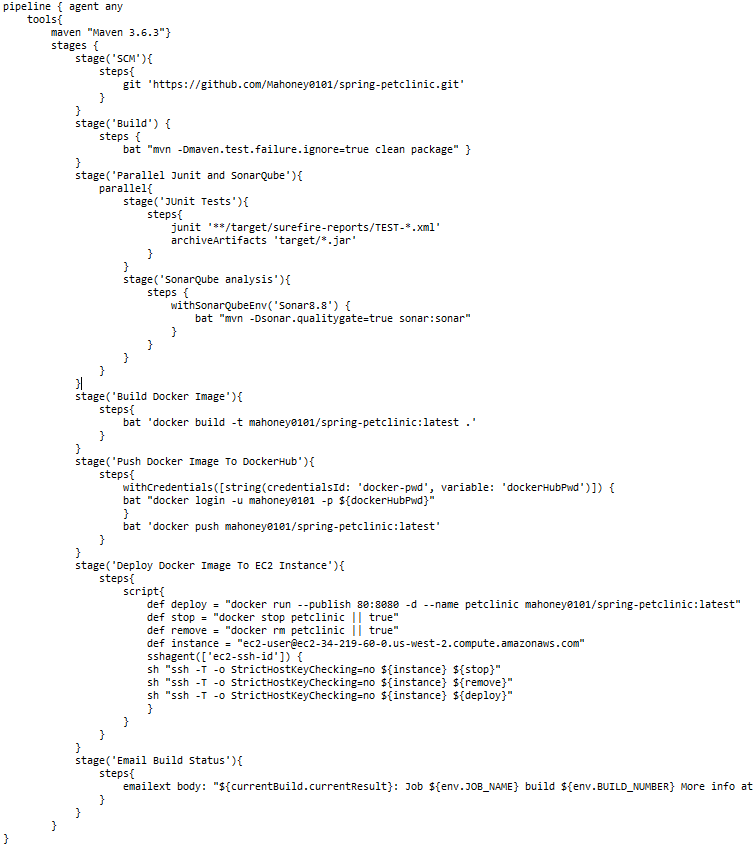
For monitoring the Infrastructure AWS CloudWatch was used. It was very simple to access. Just search for CloudWatch in AWS list of services and choose your EC2 instance and you can monitor all metrics.



# Orchestrating application deployment

The entire pipeline script can be seen below.

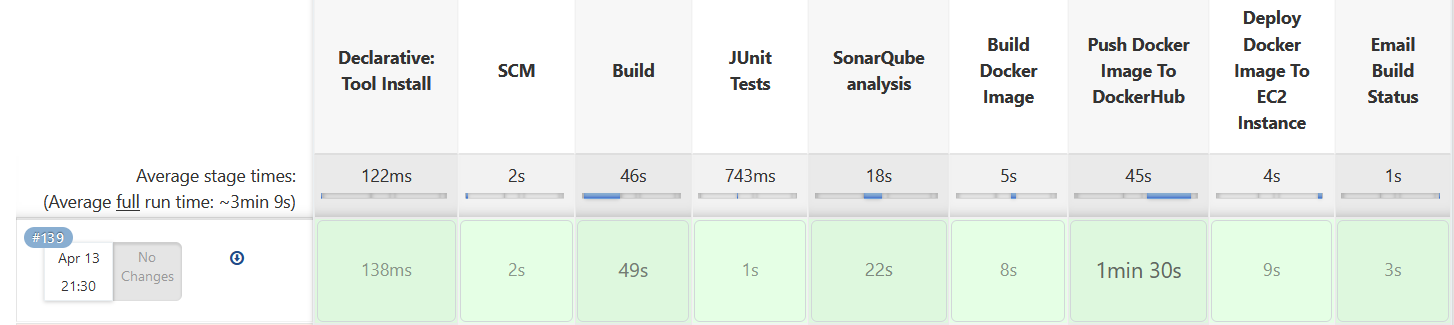
**Full pipeline script**



The pipeline steps are as follows.

* Install Maven
* Git clone
* Build
* Test
* Build Docker Image
* Push Docker image to Docker Hub
* Deploy Docker image to EC2
* Email status

**Successful pipeline**



# Conclusion

The goal of this report was to Build an automated deployment pipeline for an application that I have built or sourced. The status of this goal was successful. A pipeline was set up that will install tools, git clones a repo, builds the repo using maven, tests using Junit, scans using SonarQube, builds a Docker image, pushes that image to Docker Hub, then uses secure shell to clean up the EC2 instance, deploy the image to the EC2 instance and finally emails the build status to a mailing group.

The pipeline can be run without any manual configuration once the EC2 instance is up and running everything is automated.

An issue when developing this pipeline was that sshagent plugin was not updated to use OpenSSH when it was installed on windows so the path environment variable had to be updated and the git sshagent had to be added so that Jenkins would hit the git version first. Once this was done the sshagent plugin worked fine. The issue is well documented but has not been fixed.

There was also an issue with Docker Desktop where the engine settings had to be changed from “buildkit: true” to “buildkit:false”. This change was necessary for docker to find the build artifact. When buildkit was set to true, Docker was looking in a Linux filesystem that does not exist.