ShipIT Project Definition

Members: Cameron Bay, Adam Thiriot, Charles Robinson, Erik Pinho, Trevor Garn
Sponser: Jackson Behling
Coach: Jackson Behling

Project Statement

Design and develop an AWS code configured pipeline with an API, able to build asynchronously and with an interface for monitoring by April 10, 2019.

Introduction

Stakeholders

Developers: Primary code writers. They will submit code to the pipeline. Easy of use is essential they want to submit and forget. A plus would be the ability to submit new changes while the last is still being tested with the pipeline just queuing the next code set to test.

DevOps Chapter: Team that automates processes between development and IT operations. Their goal is to improve the relationship by simplifying how code is built, tested, and released. There are many different tools for DevOps teams to use, but many are difficult to configure or don't have a complete pipeline with build agents, testing, manual gates, and deployment all together. DevOps teams need a tool that is easily configurable, and contains all parts of a pipeline service. This will help them to be more effective, and be able to set up a simple process for the developers to deploy code.

Testing Automation Team: They maintain an in-house developed testing framework, write the tests, and are responsible for making sure the tests are run. Their problem is to have to manually test things. They need our product to be able to integrate with this testing framework so that tests can can be kicked off automatically.

Management: The ones in charge of resource allocation from personal to finances. They need an easy simple way to get statistics to know that resources are being used efficiently and if more or less need to be allocated.

Manual Testers: Testers that follow a sequence of tests on a physical phone to test code. This is an integral part of the testing phase of the pipeline. They need a manual gate to stop the code from deploying until they have performed their tests.

Market Research

Our project is based on building a Continuous Integration(CI) and a Continuous Delivery (CD), environment. Several researches were made to understand the market, concept and tools available to understand how this project should be done. Some methods used for this research was pinterviews, competitive products and understand the constraints of this project.

We noticed that for this type of project there is no one fit all tool that can be used. It is necessary to integrate several tools to accomplish the enterily of the project. Some area and different tools where identified below, with some of their strengths and weaknesses. Some of the areas identified were:

- Source Control
    - Github - Most commonly used. No pipeline
    - Bitbucket - Have a pipeline
- Configuration Management
    - Puppet - Older Tool, harder to setup, have to know Ruby (programing Language)
    - Chef - Easy to use, lack some functionalities
    - Ansible - Easy to use, easier to program with
    - SaltStack - Easy to use, easier to program with
- Pipelines tools (Testing, Integration)
    - GoCD – Good interface, manual gates, OpenSource, No CI, Configuration as code, expensive support.
    - TeamCity – Simple, well packaged, lots of options, Expensive as it scales up, limits number of build agents.
    - Jenkins – Open source, plugins w/ documentation, has support for build pipelines, Dedicated server, Setup.
    - Travis CI – Cloud Hosting, Build matrix feature, multiple testing env, Limited customization, Costs enterprises.
    - Drone.io - Specialized for containers, out of the box functionality, Expensive, not very flexible.
    - Circle.ci - Really easy to use, has a 30 days free trial.

To better understand what is used by people on the job market we interviewed two Senior developers. One that work at SaltStack and one that works at observepoint. We noticed by talking to these Senior engineers that, there seems to have a preference for some of the tools, one of them is “Jenkins”, used as a pipeline tool. However most of the time people use tools that are their preference. The SaltStack engineer prefers using “SaltStack” as a configuration management tool and the ObservePoint engineer prefers “Ansible”. There are some bias for the tools being used at the moment, even though some are better then other, there are preferences when choosing one of the tools to use. Some more research and prototypes will have to be done in order to better decide which tool would be the best to use in our project.

The more we research and ask, the more we understand our constraints. One of them is choosing the correct tools to use. Another is to understand the scope of this project. By talking with our Sponsor there are some requirements for this project such as using docker container, having manual gates and using AWS as the environment of deploys. However we have to understand better our scope and how big this project has to be in order to better represent a solution for our sponsors company. By understanding the scope, and what is wanted from our sponsor we can better create an environment where are CI/CD can be reflected asn our sponsors CI/CD, and our solutions be implemented in his company. Our Sponsor's company needs some solutions and we can solve that by trying to implement different tools and striving to achieve our sponsor requirements.

Requirements

In order to meet the needs of the stakeholders there are a few key features that need to be included:

Developers:

- Need a simple and adaptable method to test and deploy code that they don't have to worry about managing.
- Should be able to create a pipeline with their code and not have to worry about anything after that except sending code to the pipeline.

DevOps:

- Allow for pipeline to be easily set up through configuration files, with option for manual and automatic gates.
- Deploy to AWS in containers to allows for easy scale and management.
- Has some method to rollback failed code to prevent outages and bugs from reaching end users.

Management:

- Have some form of monitoring to identify problems and justify resources.

Automation Test Team:

- Be able to integrate with the pre-existing testing framework to be able to start tests and receive results.

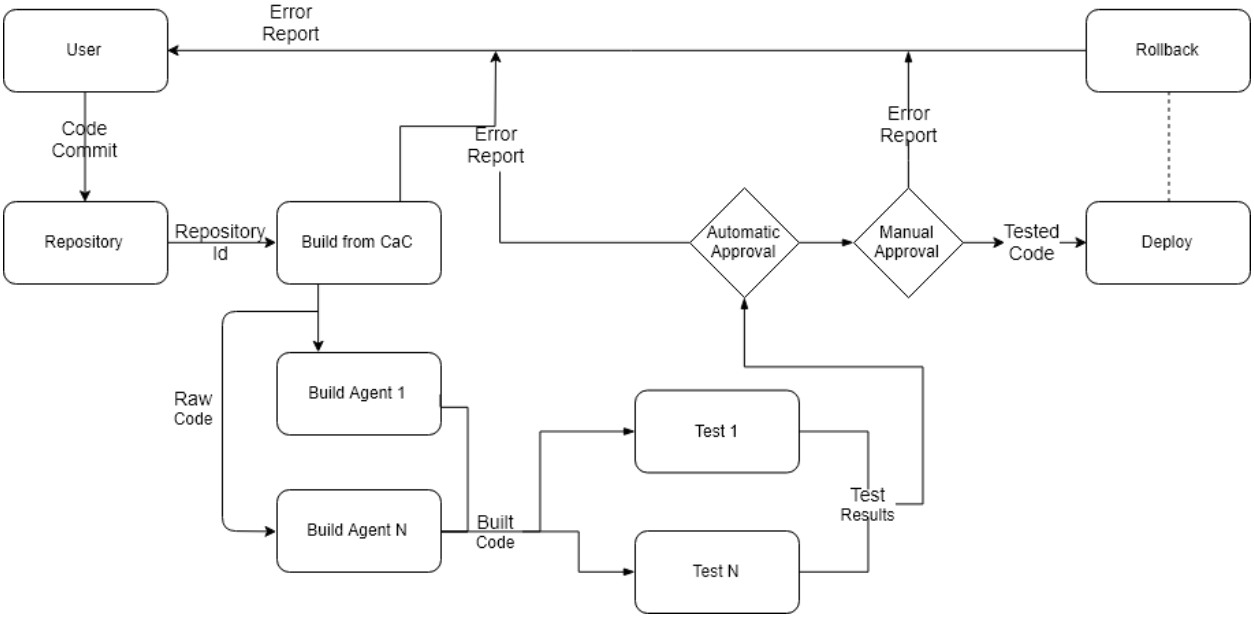
Manual Testers

- Product needs to have easy to access alerts and an interface suitable for interaction with non-technical people.

All:

- Integrate with the team's communication tool to report when pipelines are completed.

Logical Diagram



Description: The logical diagram shows the expected process for a user submitting code to be deployed to a server. This diagram is the default pipeline that would be built for developers, but the sections between ‘Build from CaC’ and ‘Deploy’ are dependant upon the CaC file that the user submits.

When a user submits code to the repository, the Build server gathers the code stored in the repository, and builds a program based on the CaC file and Dockerfile provided. It will then run through the tests provided or connect to an external testing framework to test the code. Also provided are automatic and manual gates so that developers can verify code is successful before complete deployment.

Error reporting and the ability to easily rollback are critical in this rapid deployment strategy. In the case of any issues, the Build server can rollback changes if an error is found in the deployment, build, or testing stages.

Prototype / Preliminary Implementation

c. Add new section about the prototype / preliminary implementation, and what questions it answered

We were able to build a prototype that integrated multiple tools to create what is called “Continuous Integration and Continuous Delivery” (CICD). The tools that we used were GitHub, Jenkins, Amazon Web Services (ECS and EC2).

The Goal was to push a commit to Github and automatically trigger builds and deploy to AWS on a live environment. We were able to reach this goal by creating a webhook that

connected Github to Jenkins. This would send a notification to Jenkins whenever a new push was done to Github. Jenkins then builds the code and Deploys it to AWS.

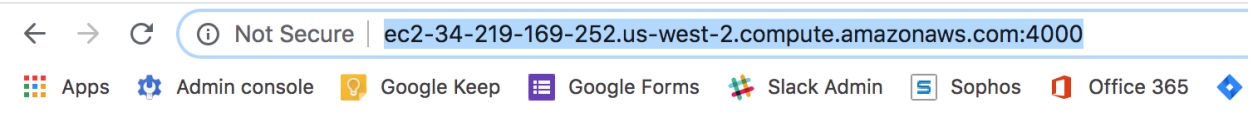
Even though the Demo was pretty simple, we were also able to add some extra features. These features included; building a pipeline to deal with multi-steps and multithreading steps; providing a visual representation of the build, and we were also able to create a simple docker container through Jenkins. This docker image is stored in AWS ECS, and run on a live Server on AWS EC2 (http://ec2-34-219-169-252.us-west-2.compute.amazonaws.com:4000).

This demo/prototype helped us to understand better the scope of this project. We were able to answer questions about how we were going to use GitHub and Jenkins together through the use of a Webhook. It helped answer a question that we had about developing our own web GUI as the one provided doesn't meet the needs/desires of our sponsor.

Along with answering some questions this demo/prototype has however, brought to light some questions that we didn't previously have:

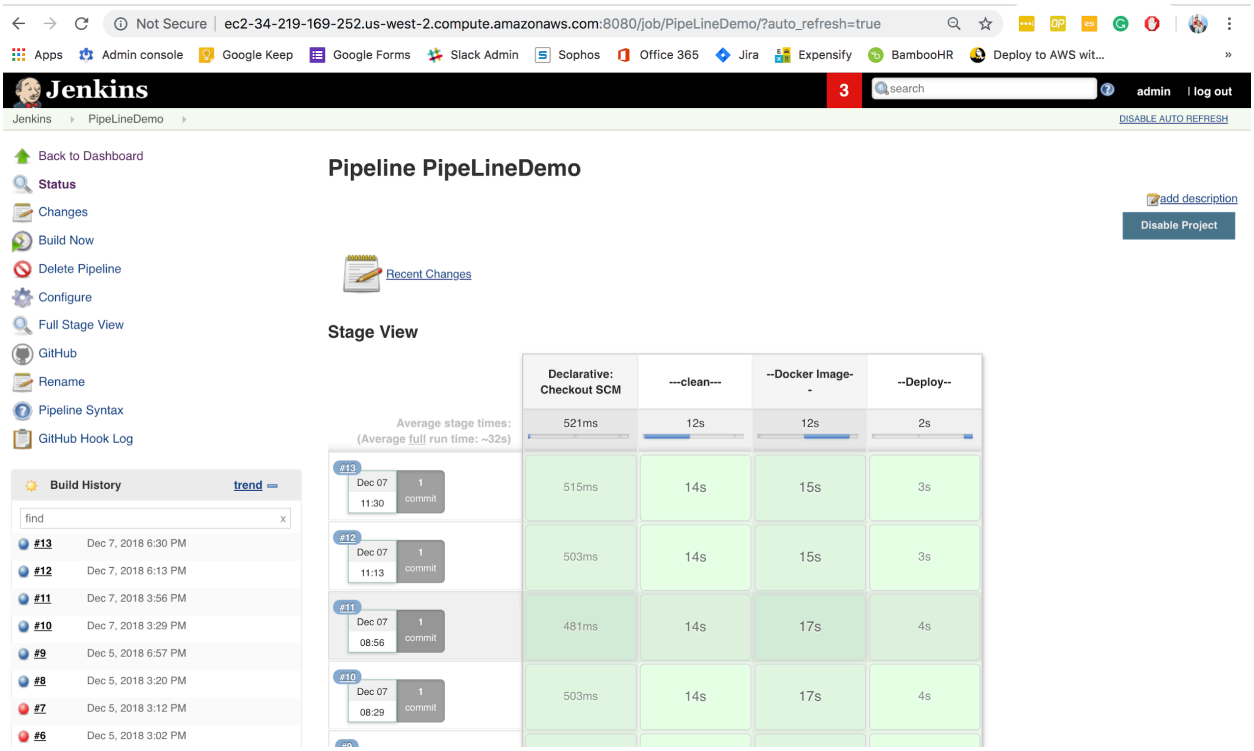
- Which tools could we use to better manage our docker images?
- How to make a reliable open source project?
- What other features could we add?

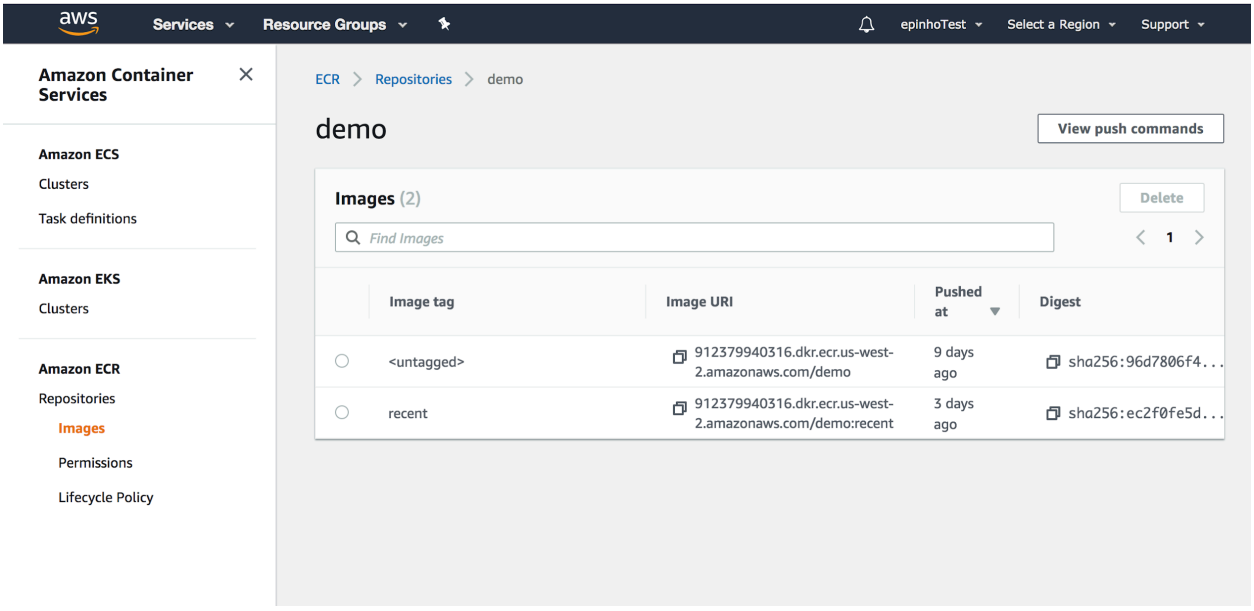
These questions, even though they are not answered, will be just as helpful to our project as the questions that have been answered through building this demo/prototype.

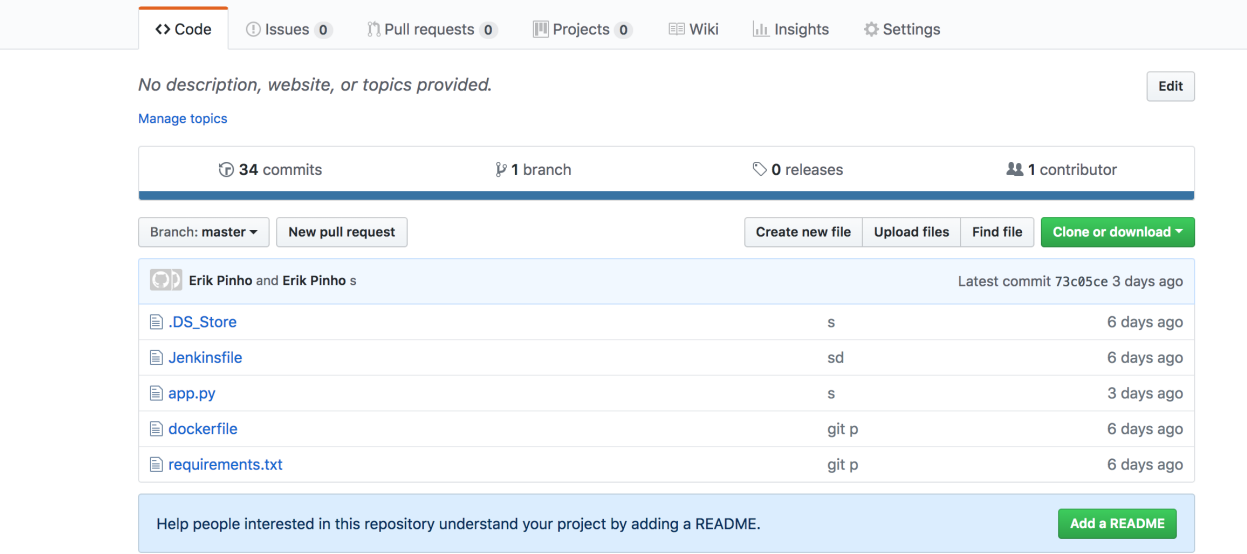


ShipIt World!

Hostname: 2469ae4ce169
Visits: *cannot connect to Redis, counter disabled*


Not Secure | ec2-34-219-169-252.us-west-2.compute.amazonaws.com:8080/job/PipeLineDemo/?auto_refresh=true
Apps
Admin console
Google Keep
Google Forms
Slack Admin
Sophos
Office 365
Jira
Expensify
BambooHR
Deploy to AWS wit...
Jenkins
3
search
admin
log out
Jenkins
PipeLineDemo
DISABLE AUTO REFRESH
Back to Dashboard
Status
Changes
Build Now
Delete Pipeline
Configure
Full Stage View
GitHub
Rename
Pipeline Syntax
GitHub Hook Log
Build History
trend
find
#13 Dec 7, 2018 6:30 PM
#12 Dec 7, 2018 6:13 PM
#11 Dec 7, 2018 3:56 PM
#10 Dec 7, 2018 3:29 PM
#9 Dec 5, 2018 6:57 PM
#8 Dec 5, 2018 3:20 PM
#7 Dec 5, 2018 3:12 PM
#6 Dec 5, 2018 3:02 PM
Pipeline PipeLineDemo
add description
Disable Project
Recent Changes
Stage View
Declarative: Checkout SCM
---clean---
--Docker Image--
--Deploy--
Average stage times:
(Average full run time: ~32s)
521ms
12s
12s
2s
#13 Dec 07 11:30 1 commit
515ms
14s
15s
3s
#12 Dec 07 11:13 1 commit
503ms
14s
15s
3s
#11 Dec 07 08:56 1 commit
481ms
14s
17s
4s
#10 Dec 07 08:29 1 commit
503ms
14s
17s
4s


aws
Services
Resource Groups
epinhoTest
Select a Region
Support
Amazon Container Services
Amazon ECS
Clusters
Task definitions
Amazon EKS
Clusters
Amazon ECR
Repositories
Images
Permissions
Lifecycle Policy
ECR > Repositories > demo
demo
View push commands
Images (2)
Delete
Find Images
1
Image tag
Image URI
Pushed at
Digest
<untagged>
912379940316.dkr.ecr.us-west-2.amazonaws.com/demo
9 days ago
sha256:96d7806f4...
recent
912379940316.dkr.ecr.us-west-2.amazonaws.com/demo:recent
3 days ago
sha256:ec2f0fe5d...



Deliverables

Deliverables:

User receipts
Documentation - README, documentation repository
Software - Build agent, pipeline server, test environment, user interface
Demo - Working baseline code, practice demo
Training (physical/video) - Editing software, practice training, video
Kickstarter Trailer - Video, editing software

Schedule:

Phase 1: Establish Context (Completed)

- Understand requirements for successful project. (Completed)
    - Understand what user really wants. (Completed)
    - Establish what is and is not part of the project. (Completed)
- Understand competitive environment. (Completed)
    - What tools or resources are already available. (Completed)
    - What is unique that we can bring to the table. (Completed)
- Establish team rules. (Completed)
- Determine deliverables. (Completed)

Phase 2: Plan (In Progress)

- Set goals. (Completed)
- Layout schedule. (In Progress)

    - Decide when and how deliverables will be accomplished. (In Progress)
- Assign responsibilities to team members for parts of the project. (Completed)

Phase 3: Execute (In Progress)

- Build prototypes.
    - Low fidelity.
    - High fidelity. (Completed)
- Verify that prototypes will work. (Completed)
- Validate that prototypes are what users need.
    - Evaluate Jenkins UI compared to a custom built one
- Build Product
    - Test-case Kickoff Plugin
    - Slack Reporting
    - Manual and automatic gates
    - Cloud Formation Setup
    - Multiple Environment Deployment
    - Rollback Functionality
- Manage schedule.
- Collaborate team progress.
- Prepare to and present completed project.
    - Video (In Progress)
        - Storyboard (Completed)
        - Script (In Progress)
        - Initial Animations
        - Voiceovers
        - Final Assembly
    - Final Presentation Slides (In Progress)

Is/Is Not:

Is	**Is not**
Slack Reporting	Infrastructure
Monitoring UI	Built from scratch
Rollback on Failure	
Unit Test Kickoff	
Integration Test Kickoff	
Configurable as Code	
API for Integration	

Async build agents

Manual Gates

Automatic Gates

Multiple Environment Deploy

Deploys to AWS

Container-ized

First Slide - Concept and requirements

Matrix

	Most	Medium	Least
Time	X		
Money		X	
Scope			X

Governance Framework - Internal workings of the project. Using Trello.

Glossary

AWS - Amazon Web Services, Amazon's cloud computing service.

API - Application Program Interface, a method to make different programs be able to talk to each other easily.

DevOps- Development-Operations, a hybrid term to refer to someone who both rites code as well as works in deploying that code to various environments for use.

Container - Unit of software that packages up code and dependencies so that the code can be executed regardless of the underlying operating system running the code.

CaC - Configuration as Code: A way to write a pipeline as code instead of having to use a UI to create each pipeline.

Bibliography

AWS Documentation. (n.d.). Retrieved from https://docs.aws.amazon.com/index.html#lang/en_us

Docker Documentation. (2018, December 05). Retrieved from https://docs.docker.com/

Dotsway. (2017, June 09). AWS S3 Tutorial: Static Website with S3 Hosting. Retrieved from https://www.youtube.com/watch?v=EMXCIWW0x2o

Get started with Docker for Mac. (2018, December 05). Retrieved from https://docs.docker.com/docker-for-mac/

Getting Started with Amazon ECS - run containers in production. (n.d.). Retrieved from https://aws.amazon.com/ecs/getting-started/

Jenkins Documentation. (n.d.). Retrieved from https://jenkins.io/doc/

Laster, B. (2018, October 11). What is CI/CD? Retrieved from https://opensource.com/article/18/8/what-cicd

Lateef, Z. (2018, December 06). Jenkins Pipeline Tutorial: Introduction To Continuous Delivery | Edureka. Retrieved from https://www.edureka.co/blog/jenkins-pipeline-tutorial-continuous-delivery

Manivannan. (2017, January 02). Use Jenkins and AWS Code Deploy as a CI / CD Tool. Retrieved from https://www.youtube.com/watch?v=YM7DOpWiL0A

Pipeline. (n.d.). Retrieved from https://jenkins.io/doc/book/pipeline/

ProgrammingKnowledge. (2018, July 23). Jenkins Tutorial For Beginners 15 - Pipeline script from SCM Using Jenkinsfile in Github Project. Retrieved from https://www.youtube.com/watch?v=56jtwSrNvrs&t=105s

Ulfeldt, A. (2018, April 18). Deploy to AWS with Jenkins. Retrieved from https://www.lynda.com/Docker-tutorials/Deploy-AWS-Jenkins/672474/727833-4.html