Cloud Digital Repository Automation

Matthew Brockman, Chris Hill
CS4624: Multimedia, Hypertext, and Information Access
Virginia Tech, Blacksburg, VA 24061
Instructor: Dr. Edward Fox
Client: Yinlin Chen
5/2/2018
# Table of Contents

Table of Figures .................................................................................................................. 3

Table of Tables ................................................................................................................... 4

Abstract ............................................................................................................................... 5

Introduction ......................................................................................................................... 6
  Requirements ..................................................................................................................... 7

Design .................................................................................................................................. 8
  Tech Stack ......................................................................................................................... 9
  Implementation .................................................................................................................. 10
  Notes on AWS .................................................................................................................. 12
  Account Management .................................................................................................... 12

AWS Console Pipeline Walkthrough .................................................................................. 13

Costs .................................................................................................................................. 23

Lessons Learned ............................................................................................................... 25
  Problems ......................................................................................................................... 25
  Solutions ......................................................................................................................... 25
  Future work ..................................................................................................................... 26
  Closing Thoughts ........................................................................................................... 26

Acknowledgements ............................................................................................................. 27

References ............................................................................................................................ 28
Table of Figures

1. Representation of a Software Development Pipeline .................................................. 6
2. Different Stages within the Pipeline ........................................................................... 10
3. The Initial Pipeline Screen .......................................................................................... 13
4. Connecting a Github Account ...................................................................................... 13
5. AWS CodeBuild Selection Options ............................................................................. 14
6. AWS Build Project Configuration ................................................................................ 14
7. BuildSpec.yaml Specification ...................................................................................... 15
8. Caching Options and Service Roles ............................................................................ 15
9. Advanced Build Setting for AWS CodeBuild ............................................................... 16
10. Deployment Provider Selection ................................................................................ 17
11. Application and Environments for Beanstalk ............................................................. 17
12. Selected Environment Tier ......................................................................................... 18
13. New Environment Creation ......................................................................................... 18
14. Base Configuration for Environment ....................................................................... 19
15. Finalizing Deployment for both Application and Environment .............................. 20
16. CodePipeline Access using Services ....................................................................... 20
17. Final Review of the Pipeline ...................................................................................... 21
18. Final Pipeline format in AWS ................................................................................... 22
Table of Tables

1.1 Cost Analysis for CodeBuild...............................................................................................................23
Abstract

The Cloud Digital Repository Automation project uses AWS services to create a Continuous Integration and Continuous Deployment pipeline for the Fedora4 digital repository system. We have documented our process, services, and resources used so that the pipeline can be modified and expanded upon further.

This project is for our course Multimedia, Hypertext, and Information Access, which creates an automated deployment pipeline using AWS resources. The overall purpose of this project is to automate some of the more mundane yet essential aspects of building and deploying a codebase. Taking source code from a repository and updating based on the recent changes can be a hassle as well as manually time consuming. This process of updating and bug fixing source code is not a new concept but can be made easier if some and if not all of the building, testing, and deploying is done automatically. This project aims to help the Fedora4 development team by providing a baseline pipeline configuration that can handle updates to source and subsequently build, test, and deploy the new updates and changes.

Our project sets up an AWS pipeline that handles automatic deployment to a staging server in the cloud for Fedora4. We based our implementation of the pipeline to what was available for our access and made sure not to interfere with any existing Fedora4 code or resources. We used Amazon services such as CodePipeline, CodeBuild, and Elastic Beanstalk to create and format our automation process. Understanding and utilizing cloud automation is essential to future careers as software developers and this project aims to acclimate and understand AWS in that role with a focus on automated CI/CD.
Introduction

Continuous Integration (CI) and Continuous Deployment (CD) is at the forefront of modern software practices. Programming applications and coding projects are developed and maintained by a group of developer’s who update and write code to suit client needs or improve the product incrementally. Continuous integration with a project aims to collaborate all developer’s work to a main or master branch and essentially integrating the day’s work together. This CI in combination with Continuous Deployment allows for any application to be in a state of release and keeping software up to date with the latest changes to the code.

The concepts of CICD aim to help a software development team and automate the building/compiling, testing, and deploying of an application. The application can be passed through a pipeline of stages for these steps in the software delivery process. Automation of any or all of these steps drastically affects the amount of human interaction for any code change in the repository. This means that developer’s can spend more time designing, programming, and debugging, rather than manually testing and checking results of the most recent build or checking that the latest version deploys to production. Automatically checking for a change in the repository is the start of this process for most applications.

Figure 1 - Representation of a Software Development Pipeline

The future of software development revolves around integrating bug fixes and new features being delivered and deployed as soon as possible. By automating these some or all of these steps, much of the process for building, testing, artifacting, and deploying new versions of software becomes seamless and simply requires minimal human interaction. In this way, any problems can quickly be sought out, fixed and
automatically tested, and releasing new software becomes almost completely automated.

Handling this process for the Fedora4 team is hopefully helpful and also intuitive when the pipeline is then passed on to them because it lays important groundwork for the configuration of their deployment server in an AWS cloud environment.

Requirements

The following requirements for this project:

- Using Amazon Web Services for an automated pipeline (AWS)
- Automated Deployment Pipeline of Fedora 4 to a staging server
  - Github change triggers the following
  - Build using Maven 3 on AWS resources
  - Deploy to a cloud server (AWS EC2 instance)
- $500 AWS credit for the budget
- Documentation of steps, accounts, files in order to recreate and reconfigure this process for any updates and changes
Design

The Project deliverable is an automated continuous integration (CI) and continuous delivery (CD) release workflow for an open source digital repository software - Fedora 4 using Amazon Web Services (AWS). A working CICD deployment staging server for Fedora 4 allows for the Fedora to maintain a streamlined finished product for both clients and development team alike. Much of the pipeline is aimed at using AWS and multiple plugins can be found for any existing additions to it, like a Jenkins plug-in for AWS\textsuperscript{[1]}.

AWS CodePipeline\textsuperscript{[2]} serves as an overarching backbone that can utilize and automate the other AWS services within itself. It is built upon a series of inputs and outputs between each of the stages.

Elastic Beanstalk\textsuperscript{[3]} is a service that offers EC2 instances based on the necessary resources for a certain web application. This service automatically scales based on the requested resources or the amount from the web app. There is a reduction in complexity because the Beanstalk will handle more detailed aspects of load balancing, scaling, and application health. It also supports applications developed in Java, as well as

AWS Simple Storage Service (S3)\textsuperscript{[4]} serves as an intermediary storage space for artifacts in each stage in the pipeline. Each input and output of the Pipeline can be managed by this simple repository and provides access to the data throughout all of AWS, as long as permissions are given.

Other options we looked into on AWS were CodeDeploy and Opsworks. These tools are good options to look at in terms of configuring a testing, pre-prod, and production server. CodeDeploy handles deployment on existing EC2 instances that are already setup and configured for the production environment. Opsworks acts a tool to configure EC2 instances using either a Chef recipe approach or Puppet. Considering this was our first time deploying, we opted to use Elastic Beanstalk. However if interesting in deploying and configuring a fleet of servers, OpsWorks and CodeDeploy may be of use.

\begin{footnotesize}
\begin{enumerate}
\item "GitHub - jenkinsci/pipeline-aws-plugin: Jenkins Pipeline Step Plugin ...." \url{https://github.com/jenkinsci/pipeline-aws-plugin}, Accessed 1 May 2018.
\item "AWS CodePipeline | Continuous Integration ...." \url{https://aws.amazon.com/codepipeline/}, Accessed 1 May 2018.
\end{enumerate}
\end{footnotesize}
Tech Stack

- Github
- Amazon Web Services
  - AWS CodePipeline
  - AWS CodeBuild
  - AWS Elastic Beanstalk
    - AWS EC2 Instances
  - AWS Dashboard
- Maven

Source code: Fedora 4 Github repository
Build tool: Maven 3 with AWS CodeBuild
Deployment Server: EC2 instance encompassed in Elastic Beanstalk with Tomcat
Implementation

Implementation involves using 3 main stages within the CodePipeline. These consist of pulling source code from our forked Fedora4 github repository, building and testing Fedora4 then outputting an artifact into a S3 bucket, and finally onto the Elastic Beanstalk.

Since the Fedora4 repository is public, the option to fork was offered to us to add files for AWS services to use for an automatic deployment. The use of Amazon’s S3 bucket is necessary because of the permissions and security roles built in to both AWS services and the EC2 instance(s) within the Elastic Beanstalk. In order to retrieve the build artifact in between stages, the easiest and most effective way is to use S3 buckets that can also be used as an artifactory that can timestamp a build. This artifactory can be used to rollback to previous versions or debug a client’s specific version. Each build can be set to have a timestamp or be configured further with AWS CloudWatch.

The changes made to the forked Fedora4 repository include a BuildSpec.yml file that is used when AWS triggers a release and starts the pipeline based on the Github webhook. AWS CodeBuild can be supplied direct commands to run in a specified build environment where the BuildSpec.yml specifies what commands to be run.

Deployment to the Beanstalk is handled by passing the artifact from the specified bucket and having an .ebextensions folder within the directory of the zipped file passed to the Beanstalk. Any configuration steps for the environment and other software configurations should be handled in this folder with the use of .yml or .json files. There is a great AWS guide[^5] that specifies how these files should be formatted and used for a software environment like Fedora4. Essentially any packages that need installing or setup commands that need to be run on the Beanstalk will be placed here. Any and all configurations that need to be done to the instance or instances within the Beanstalk should be placed in this folder with the correct file ending. These Elastic Beanstalk extensions[^6] are found more in-depth and can be configured to suit the type of environment.

---


of staging server. This includes any specific file commands that need to be run or any file manipulation that occurs. It is interesting to note that the Beanstalk can restart based on a list of files that change. Once a configuration is set and functioning, make sure to save the configuration so that this environment can be saved and used for other applications.

The addition of the .ebextensions folder should be made in the pom.xml file to be included. There are 2 ways of passing the extension config files to the Beanstalk, manually extracting them once the instance(s) are up, or passing Beanstalk a .zip file containing both the 5.0.0-SNAPSHOT.war and the .ebextensions folder. Using the latter approach is the only way of making this process completely automated, to avoid ssh into the Beanstalk to manually configure files. These extensions act almost as a Dockerfile for the Beanstalk but must be added and configured in the build phase. In the BuildSpec.yml file extra commands can be given to combine the .war and .ebextensions folder.

Ultimately an .ebextensions folder can be further configured to address any specific Fedora4 needs or changes, but there is a single config file based on the Dockerfile that specifies the commands run on a preliminary configuration for the Beanstalk. Beanstalk needs a .zip file that contains both the .war and .ebextensions. This should be modified in the Pipeline overview such that the input to the Beanstalk is a .zip file so Fedora4 configurations can be passed to it.
Notes on AWS

Initially an error was due to not putting the correct path name to the artifact within the codeBuild account. If data is stored onto AWS S3 buckets like we have used, make sure that the artifact is stored on the same bucket that the pipeline is being built in.

For example if the bucket is in aws-us-east-2, do not create the pipeline in aws-us-west-1 because the pipeline will not be able to see any of the codeBuild or S3 buckets in another region.

Account Management

- user/ IAM account number: 099214287868
- IAM account name: rock_mjb
- Password: TunaMelt123

This account manages all the resources created for the AWS Pipeline. These are the resources needed for the automatic deployment to a Beanstalk and the URL for the beanstalk currently is located at:

http://trial7.us-east-2.elasticbeanstalk.com:8080/fcrepo

A sample .tar file is located at:

https://s3.console.aws.amazon.com/s3/object/fedora4configs/beanstalk_configs/fedora_w_config.zip

The .zip file should be modeled like above in terms of directory structure in order for Elastic Beanstalk to read and recognize the extensions folder with configuration inside.
AWS Console Pipeline Walkthrough

This is meant to provide help initializing a Fedora 4 pipeline using the AWS console.

Figure 3 - The initial pipeline screen

Figure 4 - Connecting a github account with AWS allows for automatic webhooks based on a git push to the repository given.
Figure 5 - This is build selection. Among the options here are Jenkins, AWS CodeBuild, and Solano CI. The CodeBuild option is the most straightforward for this. If a BuildSpec.yml file is added to the github repository, there are many options to determine build commands and environments.

Figure 6 - This is where the build process can be specified and configured for CodeBuild.

---

"Build Specification Reference for AWS CodeBuild - AWS CodeBuild."  
Figure 7 - Here is where a BuildSpec.yaml file can be used to specify any and all commands/build environment variables. Generic OS and runtime languages can be specified here as well.

Figure 8 - Here is where the option to cache the dependencies of a build can be specified at a specific S3 bucket to save on build time. Service role is to ensure that the the build can access the correct AWS resources it needs to build, ie here you would give a service role that has access to a cache bucket.
Figure 9 - These are the advanced settings for AWS CodeBuild. Any Virtual Private Cloud network can be specified for this build. A timeout can be set to make sure that the build time doesn't take up too many resources. The average build time with tests was around 12 min and without Maven unit tests, 7 min.

Remember to save this build project so that if another pipeline is made it can use the same configurations.
Figure 10 - The Deployment provider is selected as Elastic Beanstalk. And the creation of an application and an environment is prompted.

Figure 11 - This is a created application without an environment named “FinalApplication.” Creating an environment can be made by “Create one now.”
Figure 12 - The selected environment is web server because a .war file is being deployed.

Figure 13 - This specifies the new environment name and description for the specified application.
Figure 14 - This specifies the Tomcat servlet that is used for Fedora4. Upload the code from an artifactory, and in this case, we used S3 as the artifactory so providing a .zip file after CodeBuild would be optimal defined in the BuildSpec.yml file.
Figure 15 - Finalizing Deployment for both Application and Environment after they have been created. Ensure that the application being uploaded is targeted wherever the CodeBuild output bucket is specified.

Figure 16 - Let CodePipeline Service be allowed to access both the CodeBuild and S3 buckets used by this pipeline.
<table>
<thead>
<tr>
<th><strong>Review your pipeline</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Source Stage</strong></td>
</tr>
<tr>
<td><strong>Source provider</strong></td>
</tr>
<tr>
<td><strong>Repository</strong></td>
</tr>
<tr>
<td><strong>Branch</strong></td>
</tr>
<tr>
<td><strong>Build Stage</strong></td>
</tr>
<tr>
<td><strong>Build provider</strong></td>
</tr>
<tr>
<td><strong>Project name</strong></td>
</tr>
<tr>
<td><strong>Staging Stage</strong></td>
</tr>
<tr>
<td><strong>Deployment provider</strong></td>
</tr>
<tr>
<td><strong>Application name</strong></td>
</tr>
<tr>
<td><strong>Environment name</strong></td>
</tr>
<tr>
<td><strong>Pipeline settings</strong></td>
</tr>
<tr>
<td><strong>Pipeline name</strong></td>
</tr>
<tr>
<td><strong>Artifact location</strong></td>
</tr>
<tr>
<td><strong>Role name</strong></td>
</tr>
</tbody>
</table>

Review your pipeline: We will create your pipeline with the following resources. Source Stage: Source provider is GitHub, Repository is Breezus/tcrepo4, Branch is master. Build Stage: Build provider is AWS CodeBuild, Project name is FinalBuild. Staging Stage: Deployment provider is AWS Elastic Beanstalk, Application name is FinalApplication, Environment name is Finalapplication-env. Pipeline settings: Pipeline name is FinalPipeline, Artifact location is s3://codepipeline-us-east-2-457633848065/, Role name is AWS-CodePipeline-Service.

Figure 17 - Final Review step before Pipeline creation.
Finished Pipeline

The finished pipeline is now deployed and the CodePipeline on the AWS account looks like with manual and automatic deployments.

Figure 18 - Final Pipeline format in AWS
Costs

Build Time is the most important factor in this situation. It costs 1$ per pipeline and also dependent upon other AWS resources used by this pipeline (i.e. EC2 instances within Beanstalk and CodeBuild minutes). S3 storage charges are miniscule and the heaviest charge is based on CodeBuild. CodeBuild charges by build minutes, essentially based on how long the maven install command takes. Roughly 7 minutes without tests and 12 minutes with tests.

Table 1.1 - Cost Analysis for CodeBuild

<table>
<thead>
<tr>
<th>Compute instance type</th>
<th>Memory (GB)</th>
<th>vCPU</th>
<th>Price per build minute ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>build.general1.small</td>
<td>3</td>
<td>2</td>
<td>0.005</td>
</tr>
<tr>
<td>build.general1.medium</td>
<td>7</td>
<td>4</td>
<td>0.010</td>
</tr>
<tr>
<td>build.general1.large</td>
<td>15</td>
<td>8</td>
<td>0.020</td>
</tr>
</tbody>
</table>

7 min * $0.005 = 0.035 cents per build without testing
12 min * $0.005 = 0.06 cents per build

Instance use is dependent on how much traffic is outbound out of the Beanstalk and a calculator at: https://calculator.s3.amazonaws.com/index.html
This can help with determining exact costs for each staging server, testing, pre-prod, or prod.

Existing AWS Resources

All lasting AWS resources should be located in region = us-east-2
There exists a pipeline at: https://us-east-2.console.aws.amazon.com/codepipeline/home?region=us-east-2#/dashboard
There exists 3 CodeBuild builds at: https://us-east-2.console.aws.amazon.com/codebuild/home?region=us-east-2#/projects
There exists 2 Beanstalks at: https://us-east-2.console.aws.amazon.com/elasticbeanstalk/home?region=us-east-2#/applications
Timeline and Milestones

1/31: Meeting with client, understanding scope of the problem, addressing questions, identifying some requirements and understanding the budget.

2/4 - 2/5: Prep for Presentation 1. Formally write down and understand requirements, present using visuals/slides in presentation, what is expected at delivery time

2/18: Ensure that client approves of AWS tool choice (i.e., CodeDeploy, CodePipeline), formulate design based on AWS research for CICD and best optimal solution for deployment

2/28: Collect research from the AWS user guide for CICD, note the current state of the Fedora CICD pipeline, and determine the system config for deployment and adjust for the current state

3/16: Prep for Presentation 2. Be able to explain the design and methodology of the preferred solution, strengths and flaws of the pipeline we create using certain technologies (AWS).

3/20: Begin testing with the pipeline to ensure that a deployment server can be created using EC2 instances. The staging server should contain the web-app for Fedora4 after passing testing.

4/15: Prep for Presentation 3. Explain the implementation of the pipeline, tools and technologies that were used, configurations of the pipeline system.

4/29: Prep for final Presentation. Synthesize information from requirements, research, design, implementation, and testing with visuals and diagrams explaining our thought process and procedures done.

5/1: Ensure that all documents and materials are in a submitted form/turned in to canvas and that the client has a version of the finished project
Lessons Learned

Problems

Much of our problems stemmed from choosing different AWS options within the suite of developer tools, as well as understanding and configuring them correctly. Researching the plethora of different tools available for deployment was simply incredible. AWS has 3 different tools that we could find that were specific for deployment, which were EC2 instances, CodeDeploy, and Elastic Beanstalk. Each of these tools came with a different set of features and steps for configuration. Choosing between them wasn’t easy because of the research that needed to be put into them. AWS has some very simplified tutorials for deploying a web application in the form of a .war file and the suggested tutorials never made any notion of configurations for the deployment server. This required extra research and experimentation with the available options for AWS like the .ebextensions folder.

Another problem we came across with, and this may be Amazon specific, was the CodeBuild builds, configurations, and histories being region dependent. This means that if the pipeline was being built on a different region server (i.e. us-east-1) and all of the functioning builds are working correctly but are located on another server (i.e. us-east-2), the pipeline would not have access to the builds. This caused some problems were upon a pipeline would not have access to a build configuration that was in a different region.

The final problem was understanding how Beanstalk actually uploaded a zipped codebase to it’s instance(s) and where to current working directories existed and operated in. The .ebextensions are slightly confusing to work with but operate almost the same as a Dockerfile. At first, Elastic Beanstalk looks like you can configure settings and files in the instance(s) controlled by it, but in reality you must use the config yaml and json files. There are no ways to automate a config setup other than using these files. If the environment cannot be setup correctly refresh the environment entirely and create a new environment as the crashed environment is unrecoverable if the .ebextensions are incorrect.

Solutions

One of the simpler solutions to the build region and pipeline region was by placing the pipeline and builds within the same AWS region. This was an overlooked solution because it was so simple yet so detrimental. The inability to reach built .zip or .war files was a major blockage for the pipeline as no output artifacts were reachable in another region.

A solution to deployment configurations that we found is the usage of the .ebextensions folder to place the commands needed to setup a configuration on the Beanstalk. This was a sought after fix for our problem because we do not want to manually configure a Beanstalk for every development or production change. The ability to save an Elastic Beanstalk configuration means that once completely configured, an application can quickly be spun up with the correct configuration.
Future work

This pipeline was meant to be a groundwork and foundation for the Fedora4 team so that they can integrate their current workflow into the automated pipeline. There are many different arrangements for any pipeline and the fact that they can all be automated and run from afar as distinct stages allows devOps to interchange and configure settings for the needs of development or production.

There are so many plug-ins and integrations with other DevOps services like Jenkins, Chef, Puppet, and Docker. AWS can combine many 3rd-party tools to further automate a pipeline.

And on a final note, the changes made to the pipeline can always be undone by modifying the process and configurations at each step of the way.

Closing Thoughts

This project was both eye opening and informative to say the least. Amazon Web Services is a huge player in the cloud computing industry today and the tools that they provide developers are ever-changing. Part of the challenge of this project was finding the correct tools to use for the pipeline. AWS offers many different tools aimed at certain aspects of cloud computing and using just a few of these tools and seeing how they operate and work with existing AWS frameworks is just incredible. This was a great introduction and project learning about how to use AWS for building, testing, and deploying.
Acknowledgements

Client: Yinlin Chen

Special thanks to our client Yinlin Chen to support us in our project and also to the Fedora4 team and their documentation on Fedora4 configurations which can be found at the DuraSpace wiki:\footnote{"Fedora Repository Home - Fedora Repository - DuraSpace Wiki." 9 Jan. 2017, \url{wiki.duraspace.org/display/FEDORA4x/Fedora+4.x+Documentation}. Accessed 1 May 2018.} Their maintenance of the documentation was a great help for formatting our ebextensions file.
References


AWS CodePipeline Guide:

AWS CodeBuild Guide:

AWS Elastic Beanstalk Guide:
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/awseb-dg.pdf

AWS CodeDeploy Guide:

AWS EC2 Guide: