Cloud Digital Repo Optimization

By Tom Fowler and Christian Howe

Completed at Virginia Tech in Blacksburg, VA 24061

CS4624 – Multimedia, Hypertext and Information Access

5/3/2018

Instructor: Professor Edward A. Fox

Client: Yinlin Chen

Table of Contents

1. List of Figures

2. List of Tables

3. File Inventory

1. clouddigitalrepooptreport.pdf and clouddigitalrepooptreport.docx

   This report.

2. clouddigitalrepooptpresentation.pdf and clouddigitalrepooptpresentation.pptx

   The slides used in our final presentation.

3. cloudformation.zip

   A zip bundle of the files used in the User Manual to deploy the template

   a. assets

      Files uploaded to an S3 bucket for deploying to ElasticBeanstalk

   b. params

      Our default parameters for deploying the templates

   c. templates

      The CloudFormation templates

4. Executive Summary

The goal of this project is to scale down the CloudFormation templates for deploying the Hyku digital repository application. We have attempted to reduce the cost of running the Hyku application with a base level of performance, essentially reducing it to the minimum viable scale. We have accomplished this by changing these templates and their configuration parameters to use less instances at smaller sizes. After evaluating a number of different options for reducing the base cost, including using other AWS offerings, we have settled on a number of parameters that work well at the base level of performance. In testing these changes, we used a qualitative method of testing the functionality of the existing feature set on the original deployment and comparing that to the functionality of the new deployment. We have seen no changes in functionality from the original deployment.

The cost reduction we see with these reduced instance sizes is to about one third of the original cost, resulting in massive savings given that the original cost of running the application was about $800-900 a month. The new cost of running our modified templates with the parameters we have tested is about $300 a month. Given that the original feature set is still functioning as it was before, we believe that we have achieved a satisfactory reduction of cost from the original deployment, and therefore have accomplished the goal we set out to complete.

We provide documentation on our process and the changes we made, including on how to reproduce in the future the changes we have made. Since the templates require some level of maintenance, this documentation is vital for deploying them in the future. The documentation provided by this report gives future maintainers the ability to quickly get up and running with the potential problems encountered when working with the templates, and gives future groups the insight to predict the kinds of challenges they will face when working on the Hyku CloudFormation templates.

5. Introduction

Fedora [1] is a digital repository platform for indexing and storing large digital libraries. It includes functionalities for storing files of any size with their metadata, and the ability for linking data via RDF. Fedora helps with preserving the digital repository content with fixity checking and an audit trail. It is produced by the community associated with Duraspace, which is a not-for-profit organization. It serves as a backend for digital repository solutions.

Hyrax [2] is a digital repository front-end from the Samvera community. It uses Fedora on the backend to provide a web interface to managing a digital repository, including adding content to the digital repository. It also allows convenient access to the digital content with Solr-powered searching of the digital repository.

Hyku is a project by the Hydra-in-a-Box team that extends Hyrax to support multiple tenants [3]. It includes an interface for creating new digital repositories that use the Hyrax front-end. In this way, an organization can create multiple digital libraries, each with their own set of users and files.

Amazon Web Services (AWS) is a platform providing a cloud hosting solution for servers, databases, and a variety of other services provided by Amazon, a popular e-commerce website. AWS CloudFormation is a technology for writing templates that manage AWS resources and services [4]. A set of AWS CloudFormation templates exist for the Hyku application [5]. Therefore, you can use these templates to create the AWS resources necessary to deploy Hyku to the AWS cloud.

The current deployment costs using these CloudFormation templates with their default parameters start at about $800-$900 per month, which is prohibitively expensive for small deployments. Our client, Yinlin Chen, has asked us to reduce the cost necessary for the deployment of Hyku using the CloudFormation templates.

In this document, the first topics we discuss are the details of our project, including the requirements and the thought process behind our solution. We then provide several resources to both users and developers. This includes deployment steps for users of our CloudFormation template setup and an analysis of the cost requirements at the base level, compared to the previous cost requirements. For developers interested in extending our work, we provide a discussion of how the templates are developed and some helpful tips for keeping the templates up to date. Finally, we review the process we went through to complete the project, including the problems we encountered and the solutions we developed.

6. Requirements

The main deliverable for this project is a new CloudFormation template that will create the stack in Amazon Web Services. This new template needs to be designed such that it reduces the cost of running the application with our given parameters but retains full functionality under small loads. Our goal is specifically to reduce cost at the base level of performance. Therefore, performance tests are out of scope, so long as the application functions appropriately with the lowest cost.

Our solution also needs to maintain a 24/7 uptime, so it needs failovers for disaster recovery and periodic updates to databases and instances. It should ensure the instances are restarted if they crash, to maintain a minimum number of healthy instances, and databases should have a standby in a different availability zone in case of database failure.

With this template, we will provide a CSV file from Amazon's simple monthly cost calculator to show the estimated cost of our template. This calculator is not always exact, but it is a reasonably accurate representation of the cost of Amazon's services. We use this to provide a cost comparison with the previous CloudFormation template.

We will also provide quality assurance on the application created by our stack. We will be using qualitative testing paradigms. We plan to use the original CloudFormation template to create an application stack and use our updated template to create our updated stack. We will then manually test each functionality of the application on both versions of the application and note any usability changes.

An important requirement for our application is documentation for using the templates and developing them. Since the application is very specific, regarding how it needs to be deployed, and there are many steps that aren't covered well in the existing documentation, we hope to expand on its usage. Additionally, the CloudFormation templates are often in need of maintenance, so providing documentation to handling this maintenance is vital.

7. Process

7.1. Definitions

AWS has many services that are used in the CloudFormation templates to deploy the Hyku application. These services are used to run the various servers and applications that Hyku depends upon. We will now briefly discuss each of the services used in the Hyku CloudFormation templates.

A fundamental service in AWS is the AWS Virtual Private Cloud (VPC), which is a service for provisioning an isolated virtual network in the AWS cloud. This provides added security and the ability to isolate your software from other people on the AWS cloud and the public internet.

Perhaps the most well-known AWS service is AWS Elastic Compute Cloud (EC2), which is a service for managing server instances in the AWS cloud [4]. It includes the ability to start servers, put them behind a firewall with a security group, create load balancers to distribute traffic evenly between your instances, and create auto-scaling groups to scale the number of instances up and down based on the load. EC2 also allows uploading Amazon Machine Images (AMI) that let you provide your own operating system and software to run on the instance. These facilities provide a foundational offering of the AWS cloud.

One AWS service that is used frequently throughout the Hyku CloudFormation templates is AWS Elastic Beanstalk. Elastic Beanstalk is a service which allows you to deploy applications to the AWS cloud without worrying about the infrastructure that runs them [4]. For example, you can deploy a Ruby on Rails application without having to create an AMI with your software to run on the EC2 instance. You can choose an Elastic Beanstalk Solution Stack that works with

the requirements of your application. Elastic Beanstalk has many such Solution Stacks for a variety of common applications.

AWS provides a number of services for the management of database instances, including many of their own custom databases. AWS Relational Database Services (RDS) is a service for provisioning and scaling relational databases that use SQL, the Structured Query Language [4]. One such relational database that RDS supports is PostgreSQL, a popular open source SQL database [6]. PostgreSQL is used as the primary database in the CloudFormation templates for deploying Hyku.

Another service provided by AWS for data management is AWS ElastiCache. ElastiCache is a service for deploying in-memory caches which is often used for ephemeral data to improve performance [4]. One in-memory cache that ElastiCache supports is Redis [7], a popular open source option, which is used in the Hyku CloudFormation templates.

AWS Simple Queue Service (SQS) is an AWS service for storing messages as they are passed between servers in a scalable manner [4].

Finally, AWS Elastic Container Service (ECS) is a service for scaling Docker containers on the AWS cloud [4]. Docker is a platform that allows you to package an application with all of its dependencies and run it in an isolated manner [8]. This makes it an excellent choice for application developers to be able to test locally with the entire environment that will be used in production. Elastic Container Service takes advantage of this to distribute Docker containers to run across EC2 instances in a balanced manner.

7.2. Existing Architecture

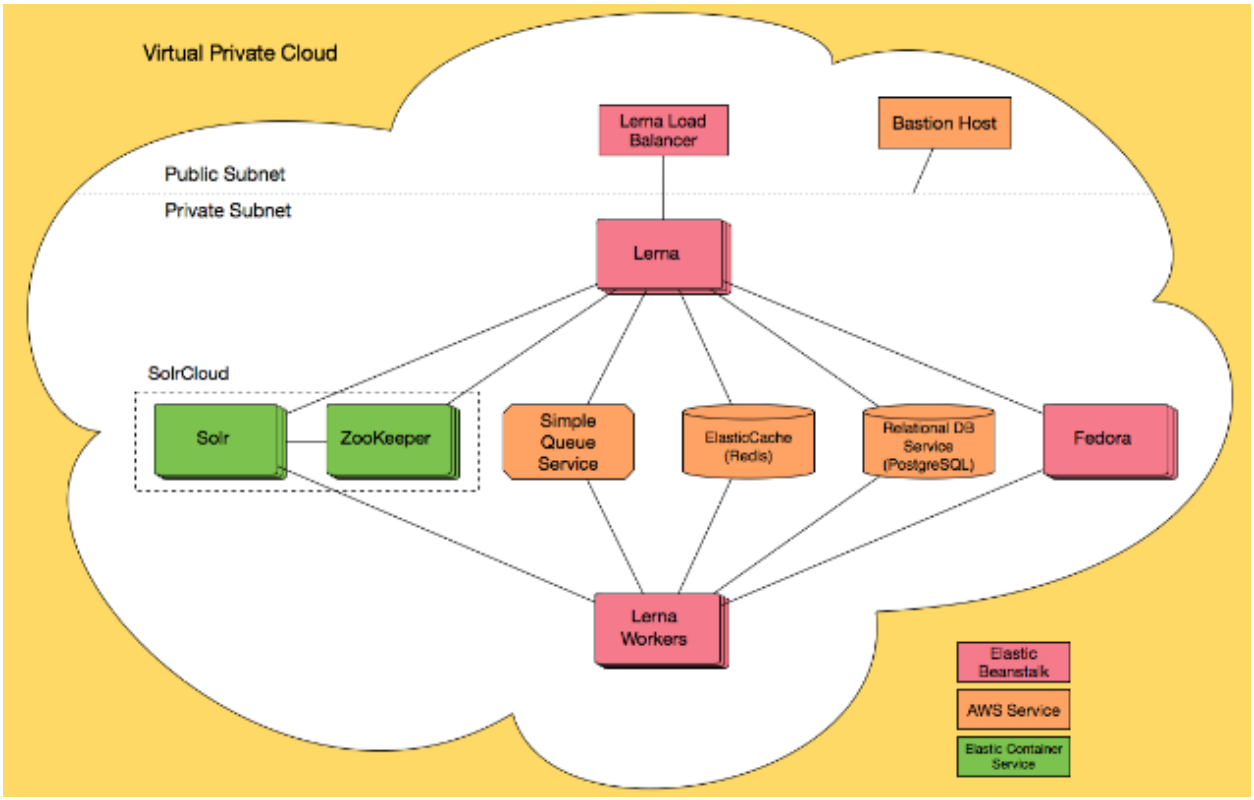

Figure 1 - Existing Architecture Diagram [5]

The Hybox architecture consists of many components hosted in AWS. All of these components exist in the same public cloud, with the load balancer being exposed to the internet for access to the frontend of the application. A bastion host is also exposed for accessing the private subnet for debugging and development purposes. In the private subnet, a Rails application called Lerna is run to host the frontend of the application, which is exposed through the load balancer. This Rails application accesses many services, including SolrCloud, Amazon SQS, Redis, PostgreSQL, and the Fedora digital repository. The Lerna application also runs a number of workers that access these same services.

The SolrCloud cluster is run under an Amazon ECS cluster, which provides a way for deploying Docker containers onto EC2 instances. This includes Solr, a service used for search

indexing, and a Zookeeper instance, which is necessary for the operation of SolrCloud and provides configuration management for SolrCloud. The Amazon SQS service, along with Redis running under Amazon ElastiCache, and PostgreSQL running under Amazon RDS, are managed by the AWS service, and don't require an additional service to run. Finally, the Lerna application and the Fedora repository run under Elastic Beanstalk, which is a service for simplifying deployment of services to EC2 instances.

7.3. Updates to Architecture

Due to the main goal of the design being to reduce cost of the base deployment of the Hybox CloudFormation templates, we start by looking at the cost of the existing deployment. The majority of the cost of the existing deployment is centered around a few major components, including the EC2 instances of the SolrCloud component, and the large SQL databases deployed. Therefore, we chose to focus our attention on these high-cost components.

Due to the large size of these EC2 instances, the EC2 portion of the application is the bulk of the cost. The parts of the application using EC2 are Fedora, SolrCloud (Solr and Zookeeper), and the Hyku application using Ruby on Rails. [5] We evaluated each of these parts to determine the correct size and quantity of each instance.

First we looked at the instance running Fedora. The existing application uses a t2.large EC2 instance, which provides 8 GiB of memory [9]. The minimum amount of memory required to run Fedora is 4GiB. t2.medium instances provide exactly 4GiB of memory [9]. Therefore, we scaled down the Fedora instance to be t2.medium. This change cuts the cost of this instance in half, from $0.0928/hour to $0.0464 per hour. [9]

Next, we looked at SolrCloud. SolrCloud is made up of two sections: Solr and Zookeeper. For production, you should have at least three Zookeeper nodes [10]; however, Solr can be used with less than three instances. Therefore, we chose to keep the three Zookeeper

nodes but reduce the number of Solr instances to a minimum of one, with a maximum of two. This lowers the base cost while maintaining some scalability. We attempted to reduce the size of the Zookeeper instances, but they had out-of-memory errors, so we set them to remain as is. We also reduced the size of the Solr instance from t2.large to t2.micro, which cut the cost from $0.0928/hour to $0.0116/hour. [9]

The remaining EC2 instances are the Hyku application and a worker to assist it. These do not have documentation on the amount of memory required, so we scaled them down first to t2.micro, then to t2.nano. The application failed due to out-of-memory errors at a t2.nano size, so we decided to increase the size back to t2.micro. Bringing the Hyku app from t2.large to t2.micro cut the cost from $0.0928/hour to $0.0116/hour, and reducing the worker from t2.medium to t2.micro cut the cost from $0.0464/hour to $0.0116/hour [9].

After reducing the size and quantity of the EC2 instances, we looked at the second largest cost in the overall application: the database instances. The two PostgreSQL databases were initially db.t2.medium. The minimum size for a PostgreSQL instance in RDS is db.t2.micro [11]. We were able to reduce the size of the databases to db.t2.micro without experiencing any performance issues. This change in size reduces the cost of each database from $0.073/hour to $0.018/hour [11].

Finally, we looked at the Redis cache. The initial size of the ElastiCache node was cache.m1.small. This is a previous generation node type, which costs $0.055 per hour [12]. We updated the template to use the smallest version of the current generation's node: a cache.t2.micro node [13]. The cost of this node is $0.017/hour [13].

Through individually reducing the sizes of these instances, we were able to create a much cheaper working architecture within AWS.

7.4. Options Explored

This section will discuss two more ways to update the architecture that we evaluated. Both methods, Amazon Aurora and AWS Fargate, would most likely have been able to replace a section of architecture, but during our cost evaluation, we found that they would be more expensive than our scaled-down architecture.

Amazon Aurora is a relational database service provided solely by Amazon that advertises higher throughput, lower latency, and lower costs than the equivalent PostgreSQL databases [14]. However, the smallest available Aurora instance is db.r4.large, which costs $0.29/hour [15]. The database instance we used in our solution was a db.t2.micro, which costs only $0.018/hour [11]. Had we needed databases of much larger size, Amazon Aurora may have been a more cost-effective solution. However, because our architecture has minimal performance requirements, an Aurora database was more expensive and higher-performing than needed.

The second update that we evaluated was moving several parts of the application to use AWS Fargate. AWS Fargate is a technology that allows the user to run applications that use containers without having to manage servers or clusters [16]. The Solr and Zookeeper instances are managed by Elastic Beanstalk, and are running as Multi-Container Docker Instances. In this configuration, the developer can define one or more containers in the CloudFormation template, and Elastic Beanstalk will run one of each container on each instance [17]. Figure 2 shows an

example Multi-Container Docker environment.

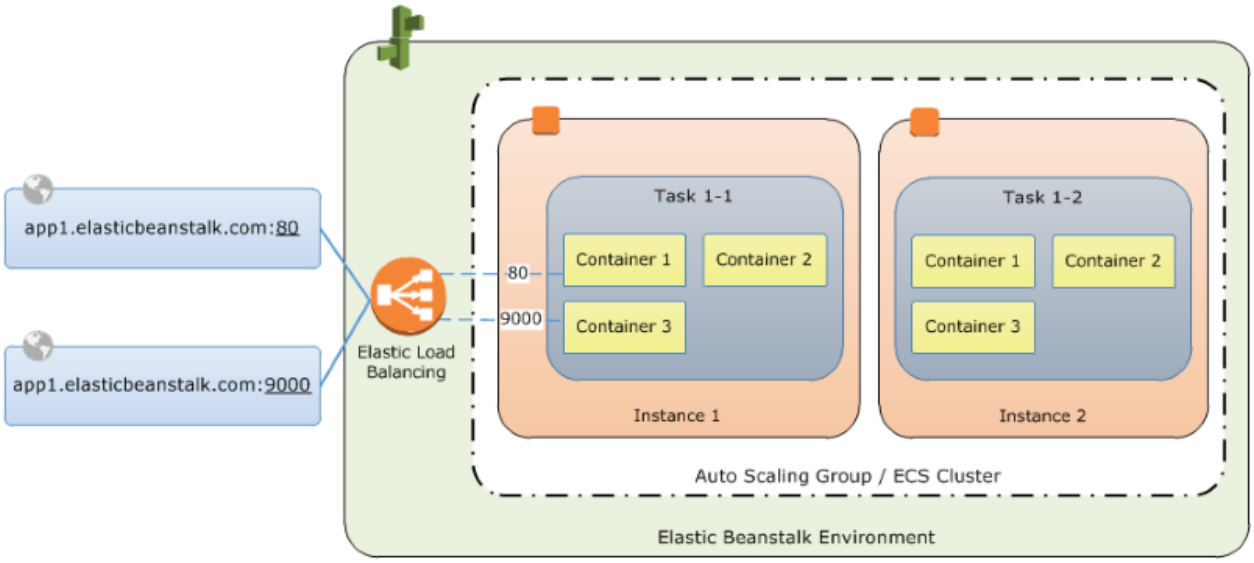

Figure 2 - Example Elastic Beanstalk Multi-container environment [17]

Our scaled down application is running exactly four containers: three containing Zookeeper and one containing Solr. Using Fargate would allow us to deploy these containers as a part of our application without creating server instances, as Elastic Beanstalk does. While this solution should theoretically work for deploying these containers, we chose not to do this because of cost. As discussed earlier, each of these containers is running on a t2.micro instance which costs $0.016/hour, providing one vCPU and one GiB of memory [9]. To compare the AWS Fargate pricing, see Table 1.

Table 1 - Fargate supported configurations [18]

CPU	Memory
0.25 vCPU	0.5GB, 1GB, and 2GB
0.5 vCPU	Min. 1GB and Max. 4GB, in 1GB increments
1 vCPU	Min. 2GB and Max. 8GB, in 1GB increments
2 vCPU	Min. 4GB and Max. 16GB, in 1GB increments
4 vCPU	Min. 8GB and Max. 30GB, in 1GB increments

The cost of each container is the sum of the vCPU requested and the memory resources requested. The rates are $0.0506/vCPU/hour and $0.0127/GB memory/hour [18]. From our testing while reducing the size of the EC2 instance, we know that Zookeeper and Solr need at least one GiB of memory. Even if they were to run on Fargate with a quarter of the vCPU resources provided by Elastic Beanstalk, the price to run each container would be 0.25*$0.0506+1*$0.0127=$0.02535/hour. Compared to Elastic Beanstalk's price of $0.016/hour, this is clearly not a cost effective solution in our situation.

8. Cost Assessment

The main goal of this project is the reduction of cost, and we will now discuss how our project achieves that goal and to what extent. We have determined that the cost reduction described is an acceptable baseline of performance and usability for the software in the Hybox repository. The cost assessments were made by entering each part of the solution stack into AWS Simple Monthly calculator [19]. Note that these costs were evaluated on 4/29/2018 in the US West 2 (Northern California) region, and costs can change based on date and region. The CSV files for the following tables can be downloaded at the associated links to the AWS Simple Monthly Calculator provided in the bibliography.

8.1. Original Cost Assessment

Table 2 - Initial Cost Breakdown [20]

Your Estimate				
Service Type	Components	Region	Component Price	Service Price
Amazon EC2 Service (US West (Northern California))				$587.33
	Compute:	US West (Northern California)	$479.90	
	EBS Volumes:	US West (Northern California)	$25.44	
	EBS IOPS:	US West (Northern California)	$0	
	Classic LBs:	US West (Northern California)	$81.99	
Amazon S3 Service (US West (Northern California))				$2.60
	Standard Storage:	US West (Northern California)	$2.60	
Amazon RDS Service (US West (Northern California))				$280.92
	DB instances:	US West (Northern California)	$278.16	
	Storage:	US West (Northern California)	$2.76	
Amazon ElastiCache Service (US West (Northern California))				$43.92
	On-Demand Cache Nodes:	US West (Northern California)	$43.92	
AWS Data Transfer Out				$4.41
	US West (Northern California) Region:	Global	$4.41	
AWS Support (Basic)				$0
	Support for all AWS services:		$0	
		Total Monthly Payment:		$919.18

The above table shows line-by-line the various expenses involved in running the current baseline deployment of the Hybox repository. This comes out to a monthly payment of $919.18. Most of that cost consists of the EC2 compute instances and the RDS database instances. These were the main focus of our reduction in cost.

8.2. Final Cost Assessment

Table 3 - Final Cost Breakdown [21]

Your Estimate				
Service Type	Components	Region	Component Price	Service Price
Amazon EC2 Service (US West (Northern California))				$206.36
	Compute:	US West (Northern California)	$106.13	
	EBS Volumes:	US West (Northern California)	$18.24	
	EBS IOPS:	US West (Northern California)	$0	
	Classic LBs:	US West (Northern California)	$81.99	
Amazon S3 Service (US West (Northern California))				$2.60
	Standard Storage:	US West (Northern California)	$2.60	
Amazon RDS Service (US West (Northern California))				$73.04
	DB instances:	US West (Northern California)	$70.28	
	Storage:	US West (Northern California)	$2.76	
Amazon ElastiCache Service (US West (Northern California))				$16.11
	On-Demand Cache Nodes:	US West (Northern California)	$16.11	
AWS Data Transfer Out				$4.41
	US West (Northern California) Region:	Global	$4.41	
AWS Support (Basic)				$0
	Support for all AWS services:		$0	
		Total Monthly Payment:		$302.52

Our final cost assessment takes note of all of the changes mentioned in the Process section of this report. The net monthly cost of running an instance of this application is reduced to $302.52 using our solution. This is about one-third of the original monthly cost [19] [20] [21].

9. Testing

9.1. Results

The client for this project requested that we do qualitative testing of this application, and said that quantitative testing is unnecessary for his purposes. The qualitative tests, as well as results for both the original and scaled down application, are given in Table 4. The tests were chosen based on the Hyku user manual and features overview [22].

Table 4 - Testing metrics and results

Metric	Original application	Scaled-down application
Home page loads in under 10 seconds	True	True
New repositories appear in DB	True	True
New users appear in DB	True	True
Repository labels appear for all users	True	True
Home page image uploads and appears for all users	True	True
Repository Admin can create and manage groups, collections, and works	True	True
Work creation saves metadata provided by user	True	True
Work creation uploads file to Fedora repository	False	False
Batch upload saves multiple files to repository simultaneously	False	False
Search returns relevant results within 10 seconds	True	True

You may notice that several of these tests fail. However, these tests fail on both the original application and our scaled-down version. After discussing the issue of uploading with the developer, we found out that this is a known issue in the application and concluded that our change in the architecture most likely did not cause this failure. Therefore, for our comparison, we found the failure of these tests to be acceptable.

9.2. Methods

Most of the tests in Table 4 are tested qualitatively within the application. The methods for these tests are mostly straightforward. Following the instructions found in the Hyku user manual [22], we create, upload, or edit various resources. Subsequently, we logged in as another user to observe that the changes have indeed occurred. There also are a couple of tests that involve connecting to the database instance and verifying that tables within the database have been updated.

To connect to the databases, we will start a secure shell session connecting to the bastion host, and then use the command line "psql" command to connect to and query the database. To connect to the bastion host, you will have to add an inbound rule to your bastion host's security group allowing SSH traffic on port 22 from your local machine's IP address. Then, you can use any ssh client to log into the bastion host using the private key specified in your params file. However, the bastion instance as configured by this template is out of date, so you will have to run two commands to install postgresql:

```
$ sudo yum update
$ sudo yum install postgresql
```

After these commands complete, the necessary commands will be installed on the bastion host. However, the security groups for the databases do not automatically allow the bastion host to connect to them. Go to the AWS EC2 dashboard. Under "Network & Security," select security groups. Find the security group with the name "hybox-bastion" and note its security group ID.

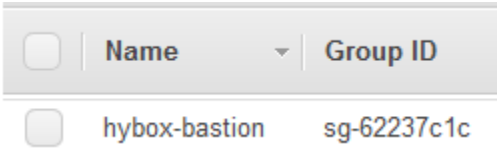

Figure 3 - Security Group within AWS console

Next, find the “hybox-db” and “hybox-fcrepodb” security groups. Add an inbound rule to each of them that allows traffic from port 5432 to those security groups.

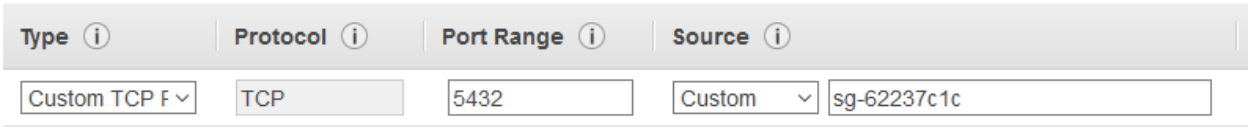

Figure 4 - New security group rule for hybox-db and hybox-fcrepodb

Now, on your bastion instance, you should be able to use the “psql” command to connect to each of the databases. For information on how to use the psql command to connect to RDS instances, read Amazon’s guide on Connecting to a PostgreSQL DB Instance [23]. The tables you need to inspect for the purposes of these tests are on the hybox-db instance, and are named “accounts” and “users”.

10. Users’ Manual

The intended user for this project is a system administrator who wishes to host a digital repository, Hyku, on Amazon Web Services. The best step-by-step reference explaining how to do this is on the project’s GitHub [5]. However, explanations of each necessary step are found below, with screenshots of the steps.

10.1. Configuring the CloudFormation Parameters

- Step 0: Choose an AWS region. AWS regions include:
    - US East (N. Virginia or Ohio)
    - US West (N. California or Oregon)
    - Asia Pacific (Mumbai, Seoul, Singapore, or Tokyo)
    - Canada (Central)
    - EU (Frankfurt, Ireland, London, or Paris)

- South America (Sao Paulo)

- Step 1: Create an EC2 Key-Pair in the chosen region
  - In the AWS EC2 dashboard, select the “key pairs” page.

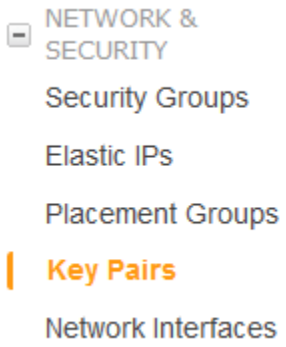

Figure 5 - Key pairs selection in EC2 dashboard

  - Select “Create Key Pair”, and enter a name for the key pair, then select “Create”

Figure 6 - Create Key Pair Button

Figure 7 - Choose Key Pair Name

  - Save the created .pem file, and remember the name of the key pair for later.

- Step 2: Create a hosted zone using Route53. A registered domain name is necessary for this step.

- Navigate to the AWS Route53 dashboard, select “Create Hosted Zone”, and enter the domain for which you are creating this zone in the subsequent screen.

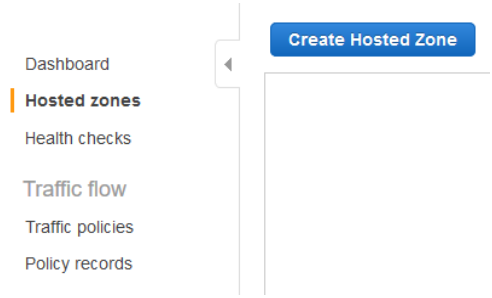

Figure 8 - Create Hosted Zone button

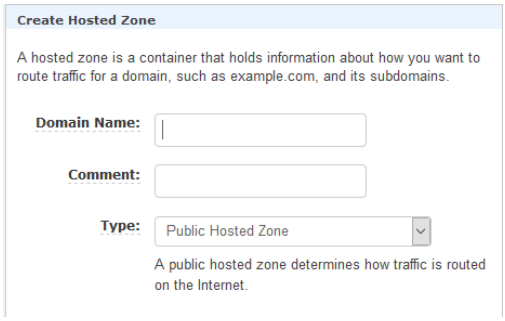

Figure 9 - Hosted zone settings screen

- Step 3: Create an S3 bucket to store binary content

- Navigate to the AWS S3 dashboard and select the “Create bucket” option. Enter the desired bucket name, and keep all other options default.

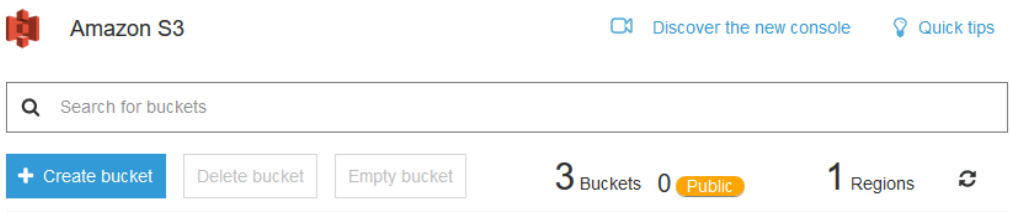

Figure 10 - S3 Dashboard with "Create bucket" button in lower left

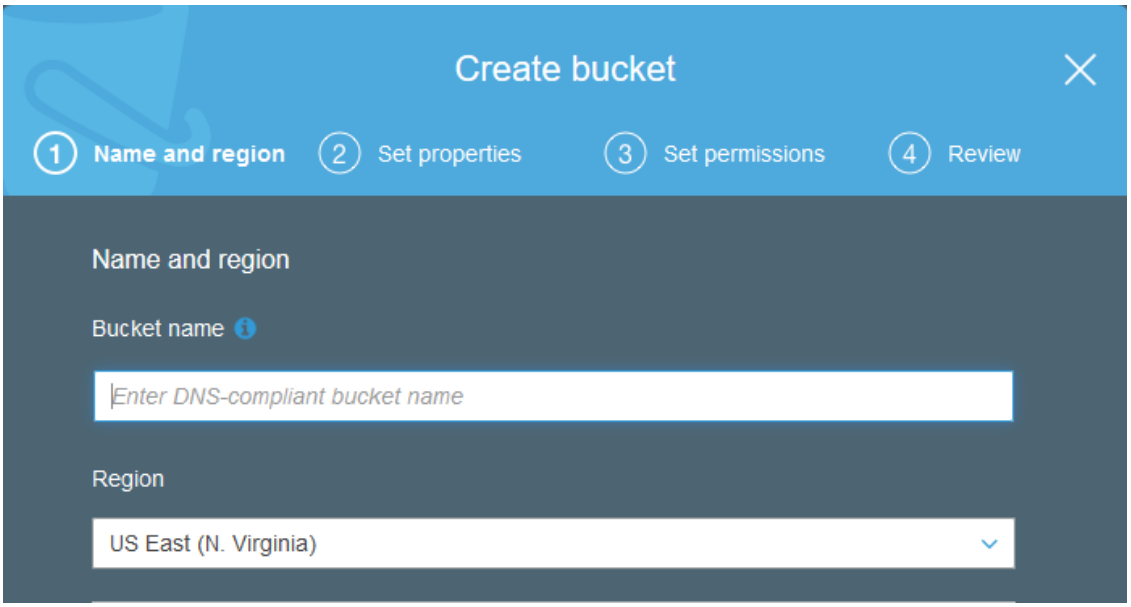

Figure 11 - Bucket settings screen

- Step 4: Create an IAM user with permission to access the aforementioned bucket.

- Navigate to the IAM dashboard, and select “Users” on the left-hand column, then select “New user”

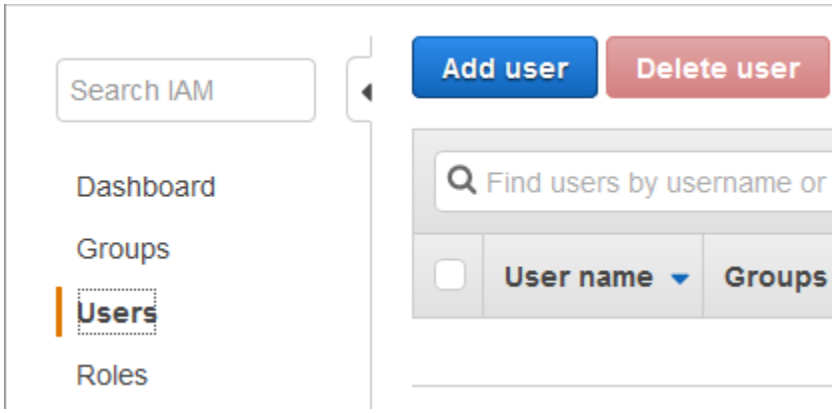

Figure 12 - IAM dashboard with user section and add user button.

- Enter the desired username, and give the user programmatic access

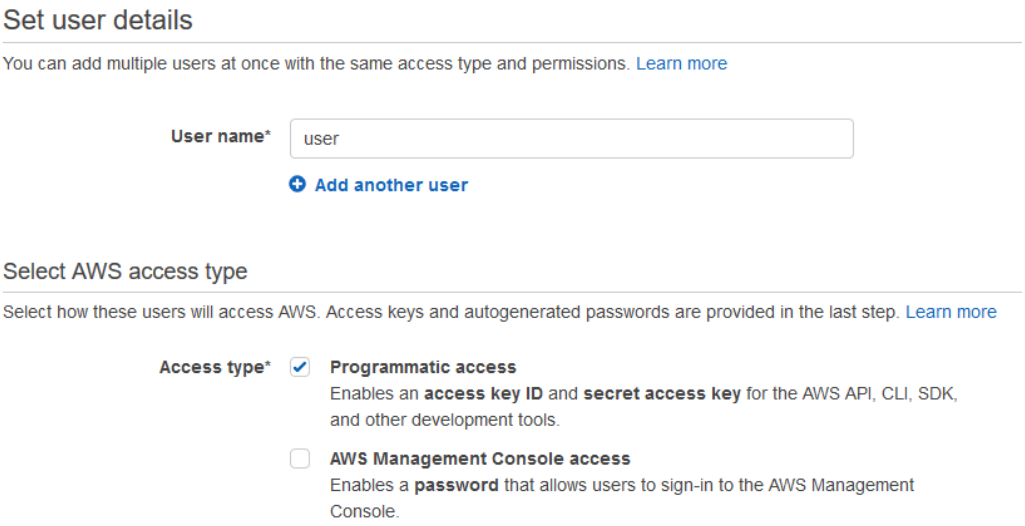

Figure 13 - User creation screen with correct settings

- On the permissions page, select “Attach existing policies directly”, followed by “Create policy”

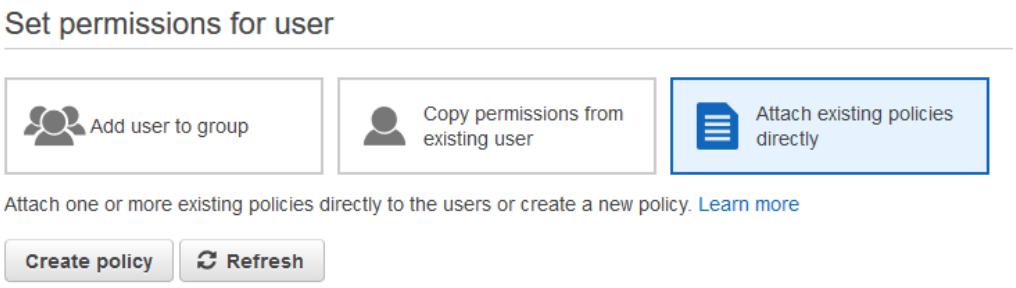

Figure 14 - Policy attachment screen

- Select “JSON” on the following screen, and enter the following JSON into the policy. Replace “test” on lines 7 and 16 with the name of your s3 bucket:
  - {

```
"Version": "2012-10-17",
"Statement": [
 {
  "Effect": "Allow",
  "Action": ["s3:ListBucket"],
  "Resource": ["arn:aws:s3:::test"]
 },
 {
  "Effect": "Allow",
  "Action": [
   "s3:PutObject",
   "s3:GetObject",
   "s3:DeleteObject"
  ],
  "Resource": ["arn:aws:s3:::test/*"]
 }
```

]

}

- After the user has been created, make sure you save the Access key ID and secret access key for this user. Download the .csv file if necessary.

Figure 15 - User Secret Access Key screen

- Step 5: Create an S3 bucket with public read-only permissions. This bucket will be used to store the source bundles and CloudFormation templates that we have updated. Creating an S3 bucket for this step will be the same as in step 3, except that on the "Permissions" screen, select "Grant Public Permissions."

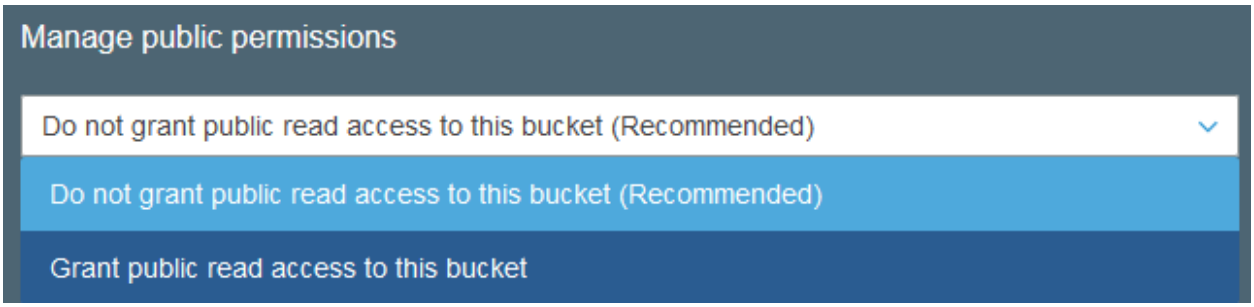

Figure 16 - S3 public permission selection

Inside of this S3 bucket, you will need the source bundles for Solr, Zookeeper, Hyku, and Fedora. Clone the repository to your computer, then navigate to the "assets/solr" directory. Zip the contents of this directory into a file called "solr.zip". Then, navigate to the "assets/zookeeper" directory and zip those contents into "zookeeper.zip". Use the following two commands to download the Hyku and Fedora source bundles:

```
$ wget -O hyku.zip https://github.com/samvera-
labs/hyku/archive/master.zip
$ wget https://hybox-deployment-artifacts.s3.amazonaws.com/fcrepo-
webapp-ext-4.8.0-SNAPSHOT.war
```

Upload “hyku.zip”, “solr.zip”, “zookeeper.zip”, and “fcrepo-webapp-ext-4.8.0-SNAPSHOT.war” to the source bundles S3 bucket created in this step.

Next, download the CloudFormation templates included in the VTechWorks submission for this project. These templates should be in the folder “cloudformation/current/templates”. Copy this entire folder structure into the source bundles S3 bucket.

- Step 6: Edit the `params/defaults.json` file with the values you created earlier. A list of the parameters to change are below, from the github:
    - KeyName: the name of the key-pair created in step 1
    - PublicZoneName: the name of the hosted zone created in step 2 (with a trailing period)
    - DatabasePassword and FcrepoDatabasePassword: password for Hyku and Fedora databases
    - FcrepoS3BucketName: the name of the S3 bucket created in step 3
    - FcrepoS3AccessKey and FcrepoS3SecretKey: API credentials for user created in step 4
    - SecretKeyBase: Rails key generation base
    - S3BucketEB and S3Bucket: name of the S3 bucket that contains the Beanstalk source bundles and CloudFormation templates created in step 5
    - S3KeyPrefix: change this to “current”
    - WebappS3Key: change this to “hyku.zip”

10.2. Deploying the Application

Once the parameters are set as outlined above, deploying the application should be fairly simple. A single command should create the stack within your AWS account.

```
$ aws --region $AWS_DEFAULT_REGION cloudformation create-stack --
disable-rollback --stack-name hybox --template-body
https://s3.amazonaws.com/<S3BucketEB>/cloudformation/current/templates
/stack.yaml --capabilities CAPABILITY_IAM --parameters
file://params/defaults.json
```

Once you have executed this command, the stack should create within your AWS environment. It may take up to an hour. After the stack is successfully created, the application will be accessible at the domain you specified earlier in the tutorial.

10.3. Bringing the Application Down

After using the CloudFormation template to create the application stack, you can go to the CloudFormation dashboard within AWS and simply select the “hybox” stack and click the delete stack button. This will take some time to delete, potentially up to a half of an hour. Deleting the stack from CloudFormation will not delete a number of S3 buckets, including hybox-solrcloud and hybox-mail. You will have to delete these on your own from the S3 dashboard.

11. Developers' Manual

This guide explains the architecture at a high level, including the various components and how they are deployed, then explains the layout of the CloudFormation templates. Finally, we discuss some aspects of the development process to help developers get started with the project quickly.

11.1. Templates

The CloudFormation templates for Hyku are available open source on GitHub under the Apache 2.0 license [5]. They can be found in the templates directory inside that repository. Various services have been split out into their own CloudFormation templates, which are included in the other templates. This splits up the large level of configuration into separate files to make it more manageable.

The initial CloudFormation template is the stack.yaml file in the templates directory, which hierarchically includes all other CloudFormation templates. It specifies a number of parameters that are configurable during the deployment; the use of these options is discussed in the user manual. You'll see many CloudFormation Stack resources in this CloudFormation template, which is how it includes the major components of the service in a single CloudFormation template.

Our implementation relies heavily on changes to the CloudFormation templates and their input parameters for deploying the Hyku application. We have proposed to merge the changes discussed below back to the original repository, with their approval, using a GitHub pull request [24].

11.2. Elastic Beanstalk

The first change we've made to the templates are to update the EBS solution stacks. As a reminder, Elastic Beanstalk is a service to help with the deployment of applications onto EC2 instances, including Ruby on Rails applications. However, when deploying to Elastic Beanstalk, you have to specify a Solution Stack version which you want to run the application on, which includes the version of the software that EBS runs, for example the Ruby version. When we tried to deploy the application, we found that the Solution Stacks that were being used for several of the components deployed on EBS were no longer supported.

The Hyku web application uses Ruby and Puma on Elastic Beanstalk, and its Solution Stack needed to be updated to the 2017.09 v2.7.2 version of the Ruby Solution Stack [25] on line 260 of webapp.yaml. The Hyku worker application also uses Ruby and Puma, so it needed the same update at line 280 of worker.yaml. The Fedora application uses Tomcat 8 with Java 8, and its Solution Stack needed to be updated to the 2017.09 v2.7.7 version of the Tomcat Solution Stack at line 157 of fcrepo.yaml. The Solr application uses multi-container Docker, and its Solution Stack needed to be updated to the 2017.09 v2.9.2 version of the Multi-container Docker Solution Stack at line 191 of solr.yaml. Finally, the Zookeeper cluster uses single-container Docker, and it needed to be updated to the 2017.09 v2.9.2 version of the single-container Docker Solution Stack at line 205 of zookeeper.yaml. Visit the pull request to see exactly what these changes involve [24]. The latest Solution Stacks are available on the AWS website [25]. They will certainly need to be updated again in the future, because the availability of Elastic Beanstalk Solution Stacks changes frequently for new AWS accounts.

11.3. CloudFormation Helper Scripts

After this, we had an issue with the configuration of the bastion instance. AWS has the concept of a VPC, which is used to keep servers isolated on the network from other VPCs and

the public internet. The only instances that are publicly accessible to the internet are the load balancers and the bastion instance. The load balancers distribute network traffic to the instances internal to the VPC. A bastion instance is a server which developers can use to access the Amazon VPC to debug problems with instances. This instance had a configuration problem with the CloudFormation Helper scripts, which we had to fix.

The CloudFormation Helper scripts are responsible for managing the services and configuration files on an instance [26]. In this case, it is used to reconfigure the bastion instance when a CloudFormation update is deployed that affects it. The problem with the current configuration is that the configuration files included in the bastion.yaml template at lines 69 and 77 included extra indentation, which needed to be removed. This indentation is unnecessary and breaks the scripts, which do not expect to find indentation. This change can be found in the GitHub pull request [24].

11.4. Changes to Parameters

We made a number of changes to the parameters in “params/default.json” when deploying the template, in addition to what is discussed in the User Manual, which is vital for reducing the cost. The following changes are necessary to reduce the instances to their minimum viable sizes based on our observations. Additionally, we discuss a change to the number of Solr instances that greatly contributes to the reduced cost. Discussion about how we decided on these instance sizes can be found in the Implementation section.

The “FcRepoInstanceType” key at line 92 in the parameters file refers to the size of the Fedora instance, and we set its value to “t2.medium”. The “SolrCloudInstanceType” key at line 108 in the parameters file refers to the size of the Solr instances, and we set its value to “t2.micro”. The “WorkerInstanceType” key at line 144 in the parameters file refers to the size of the Hyku worker instance, and we set its value to “t2.micro”. The “WebappInstanceType” key at

line 156 in the parameters file refers to the size of the Hyku web application instance, and we set its value to “t2.micro”. The “RedisInstanceType” key at line 168 in the parameters file refers to the size of the Redis ElastiCache instance, and we set its value to “cache.t2.micro”. The “DatabaseInstanceType” key at line 176 in the parameters file refers to the size of the RDS database instance for the Hyku application, and we set its value to “db.t2.micro”. The “FcrepoDatabaseInstanceType” key at line 180 in the parameters file refers to the size of the RDS database instance for the Fedora application, and we set its value to “db.t2.micro”. Any of these parameters may be scaled up to a larger instance size later, based on the performance of the various components of the Hyku application, and we suggest scaling with this method initially.

To be able to reduce the instance size of Solr as discussed above, the assets/solr/Dockerrun.aws.json has to be changed. This is because it defines the memory requirements of the container, which changes based on the instance size. Before uploading the solr.zip file to the S3 bucket in the User Manual, we had to change line 22 of this file to have a “memory” value of 950, which is in MB. This is so it fits on the t2.micro instance size, which has 1GB of memory.

Finally, we reduced the number of Solr instances. The “SolrCloudSize” key at line 112 refers to the desired number of instances in the Solr auto-scaling group, and we set its value to 1. The “SolrCloudMaxSize” key at line 116 refers to the maximum number of instances in the Solr auto-scaling group, and we set its value to 2. To be able to make this change, we had to fix a hardcoded “MinInstancesInService” key in the solr.yaml template at line 160. This defines the minimum number of instances while AWS updates the Solr application, so it must be less than the maximum number of instances. Therefore, we set its value to 1.

12. Lessons Learned

We believe that we met the goals of the project by greatly reducing the base cost of the template, as discussed in our Cost Assessment. While this may be true, we could have continued reducing the cost of the template, given we had the time. Having to work through unforeseen problems to get the template functioning definitely affected the time we could spend on optimizing the template's cost. If we had the experience with CloudFormation that we do now, we could have more accurately predicted our time requirements and set better deadlines, giving us more time for optimization. We discuss these issues below, ending with some ideas about future work that could be done to optimize the cost further.

12.1. Schedule

2/2 - Meet with client to establish requirements and expected deliverables.

2/6 - Prepare presentation 1 based on our requirements. Explain the deliverables for this project to our peers and lay out our plan for meeting the requirements.

2/20 - Inspect existing template and research options within AWS to implement improvements.

3/6 - Decide on optimization plan

3/13 - Presentation 2.

3/29 – Meet with client to discuss optimization plan

4/1 - Implement CloudFormation template and deploy a test environment

4/3 - Presentation 3.

4/17 - Assess implemented solution, compare to existing template. Tweak implementation to improve performance.

4/19 – Meet with client to discuss implemented solution

4/24 - Create graphics to show solution and assessment.

5/1 - Final presentation

We initially started out with a lot of ideas on how to improve the template and meet our deadlines near the beginning of the project. However, we didn't discover the main problems we'd encounter in the project until the implementation work started. We met all our deadlines before April, but when we went to start implementation, we started to fall behind. This was exacerbated by the amount of time it took to deploy the templates, which is discussed in detail below. Due to a lack of knowledge about the main problems we'd be dealing with in this project, we fell behind schedule in the second half of the project.

It's hard to evaluate what we could have done about this. Perhaps if we had known more about the maintenance requirements of CloudFormation templates, especially with Elastic Beanstalk, we would have had a more pessimistic view of the recency of CloudFormation templates. If we had known about how long a CloudFormation template takes to re-create, we would have planned more time for implementation. Now that we have that experience, in the future we will be able to make more accurate scheduling estimates in relationship to deployment via CloudFormation to AWS.

The "waterfall" method of software engineering, or in other words, planning the entire process up front and executing on it without falling behind schedule, works great when the problems that will be encountered can be easily predicted. When there is a lot of uncertainty about exactly what types of problems we may encounter, it's hard to accomplish this method effectively. Timeboxing could have helped us to evaluate our process and adjust our schedule as necessary. This could potentially have allowed us greater time to attempt scaling down the templates.

12.2. Problems and Solutions

The last commit to the GitHub repository containing the AWS CloudFormation templates for Hyku was on December 7, 2017, not including our pull request [5]. This is probably related to the fact that the Hydra-in-a-Box team ran out of grant money in November of 2017 [3]. Due to the repository being somewhat stale, we found that the templates were not working when we tried to deploy, due to a variety of issues which were discussed in the Developer Manual. Our solution to the GitHub repository being out of date involved documenting the process to fix it in the Developer Manual and creating a GitHub pull request to contribute the fixes back to the original repository.

In dealing with the variety of problems in the CloudFormation template, including both problems deploying the application and using it after deployment, we found that we tried to solve a lot of the problems ourselves or by consulting our client. However, our client recommended us to a Samvera Slack channel, which provides a community communication channel. If we had consulted this earlier and more frequently, instead of trying to approach some of the problems ourselves, we could have reached to solution more quickly. In some cases, the problems we dealt with were known issues that we weren't prepared to solve ourselves. We discuss one such problem in the Testing section.

One specific problem we had with the CloudFormation template was how long it took to update. On average it takes one and a half hours to delete the CloudFormation template and re-create it from scratch. While this isn't always necessary when updating the CloudFormation template, if the CloudFormation deployment fails, it is required. While this seems unnecessarily long, it actually takes much less time than it would take to deploy the individual services manually. However, if you deploy the individual services manually, you can fix failures in them piece by piece, rather than having to re-create all of the resources to fix the failure.

Another problem we had with the CloudFormation templates was its inability to consistently update a running CloudFormation stack, which forced us to re-create the stack more than necessary. When dealing with Elastic Beanstalk's auto-scaling groups and instances, we often found that changes in parameters didn't propagate through Beanstalk. Even when re-building the Elastic Beanstalk environments, the problem persisted. This seems to be a failure with how CloudFormation applies its changes to the stack. The time cost to make these changes by re-creating the stack is prohibitive, and greatly limited our ability to iterate and improve on the templates.

We feel that while CloudFormation is an excellent tool for automating manual deployments that can take a long time, it lacks the ability to easily debug and repair failures in the CloudFormation stack. If the template isn't in a working state, it can take many days to debug and fix a failure due to the re-creation time. Additionally, other problems with CloudFormation exacerbate this problem, leading to a frustrating experience with an unmaintained template. Nevertheless, it certainly is an excellent tool to use when well-maintained and used in the right contexts.

12.3. Future Work

While we believe we achieved the goals of the project, there is some further work that could be done to optimize the cost of deploying the CloudFormation templates. These potential improvements are discussed below.

First, we think we could scale down Fedora further. Right now, it is running with 4GB of memory, but we believe this is based on the JVM memory options that are hardcoded into the CloudFormation templates. We couldn't find any source saying that the minimum requirement must be 4GB. Our evidence based on what we could successfully deploy suggests that 4GB is the minimum, but perhaps playing around with the memory configuration of the JVM could result

in a lower requirement. Especially if autoscaling could be enabled for Fedora, this could provide an excellent way to lower costs.

Finally, although we had problems attempting to reduce the Hyku application below t2.micro, perhaps it could be done with further debugging. It seems that native dependencies don't have the memory to successfully build on the smaller instances while Elastic Beanstalk is deploying. However, when we test it manually, it works. There may be some way to work around this memory requirement in the building of the native gem, or some way to reduce the memory usage while deploying with Elastic Beanstalk. Further research could uncover possible options that would allow the instance size to be reduced. Another option is to forego Elastic Beanstalk altogether and use a Docker container. This would move the installation of gems off of the instance, removing the memory requirement altogether.

13. Acknowledgements

Our client for this project is Yinlin Chen.

ylchen@vt.edu

14. References

[1] Duraspace, "Fedora Repository," 2018. [Online]. Available: http://fedorarepository.org/. [Accessed 29 April 2018].

[2] Samvera, "Hyrax: a community-supported repository front-end," 2018. [Online]. Available: http://hyr.ax/. [Accessed 29 April 2018].

[3] Hydra-in-a-Box, "Hydra-in-a-Box," 2017. [Online]. Available: http://hydrainabox.samvera.org/. [Accessed 29 April 2018].

[4] Amazon Web Services, Inc., "Amazon Web Services Glossary," 2018. [Online]. Available: https://docs.aws.amazon.com/general/latest/gr/glos-chap.html. [Accessed 29 April 2018].

[5] C. Beer, "Hybox/Aws," 7 December 2017. [Online]. Available: github.com/hybox/aws. [Accessed 29 April 2018].

[6] The PostgreSQL Global Development Group, "PostgreSQL: The world's most advanced open source database," 2018. [Online]. Available: https://www.postgresql.org/. [Accessed 29 April 2018].

[7] S. Sanfilippo, "Redis," 2018. [Online]. Available: https://redis.io/. [Accessed 29 April 2018].

[8] Docker, Inc., "Docker - Build, Ship, and Run Any App, Anywhere," 2018. [Online]. Available: https://www.docker.com/. [Accessed 29 April 2018].

[9] AWS, "Amazon EC2 Pricing," AWS, 2018. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/. [Accessed 29 April 2018].

[10] Apache Software Foundation, "Solr System Requirements," 26 September 2017. [Online]. Available: https://lucene.apache.org/solr/guide/7_0/solr-system-requirements.html. [Accessed 29 April 1028].

[11] AWS, "Amazon RDS for PostgreSQL pricing," 2018. [Online]. Available: https://aws.amazon.com/rds/postgresql/pricing/. [Accessed 29 April 2018].

[12] AWS, "Previous Generation Nodes," 2018. [Online]. Available: https://aws.amazon.com/elasticache/previous-generation/. [Accessed 29 April 2018].

[13] AWS, "AWS Elasticache Pricing," 2018. [Online]. Available: https://aws.amazon.com/elasticache/pricing/. [Accessed 29 April 2018].

[14] AWS, "Amazon Aurora Product Details: PostgreSQL-Compatible Edition," 2018. [Online]. Available: https://aws.amazon.com/rds/aurora/details/postgresql-details/. [Accessed 29 April 2018].

[15] AWS, "Amazon Aurora Pricing," 2018. [Online]. Available: https://aws.amazon.com/rds/aurora/pricing/. [Accessed 29 April 2018].

[16] AWS, "AWS Fargate," 2018. [Online]. Available: https://aws.amazon.com/fargate/. [Accessed 29 April 2018].

[17] AWS, "Multicontainer Docker Environments," 2018. [Online]. Available: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_docker_ecs.html. [Accessed 29 April 2018].

[18] AWS, "AWS Fargate Pricing," 2018. [Online]. Available: https://aws.amazon.com/fargate/pricing/. [Accessed 29 April 2018].

[19] "Amazon Web Services Simple Monthly Calculator," Amazon Web Services, 2018. [Online]. Available: calculator.s3.amazonaws.com/index.html. [Accessed 19 April 2018].

[20] T. Fowler and C. Howe, "AWS Simple Monthly Calculator - Full Application," 2018. [Online]. Available: https://calculator.s3.amazonaws.com/index.html#r=SFO&key=calc-94D608B0-972F-4C2E-82CF-8B18D55A9974. [Accessed 29 April 2018].

[21] T. Fowler and C. Howe, "AWS Simple Monthly Calculator - Scaled-down Application," 2018. [Online]. Available: https://calculator.s3.amazonaws.com/index.html#r=SFO&key=calc-8F3D16F4-0285-4637-B462-2C50DAB63FEB. [Accessed 29 April 2018].

[22] H. Frost, "Hyku Documentation," Duraspace, 22 November 2017. [Online]. Available: https://wiki.duraspace.org/display/hyku/User+Documentation. [Accessed 25 April 2018].

[23] AWS, "Creating a PostgreSQL DB Instance and Connecting to a Database on a PostgreSQL DB Instance," 2018. [Online]. Available: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_GettingStarted.CreatingConnecting.PostgreSQL.html#CHAP_GettingStarted.Connecting.PostgreSQL. [Accessed 29 April 2018].

[24] T. Fowler and C. Howe, "Update to latest EB solution stacks," 29 April 2018. [Online]. Available: https://github.com/hybox/aws/pull/178. [Accessed 29 April 2018].

[25] Amazon Web Services, Inc., "Elastic Beanstalk Supported Platforms," 2018. [Online]. Available:

https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts.platforms.html. [Accessed 29 April 2018].

[26] Amazon Web Services, Inc., "CloudFormation Helper Scripts Reference," 2018. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-helper-scripts-reference.html. [Accessed 29 April 2018].

[27] H. Frost, "Frequently Asked Questions - Hyku Documentation," Hyku Product Beta, June 2017. [Online]. Available: wiki.duraspace.org/display/hyku/Hyku Product Beta - Frequently Asked Questions. [Accessed 19 April 2018].

[28] E. Phoenix, "Puma," Puma, 2018. [Online]. Available: github.com/puma/puma. [Accessed 19 April 2018].

[29] Samvera, "Hyku," Hyku, 2018. [Online]. Available: github.com/samvera-labs/hyku. [Accessed 19 April 2018].

[30] Samvera, "Hydra-In-a-Box," Samvera, 2018. [Online]. Available: hydrainabox.samvera.org. [Accessed 19 April 2018].

[31] C. Howe and T. Fowler, "cjhowe7/aws," 19 April 2018. [Online]. Available: github.com/cjhowe7/aws. [Accessed 19 April 2018].

[32] "AWS CloudFormation Template Formats," Amazon Web Services, 2018. [Online]. Available: docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-formats.html. [Accessed 19 April 2018].

[33] "AWS Well-Architected Framework," 2017. [Online]. [Accessed 19 April 2018].

[34] Apache Software Foundation, "SolrCloud on AWS EC2," 18 December 2017. [Online]. Available: https://lucene.apache.org/solr/guide/7_2/aws-solrcloud-tutorial.html. [Accessed 29 April 2018].