Exploring AWS DevOps

ALAN LANDUCCI-RUIZ

Exploring AWS DevOps

Experimentation in ElasticBeanstalk and OpsWorks

Alan T. Landucci-Ruiz

Table of Contents

1. Identity & Access Management (IAM)

Most of the access control that AWS uses is controlled by Identity & Access Management. When you first start using AWS, it's a good idea to start at the IAM panel and get familiar with everything. You should also create an administrative user so that you're not constantly using the root account. The "Security Status" section of the Dashboard will help walk you through all the necessary steps to ensure your account is properly secured. For DevOps and DevOps support, users will need to be created for the DevOps team and the support staff (AWS Administrators).

Best practices recommends creating a group first, with all of the rights that you think you would need. I started by creating an Administrators group, which has an attached policy named "AdministratorAccess"; I can adjust this later or create new ones, if needed. If you're unsure about the groups you'll need, think about the modules the users might need, then click on the "Access Advisor" tab. For example, I would want to give a DBA access to RDS, but wouldn't want them having access to EC2. A Build Engineer might only need access to OpsWorks or CloudFormation, but not the EC2 services.

Finally, create the User accounts (be sure to create access keys and distribute appropriately) and specify a group for each. Once you've created the accounts, make a note of the IAM users sign-in link at the top of the Dashboard (this can also be customized to your own preference). Copy the link, then sign out. Paste the link into the location bar of your browser, then sign in with your new credentials. If your access is correct, you should now be able to go back to the IAM page and add more users. You can generate passwords for the accounts if they will need to log into the AWS console, but these will otherwise not be needed, since the majority of AWS uses access keys.

2. Exploring Elastic Beanstalk

Introduction

There are many advantages to using Elasticbeanstalk. As a DevOps for simple applications in PHP, Java, Node.js, it has scaling abilities that developers shouldn't need to worry about. It reduces the costs associated with time lost in operating costs. In addition to the advantages, there are many advanced features that come with it. In this exploration, I intend to look at some of these features and how they can be applied to an existing development environment.

Before we begin, let's start with a simple explanation of Elastic Beanstalk. Let's assume each of our applications is considered its own container. We have an application called "Web Notifier." Within the "Web Notifier" container, we can create different environments: dev (developer), prod (production), uat (user applied testing), qa (quality assurance). We can actually have as many environments as we need, but every environment creates a new web space, or EC2 container[1]. Each environment can house several configurations, and each configuration can be applied to each environment. In fact, any setting for the container itself can be applied to any environment, and many are inherited.

Our experimentation and exploration need goals to be defined. The requirements I'm looking for in my environment are as below:

1. A Development Stack that is consistent across all environments.
2. Limits change in our development workflow, or supports our current development workflow:
    1. Git availability
    2. Support existing large site (>13G).
    3. Local accounts on the system.
3. Easily promote publication between environments.
4. PHP Customization in the .ini file.

Items 2c and 3 are ideal if they can be done on-the-fly, but are acceptable “during build”. Also, we aren’t hard-locked into keeping our current development workflow, as long as a migration to a new workflow is easy. This is because Elasticbeanstalk advertises being able to easily publish to environments.

Experimentation

Experiment 1: Exploration of the CLI

Experimentation

During the course of this exploration, we're going to use the Elastic Beanstalk command-line interface (CLI). The installation instructions for the CLI can be found at https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb3-init.html. For simplicity, since our exploration will use a RedHat Enterprise version of Linux, I've included the steps for installing the CLI on RedHat. If you're using a different distribution of Linux, please follow the aforementioned link.

```
$ sudo yum -y install python
$ curl -O https://bootstrap.pypa.io/get-pip.py
$ sudo python ./get-pip.py && rm ./get-pip.py
$ sudo pip install awsebcli
```

We should also install the AWS CLI, which gives us command-line interface to all of AWS tools:

```
pip install awscli
```

The AWS CLI will allow us to do advanced CLI commands, such as destroying full applications. After installing the AWS CLI, we will need to create the credentials to allow our computer's CLI access to make changes to the cloud.

From your AWS console, click on Identity and Access Management. Rather than using accounts or usernames, AWS uses identities and secret access keys for allowing and disallowing permissions to specific areas within AWS, such as S3 or EC2 containers. AWS designates these as Identities because, sometimes, these will be assigned to applications themselves, rather than people, but for all practical purposes, these AWS Identities are also called Users. These identities can also be allowed specific permissions, such as read-only, to each area.[2] Thus, with some advanced IAM JSON coding, you can give your developers permissions to write to the development area of Elasticbeanstalk, but not give them the freedom to create any environment they want. To ease our frustrations with managing these identities, AWS allows us to create groups. We will create our first group, named "EBAdmins", and give it the AWSElasticBeanstalkFullAccess. Now that we have our new group, we can create a new Identity by clicking "Users" (remember, in AWS, "Identity" and "User" are interchangeable). Note that in the User creation pane, we can add up to 5 users at once, and the User Name can be any descriptive name that we want, as long as it is less than 65 alphanumeric characters (and can include any of +=,.@-_). I'll call mine "Ashley.J.Williams."

The user is created and it gives me the User Security Credentials page. It's a good idea to download the credentials of the user(s) that we just created, because if we lose them, we will have to re-create them. We will not need to download new credentials if we change any of the user's permissions. We can also click on "Show User Security Credentials" and copy them down, but we'll download them, which creates a "credentials.csv" file that we can later parse through using a script to create other files that we need. If you created these for other people, then split them out into their own file, encrypt the file, then send the encrypted credentials to the intended user. A screenshot walkthrough can be found in Appendix A.

To use these credentials in the Elasticbeanstalk CLI, we need to create a file that houses our credentials. The filename is called "aws_credential_file" and is placed in the .elasticbeanstalk directory of our home directory. In this example, we've put the credentials.csv in our home directory.

```
$ cd ~
$ mkdir .elasticbeanstalk
$ awk -F, \
> '/Ashley/{NAME=$1;KEYID=$2;SECRET=$3;print "AWSAccessKeyId="KEYI
> credentials.csv > .elasticbeanstalk/aws_credential_file
$ chmod 600 .elasticbeanstalk/aws_credential_file
$ rm credentials.csv
```

The chmod line sets the file so that only the current user can read it. If you've installed the AWS CLI, you will also need to create the AWS credentials.

```
$ aws configure
AWS Access Key ID [None]: AKPAJE2DRPZAFRXSFOAA
AWS Secret Access Key [None]: ZlFCtagDrr4ZnHNMqzlsgIOefA3L+qvRwTyR
Default region name [None]: us-west-2
Default output format [None]:
```

Let's start with a clean (simple) application. In our application directory, we will want to remove any existing .git, .elasticbeanstalk, and .ebextensions subdirectories. If you've never worked with Elasticbeanstalk, you won't have these last two subdirectories.

```
$ cd ~
$ cd WebNotifier
WebNotifier $ rm -r .git
WebNotifier $ rm -r .elasticbeanstalk
WebNotifier $ rm -r .ebextensions
```

Now that we have our directory ready, we can start configuring the Elasticbeanstalk. One note before we begin: login to the AWS console and take a look at the Elasticbeanstalk configuration there. We want to make sure there isn't already a project with the same name; if there is, we'll want to pick a different name for our application. When we initialize our Elasticbeanstalk from the CLI, it detects any other applications that we have created (including ones we've already destroyed) and will ask us if we want to use those or create a new one. It will also ask us for a default region. This is generally the region that is closest to you. For mine, I would select us-west-2 (Oregon). Information on the different "eb" subcommands can be found with the "–help" switch and at http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3-getting-started.html.

```
WebNotifier $ eb init

Select a default region
1) us-east-1 : US East (N. Virginia)
2) us-west-1 : US West (N. California)
3) us-west-2 : US West (Oregon)
4) eu-west-1 : EU (Ireland)
5) eu-central-1 : EU (Frankfurt)
6) ap-southeast-1 : Asia Pacific (Singapore)
7) ap-southeast-2 : Asia Pacific (Sydney)
8) ap-northeast-1 : Asia Pacific (Tokyo)
9) ap-northeast-2 : Asia Pacific (Seoul)
10) sa-east-1 : South America (Sao Paulo)
```

```
11) cn-north-1 : China (Beijing)
(default is 3): 3


Select an application to use
1) ad1app
2) awsmigrate
3) [ Create new Application ]
(default is 3): 3


Enter Application Name
(default is "WebNotifier"): <enter>
Application WebNotifier has been created.


It appears you are using PHP. Is this correct?
(y/n): y


Select a platform version.
1) PHP 5.4
2) PHP 5.5
3) PHP 5.6
4) PHP 5.3
(default is 1): 3
Do you want to set up SSH for your instances?
(y/n): y


Select a keypair.
1) awskeypair
2) [ Create new KeyPair ]
(default is 10): 1
```

Elasticbeanstalk detected that the application, from the files in my current directory, was a PHP application. With the “eb init” command, we can control other things like the operating system version, but this simple init subcommand gives us the latest Amazon Linux distribution. I’ve selected PHP 5.6 because I know my app supports it, but you may want to check the requirements for your application. After running this command, it’s created a .gitignore file and an .elasticbeanstalk directory with a file named “config.yml”. This file can also be created before you run “eb init”, and it will take the configuration syntax from the “config.yml” file to create the application. This file is also where you can specify your OS version. Our auto-generated file is very small with very few specifications.

```
WebNotifier $ cat .elasticbeanstalk/config.yml
branch-defaults:
  default:
    environment: null
    group_suffix: null
global:
  application_name: WebNotifier
  default_ec2_keyname: awskeypair
  default_platform: PHP 5.6
  default_region: us-west-2
  profile: eb-cli
  sc: null
```

The config files are written in YAML syntax, which uses the spaces as delineators within each sub-group. The advantage over JSON is that we don’t have to worry about whether or not we’ve closed a brace, bracket, or quotation mark (okay, we do have to worry about closing quotation marks). Before we create the environment, we’re going to add a couple of users to our configuration. This now requires us to use the .ebextensions directory. The documentation to the .ebextensions and everything that you can do can be found here: http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/customize-containers-ec2.html.

We can create separate configuration files for everything, as long as our file names end in the “.config” extension. We’ll create one file for both users and groups, because we want to make sure our group gets created before the user is, otherwise we’ll get an error that the group doesn’t exist. Also, there was a time when usernames could only be 8 characters long, but more than 8 characters can get exhausting to type, so we’ll keep it less than 8 characters.

```
WebNotifier $ mkdir .ebextensions
WebNotifier $ vi .ebextensions/users.config
groups:
  deadite:
    gid: "118"
users:
  williama:
    groups:
     - deadite
    uid: "2000"
    homeDir: "/home/williama"
```

We can try to spin up our environment now, but let’s add one more thing. We have a public git repository where we’ve stored our stuff, and want to keep it up to date. Whenever I start up a new environment, scale an environment, or upgrade an environment, the Elasticbeanstalk will use the last good “push” of the environment (we’ll get to that later). This is fine, but what if our site is extremely large? An initial push will timeout when doing our git push to the beanstalk. There are a couple of ways around this, but the simplest way is to try to fetch our entire site down from our git repository. We do this with a few files. Let’s start with the commands. Commands are run in alphabetical order by the name that we designate and before the application is extracted and run.

```
WebNotifier $ vi .ebextensions/commands.config
files:
  "/home/ec2-user/.netrc":
    mode: "000600"
```

```
      owner: ec2-user
      content: |
        machine bitbucket.org login haxxon_hax password abc12345
    "/home/root/.netrc":
          mode: "000600"
          owner: root
          content: |
            machine bitbucket.org login haxxon_hax password abc12345
    "/tmp/getmyapp.sh":
      mode: "000755"
      content: |
        #!/bin/sh
        git init
        git remote add origin https://bitbucket.org/haxxon_hax/allsta
        git fetch
        rm /home/ec2-user/.netrc
        rm /home/root/.netrc
  commands:
    gitmyappfrombitbucket:
      cwd: /var/app/current
      command: /tmp/getmyapp.sh
```

If our git repository were publicly available, then we wouldn't need the first file, the .netrc file. This gets placed in the ec2-user's home directory. We've also placed it in root's home directory, because, at this moment, we don't know which account these commands will use to run as. We're now ready to create our environment. As our environment runs, we are able to hit CTRL-C to exit out of the output and do other work, and the environment will still create in the background.

```
WebNotifier $ eb create WebNotifier-dev
Creating application version archive "app-160629_163607".
Uploading WebNotifier/app-160629_163607.zip to S3. This may take a
Upload Complete.
Environment details for: WebNotifier-dev
  Application name: WebNotifier
  Region: us-west-2
  Deployed Version: app-160629_163607
  Environment ID: e-iaz8vvtdsc
```

```
    Platform: 64bit Amazon Linux 2016.03 v2.1.3 running PHP 5.6
    Tier: WebServer-Standard
    CNAME: UNKNOWN
    Updated: [a]2016-06-25 23:09:52.506000+00:00
  Printing Status:
  INFO: createEnvironment is starting.
  INFO: Using elasticbeanstalk-us-west-2-550807172100 as Amazon S3 st
  nt data.
  INFO: Created security group named: sg-1ed04678
  INFO: Created load balancer named: awseb-e-i-AWSEBLoa-13A1LJMMNCKV(
  INFO: Created security group named: awseb-e-iaz8vvtdsc-stack-AWSEBS
  INFO: Environment health has transitioned to Pending. Initializatio
   16 seconds). There are no instances.
  INFO: Added instance [i-3a8908e7] to your environment.
  INFO: Waiting for EC2 instances to launch. This may take a few minu
  INFO: Created Auto Scaling group named: awseb-e-iaz8vvtdsc-stack-AW
  SEVMGGQ
  INFO: Created Auto Scaling group policy named: arn:aws:autoscaling
  lingPolicy:725b85dc-8fb9-499b-b477-62913ea1078e:autoScalingGroupNar
  -AWSEBAutoScalingGroup-1H1O52SEVMGGQ:policyName/awseb-e-iaz8vvtdsc-
  eUpPolicy-107A3TL4ZMCWY
  INFO: Created Auto Scaling group policy named: arn:aws:autoscaling
  lingPolicy:c8f59b45-cc15-4e3e-bbe5-6fbfe511c641:autoScalingGroupNar
  -AWSEBAutoScalingGroup-1H1O52SEVMGGQ:policyName/awseb-e-iaz8vvtdsc-
  eDownPolicy-1XWR3Q2GK2GVZ
  INFO: Created CloudWatch alarm named: awseb-e-iaz8vvtdsc-stack-AWSI
  HBLJLE1
  INFO: Created CloudWatch alarm named: awseb-e-iaz8vvtdsc-stack-AWSI
  JAIDFU
  ERROR: [Instance: i-3a8908e7] Command failed on instance. Return co
  ..to possible repository corruption on the remote side.
  fatal: protocol error: bad pack header
  remote: warning: suboptimal pack - out of memory
  remote: aborting due to possible repository corruption on the remot
  fatal: protocol error: bad pack header.
  command gitmyappfrombitbucket in .ebextensions/commands.config fail
   /var/log/eb-activity.log using console or EB CLI.
  INFO: Command execution completed on all instances. Summary: [Succe

  ERROR: The operation timed out. The state of the environment is unl
  et using the --timeout option.
```

The operation timed out. This is likely due to the very large site that we have. We’ll log into the instance and take a look at the activity log to verify.

```
$ eb ssh
$ less /var/log/eb-activity.log
[...]
[[c]2016-06-25T23:08:36.664Z] INFO  [2852]  - [Application deployme
rtupStage0/EbExtensionPreBuild/Infra-EmbeddedPreBuild/prebuild_0_We
frombitbucket] : Activity execution failed, because: Initialized em
/app/current/.git/
  remote: warning: suboptimal pack - out of memory
  remote: aborting due to possible repository corruption on the rem
  fatal: protocol error: bad pack header
  remote: warning: suboptimal pack - out of memory
  remote: aborting due to possible repository corruption on the rem
  fatal: protocol error: bad pack header
   (ElasticBeanstalk::ExternalInvocationError)[d]
```

The “out of memory” error is usually due to the size of the repository. Instead of fiddling around with the .git/config files to get the site, we want to first make sure our .ebextensions configurations work, so we’re going to create our environment again, but point to a smaller repository.

```
WebNotifier $ eb terminate WebNotifier-dev
WebNotifier $ vi .ebextensions/commands.config
files:
  "/home/ec2-user/.netrc":
    mode: "000600"
    owner: ec2-user
    content: |
      machine bitbucket.org login haxxon password abc12345
  "/home/root/.netrc":
    mode: "000600"
    owner: root
    content: |
      machine bitbucket.org login haxxon_hax password abc12345
  "/tmp/getmyapp.sh":
    mode: "000755"
    content: |
      #!/bin/sh
      git init
      git remote add origin https://bitbucket.org/haxxonhax/allsta
      git fetch
      rm /home/ec2-user/.netrc
commands:
  gitmyappfrombitbucket:
    cwd: /var/app/current
    command: /tmp/getmyapp.sh
WebNotifier $ eb create WebNotifier-dev
Creating application version archive "app-160120_160949".
Uploading WebNotifier/app-160120_160949.zip to S3. This may take a
Upload Complete.
Environment details for: WebNotifier-dev
  Application name: WebNotifier
  Region: us-west-2
  Deployed Version: app-160120_160949
  Environment ID: e-9huqtp9jht
  Platform: 64bit Amazon Linux 2015.09 v2.0.6 running PHP 5.6
  Tier: WebServer-Standard
  CNAME: UNKNOWN
  Updated: 2016-06-25 23:09:52.506000+00:00
Printing Status:
INFO: createEnvironment is starting.
INFO: Using elasticbeanstalk-us-west-2-550807172100 as Amazon S3 s
nt data.
INFO: Created security group named: sg-5930da3e
INFO: Created load balancer named: awseb-e-9-AWSEBLoa-1GJMO9D3SPTO
INFO: Environment health has transitioned to Pending. There are no
INFO: Created security group named: awseb-e-9huqtp9jht-stack-AWSEB
INFO: Created Auto Scaling launch configuration named: awseb-e-9hu
ingLaunchConfiguration-1IHHDAXFXIY24
INFO: Created Auto Scaling group named: awseb-e-9huqtp9jht-stack-A
28T1LZR
INFO: Waiting for EC2 instances to launch. This may take a few min
INFO: Created Auto Scaling group policy named: arn:aws:autoscaling
lingPolicy:f1d097f9-f2fd-4d7c-a39c-67c0e95ef830:autoScalingGroupNa
-AWSEBAutoScalingGroup-1DG34128T1LZR:policyName/awseb-e-9huqtp9jht
eDownPolicy-SD8XTGPWXF3L
INFO: Created Auto Scaling group policy named: arn:aws:autoscaling
lingPolicy:051230d7-8c13-4f09-9b03-bab571a68ca8:autoScalingGroupNa
-AWSEBAutoScalingGroup-1DG34128T1LZR:policyName/awseb-e-9huqtp9jht
eUpPolicy-5ELDCE8ZOZYX
INFO: Created CloudWatch alarm named: awseb-e-9huqtp9jht-stack-AWS
5HK6B0T
INFO: Created CloudWatch alarm named: awseb-e-9huqtp9jht-stack-AWS
VV4FEV4
INFO: Added instance [i-c0736619] to your environment.
INFO: Environment health has transitioned from Pending to Ok.
INFO: Successfully launched environment: WebNotifier-dev
```

Let’s SSH into the system and check our work. The EB CLI provides an ‘ssh’ subcommand, but will require access to the SSH key that was referred to during our ‘git init’ command. Since we are using a pre-existing key, this will need to be placed in our home directory’s .ssh directory.

```
WebNotifier $ eb ssh
INFO: Attempting to open port 22.
INFO: SSH port 22 open.
The authenticity of host '52.33.50.27 (52.33.50.27)' can't be esta
ECDSA key fingerprint is be:30:6c:b6:f7:7a:79:cd:fb:90:69:8f:03:95
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.33.50.27' (ECDSA) to the list of kn
 _____ _           _   _      ____                       _        
| ____| | __ _ ___| |_(_) ___| __ )  ___  __ _ _ __  ___| |_ __ _|
|  _| | |/ _` / __| __| |/ __|  _ \ / _ \/ _` | '_ \/ __| __/ _` |
| |___| | (_| \__ \ |_| | (__| |_) |  __/ (_| | | | \__ \ || (_| |
|_____|_|\__,_|___/\__|_|\___|____/ \___|\__,_|_| |_|___/\__\__,_|
                                       Amazon Linux AMI


This EC2 instance is managed by AWS Elastic Beanstalk. Changes mad
WILL BE LOST if the instance is replaced by auto-scaling. For more
on customizing your Elastic Beanstalk environment, see our documen
http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/customize-co


[ec2-user@ip-172-3-3-7 ~]$ grep deadite /etc/group
deadite:x:118:williama
[ec2-user@ip-172-3-3-7 ~]$ ls -l /tmp/getmyapp.sh
-rwxr-xr-x 1 root root 134 Jan 20 23:13 /tmp/getmyapp.sh
[ec2-user@ip-172-3-3-7 current]$ ls -a /var/app/current
.                   contributors.txt  loginmodule.css     readme.ht
..                  license.txt       logout.php          README.md
access-denied.php   login-exec.php    member-index.php    register-
auth.php            login-failed.php  member-profile.php  register-
config.php          login-form.php    mysql.sql           register-
[ec2-user@ip-172-3-3-7 ~]$ cd /var/log
[ec2-user@ip-172-3-3-7 log]$ less eb-activity.log
[2016-06-25T23:13:42.465Z] INFO  [2743]  - [Application deployment
/EbExtensionPreBuild/Infra-EmbeddedPreBuild/prebuild_0_WebNotifier
appfrombitbucket] : Completed activity. Result:
  Initialized empty Git repository in /var/app/current/.git/
```

```
    remote: Not Found
    fatal: repository 'https://bitbucket.org/haxxonhax/allstars.git/
```

We can see that our user and group were created, but when we look at our eb-activity.log file, there was a problem with the script we ran. “Remote: Not Found” doesn’t look good. Upon closer examination, the bitbucket URL was wrong. We put “https://bitbucket.org/haxxonhax/allstars.git”, which should have been “https://bitbucket.org/haxxon_hax/allstars.git” instead. Before we fix this, however, let’s look in the /var/app/current directory.

```
[ec2-user@ip-172-3-3-7 log]$ ls /var/app/current/
access-denied.php  login-exec.php    member-index.php    register-e
auth.php           login-failed.php  member-profile.php  register-
config.php         login-form.php    mysql.sql           register-
contributors.txt   loginmodule.css   readme.html
license.txt        logout.php        README.md
```

All of our files are there, while our “git fetch” command failed. How is this possible? When we created the environment, we created it within our WebNotifier application directory. Elasticbeanstalk took this as the source directory for the files when we created the application. As soon as we created our first environment, it created an S3 storage bucket, zipped everything, uploaded it to the Amazon AWS S3 storage bucket, and extracted it there. This is a good thing because we have everything that was on our application. This can also be a bad thing because if your current directory is behind on revisions from your Git repository, then the newer files will get overwritten by the uploaded ones. This happens because the commands are run before the application itself is extracted. A little later, we’ll do an experiment, but for now, let’s fix our ‘git fetch’ command.

```
WebNotifier$ vi .ebextensions/commands.config
files:
  "/home/ec2-user/.netrc":
    mode: "000600"
    owner: ec2-user
    content: |
      machine bitbucket.org login haxxon_hax password abc12345
  "/home/root/.netrc":
    mode: "000600"
    owner: root
    content: |
      machine bitbucket.org login haxxon_hax password abc12345
  "/tmp/getmyapp.sh":
    mode: "000755"
    content: |
      #!/bin/sh
      git init
      git remote add origin https://bitbucket.org/haxxon_hax/allsta
      git fetch
      rm /home/ec2-user/.netrc
      rm /home/root/.netrc
commands:
  gitmyappfrombitbucket:
    cwd: /var/app/current
    command: /tmp/getmyapp.sh
WebNotifier$ eb deploy
WARNING: You have uncommitted changes.
Creating application version archive "app-91e1-160122_100830".
Uploading WebNotifier/app-91e1-160122_100830.zip to S3. This may ta
Upload Complete.
INFO: Environment update is starting.
INFO: Deploying new version to instance(s).
INFO: New application version was deployed to running EC2 instance
INFO: Environment update completed successfully.
WebNotifier $ eb ssh
INFO: Attempting to open port 22.
INFO: SSH port 22 open.

 _____ _           _   _      ____                       _        _
| ____| | __ _ ___| |_(_) ___| __ )  ___  __ _ _ __  ___| |_ __ _| |
|  _| | |/ _` / __| __| |/ __|  _ \ / _ \/ _` | '_ \/ __| __/ _` | |
| |___| | (_| \__ \ |_| | (__| |_) |  __/ (_| | | | \__ \ || (_| | |
|_____|_|\__,_|___/\__|_|\___|____/ \___|\__,_|_| |_|___/\__\__,_|_|
                                       Amazon Linux AMI
```

```
This EC2 instance is managed by AWS Elastic Beanstalk. Changes made
WILL BE LOST if the instance is replaced by auto-scaling. For more
on customizing your Elastic Beanstalk environment, see our document
http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/customize-co
[ec2-user@ip-172-3-3-7 ~]$ less /var/log/eb-activity.log
[2016-06-25T23:19:54.502Z] INFO  [2301]  - [Application update/App
Build/Infra-EmbeddedPreBuild/prebuild_0_WebNotifier/Command gitmya
d activity. Result:
  Initialized empty Git repository in /var/app/current/.git/
  From https://bitbucket.org/haxxon_hax/allstars
  * [new branch]      master     -> origin/master
  * [new branch]      working    -> origin/working
[2016-06-25T23:19:54.856Z] INFO  [2301]  - [Application update/App
ok/01_unzip.sh] : Completed activity. Result:
  ++ /opt/elasticbeanstalk/bin/get-config container -k app_user
  + EB_APP_USER=webapp
  ++ /opt/elasticbeanstalk/bin/get-config container -k app_staging_
  + EB_APP_STAGING_DIR=/var/app/ondeck
  ++ /opt/elasticbeanstalk/bin/get-config container -k source_bund
  + EB_SOURCE_BUNDLE=/opt/elasticbeanstalk/deploy/appsource/source_
  + rm -rf /var/app/ondeck
  + /usr/bin/unzip -d /var/app/ondeck /opt/elasticbeanstalk/deploy/
  Archive:  /opt/elasticbeanstalk/deploy/appsource/source_bundle
  11e1abd093810b84242455b8193d5e0591ded590
```

Our git repository script worked, but notice that it later starts the unzip of our source bundle. If we recall, when we did the “eb deploy”, it said something about uploading a zip file to S3: “Uploading WebNotifier/app-91e1-160122_100830.zip to S3. This may take a while.” This is what it unzips after all the commands are run. What does this mean? Well, our script worked pulling the code from the git repository, but it also zipped up the current directory, uploaded the zip file to an S3 bucket on AWS, then overwrote the application directory by extracting the zip file on top of the files we just pulled from git. This isn’t necessarily a bad thing, but it’s doing double the work. Plus, if we have a really large site, we don’t want everything to have to get zipped and sent to Elasticbeanstalk every time; we want only the changes to be populated. We’ll verify the overwriting of the files by changing the “index.php” file in our current directory, adding a timestamp to the text. Then, we’ll create a new environment with “eb create” (reasons for this later toward the end of Experiment 2), and take a look at the activity log. Notice that we haven’t initialized Git in our current application directory, so we’re not tracking changes to these files.

```
WebNotifier $ ls -al .git
ls: cannot access .git: No such file or directory
WebNotifier $ vi index.php
[...]
<h1>Welcome 2016-06-25</h1>
[...]
WebNotifier $ eb create WebNotifier-qa
[...]
WebNotifier $ eb use WebNotifier-qa
WebNotifier $ eb ssh
[...]
[ec2-user@ip-172-3-1-9 log]$ less eb-activity.log
[...]
Result:
  Initialized empty Git repository in /var/app/current/.git/
  From https://bitbucket.org/haxxon_hax/allstars
  * [new branch]          master         -> origin/master
  * [new branch]          working        -> origin/working
  From https://bitbucket.org/haxxon_hax/allstars
  * branch                master         -> FETCH_HEAD
[ec2-user@ip-172-3-1-9 current]$ grep 2016 index.php
<h1>Welcome 2016-07-01</h1>
[ec2-user@ip-172-3-1-9 current]$ exit
```

The index.php file from our repository was overwritten by the deployed files in our directory. The simple reason is because our commands run before the documents are deployed.

Observations

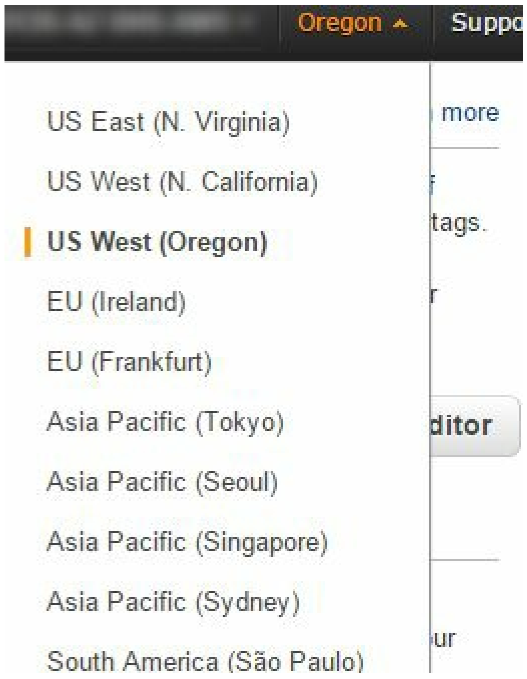
We can change different settings in the Elasticbeanstalk environments using the .ebextensions directory. We've been able to successfully add users and create custom commands. The documentation on .ebextensions shows that we can also add packages, groups, sources, services, and container commands. When using the .ebextensions, we've noticed that the extensions get run before extracting our site, which defaults to the /var/app/current directory.

Before we continue with some more tests on the Elasticbeanstalk CLI, we're going to explore the visual portion, the web console. Then, we'll explore the customization that we can do with Elasticbeanstalk before we come back to trying to deploy our site with git.

Experiment 2: The Web User Interface

Experimentation

From the AWS console, we can browse the AWS services and click on "Elastic Beanstalk". If you go here and you don't see any of your applications, check to make sure you're in the same region as when you created your beanstalk. You can find this in the drop-down in the upper right-hand corner of the console, between your account name and the word "Support." These names aren't listed exactly the same as your selections in the "eb init" earlier, but they are pretty intuitive. Since I selected "us-west-2" for my region, I select the second US West from the drop-down, which is "Oregon."



[redacted] Oregon Suppo

more

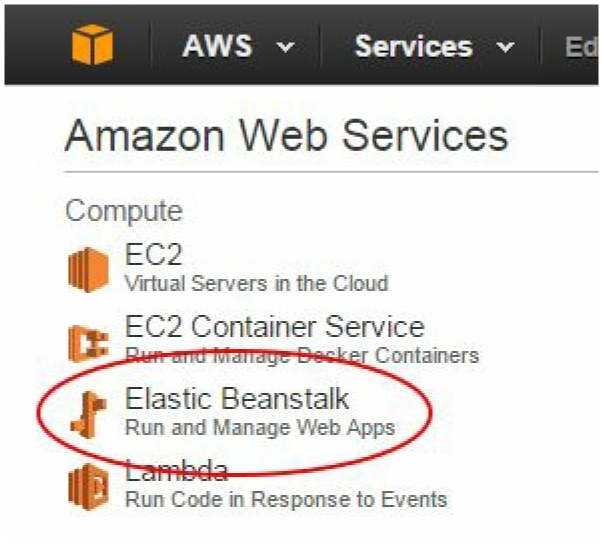
tags.

ditor

ur

US East (N. Virginia)
US West (N. California)
US West (Oregon)
EU (Ireland)
EU (Frankfurt)
Asia Pacific (Tokyo)
Asia Pacific (Seoul)
Asia Pacific (Singapore)
Asia Pacific (Sydney)
South America (São Paulo)

Clicking on the Elastic Beanstalk option under the “Compute” category takes us to the landing page where we can see all of our applications.



AWS Services Ed

Amazon Web Services

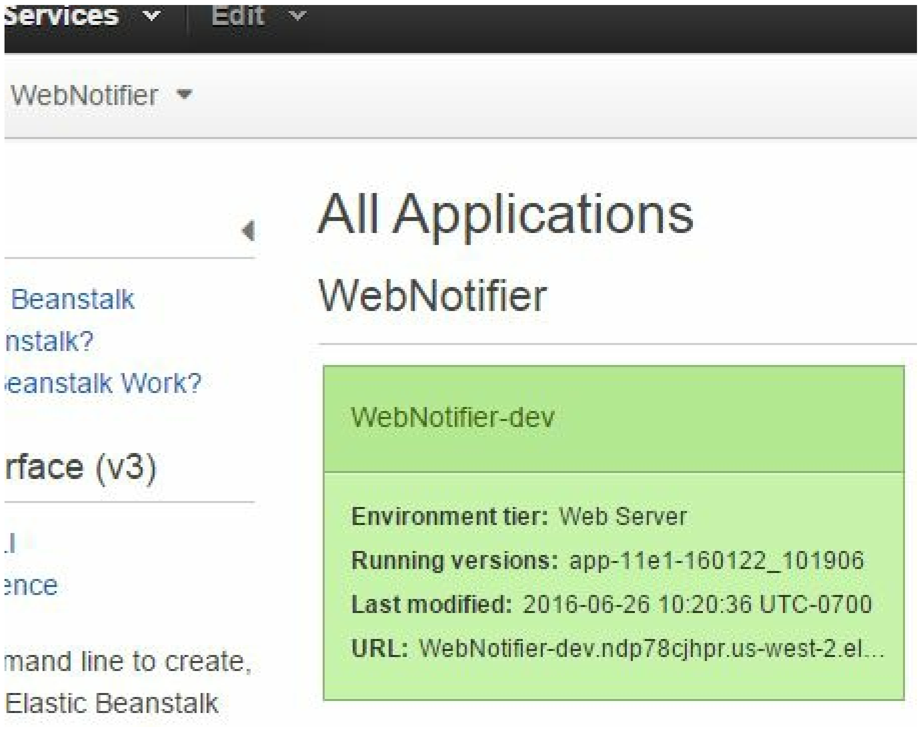
Compute

EC2
Virtual Servers in the Cloud

EC2 Container Service
Run and Manage Docker Containers

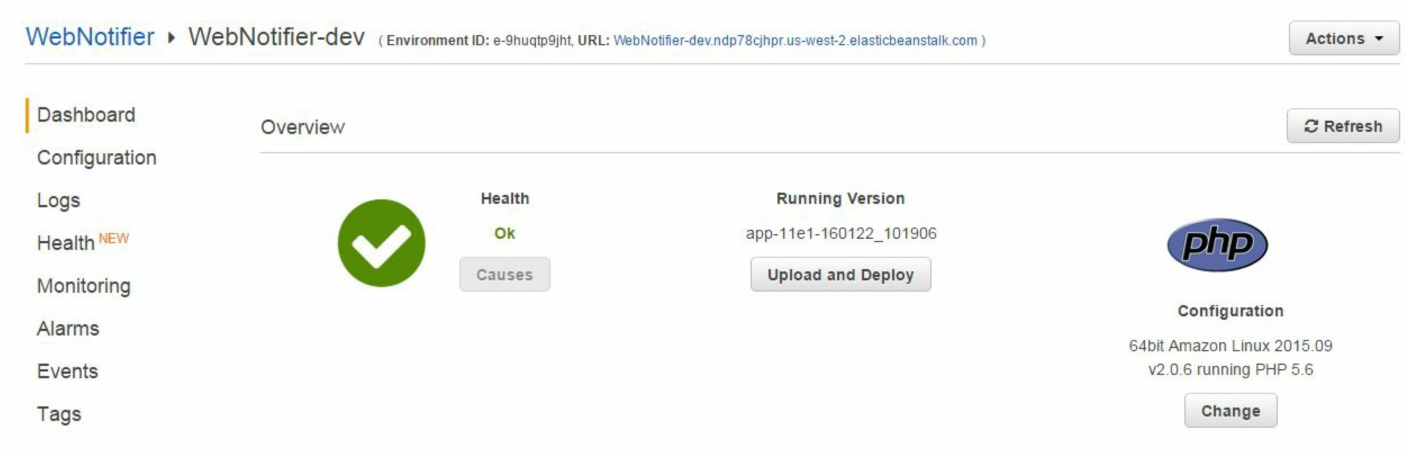
Elastic Beanstalk
Run and Manage Web Apps

Lambda
Run Code in Response to Events



Environment tier: Web Server
Running versions: app-11e1-160122_101906
Last modified: 2016-06-26 10:20:36 UTC-0700
URL: WebNotifier-dev.ndp78cjhpr.us-west-2.el...

We are able to see the environment that we set up, along with some summary information about it. If we hover over the URL, we can see the full URL. If we click anywhere in the box, we'll get to the dashboard of the environment.

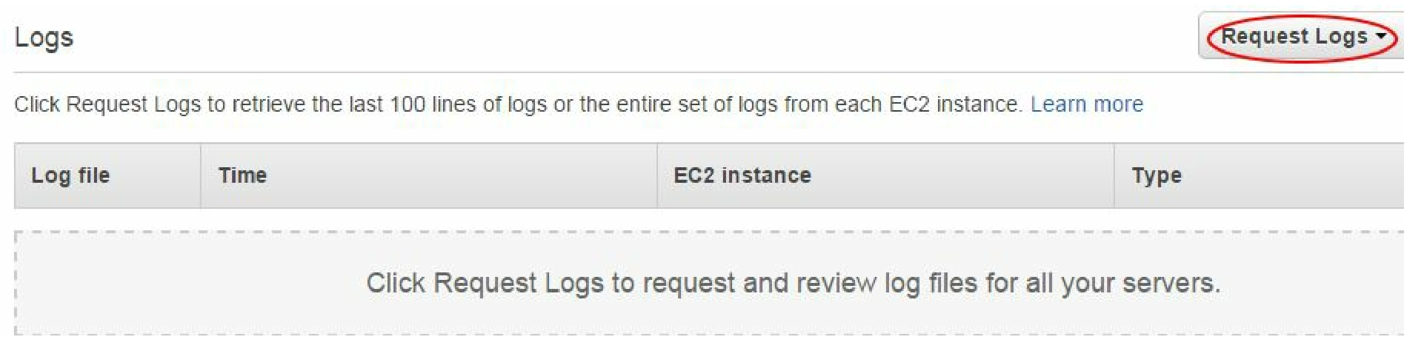


(Environment ID: e-9huqtp9jht, URL: WebNotifier-dev.ndp78cjhpr.us-west-2.elasticbeanstalk.com)

app-11e1-160122_101906

64bit Amazon Linux 2015.09 v2.0.6 running PHP 5.6

We can do many different things here, such as upload and deploy, change the application version we're running, load and save configurations, and swap environment URLs. Let's take a look at the logs.


Logs

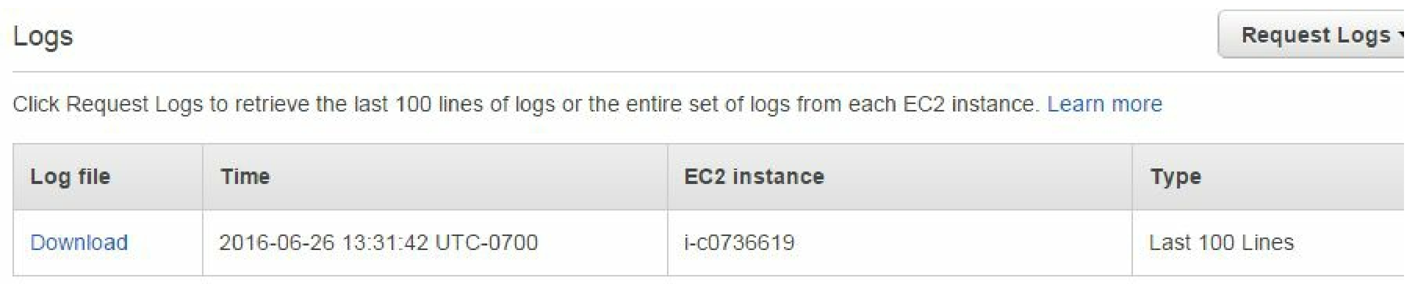
Request Logs

Click Request Logs to retrieve the last 100 lines of logs or the entire set of logs from each EC2 instance. Learn more

Log file	Time	EC2 instance	Type

Click Request Logs to request and review log files for all your servers.

It doesn’t look like there are any logs listed. Let’s click on “Request Logs”, and download the last 100.


Logs

Request Logs

Click Request Logs to retrieve the last 100 lines of logs or the entire set of logs from each EC2 instance. Learn more

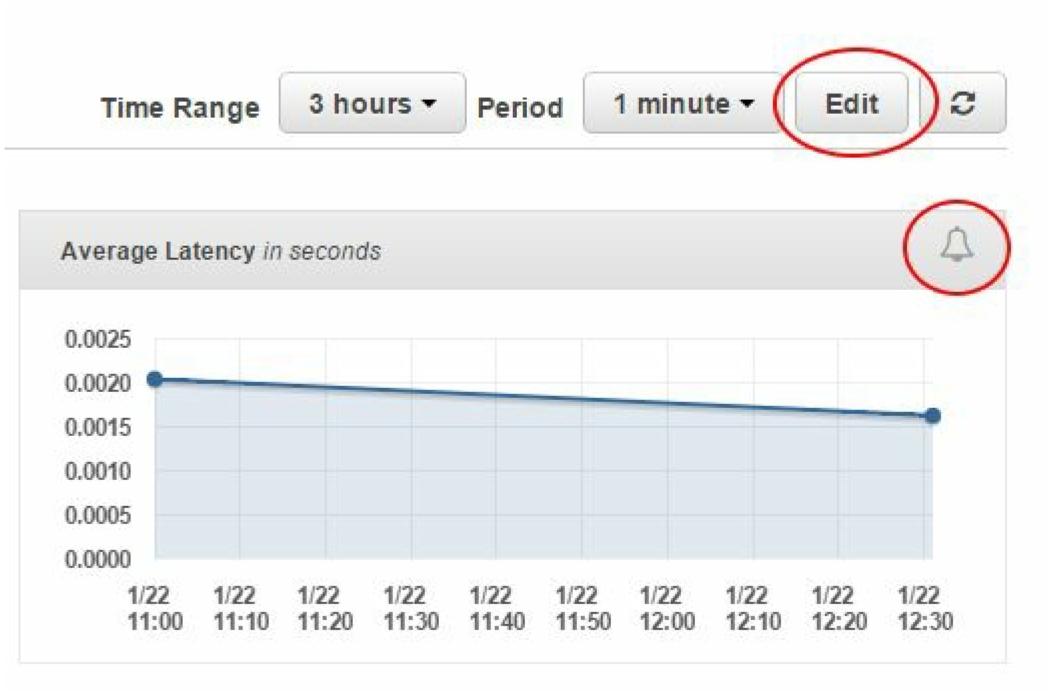
Log file	Time	EC2 instance	Type
Download	2016-06-26 13:31:42 UTC-0700	i-c0736619	Last 100 Lines

image19.png

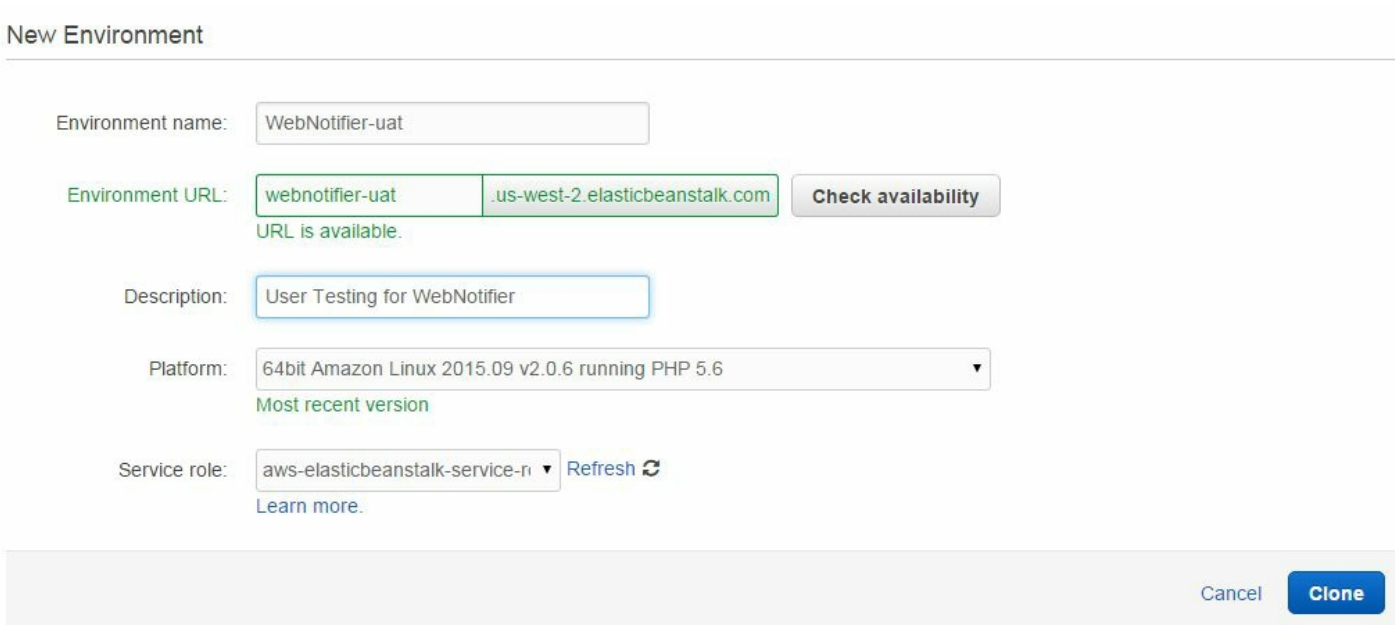
Clicking on “Download” gives us the logs from the following: /var/log/eb-version-deployment.log, /var/log/httpd/error_log, /var/log/httpd/access_log, /var/log/eb-activity.log, and /var/log/eb-commandprocessor.log. The eb-activity.log states “requested file /var/log/eb-activity.log is not a plain text file, skipping…”, but we were able to view that in SSH, weren’t we? What’s going on? Well, the file is readable because it’s a UTF-8 Unicode text file. Elasticbeanstalk’s log collector will only show files in its list that are ASCII text, and UTF-8 Unicode is not ASCII.

Let’s take a look at the Configuration pane next. The Configuration pane allows us to configure different areas of the configuration: Scaling, Notifications, Instances, Software Configuration, Network Tier, Updates and Deployments, Health, and VPC. If we click on Software Configuration, we’ll notice that it’s not as granular as our CLI allows us to be. We can change the path of our application, adjust memory limits, set zlib output compression, enable URL fopen, display errors, and set the maximum execution time. Just as with the CLI, we can set environment variables and enable log rotation.

The Events pane lets us see when the application environment had updates to its deployment or if it failed a deployment. The Monitoring pane allows us to look at the performance of the environment and set alarms if we need to. We can add new graphs by clicking on “Edit”, and set alarms by clicking on the bell icon for each graph.



Another nice feature that the Web Console allows is the ability to clone our environment. We can also do this with the CLI using the “eb clone” command. In both cases, we have the ability to set the environment name, the environment URL, and the platform. In the CLI, we can also set the scaling, tags, profile, region, and environment variables. We are not, however, able to set the description or the service role.



New Environment

Environment name: WebNotifier-uat

Environment URL: webnotifier-uat .us-west-2.elasticbeanstalk.com

URL is available.

Description: User Testing for WebNotifier

Platform: 64bit Amazon Linux 2015.09 v2.0.6 running PHP 5.6

Most recent version

Service role: aws-elasticbeanstalk-service-r

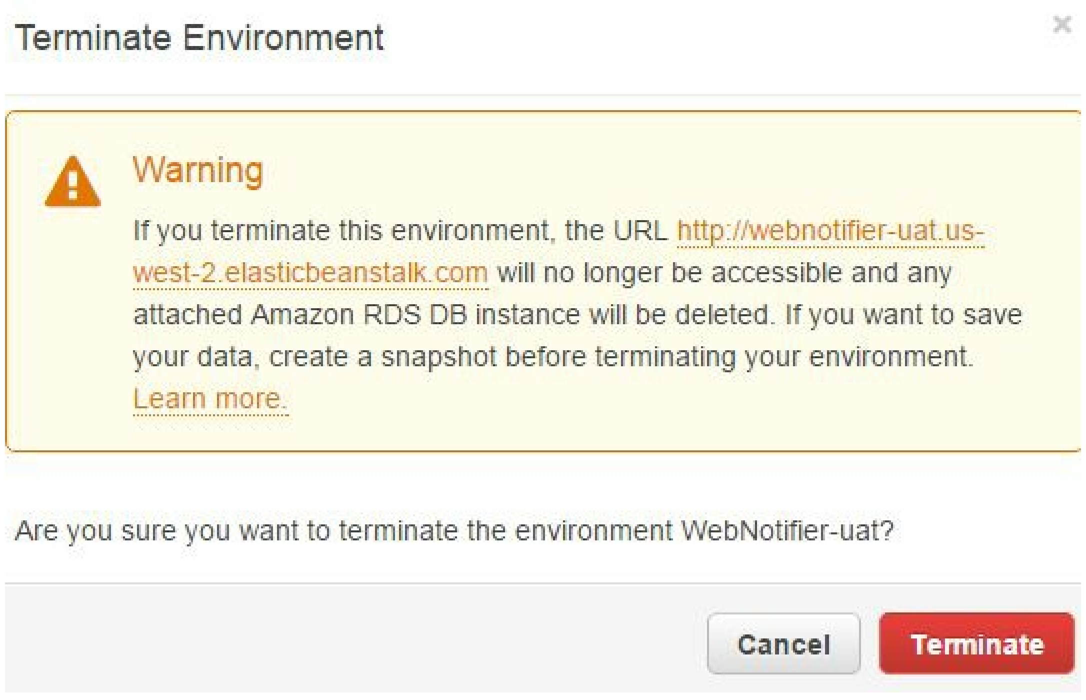
The clone also copies the entire configuration, including alarm thresholds, notifications, and monitors. After cloning the environment, if we SSH into the clone and look at the eb-activity.log file, we'll see that there's an error on the "git fetch" command.

```
[2016-06-26T21:24:16.177Z] INFO  [2731]  - INFO  [3386]  - [Applica
824@3/AppDeployStage0/EbExtension
PreBuild/Infra-EmbeddedPreBuild/prebuild_0_WebNotifier/Command gitr
eted activity. Re
sult:
  Initialized empty Git repository in /var/app/current/.git/
  From https://bitbucket.org/haxxon_hax/allstars
  * [new branch]          master         -> origin/master
  * [new branch]          working        -> origin/working
  From https://bitbucket.org/haxxon_hax/allstars
  * branch                master         -> FETCH_HEAD
  error: The following untracked working tree files would be overw
  README.md
  access-denied.php
  auth.php
  config.php
  contributors.txt
  index.php
  license.txt
  login-exec.php
  login-failed.php
  login-form.php
  loginmodule.css
```

```
    logout.php
    member-index.php
    member-profile.php
    mysql.sql
    readme.html
    register-exec.php
    register-form.php
    register-success.php
    users.config
    Please move or remove them before you can merge.
    Aborting
```

This happens because the files were overwritten on our source environment after the "git" was already initialized. We can verify this by looking at the index.html file, which we changed in Experiment 1.

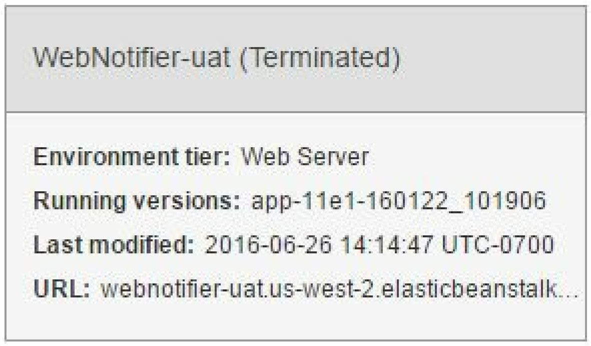
Just as we created the environments, we can also destroy (terminate) the environments. As with AWS EC2, terminated items will remain visible for approximately an hour after they've been terminated. Terminating an environment can be done from the drop-down Actions menu, and will bring up a dialog box that asks if we are sure we want to terminate. In the CLI, it asks us to type in the full name of the environment to confirm.



If you terminate this environment, the URL http://webnotifier-uat.us-west-2.elasticbeanstalk.com will no longer be accessible and any attached Amazon RDS DB instance will be deleted. If you want to save your data, create a snapshot before terminating your environment. Learn more.

Are you sure you want to terminate the environment WebNotifier-uat?

After clicking on “Terminate”, we are taken back to the dashboard, where our environment will be grey instead of its healthy green color.



WebNotifier-uat (Terminated)

Environment tier: Web Server

Running versions: app-11e1-160122_101906

Last modified: 2016-06-26 14:14:47 UTC-0700

URL: webnotifier-uat.us-west-2.elasticbeanstalk...

Observations

In the web interface, we're able to create Elasticbeanstalk applications and environments. We can see the health of the environments and their logs. While we are unable to do any .ebextensions, any clones of the environments that already have the extensions will not re-run the extensions. It assumes that all of the extensions are already applied within the source environment.

Experiment 3: Testing Web Integration

Experimentation

Next, we'll test integration between the web console and the CLI. What we want to do is create our application using the web console. I've created one called 'ad1app'. When we go into our code directory and begin an "eb init," we'll be asked which environment to use. As long as we stay within the correct region, we'll be given the correct choices. Again, the CLI detects that we're using a PHP application.

```
$ rm -r .elasticbeanstalk
$ eb status
ERROR: This directory has not been set up with the EB CLI
You must first run "eb init".
$ eb init


Select a default region
1) us-east-1 : US East (N. Virginia)
2) us-west-1 : US West (N. California)
3) us-west-2 : US West (Oregon)
4) eu-west-1 : EU (Ireland)
5) eu-central-1 : EU (Frankfurt)
6) ap-southeast-1 : Asia Pacific (Singapore)
7) ap-southeast-2 : Asia Pacific (Sydney)
8) ap-northeast-1 : Asia Pacific (Tokyo)
9) ap-northeast-2 : Asia Pacific (Seoul)
10) sa-east-1 : South America (Sao Paulo)
11) cn-north-1 : China (Beijing)
```

```
(default is 3): 3


Select an application to use
1) ad1app
2) WebNotifier
3) [ Create new Application ]
(default is 3): 1


It appears you are using PHP. Is this correct?
(y/n): y


Select a platform version.
1) PHP 5.4
2) PHP 5.5
3) PHP 5.6
4) PHP 5.3
(default is 1): 3
Do you want to set up SSH for your instances?
(y/n): y


Select a keypair.
1) awskeypair
2) [ Create new KeyPair ]
(default is 10): 1
$
```

We've initialized the Elasticbeanstalk for the application directory using the CLI, and we've created the application using the web console. All that's left is to create the environment. We'll create two environments in two different ways. First, let's create our dev environment using the CLI. We're also going to introduce the –cname option, which gives us control over the short name to access the application. Because our application is in the us-west-2 region, giving it a cname of ad1app-dev will put our application at http://ad1app-dev.us-west-2.elasticbeanstalk.com.

```
$ eb create --cname ad1app-dev ad1app-dev
Creating application version archive "app-db0a-160208_095739".
Uploading ad1app/app-db0a-160208_095739.zip to S3. This may take a
Upload Complete.
Environment details for: ad1app-dev
  Application name: ad1app
  Region: us-west-2
  Deployed Version: app-db0a-160208_095739
  Environment ID: e-xb9u3hk9pq
  Platform: 64bit Amazon Linux 2015.09 v2.0.6 running PHP 5.6
  Tier: WebServer-Standard
  CNAME: ad1app-dev.us-west-2.elasticbeanstalk.com
  Updated: 2016-06-25 16:57:46.355000+00:00
Printing Status:
INFO: createEnvironment is starting.
INFO: Using elasticbeanstalk-us-west-2-550807172100 as Amazon S3 s
nt data.
INFO: Created security group named: sg-c88e5aaf
INFO: Created load balancer named: awseb-e-x-AWSEBLoa-1XTQRB66SVHH
INFO: Environment health has transitioned to Pending. There are no
INFO: Created security group named: awseb-e-xb9u3hk9pq-stack-AWSEB
INFO: Created Auto Scaling launch configuration named: awseb-e-xb9
ingLaunchConfiguration-VU52YN76FFLN
INFO: Added instance [i-7e513aa4] to your environment.
INFO: Waiting for EC2 instances to launch. This may take a few min
INFO: Created Auto Scaling group named: awseb-e-xb9u3hk9pq-stack-A
HM9MDE
INFO: Created Auto Scaling group policy named: arn:aws:autoscaling
lingPolicy:b8a92633-edd5-4ed1-93c9-274fbe1b39d5:autoScalingGroupNa
-AWSEBAutoScalingGroup-AJ5HS2HM9MDE:policyName/awseb-e-xb9u3hk9pq-
UpPolicy-1P2YWEEV525Z2
INFO: Created Auto Scaling group policy named: arn:aws:autoscaling
lingPolicy:6753cd0f-3074-478e-8654-cfb93729bdd0:autoScalingGroupNa
-AWSEBAutoScalingGroup-AJ5HS2HM9MDE:policyName/awseb-e-xb9u3hk9pq-
DownPolicy-4CSQW9Y08XE8
INFO: Created CloudWatch alarm named: awseb-e-xb9u3hk9pq-stack-AWS
UT7YQKSL
INFO: Created CloudWatch alarm named: awseb-e-xb9u3hk9pq-stack-AWS
NYYPQX
INFO: Environment health has transitioned from Pending to Ok.
INFO: Successfully launched environment: ad1app-dev


$ eb status
Environment details for: ad1app-dev
  Application name: ad1app
  Region: us-west-2
  Deployed Version: app-db0a-160208_095739
  Environment ID: e-xb9u3hk9pq
  Platform: 64bit Amazon Linux 2015.09 v2.0.6 running PHP 5.6
```

```
    Tier: WebServer-Standard
    CNAME: ad1app-dev.us-west-2.elasticbeanstalk.com
    Updated: 2016-06-25 17:02:35.331000+00:00
    Status: Ready
    Health: Green
  $ eb ssh
  [ec2-user@ip-172-3-2-4 ~]$ grep deadite /etc/group
  deadite:x:118:williama
```

Our first environment is created. Let’s create a production environment in the Web UI. When we log in and browse to the “Elastic Beanstalk” service, we see our WebNotifier App, with one green box containing our dev environment. We’ll create the second environment in the Web UI by clicking on the environment “ad1app-dev”. In the new pane, there’s a dropdown that says “Actions.” From there, we click on “Clone Environment”, answer a few questions about our new environment, giving it the ad1app-prod name, and we now have a production environment that’s separate from ad1app-dev, but looks exactly like it. When we changed the name in the Web UI, it also auto-completed the environment URL, so when we browse there, we see everything the same as it is in dev.

The problem now is that we need to publish to this environment. We configured our CLI to publish to the dev environment, so how do we publish to production? The workflow for Elasticbeanstalk is the solution. When developing on an Elasticbeanstalk-controlled application, you make the changes locally, then push to your dev environment with the “eb deploy” command. You can then view your changes on the dev environment, and if everything looks and operates correctly, you can select the new environment to push onto using the “eb use” command. This command requires the environment as its last argument. To find out the list of environments to choose from, we use the “eb list” command.

```
$ eb list
* ad1app-dev
ad1app-prod
$ eb use ad1app-prod
$ eb list
ad1app-dev
* ad1app-prod
$
```

The asterisk indicates which environment you are currently working on. When using Elasticbeanstalk, it’s a good idea to get into the practice of checking this periodically to ensure you’re not publishing onto the wrong environment.

Observations

Our integration between the CLI and the Web UI works well, allowing us to use the Web UI to do some modifications after we've customized everything with our first init. With this, we can also delegate control of the environment build-outs to a build team, or person, while the developers can focus on developing. Next, we're going to test customizing the PHP configuration, specifically, adding custom features to the /etc/php.ini file.

Experiment 4: Testing PHP Configuration

Experimentation

Let's pose a hypothetical: our system is created and everything is ready to go, but when I browse to my CNAME, there's a 500 error. What's wrong? Well, our web site is dependent on a PHP setting called "auto_prepend_file". This can be set in a .htaccess file, but we didn't include that in our code. We can also set it this in the /etc/php.ini file. Also, changes within a php.ini file are minor changes that shouldn't require a rebuild or reboot, so we're going to try doing this without rebuilding or rebooting the machine, so we'll try several options. First, we'll use the "eb deploy" command.

```
$ mkdir .ebextensions
$ cd .ebextensions/
$ vi commands.config
files:
  "/home/ec2-user/.netrc":
    mode: "000600"
    owner: ec2-user
    content: |
      machine bitbucket.org login haxxon_hax password abc12345
  "/home/root/.netrc":
    mode: "000600"
    owner: root
    content: |
      machine bitbucket.org login haxxon_hax password abc12345
  "/tmp/getmyapp.sh":
    mode: "000755"
    content: |
      #!/bin/sh
      git init
      git remote add origin https://bitbucket.org/haxxon_hax/allsta
      git fetch
      rm /home/ec2-user/.netrc
  "/tmp/fixphp.sh":
    mode: "000755"
    content: |
      #!/bin/sh
      echo 'auto_prepend_file = "/var/app/current/includeManager/i
> /etc/php.ini
commands:
  gitmyappfrombitbucket:
    cwd: /var/app/current
    command: /tmp/getmyapp.sh
  fixphp:
    cwd: /var/app/current
    command: /tmp/fixphp.sh
$ eb deploy
Creating application version archive "app-db0a-160208_101739".
Uploading ad1app/app-db0a-160208_101739.zip to S3. This may take a
Upload Complete.
INFO: Environment update is starting.
INFO: Deploying new version to instance(s).
INFO: Environment health has transitioned from Severe to Ok.
INFO: New application version was deployed to running EC2 instance
INFO: Environment update completed successfully.


$ eb ssh
INFO: Attempting to open port 22.
INFO: SSH port 22 open.
 _____ _           _   _      ____                       _        -
| ____| | __ _ ___| |_(_) ___| __ )  ___  __ _ _ __  ___| |_ __ _|
|  _| | |/ _` / __| __| |/ __|  _ \ / _ \/ _` | '_ \/ __| __/ _` |
| |___| | (_| \__ \ |_| | (__| |_) |  __/ (_| | | | \__ \ || (_| |
|_____|_|\__,_|___/\__|_|\___|____/ \___|\__,_|_| |_|___/\__\__,_|_
                                       Amazon Linux AMI


This EC2 instance is managed by AWS Elastic Beanstalk. Changes made
WILL BE LOST if the instance is replaced by auto-scaling. For more
on customizing your Elastic Beanstalk environment, see our documen
http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/customize-co
[ec2-user@ip-172-3-1-5 ~]$ grep auto_prepend_file /etc/php.ini
auto_prepend_file =
auto_prepend_file = "/var/app/current/includeManager/includeManage
[ec2-user@ip-172-3-1-5 ~]$ ls /tmp/fixphp.sh
/tmp/fixphp.sh
[ec2-user@ip-172-3-1-5 ~]$ exit
```

The new command was placed and run. Ideally, this is not the best way to make the changes necessary in the /etc/php.ini file. The best scenario is to create a file resource in the .ebextensions directory, and point it to a source, which will overwrite the php.ini file. The source must be a URL that the Elasticbeanstalk instance can get to.

Notice that we're using the "eb deploy" script to push out changes to the .ebextensions; we're not pushing to a git repository, then pulling from a git repository. In the documentation on "eb deploy" from AWS, using a git-initialized application requires commits to be done before running "eb deploy". We'll take a look at this next, initializing our directory with git, then changing our environment by adding a user in the users.config file.

```
WebNotifier $ git init
WebNotifier $ git remote add origin https://bitbucket.org/haxxon_ha
WebNotifier $ git fetch
WebNotifier $ git pull origin master
[...]
WebNotifier $ vi .ebextensions/users.config
groups:
  deadite:
    gid: "118"
users:
  williama:
    groups:
     - deadite
    uid: "2000"
    homeDir: "/home/williama"
  brownc:
    groups:
     - deadite
    uid: "2001"
    homeDir: "/home/brownc"
```

```
WebNotifier $ eb deploy
ERROR: This branch does not have a default environment. You must e
t by typing "deploy my-env-name" or set a default environment by ty
WebNotifier $ cat .elasticbeanstalk/config.yml
branch-defaults:
  default:
    environment: ad1app-dev
global:
  application_name: ad1app
  default_ec2_keyname: awskeypair
  default_platform: PHP 5.6
  default_region: us-west-2
  profile: eb-cli
  sc: null
WebNotifier $
```

Initializing git caused our “eb deploy” to realize something is different with the environment, but the .elasticbeanstalk/config.yml file still has the same environment specified. We can specify the environment at the CLI.

```
WebNotifier $ eb list
ad1app-dev
WebNotifier $ eb use WebNotifier-dev
WebNotifier $ eb deploy
WARNING: You have uncommitted changes.
Creating application version archive "app-bcb7-160701_143746".
Uploading WebNotifier/app-bcb7-160701_143746.zip to S3. This may ta
Upload Complete.
INFO: Environment update is starting.
INFO: Deploying new version to instance(s).
INFO: New application version was deployed to running EC2 instance
INFO: Environment update completed successfully.
WebNotifier $ eb ssh
[...]
[ec2-user@ip-172-3-1-5]$ id brownc
id: brownc: no such user
[ec2-user@ip-172-3-1-5]$ tail -2 /etc/php.ini
auto_prepend_file = "/var/app/current/includeManager/includeManage
auto_prepend_file = "/var/app/current/includeManager/includeManage
[ec2-user@ip-172-3-1-5]$ ls /tmp/fixphp*
```

```
/tmp/fixphp.sh  /tmp/fixphp.sh.bak
[ec2-user@ip-172-3-1-5]$ exit
```

Well, it doesn’t look like an “eb deploy” created the user. Any changes that we make to our extensions look to be lost. The /etc/php.ini file has an additional auto_prepend_file entry in it, which means that our extensions were looked at, files created, and run. Elasticbeanstalk also automatically backs up our old scripts in /tmp. Notice at the head of our “eb deploy” output, though, that we have uncommitted changes. Let’s run an “eb deploy” again, but push our changes up to git.

```
WebNotifier $ git add .
WebNotifier $ git commit -m "added a user."
[master a26adb0] added a user.
 1 file changed, 5 insertions(+)
WebNotifier $ git push origin master
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 396 bytes | 0 bytes/s, done.
Total 4 (delta 2), reused 0 (delta 0)
To https://bitbucket.org/haxxon_hax/allstars.git
   bcb7bb0..a26adb0  master -> master
WebNotifier $ eb deploy
reating application version archive "app-a26a-160701_144900".
Uploading WebNotifier/app-a26a-160701_144900.zip to S3. This may ta
Upload Complete.
INFO: Environment update is starting.
INFO: Environment health has transitioned from Ok to Info. Applicat
nning for 11 seconds).
INFO: Deploying new version to instance(s).
INFO: New application version was deployed to running EC2 instances
INFO: Environment update completed successfully.

WebNotifier $ eb ssh
INFO: Attempting to open port 22.
INFO: SSH port 22 open.
 _____ _           _   _      ____                       _        _
| ____| | __ _ ___| |_(_) ___| __ )  ___  __ _ _ __  ___| |_ __ _| |
|  _| | |/ _` / __| __| |/ __|  _ \ / _ \/ _` | '_ \/ __| __/ _` | |
| |___| | (_| \__ \ |_| | (__| |_) |  __/ (_| | | | \__ \ || (_| | |
|_____|_|\__,_|___/\__|_|\___|____/ \___|\__,_|_| |_|___/\__\__,_|_
                                                  Amazon Linux AMI


This EC2 instance is managed by AWS Elastic Beanstalk. Changes made
WILL BE LOST if the instance is replaced by auto-scaling. For more
on customizing your Elastic Beanstalk environment, see our document
http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/customize-co
[ec2-user@ip-172-3-1-5 ~]$ id brownc
uid=2001(brownc) gid=494(brownc) groups=494(brownc),118(deadite)
[ec2-user@ip-172-3-1-5 ~]$ exit
```

When we ran the “eb deploy” after committing everything in Git, we didn’t get the “commit” warning message. After logging in, we have successfully added the brownc user. When an Elasticbeanstalk environment is associated with a git repository, any changes to the environment code, including the ebextensions, must be committed before running “eb deploy”, or they will not show up on the environment in AWS.

Observations

We are able to make changes to our php environment, so we can now customize how our system behaves. While we used custom commands, it would be preferable to place the files that we need in a publicly-accessible environment (usually an S3 bucket), and refer to that file by its URL. We've also noticed that, when our environment is under Git control, we need to commit changes before deploying to the environment. In the next section, we will look at how to deploy our application using an S3 bucket.

Experiment 5: S3 Storage

Experimentation

Now, let's get back to one of our original problems. We want to put 47G of data into our Elasticbeanstalk. When we create the Elasticbeanstalk, we know that it zips the site and uploads it into our instance. The first question to ask is, "where does it put the zip file?" That's an easy one. It puts it in our /tmp folder, so we'd have to make sure we have enough from for the zip file in our /tmp folder. The second question is, "if it zips all that data, does it transfer it all at once?" The answer: yes, it does! And, at the time of this document, there seems to be a limit of about 5G through that pipe until it dies. This means that if we have a large amount of data to transfer into your Elasticbeanstalk, we need to find an alternative way to get the data up there. Luckily, there is another way! We've explored using 'git' to pull the data from a git repository. Now, let's explore using an S3 bucket.

The Elasticbeanstalk CLI allows us to specify our own S3 bucket for the beanstalk itself. When we do this, the beanstalk uses that bucket for our application instead of creating a new one. Note that this also eliminates our original problem that the files are overwritten after our 'git fetch'. The first thing we'll do is create our bucket. Each amazon S3 bucket has a namespace that is global across all of AWS, so be sure to make yours unique (AWS will tell you if it’s not). The other advantage that this gives us is that we can give the S3 a descriptive name, so we can distinguish it from the other S3 buckets, and we know exactly what it is and who it’s for. Like the Elasticbeanstalk, S3 also has limits to the upload size, but I’ve been able to upload 13G via the Web UI without any problems (but it takes a really long time).

After speaking with Amazon, the S3 command-line should be able to upload the entire site. This is done, however, using the S3 route, which does charge per request, whereas EC2 access to S3 does not (charges for keeping storage still accrues).[3] . However, the process is as such:

1. Create an AWS S3 bucket.
2. From the command-line, copy (or sync) your files to the AWS S3 bucket:

```
$ ls -lh /var/www/html_site.zip
-rw-r--r--. 1 nobody nobody 13G Feb 17 13:25 /var/www/html_site.zi
$ aws s3 cp /var/www/html_site.zip s3://webnotifier-new-bucket/
[...]
Completed 54 part(s) with ... file(s) remaining
[...]
Completed 353 part(s) with ... file(s) remaining
[...]
Completed 1025 of 1548 part(s) with 1 file(s) remaining
[...]
upload: ./html_site.zip to s3://webnotifier-new-bucket/html_site.z
```

For Elasticbeanstalk to use the S3 as a source, the site has to be one complete zip file. Once the files is copied, we can modify the .ebextensions to point to the bucket:

```
WebNotifier $ eb init
Application WebNotifier has been created.
WebNotifier $ vi .ebextensions/bucket.config
sources:
  /var/app/WebNotifier: http://s3.amazonaws.com/webnotifier-new-buc
WebNotifier $ eb status
ERROR: Environment "WebNotifier-dev" not Found.
WebNotifier $ eb create WebNotifier-dev
Creating application version archive "app-11e1-160525_103658".
Uploading WebNotifier/app-11e1-160525_103658.zip to S3. This may ta
Upload Complete.
Application WebNotifier has been created.
Environment details for: WebNotifier-dev
  Application name: WebNotifier
  Region: us-west-2
  Deployed Version: app-11e1-160525_103658
  Environment ID: e-ipzhprfbnp
  Platform: 64bit Amazon Linux 2016.03 v2.1.2 running PHP 5.6
  Tier: WebServer-Standard
  CNAME: UNKNOWN
  Updated: 2016-06-25 17:37:03.590000+00:00
Printing Status:
INFO: createEnvironment is starting.
INFO: Using elasticbeanstalk-us-west-2-550807172100 as Amazon S3 st
nt data.
INFO: Created security group named: sg-cd35c1ab
INFO: Created load balancer named: awseb-e-i-AWSEBLoa-XKJGTB19HSGO
INFO: Environment health has transitioned to Pending. Initializatio
 20 seconds). There are no instances.
INFO: Created security group named: awseb-e-ipzhprfbnp-stack-AWSEBS
INFO: Created Auto Scaling launch configuration named: awseb-e-ipzh
ingLaunchConfiguration-X4RINE18IBB8
INFO: Created Auto Scaling group named: awseb-e-ipzhprfbnp-stack-AW
YCF9PV
INFO: Waiting for EC2 instances to launch. This may take a few minu
INFO: Created Auto Scaling group policy named: arn:aws:autoscaling
lingPolicy:6b107cd8-2c2c-47bc-b2f1-d2c532d7df76:autoScalingGroupNam
-AWSEBAutoScalingGroup-7CCZ0OYCF9PV:policyName/awseb-e-ipzhprfbnp-s
UpPolicy-JWHW057JV7CD
INFO: Created Auto Scaling group policy named: arn:aws:autoscaling
lingPolicy:799d761b-def4-434f-93e7-3c610eef15da:autoScalingGroupNam
-AWSEBAutoScalingGroup-7CCZ0OYCF9PV:policyName/awseb-e-ipzhprfbnp-s
DownPolicy-1H9MG40IV4L6J
INFO: Created CloudWatch alarm named: awseb-e-ipzhprfbnp-stack-AWSE
VJD37T
INFO: Created CloudWatch alarm named: awseb-e-ipzhprfbnp-stack-AWSE
KH69ATRX
INFO: Added instance [i-b391801f] to your environment.
INFO: Environment health has transitioned from Pending to Ok. Initi
```

```
onds ago and took 3 minutes.
INFO: Successfully launched environment: WebNotifier-dev
```

The build looks okay, so let’s now take a look at the EC2 instance itself:

```
WebNotifier $ eb ssh
INFO: Attempting to open port 22.
INFO: SSH port 22 open.
The authenticity of host '52.3.12.21 (52.3.12.21)' can't be establi
ECDSA key fingerprint is cc:30:a2:a7:a2:4d:70:fa:d6:ae:71:d8:20:bc
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.3.12.21' (ECDSA) to the list of know
 _____ _           _   _      ____                       _        -
| ____| | __ _ ___| |_(_) ___| __ )  ___  __ _ _ __  ___| |_ __ _|
|  _| | |/ _` / __| __| |/ __|  _ \ / _ \/ _` | '_ \/ __| __/ _` |
| |___| | (_| \__ \ |_| | (__| |_) |  __/ (_| | | | \__ \ || (_| |
|_____|_|\__,_|___/\__|_|\___|____/ \___|\__,_|_| |_|___/\__\__,_|_
                                           Amazon Linux AMI


This EC2 instance is managed by AWS Elastic Beanstalk. Changes made
WILL BE LOST if the instance is replaced by auto-scaling. For more
on customizing your Elastic Beanstalk environment, see our document
http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/customize-con
[ec2-user@ip-172-3-8-9 ~]$ ls -l /var/app/WebNotifier
ls: cannot access /var/app/WebNotifier: No such file or directory
[ec2-user@ip-172-3-8-9 ~]$
[ec2-user@ip-172-3-8-9 ~]$ du -sh /var/app/current
88K         /var/app/current
[ec2-user@ip-172-3-8-9 log]$ sudo grep -i webnotifier-new-bucket *
grep: healthd: Is a directory
grep: httpd: Is a directory
secure:May 25 17:47:41 ip-172-3-8-9 sudo: ec2-user : TTY=pts/0 ; PW
OMMAND=/bin/grep -i webnotifier-new-bucket boot.log cfn-hup.log cfn
 cfn-wire.log cloud-init.log cloud-init-output.log cron dmesg eb-ac
l.log eb-cfn-init.log eb-commandprocessor.log eb-publish-logs.log e
lastlog maillog messages secure spooler wtmp yum.log
[ec2-user@ip-172-3-8-9 log]$
```

It doesn’t look like it extracted our new 13G bucket, nor is there any tracking of this in the logs. This didn’t work, but we also haven’t committed anything to the repository. We didn’t get a warning about this because we were creating an environment, rather than deploying to one. Before we commit, let’s run a deploy.

```
WebNotifier $ eb deploy
Creating application version archive "app-a26a-160701_155713".
Uploading WebNotifier/app-a26a-160701_155713.zip to S3. This may ta
Upload Complete.
[...]
```

The “eb deploy” command didn’t warn us that we haven’t committed our changes, but let’s try to add our changes to the repository and deploy again.

```
WebNotifier $ git add .
WebNotifier $ git commit -m "Added S3 bucket"
[master 1d56364] Added S3 bucket
 1 file changed, 2 insertions(+)
 create mode 100644 .ebextensions/bucket.config
WebNotifier $ git push origin master
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 459 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To https://bitbucket.org/haxxon_hax/allstars.git
   a26adb0..1d56364  master -> master
WebNotifier $ eb deploy
Creating application version archive "app-1d56-160701_160018".
Uploading WebNotifier/app-1d56-160701_160018.zip to S3. This may ta
Upload Complete.
INFO: Environment update is starting.
INFO: Deploying new version to instance(s).
ERROR: [Instance: i-fba9bb6e] Command failed on instance. Return co
source file (not zip or tarball): http://s3.amazonaws.com/webnotifi
p.
```

```
EBExtension failed. For more detail, check /var/log/eb-activity.lo
INFO: Command execution completed on all instances. Summary: [Succ
ERROR: Unsuccessful command execution on instance id(s) 'i-fba9bb6
ERROR: Failed to deploy application.
```

This time, we get an error and a failure, indicating that Elasticbeanstalk did not like our source file. Checking the documentation at http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/customize-containers-ec2.html#linux-sources, there is a note that the S3 bucket should be publically accessible. Going back to the S3 settings, we can change the permissions to allow “Everyone” to “List”, and enable “Static Website Hosting” (note the URL prefix: http://). After making these changes, we can try to rebuild the environment.

```
WebNotifier $ eb deploy
[...]
ERROR: Failed to deploy application.
WebNotifier $
```

Again, the deployment failed. The url we’re trying to pull from is an S3 bucket, which points to the correct location. Let’s test by trying to pull down the html_site.zip file directly using curl or wget.

```
WebNotifier $ wget http://s3.amazonaws.com/webnotifier-new-bucket/
[...]
HTTP request sent, awaiting response... 301 Moved Permanently
```

Well, it's mentioned that it's moved permanently. The number 301 correlates to a redirect, meaning that if we browse to that site, we would most likely get more information. From the browser, we see:

```
<Error>
  <Code>PermanentRedirect</Code>
  <Message>The bucket you are attempting to access must be address
oint. Please send all future requests to this endpoint.</Message>
  <Bucket>wnawss3bucket</Bucket>
  <Endpoint>wnawss3bucket.s3.amazonaws.com</Endpoint>
  <RequestId>C140F1EDB042EFD1</RequestId>
  <HostId>{redacted}</HostId>
</Error>
```

So, now we're getting closer. One noticeable thing is that we copied the S3 URL directly from the ebextensions documentation page, and just changed the bucket name and the zip file. Attempting to access that URL, we are unable to get to the file. Browsing back to our S3, if we click on the zip file, we'll find that we need to grant open/download permissions to Everyone. Additionally, the link provided includes the region: http://s3-us-west-2.amazonaws.com/webnotifier-new-bucket/html_site.zip. We can change this in the bucket.config file, then try to redeploy (we can deploy since the system hasn't actually deployed anything yet):

```
WebNotifier $ vi .ebextensions/bucket.config
[...]
WebNotifier $ git add .
WebNotifier $ git commit -m "Changed bucket location"
[master 6d3267f] Changed bucket location
 1 file changed, 1 insertion(+), 1 deletion(-)
WebNotifier $ git push -u origin master
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
```

```
Writing objects: 100% (4/4), 467 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To https://bitbucket.org/haxxon_hax/allstars.git
   1d56364..6d3267f  master -> master
Branch master set up to track remote branch master from origin.
WebNotifier $
NewWebNotifier $ eb deploy
Creating application version archive "app-160525_135320".
Uploading NewWebNotifier/app-160525_135320.zip to S3. This may tak
Upload Complete.
INFO: Environment update is starting.
INFO: Deploying new version to instance(s).
ERROR: [Instance: i-123cedcf] Command failed on instance. Return c
trieve https://s3-us-west-2.amazonaws.com/webnotifier-new-bucket/h
 space left on device.
EBExtension failed. For more detail, check /var/log/eb-activity.lo
INFO: Command execution completed on all instances. Summary: [Succ
ERROR: Unsuccessful command execution on instance id(s) 'i-123cedc
ERROR: Failed to deploy application.

ERROR: Failed to deploy application.
NewWebNotifier $
```

“No space left on device” means that our zip archive is too large for where we’re trying to put it. We’re going to try one more time, placing it in the /var/app/current directory.

```
WebNotifier $ vi .ebextensions/bucket.config
sources:
  /var/app/current: http://s3-us-west-2.amazonaws.com/webnotifier-
WebNotifier $ git add .
WebNotifier $ git commit -m "Changed bucket location"
[master 6d3267f] Changed bucket location
 1 file changed, 1 insertion(+), 1 deletion(-)
WebNotifier $ git push -u origin master
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 467 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To https://bitbucket.org/haxxon_hax/allstars.git
   1d56364..6d3267f  master -> master
Branch master set up to track remote branch master from origin.
```

```
WebNotifier $ eb deploy
Creating application version archive "app-6d32-160701_162540".
Uploading WebNotifier/app-6d32-160701_162540.zip to S3. This may ta
Upload Complete.
INFO: Environment update is starting.
INFO: Deploying new version to instance(s).

ERROR: Timed out while waiting for command to Complete. The timeout
meout option.
WebNotifier $
```

This time, it times out. We'll deploy again, increasing the timeout value with the "–timeout" switch. 20 minutes should do it.

```
WebNotifier $ eb deploy --timeout 18
Creating application version archive "app-6d32-160701_164221".
Uploading WebNotifier/app-6d32-160701_164221.zip to S3. This may ta
Upload Complete.
INFO: Environment update is starting.
INFO: Deploying new version to instance(s).
INFO: Batch 1: Starting application deployment on instance(s) [i-5e
INFO: Batch 1: De-registering instance(s) from the load balancer an
ut of service.
INFO: Batch 1: Starting application deployment command execution.
WARN: The following instances have not responded in the allowed cor
ght still finish eventually on their own): [i-5ed40ef1].
INFO: Batch 1: Completed application deployment command execution.
INFO: Command execution completed on 1 of 2 instances in environmer
ERROR: Unsuccessful command execution on instance id(s) 'i-5ed40ef1
Leaving instances running for troubleshooting. Instances have not b
lancer.
ERROR: Failed to deploy application.
```

This takes a long time, but eventually errors out. When we log into the instance and check the /var/log/eb-activity logs, we find a "no space left on device" error. We'll try to deploy the app again using the zip file, but to make sure that we get a clean build, we'll terminate the environment, destroy the app, then create a new one.

```
WebNotifier3 $ cp -rp ../NewWebNotifier/.ebextensions ./
WebNotifier3 $ eb init
[...]
WebNotifier3]$ eb create WebNotifier3-dev
Creating application version archive "app-160525_142849".
Uploading WebNotifier3/app-160525_142849.zip to S3. This may take a
Upload Complete.
Environment details for: WebNotifier3-dev
  Application name: WebNotifier3
  Region: us-west-2
  Deployed Version: app-160525_142849
  Environment ID: e-etmgkjad8q
  Platform: 64bit Amazon Linux 2016.03 v2.1.2 running PHP 5.6
  Tier: WebServer-Standard
  CNAME: UNKNOWN
  Updated: 2016-06-25 21:28:53.169000+00:00
Printing Status:
INFO: createEnvironment is starting.
INFO: Using elasticbeanstalk-us-west-2-550807172100 as Amazon S3 st
nt data.
INFO: Environment health has transitioned to Pending. Initializatio
 7 seconds). There are no instances.
INFO: Created security group named: sg-c45aaea2
INFO: Created load balancer named: awseb-e-e-AWSEBLoa-42XC4RWX0KQW
INFO: Created security group named: awseb-e-etmgkjad8q-stack-AWSEBS
INFO: Created Auto Scaling launch configuration named: awseb-e-etmg
ingLaunchConfiguration-122O6K3BY3OMT
INFO: Added instance [i-3353a1e8] to your environment.

ERROR: The operation timed out. The state of the environment is unl
et using the --timeout option.
```

We got a different error than the last one. The timeout is possibly due to the size of the zip file, retrieving and unzipping it. According to the help page, the –timeout option for “eb create” is in minutes, just like int “eb deploy.” We’ll increase the timeout, setting it for 40.

```
$ mkdir WebNotifier4
$ cd WebNotifier4
WebNotifier4 $ cp ../WebNotifier/index.php ./
```

```
WebNotifier4 $ cp -rp ../WebNotifier/.ebextensions ./
WebNotifier4 $ cat .ebextensions/bucket.config
sources:
  /var/app/WebNotifier: http://s3-us-west-2.amazonaws.com/webnotif
WebNotifier4 $ eb init
[...]
WebNotifier4 $ eb create WebNotifier4-dev --timeout 40
Creating application version archive "app-160525_150847".
Uploading WebNotifier4/app-160525_150847.zip to S3. This may take
Upload Complete.
Environment details for: WebNotifier4-dev
  Application name: WebNotifier4
  Region: us-west-2
  Deployed Version: app-160525_150847
  Environment ID: e-a2anemzege
  Platform: 64bit Amazon Linux 2016.03 v2.1.2 running PHP 5.6
  Tier: WebServer-Standard
  CNAME: UNKNOWN
  Updated: 2016-06-25 22:08:51.459000+00:00
Printing Status:
INFO: createEnvironment is starting.
INFO: Using elasticbeanstalk-us-west-2-550807172100 as Amazon S3 s
nt data.
INFO: Created security group named: sg-2fb34449
INFO: Created load balancer named: awseb-e-a-AWSEBLoa-1JWP1QCYJJZQ
INFO: Environment health has transitioned to Pending. Initializati
 10 seconds). There are no instances.
INFO: Created Auto Scaling launch configuration named: awseb-e-a2a
ingLaunchConfiguration-1ENXEUBQ6Y9SZ
INFO: Added instance [i-005f8edd] to your environment.
INFO: Created Auto Scaling group policy named: arn:aws:autoscaling
lingPolicy:5f8f8065-fddc-4e60-be91-08a609e9b73a:autoScalingGroupNa
-AWSEBAutoScalingGroup-226QJKZWFPRF:policyName/awseb-e-a2anemzege-
DownPolicy-LNINIIRX8K7I
INFO: Created Auto Scaling group policy named: arn:aws:autoscaling
lingPolicy:6e00c147-4a3b-45c9-8857-b15b60db6bf2:autoScalingGroupNa
-AWSEBAutoScalingGroup-226QJKZWFPRF:policyName/awseb-e-a2anemzege-
UpPolicy-O1Z3KWFT35KE
INFO: Created CloudWatch alarm named: awseb-e-a2anemzege-stack-AWS
WI7H5WJ
INFO: Created CloudWatch alarm named: awseb-e-a2anemzege-stack-AWS
K5I4JGDP
WARN: Environment health has transitioned from Pending to Severe.
on 1 instance. 0 out of 1 instance completed (running for 12 minut
or not available for all instances.
WARN: The following instances have not responded in the allowed co
ght still finish eventually on their own): [i-005f8edd].
INFO: Command execution completed on all instances. Summary: [Succ
ERROR: Create environment operation is complete, but with command
e timeout period. For more information, see troubleshooting docume
```

```
INFO: Added instance [i-754bb9ae] to your environment.
WARN: Environment health has transitioned from Severe to Warning. (
ut of 2 instances. Command failed on 1 out of 2 instances.
WARN: Environment health has transitioned from Warning to Degraded
 out of 2 instances. Command failed on 1 out of 2 instances. ELB he
ilable for 1 out of 2 instances.
INFO: Added instance [i-2b6b7b87] to your environment.
INFO: Added instance [i-4247969f] to your environment.

ERROR: The operation timed out. The state of the environment is unl
et using the --timeout option.
WebNotifier4 $ eb ssh


Select an instance to ssh into
1) i-005f8edd
2) i-2b6b7b87
3) i-4247969f
4) i-754bb9ae
INFO: Attempting to open port 22.
INFO: SSH port 22 open.
The authenticity of host '52.40.72.188 (52.40.72.188)' can't be est
ECDSA key fingerprint is 0d:25:3d:4d:3d:71:81:57:46:7e:cd:80:21:e0
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.40.72.188' (ECDSA) to the list of kn
 _____ _           _   _      ____                       _        -
| ____| | __ _ ___| |_(_) ___| __ )  ___  __ _ _ __  ___| |_ __ _|
|  _| | |/ _` / __| __| |/ __|  _ \ / _ \/ _` | '_ \/ __| __/ _` |
| |___| | (_| \__ \ |_| | (__| |_) |  __/ (_| | | | \__ \ || (_| |
|_____|_|\__,_|___/\__|_|\___|____/ \___|\__,_|_| |_|___/\__\__,_|_
                                           Amazon Linux AMI


This EC2 instance is managed by AWS Elastic Beanstalk. Changes made
WILL BE LOST if the instance is replaced by auto-scaling. For more
on customizing your Elastic Beanstalk environment, see our document
http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/customize-con
[ec2-user@ip-172-31-5-39 ~]$ ls -l /var/app/current
total 0
[ec2-user@ip-172-31-5-39 ~]$
```

Again, this times out. The system did recognize that the health was severe and autoscaled it to four instances.

Observations

The autoscaling feature is a definite plus! However, we still don't have a working application from our S3 bucket. We haven't been able to use the S3 storage for a very large file base (although, I think nowadays anything less than 15G isn't that excessive). For our purposes, we're going to mark this one as a failure. With some fiddling, we may be able to piece together about 2-5G of data at a time, but at that point, it just feels too kludgey. To clean up, we'll terminate the environment and delete the application.

```
$ eb terminate
$ aws elasticbeanstalk delete-application --application-name WebNo
```

Experiment 6: Container Commands

Experimentation

We've talked about the commands feature of the ebextensions, but the one problem we ran into was that the commands are run before the application is extracted from the S3 bucket. The other option is to use container commands. These commands are actually run on the staging area of your application before it is deployed to your beanstalk.

```
container_commands:
  container_gitmyappfrombitbucket:
        command: /tmp/getmyapp.sh
        leader_only: true
```

We've set this to leader_only, because this will only be run on the instance that is marked as the leader. Later environments will use the existing application at the time of creation (after the container command has been run). For environments that use Git, this can be a great feature since we only want to follow what the leader is doing, and newer commits may be in the Git repository. Leader instances are the first instance that's run.

```
WebNotifier $ eb init
[...]
WebNotifier $ eb create WebNotifier-prod
[ec2-user@ip-172-31-29-87 current]$ less /var/log/eb-activity.log
[...]
[2016-06-25T18:46:26.522Z] INFO  [2854]  - [Application deployment
tartupStage0/EbExtensionPreBuild/Infra-EmbeddedPreBuild/prebuild_0_
ppfrombitbucket] : Completed activity. Result:
  Initialized empty Git repository in /var/app/current/.git/
  fatal: could not read Username for 'https://bitbucket.org': No su
```

Notice that there's a failure in the application, unable to find the Username for bitbucket.org. Because this script is failing, this may be why our other configurations aren't getting run. Let's fix this script for now, allowing it to pass by bypassing the git commands:

```
files:
  "/home/ec2-user/.netrc":
    mode: "000600"
    owner: ec2-user
    content: |
      machine bitbucket.org login haxxon_hax password abc12345
  "/home/root/.netrc":
    mode: "000600"
    owner: root
    content: |
      machine bitbucket.org login haxxon_hax password abc12345
  "/tmp/getmyapp.sh":
        mode: "000755"
        content: |
          #!/bin/sh
          git init
          #git remote add origin https://bitbucket.org/haxxon_hax/a
          #git fetch
          #rm /home/ec2-user/.netrc
       #rm /home/root/.netrc
```

To ensure everything takes, we re-initialize the application, then create a new environment. This time, however, we'll create a whole different application with the same extensions:

```
NewWebNotifier $ eb init
NewWebNotifier $ eb create NewWebNotifier-dev
Creating application version archive "app-160525_131924".
Uploading NewWebNotifier/app-160525_131924.zip to S3. This may take
Upload Complete.
Environment details for: NewWebNotifier-dev
  Application name: NewWebNotifier
  Region: us-west-2
  Deployed Version: app-160525_131924
  Environment ID: e-3tup5fnfep
  Platform: 64bit Amazon Linux 2016.03 v2.1.2 running PHP 5.6
  Tier: WebServer-Standard
  CNAME: UNKNOWN
  Updated: 2016-06-25 20:19:27.899000+00:00
Printing Status:
INFO: createEnvironment is starting.
INFO: Using elasticbeanstalk-us-west-2-550807172100 as Amazon S3 st
nt data.
INFO: Created security group named: sg-0f788c69
INFO: Created load balancer named: awseb-e-3-AWSEBLoa-633CA7U83W9X
INFO: Created security group named: awseb-e-3tup5fnfep-stack-AWSEBS
INFO: Created Auto Scaling launch configuration named: awseb-e-3tup
ingLaunchConfiguration-FSBSD2IM24QI
INFO: Environment health has transitioned to Pending. Initializatio
 58 seconds). There are no instances.
INFO: Waiting for EC2 instances to launch. This may take a few minu
INFO: Created Auto Scaling group named: awseb-e-3tup5fnfep-stack-AW
6ENP8A
INFO: Created Auto Scaling group policy named: arn:aws:autoscaling
lingPolicy:34f74788-d36b-4577-a487-ca74ecb8ad22:autoScalingGroupNam
```

```
-AWSEBAutoScalingGroup-Q2YRDP6ENP8A:policyName/awseb-e-3tup5fnfep-s
DownPolicy-ZLBJKVHLY7CN
INFO: Created Auto Scaling group policy named: arn:aws:autoscaling
lingPolicy:715930dc-ba24-4fc9-8d5f-b781132d4fbf:autoScalingGroupNa
-AWSEBAutoScalingGroup-Q2YRDP6ENP8A:policyName/awseb-e-3tup5fnfep-s
UpPolicy-NF5ZM80T0UD8
INFO: Created CloudWatch alarm named: awseb-e-3tup5fnfep-stack-AWSE
F8YF0ID
INFO: Created CloudWatch alarm named: awseb-e-3tup5fnfep-stack-AWSE
6ZKEVV
INFO: Added instance [i-123cedcf] to your environment.
ERROR: [Instance: i-123cedcf] Command failed on instance. Return co
source file (not zip or tarball): http://s3.amazonaws.com/webnotifi
p.
EBExtension failed. For more detail, check /var/log/eb-activity.log
INFO: Command execution completed on all instances. Summary: [Succe
WARN: Environment health has transitioned from Pending to Degraded
tances. Initialization completed 27 seconds ago and took 3 minutes
ERROR: Create environment operation is complete, but with errors. F
roubleshooting documentation.
WARN: Environment health has transitioned from Degraded to Severe.
ances. ELB health is failing or not available for all instances.
 -- Events -- (safe to Ctrl+C)
```

Our environment tried to access our S3 bucket, but failed because (again) we forgot to change our source bucket back to the regional URL. Instead, we’ll remove the bucket itself, since we’re trying to replace the entire repo using a git pull.

```
$ mkdir WebNotifier5
$ cd WebNotifier5/
WebNotifier5 $ cp -rp ../WebNotifier/.ebextensions ./
WebNotifier5 $ vi .ebextensions/
bucket.config        commands.config  users.config
WebNotifier5 $ rm .ebextensions/bucket.config
WebNotifier5 $ cp ../WebNotifier/index.php ./
WebNotifier5 $ vi .ebextensions/commands.config
WebNotifier5 $ cat .ebextensions/commands.config
files:
  "/home/ec2-user/.netrc":
    mode: "000600"
    owner: ec2-user
```

```
      content: |
        machine bitbucket.org login haxxon_hax password abc12345
    "/home/root/.netrc":
      mode: "000600"
      owner: root
      content: |
        machine bitbucket.org login haxxon_hax password abc12345
    "/tmp/getmyapp.sh":
      mode: "000755"
      content: |
        #!/bin/sh
        git init
        git remote add origin https://bitbucket.org/haxxon_hax/allsta
        git fetch
        rm /home/ec2-user/.netrc
  container_commands:
    container_gitmyappfrombitbucket:
      command: /tmp/getmyapp.sh
      leader_only: true
  WebNotifier5 $ eb init
  [...]
  WebNotifier5 $ eb create WebNotifier5-dev
  Creating application version archive "app-160525_160813".
  Uploading WebNotifier5/app-160525_160813.zip to S3. This may take a
  Upload Complete.
  Environment details for: WebNotifier5-dev
    Application name: WebNotifier5
    Region: us-west-2
    Deployed Version: app-160525_160813
    Environment ID: e-y72qtbuuiy
    Platform: 64bit Amazon Linux 2016.03 v2.1.2 running PHP 5.6
    Tier: WebServer-Standard
    CNAME: UNKNOWN
    Updated: 2016-06-25 23:08:17.890000+00:00
  Printing Status:
  INFO: createEnvironment is starting.
  INFO: Using elasticbeanstalk-us-west-2-550807172100 as Amazon S3 st
  nt data.
  INFO: Environment health has transitioned to Pending. Initializatio
   3 seconds). There are no instances.
  INFO: Created security group named: sg-0995626f
  INFO: Created load balancer named: awseb-e-y-AWSEBLoa-1ALIT6A06ZDCA
  INFO: Created security group named: awseb-e-y72qtbuuiy-stack-AWSEBS
  INFO: Created Auto Scaling launch configuration named: awseb-e-y72q
  ingLaunchConfiguration-IHVUQ0HJQBB3
  INFO: Added instance [i-8071a05d] to your environment.
  INFO: Created Auto Scaling group named: awseb-e-y72qtbuuiy-stack-AW
  8CHM2X
  INFO: Waiting for EC2 instances to launch. This may take a few minu
  INFO: Created Auto Scaling group policy named: arn:aws:autoscaling
```

```
lingPolicy:fdb6e9bb-7700-4ecf-b966-da2e078999d0:autoScalingGroupNar
-AWSEBAutoScalingGroup-6PF4U28CHM2X:policyName/awseb-e-y72qtbuuiy-s
UpPolicy-1HXB41PT9S02E
INFO: Created Auto Scaling group policy named: arn:aws:autoscaling
lingPolicy:8424b39c-243b-4dc5-ac38-77c3f2ad7538:autoScalingGroupNar
-AWSEBAutoScalingGroup-6PF4U28CHM2X:policyName/awseb-e-y72qtbuuiy-s
DownPolicy-69UREXTMUB0I
INFO: Created CloudWatch alarm named: awseb-e-y72qtbuuiy-stack-AWSI
QUMI52P
INFO: Created CloudWatch alarm named: awseb-e-y72qtbuuiy-stack-AWSI
UNT2AYZ
INFO: Environment health has transitioned from Pending to Ok. Init
onds ago and took 3 minutes.
INFO: Successfully launched environment: WebNotifier5-dev
```

There are no errors now. However, inspecting the system, we find that the command was successful, but the files are nowhere to be found:

```
WebNotifier5 $ eb ssh
INFO: Attempting to open port 22.
INFO: SSH port 22 open.
The authenticity of host '52.35.62.2 (52.35.62.2)' can't be establ
ECDSA key fingerprint is e3:b3:cd:04:30:45:51:69:b9:01:3e:0a:d1:52
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.35.62.2' (ECDSA) to the list of know
 _____ _           _   _      ____                       _        _
| ____| | __ _ ___| |_(_) ___| __ )  ___  __ _ _ __  ___| |_ __ _|
|  _| | |/ _` / __| __| |/ __|  _ \ / _ \/ _` | '_ \/ __| __/ _` |
| |___| | (_| \__ \ |_| | (__| |_) |  __/ (_| | | | \__ \ || (_| |
|_____|_|\__,_|___/\__|_|\___|____/ \___|\__,_|_| |_|___/\__\__,_|_
                                           Amazon Linux AMI


This EC2 instance is managed by AWS Elastic Beanstalk. Changes made
WILL BE LOST if the instance is replaced by auto-scaling. For more
on customizing your Elastic Beanstalk environment, see our document
http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/customize-cor
[ec2-user@ip-172-31-10-142 ~]$ less /var/log/eb-activity.log
[...]
[2016-06-25T23:11:24.025Z] INFO  [2856]  - [Application deployment
pStage0/EbExtensionPostBuild/Infra-EmbeddedPostBuild/postbuild_0_We
d container_gitmyappfrombitbucket] : Starting activity...
```

```
[2016-06-25T23:11:24.028Z] INFO  [2856]  - [Application deployment
pStage0/EbExtensionPostBuild/Infra-EmbeddedPostBuild/postbuild_0_We
d container_gitmyappfrombitbucket] : Completed activity. Result:

  Completed successfully.
[2016-06-25T23:11:24.028Z] INFO  [2856]  - [Application deployment
pStage0/EbExtensionPostBuild/Infra-EmbeddedPostBuild/postbuild_0_We
er_gitmyappfrombitbucket] : Starting activity...
[2016-06-25T23:11:26.365Z] INFO  [2856]  - [Application deployment
pStage0/EbExtensionPostBuild/Infra-EmbeddedPostBuild/postbuild_0_We
er_gitmyappfrombitbucket] : Completed activity. Result:
  Initialized empty Git repository in /var/app/ondeck/.git/
  From https://bitbucket.org/haxxon_hax/allstars
  * [new branch]          master         -> origin/master
  * [new branch]          working        -> origin/working


[...]
[ec2-user@ip-172-31-10-142 ~]$ ls -l /var/app/current/
total 4
-rw-rw-r-- 1 webapp webapp 435 May 25 16:06 index.php
[ec2-user@ip-172-31-10-142 ~]$ ls -l /var/app
total 8
drwxrwxr-x 3 webapp webapp 4096 May 25 23:11 current
drwxr-xr-x 5 webapp webapp 4096 May 25 23:10 support
[ec2-user@ip-172-31-10-142 ~]$
```

Notice later in the logs that the application unzips after our custom command has run:

```
[ec2-user@ip-172-31-10-142 ~]$
[2016-06-25T23:19:37.136Z] INFO  [3197]  - [Application update app-
tage0/AppDeployPreHook/01_unzip.sh] : Completed activity. Result:
  ++ /opt/elasticbeanstalk/bin/get-config container -k app_user
  + EB_APP_USER=webapp
  ++ /opt/elasticbeanstalk/bin/get-config container -k app_staging_
  + EB_APP_STAGING_DIR=/var/app/ondeck
  ++ /opt/elasticbeanstalk/bin/get-config container -k source_bund
  + EB_SOURCE_BUNDLE=/opt/elasticbeanstalk/deploy/appsource/source_
  + rm -rf /var/app/ondeck
  + /usr/bin/unzip -d /var/app/ondeck /opt/elasticbeanstalk/deploy/
  Archive:  /opt/elasticbeanstalk/deploy/appsource/source_bundle
        inflating: /var/app/ondeck/index.php
```

```
         creating: /var/app/ondeck/.ebextensions/
        inflating: /var/app/ondeck/.ebextensions/users.config
        inflating: /var/app/ondeck/.ebextensions/commands.config
  + chown -R webapp:webapp /var/app/ondeck
  + chmod 775 /var/app/ondeck
```

This means that we’re able to pull down our code from the zip file, but the data where we’ve initialized the Elasticbeanstalk has overwritten everything we’ve extracted–or the files that matched, anyways.

Observations

We can conclude that our attempts at getting our large amount of data pulled from a git repository (and active) have been unsuccessful. We're able to run custom commands, in a similar fashion to the regular commands in the .ebextensions directory. However, the code we've extracted into the application directory gets overwritten by the code from the current Elasticbeanstalk local publication directory. Like our S3 attempts and our other Git pull requests, we have been unable to complete many of our objectives. Finally, let's clean up by terminating all our environments, deleting all environments, and deleting our S3 bucket..

Conclusion

In our first part, exploring the CLI, we found that we need to have IAM Identities configured with access keys for using the Elasticbeanstalk CLI. These credentials are placed in our home directory’’s .elasticbeanstalk/aws_credential_file with mode 600. Within our development directory, we had to create the Elasticbeanstalk structure, which includes the .elasticbeanstalk directory, plus an optional .ebextensions directory with additional config files. We can create custom commands to run after the instance is built, but is run before our application is updated with the files. Each Elasticbeanstalk environment creates its own S3 bucket, which houses the data in zipped format, then is transferred to the instance.

Connecting to the environments is very simple, using “eb ssh” to SSH into the instances, and “eb deploy” to update the instances to the latest code. Rebuilding the environment can also be done using “aws elasticbeanstalk rebuild-environment –environment-name <environment-name>”, though this creates new SSH keys, which could become a nuisance for sites that require lots

Using the Web UI, we were able to view health, logs, and monitoring of our environments. We also deleted an environment using this interface. In the Web UI, we were able to see and manipulate instances created through the CLI, provided we’re in the correct region. Logs for the environments are requested through a “request log” links, while event are automatically updated.

We could easily create applications in the AWS Web UI, too, and able to use them with the CLI, which provides an interactive selector. Finally, we found some limitations to our Elasticbeanstalk environments, being unable to deploy custom php.ini files and. We were also neither able to pull down our site from a git repository, nor from an S3 bucket.

We can now annotate our initial checklist:

1. [UNTESTED] A Development Stack that is consistent across all environments.
2. [FAIL] Limits change in our development workflow, or supports our current development workflow:
    1. Git availability
    2. Support existing large site (>13G).
    3. Local accounts on the system.
3. [UNTESTED] Easily promote publication between environments.
4. [FAIL] PHP Customization in the .ini file.

Item 2 isn’t an exact failure, as we could change our workflow using the “eb deploy”, rather than using git; this makes items 2a and 2b null. However, since we couldn’t get our existing large site (failure on #2b), we can consider this a failure. So, it looks like Elasticbeanstalk works if the application has simple requirements, doesn’t use a large repository, and is flexible on the workflow.

Next, we’ll take a look at AWS OpsWorks, which uses Chef to provision EC2 systems. OpsWorks gives us a lot more control over how our EC2 instances are built–and what we can do with them. While AWS OpsWorks gives us more freedom, it does require a lot more knowledge about systems and software

3. Exploring AWS OpsWorks

Introduction

Now that we've determined that AWS Elasticbeanstalk won't work for our DevOps environment, we need to take a step back and reconsider our architecture. But first, let's go over the requirements I'm looking for in my environment:

1. A Development Stack that is consistent across all environments.
2. Limits change in our development workflow, or supports our current development workflow:
    1. Git availability
    2. Support existing large site (>13G).
    3. Local accounts on the system.
3. Easily promote publication between environments.
4. PHP Customization in the .ini file.

As in our Elasticbeanstalk system, items 2c and 3 are ideal if they can be done on-the-fly, but are acceptable in a "during build" capacity.

Our architecture actually requires a centralized location for the PHP files, since we'll be load balancing. In a unix environment, this typically uses NFS to share out its drives to multiple systems. These systems run the same software, where the load balancer manages incoming traffic to balance it between each system, so that one system doesn't get overloaded with all the requests. One of the drawbacks from using OpsWorks over Elasticbeanstalk is that OpsWorks doesn't do autoscaling. If the systems become overloaded, we will have to manually create a new instance.

OpsWorks uses Chef recipes for building the systems. What is Chef, then? It's a deployment tool, similar to Puppet and Ansible, but its platform is different. Chef allows a builder to deploy software across multiple systems using standard templates, also known as recipes. For more information about Chef, go to https://www.chef.io/. Pluralsight has a few good classes on Chef that you might want to check out if you'd like to learn more.

AWS Opsworks works by either implementing its own pre-built recipes, or by allowing the user to provide one's own Chef recipes. To use Chef recipes on OpsWorks, the recipes need to be available to AWS through either an S3 bucket or a Git repository. For our purposes, we'll use BitBucket, but we could use GitHub or a local GitLab server.

In order for AWS Opsworks to take advantage of your recipes, these need to be stored in an accessible location, such as a git repository (GitHub, GitLab, BitBucket), a subversion repository, an HTTP Archive, or an S3 Archive, with each cookbook being at the root level. Within each cookbook are recipes specific for that book. For example, I have a git repository that has a php cookbook with the following contents:

- php/
    - attributes/
    - CHANGELOG.md
    - CONTRIBUTING.md
    - files/
    - libraries/
    - MAINTAINERS.md
    - metadata.json
    - metadata.rb
    - providers/
    - README.md
    - recipes/
    - resources/
    - templates/

This is just the first level; there are other files further below in the directory tree, but I won't bore you with those contents. If you're interested in them, they can be found in the PHP cookbook in the Chef Supermarket (https://supermarket.chef.io/cookbooks/php). If you browse the Chef Supermarket, you'll find that a lot of the work has already been done for you, as far as standard recipes go. When browsing the Chef Supermarket, always verify that the platform for the recipe you're browsing has been tested on Amazon AWS; advanced options allow you to filter by platform.

For our experiments, we'll want to set up our BitBucket account so that OpsWorks can access our Chef recipes. If you don't want your repository to be public, you'll need to set up deployment keys. In your bitbucket account, create the SSH keys and add them as deployment keys (Settings->Deployment Keys). If you're using Github or Gitlab instead, follow the instructions for deployment keys in its documentation. The application of the keys are the same: public key goes to the Git server, while the Private key will gets imported into AWS (more on this later).

```
$ ssh-keygen -t rsa -b 2048
Generating public/private rsa key pair.
Enter file in which to save the key (/home/alanducci/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/alanducci/bitbucket.
Your public key has been saved in /home/alanducci/bitbucket.pub.
The key fingerprint is:
SHA256:dFV2pBe07vQEapUCy6C9kYzzS5vRK2mTBFKGxpxwQzs
The key's randomart image is:
+---[RSA 2048]----+
|  .=+oo . . ..++o|
|   .*= = + + . +o|
|   .E = * + . =..|
|     o = =   +.o |
|        S . o  o.|
|       o B o  o..|
|        X .    ..|
|       . o       |
|                 |
+----[SHA256]-----+
$ cat /home/alanducci/bitbucket
-----BEGIN RSA PRIVATE KEY-----
MII[REDACTED]gY=
-----END RSA PRIVATE KEY-----


$ cat bitbucket.pub
ssh-rsa AAAA[REDACTED]Rl
$
```

Copy the public key into the bitbucket account and give it a name. The private key will be needed later to copy into your stack settings.

If you’re already familiar with Chef, then most of this will come easy. If you’re not familiar with Chef, and have some programming skills, this will probably still come easy. AWS Opsworks removes the knife command from the recipe. This isn’t to say that the knife command can’t be used to verify your recipes, but rather, the deployments are pushed by the AWS Opsworks console or command-line.

Experimentation

Experiment 1: NFS Server

Experimentation

We'll start with the simplest experiment, an NFS server. Within the Chef Supermarket, there is an NFS cookbook, but we're going to build out our own recipe. One reason we do this is because many of the Chef Supermarket cookbooks rely on environments or roles, which AWS Opsworks does not entirely support. Rather, AWS parallels Chef roles with AWS OpsWorks Layers. AWS OpsWorks also provides compatibility with role attributes. More information can be found on this on the AWS OpsWorks User Guide.

For our Chef NFS cookbook, we need to know the requirements for a standard NFS server:

1. rpcbind enabled and running
2. NFS daemon (nfs service) enabled and running.

Chef typically uses Ruby for its file formats. Let's first create the directory structure:

```
$ mkdir opsworks
$ cd opsworks
$ mkdir alanscookbook
$ cd alanscookbook
$ cat > metadata.rb << __END__
name                      "alanscookbook"
maintainer                "Alan Landucci-Ruiz"
maintainer_email  "landucci@something-magical.net"
license                   "GNU"
description               "Configures and starts Alans OpsWorks Stacks'
version                   "1.0.0"
__END__
$ mkdir recipes
```

For AWS, this is the least amount that is needed, with the exception of the recipes. All recipe books go into the top-level of our git repository, which, in this case is the “opsworks” directory. Now, we need to create our NFS recipe.

```
$ cd recipes
$ cat > nfs.rb << __END__
service 'rpcbind' do
  action [:enable, :start]
  not_if "chkconfig --list rpcbind | grep -qx 'on'"
end


service 'nfs' do
  action [:enable, :start]
  not_if "chkconfig --list nfs | grep -qx 'on'"
end


template '/tmp/create_nfs_exports.sh' do
  source 'create_nfs_exports.sh'
  group 'root'
  owner 'root'
  mode '0755'
end


execute 'create_nfs_exportssh' do
  user 'root'
  command '/tmp/create_nfs_expots.sh'
end
__END__
```

We’ve pushed the code to the git repository so that AWS OpsWorks can access it. The Ruby file is very easy to understand, but let’s take a look at the code by sections:

```
service 'rpcbind' do
  action [:enable, :start]
  not_if "chkconfig --list rpcbind | grep -qx 'on'"
end
```

This code tells Chef that we're going to work on the 'rpcbind' resource. The action line tells Chef to enable, then start the service. Note that these are sequential. The third line tells Chef (before it gets to the "end" keyword) not to perform the actions if the service is already enabled. Because AWS OpsWorks does these at build/boot time, we don't need to worry about if someone manually enabled rpcbind, but didn't start the service, but we also want these recipes to be agnostic of the instance's state, because these can also be run manually for any EC2 instance that's controlled by OpsWorks. The same configuration is created for the nfs service:

```
service 'nfs' do
  action [:enable, :start]
  not_if "chkconfig --list nfs | grep -qx 'on'"
end
```

Templates in Chef are a way to create files that will be used by the system. The PHP resource in the Chef Supermarket, for example, uses templates for its php.ini files. These php.ini files have certain parameters with default settings that can be overridden by creating files in the "attributes" section. For large-scale organizations that have multiple types of services, where customization are needed campus-wide, this is a great benefit, but should be studied more in a Chef exploration, instead of this book. For our NFS purposes, we're going to create a script that will be run, but later will use a php.ini template that is the same across all servers.

```
template '/tmp/create_nfs_exports.sh' do
  source 'create_nfs_exports.sh'
  group 'root'
  owner 'root'
  mode '0755'
end
```

The keyword ‘template’ automatically tells Chef to look in the templates directory of our cookbook, for the source ‘create_nfs_exports.sh’, and place it in ‘/tmp/create_nfs_exports.sh’. For owner and group, this is arbitrary, because it will be run by ‘root’ anyways; we just want to make sure we have execution access, hence “mode ‘0755’”: rwxr-xr-x.

Of course, the create_nfs_exports.sh script needs to be created. Within the Templates directory, the site has to be specified. For simplicity, AWS will automatically look for ‘default’:

```
$ cd ..
$ mkdir templates
$ cd templates
$ mkdir default
$ cd default
$ cat > create_nfs_exports.sh << __END__
#!/bin/sh
EXPORT_LIST=`/bin/df -h | /bin/grep '/export[0-9]' | /bin/awk  '{p

for i in $EXPORT_LIST; do
   /bin/echo "$i            172.31.0.0/16(ro,no_root_squash)" >> /
done

/usr/sbin/exportfs -a
__END__
```

We could also have created a template file called “exports”, and created a template with the following:

```
/export1            172.31.0.0/16(ro,no_root_squash)
```

However, the script create_nfs_export.sh is agnostic of the system, as long as we configure our EBS volumes to be mounted as /export0, /export1, /export2, /export3, […] up to /export9.

The penultimate thing we need to do is put the cookbook in our Git repository, so AWS OpsWorks can use it.

```
$ cd ../../../             # We should be back in the "opsworks" d
$ ls
alanscookbook
$ git init
$ git remote add origin https://haxxon_hax@bitbucket.org/haxxon_ha
$ git add .
$ git commit -m "Initial commit."
[master (root-commit) 7d92f9d] Initial commit.
 3 files changed, 31 insertions(+)
 create mode 100644 alanscookbook/metadata.rb
 create mode 100644 alanscookbook/recipes/nfs.rb
 create mode 100644 alanscookbook/templates/default/create_nfs_exp
$ git push -u origin master
Counting objects: 9, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (9/9), 946 bytes | 0 bytes/s, done.
Total 9 (delta 0), reused 0 (delta 0)
To https://haxxon_hax@bitbucket.org/haxxon_hax/awsopsworks.git
 * [new branch]          master -> master
Branch master set up to track remote branch master from origin.
```

Finally, we need to create our stack. This can be done either through the Web UI or the AWS command-line, using the “aws opsworks” designation. However, note that if you already have the AWS command-line configured, OpsWorks only operates in the us-east-1 region, so this will need to be specified, if your AWS CLI is already configured for a different region:

```
$ aws opsworks --region us-east-1 help
$ aws opsworks --region us-east-1 create-stack \
        --name "NFSv4 Stack" \
        --stack-region us-west-2 \
             --use-custom-cookbooks \
        --default-ssh-key-name awskeypair \
        --custom-cookbooks-source '{
          "Type": "git",
          "Url": "git@bitbucket.org:haxxon_hax/awsopsworks.git",
          "Username": "haxxon_hax",
         "Password": "My.P@ssW0rd",
         "SshKey": "<insert_private_key_from_ssh_keygen>"
        }' \
        --service-role-arn \
        "arn:aws:iam::550807172100:role/aws-opsworks-service-role"
        --default-instance-profile-arn \
        "arn:aws:iam::550807172100:instance-profile/aws-opsworks-e
{
        "StackId": "d6870048-2e00-4526-98eb-a28a38cc7e96"
}
```

The ARNs for Service Role and Instance Profile are required and can be configured through AWS Identity & Access Management (IAM) tools. However, if you create the stack using the Web UI, you have the option to have AWS automatically create these for you. As far as features are concerned, this is the only relative difference between the Web UI and the CLI. The SshKey is the private key that was created in the Introduction. This can also be added later from the Web UI.

Another command-line option that is advantageous to start out is "–generate-cli-skeleton". This prints out a JSON skeleton that you can import into a file, then be used as an argument for –cli-input-json. The advantage is if you have several stacks to make and want them to be consistent across OpsWorks.

Now that we have the stack created, we can create the layers. Layers are the skeletal rules for the EC2 instances. Within the layer settings, we tell OpsWorks how we want it to build the instances and what resources. This is where we specify any volumes we want created, what security groups to include, and what EBS Volumes to include. Because we're creating an NFS server, we'll include a security group that's good for NFS. We also want to include at least one volume (we'll start with 1 100G EBS volume), and add our custom recipe. As the "–custom-cookbooks-source" format from the create-stack subcommand used an inline format, these command-line options can also parse the JSON format:

```
$ aws ec2 --region us-west-2 describe-security-groups | egrep -i '(
                "GroupName": "AWS-OpsWorks-Custom-Server",
                "GroupId": "sg-478e0823"
                "GroupName": "AWS-OpsWorks-Default-Server",
                "GroupId": "sg-448e0820"
$ aws opsworks --region us-east-1 describe-layers --stack-id d68700
7e96
{
        "Layers": []
}
$ aws opsworks --region us-east-1 create-layer \
       --stack-id d6870048-2e00-4526-98eb-a28a38cc7e96 \
       --type custom \
       --name "NFS 1x100" \
       --shortname nfs_1x100 \
       --custom-security-group-ids \
         '[ "sg-448e0820", "sg-478e0823" ]'\
       --custom-recipes \
        '{
           "Deploy": [ "alanscookbook::nfs" ]
         }'\
       --volume-configurations \
         '[
            {
              "MountPoint": "/export1",
              "Size": 100,
              "VolumeType": "standard",
              "NumberOfDisks": 1
            }
          ]'
{
        "LayerId": "4c6e21e4-cc41-43a3-acd4-ae7a60dd908b"
}
```

The “shortname” is what would be designated in Chef recipes. For example, if we were to create custom templates based on the short name, we would put them in a subdirectory called “nfs_1x100” instead of the “default” directory under the template

Now that we have the skeletal structure, we can create an instance. The instance creation does not, however, start the instance. This becomes the equivalent of going to your EC2 settings and creating a new EC2 instance. OpsWorks handles all of that, based on the rules we set.

```
$ aws opsworks --region us-east-1 create-instance help
$ aws opsworks --region us-east-1 create-instance \
      --stack-id "d6870048-2e00-4526-98eb-a28a38cc7e96" \
      --layer-ids "4c6e21e4-cc41-43a3-acd4-ae7a60dd908b" \
      --instance-type t2.small \
      --ssh-key-name awskeypair \
      --os "Amazon Linux 2016.03" \
      --root-device-type ebs \
             --block-device-mappings \
       '[
         {
            "DeviceName": "ROOT_DEVICE",
            "Ebs": {
                    "VolumeSize": 20,
                    "VolumeType": "gp2",
                    "DeleteOnTermination": true
```

```
                }
            }
        ]'
{
        "InstanceId": "211db86a-dca2-4565-8b63-e69abe444012"
}
```

Designations like VolumeType and VolumeSize are case sensitive, which means if you type “Volumetype” instead of “VolumeType”, the command will fail. Our volume size will be 20G, as the size is specified in Gigabytes. All that’s left is to start the instance. The instance has several phases when starting: pending, booting (setup, configure, deploy).

```
$ aws opsworks --region us-east-1 start-instance \
        --instance-id "211db86a-dca2-4565-8b63-e69abe444012"
```

If we look at our instance settings, we will find the Public IP where we can SSH into it after it’s booted:

```
$ ssh -i awskeypair.pem ec2-user@52.39.150.39
```

If you’re just in the testing and exploration phase of AWS, it’s good practice to shut down and delete your instances so that you don’t get charged for them.

```
$ aws opsworks --region us-east-1 stop-instance \
        --instance-id "211db86a-dca2-4565-8b63-e69abe444012"
# Wait a few minutes for the instance to stop, then delete the ins
$ aws opsworks --region us-east-1 delete-instance \
        --instance-id "211db86a-dca2-4565-8b63-e69abe444012" \
        --delete-elastic-ip \
        --delete-volumes
```

The –delete-elastic-ip and –delete-volumes options will automatically remove any EBS volumes and Elastic IPs that were created when the instance was created. In the Web UI, these are shown as checkboxes when you click on the “delete” link.

Observations

We learned that we can easily create an EC2 instance with the services we need by defining them in Ruby files. These definitions don't necessarily require programming skills, but do require understanding the structures needed to use OpsWorks. We were able to customize a lot of the EC2 instance details, too, such as the names or naming schemes used for the instances. We were even able to specify additional EBS storage and change the amount of system storage for the root disk–something we weren't able to do with Elasticbeanstalk and could eliminate our disk space issue. We're also able to understand a little bit more about how IAM roles work with OpsWorks.

Experiment 2: Keeping EBS Volumes on new instances.

Experimentation

Building out an NFS server is a good exercise and has helped us learn some of the basics of Chef and OpsWorks. The advantage of OpsWorks is that we can now quickly create NFS servers that are essentially the same. Another advantage is that we can test new distributions or patches by creating a new instance, to ensure all the same features are there, and the system operates as expected. However, creating a new instance to replace an older NFS system will require a few changes: (1) a new IP address will be allocated to the new system, and (2) all the files shared by the existing NFS server will need to be copied to the new NFS server.

Amazon EBS Volumes and Elastic IPs are, effectively, resources, and resources can be detached, moved, or [re]attached to an instance. So, let's take a look at moving an EBS volume from one instance to another. We'll start by creating a new instance.

```
$ aws opsworks --region us-east-1 create-instance \
--stack-id c9621867-084b-4244-81c7-0f65ac34af83 \
--layer-ids "07287239-1666-4f58-b266-038205d206e5" \
--instance-type t2.small \
--ssh-key-name awskeypair \
--os "Amazon Linux 2016.03" \
--root-device-type ebs \
--block-device-mappings '[
  {
        "DeviceName": "ROOT_DEVICE",
        "Ebs": {
          "VolumeSize": 100,
          "VolumeType": "gp2",
          "DeleteOnTermination": true
        }
  }
]'
{
        "InstanceId": "dc85e1da-0269-468a-9261-442efe52f1bd"
}
$ aws opsworks --region us-east-1 start-instance \
 --instance-id dc85e1da-0269-468a-9261-442efe52f1bd
$ aws opsworks --region us-east-1 describe-instances \
 --stack-id c9621867-084b-4244-81c7-0f65ac34af83 | grep -i status
                "Status": "booting",
$ aws opsworks --region us-east-1 describe-instances \
 --stack-id c9621867-084b-4244-81c7-0f65ac34af83 | grep -i status
                "Status": "online",
$ aws opsworks --region us-east-1 describe-instances \
 --stack-id c9621867-084b-4244-81c7-0f65ac34af83 | grep -i Ip
                "PrivateDns": "ip-172-31-26-207.us-west-2.compute.
                "PrivateIp": "172.31.26.207",
                "PublicIp": "52.35.62.29"
```

Now, let’s SSH into our instance and add a few files:

```
$ ssh -i awskeypair.pem ec2-user@52.35.62.29
The authenticity of host '52.35.62.29 (52.35.62.29)' can't be esta
ECDSA key fingerprint is 26:bf:12:8b:43:f7:97:32:69:d4:1a:b8:21:91
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.35.62.29' (ECDSA) to the list of kn
 This instance is managed with AWS OpsWorks.
```

```
   ######  OpsWorks Summary  ######
   Operating System: Amazon Linux AMI release 2016.03
   OpsWorks Instance: nfs-1x1001
   OpsWorks Instance ID: dc85e1da-0269-468a-9261-442efe52f1bd
   OpsWorks Layers: NFS 1x100
   OpsWorks Stack: NFSv4 Stack
   EC2 Region: us-west-2
   EC2 Availability Zone: us-west-2a
   EC2 Instance ID: i-b7ed3e71
   Public IP: 52.35.62.29
   Private IP: 172.31.26.207
   VPC ID: vpc-7d0e6f18
   Subnet ID: subnet-e96c138c


 Visit http://aws.amazon.com/opsworks for more information.
[ec2-user@nfs-1x1001 ~]$ ls -l
total 0
[ec2-user@nfs-1x1001 ~]$ df -h
Filesystem          Size  Used Avail Use% Mounted on
/dev/xvda1           99G  1.9G   97G   2% /
devtmpfs            994M   60K  994M   1% /dev
tmpfs              1002M     0 1002M   0% /dev/shm
/dev/xvdi           100G   33M  100G   1% /export1
[ec2-user@nfs-1x1001 ~]$ cd /export1
[ec2-user@nfs-1x1001 export1]$ ls
[ec2-user@nfs-1x1001 export1]$ ls -alh
total 4.0K
drwxr-xr-x  2 root root    6 May 20 21:24 .
dr-xr-xr-x 29 root root 4.0K May 20 21:24 ..
[ec2-user@nfs-1x1001 export1]$ sudo git init
Initialized empty Git repository in /export1/.git/
[ec2-user@nfs-1x1001 export1]$ sudo git remote add origin https://
xxon_hax/allstars.git
[ec2-user@nfs-1x1001 export1]$ sudo git pull origin master
Password for 'https://haxxon_hax@bitbucket.org':
remote: Counting objects: 51, done.
remote: Compressing objects: 100% (49/49), done.
remote: Total 51 (delta 21), reused 0 (delta 0)
Unpacking objects: 100% (51/51), done.
From https://bitbucket.org/haxxon_hax/allstars
 * branch               master        -> FETCH_HEAD
 * [new branch]         master        -> origin/master
[ec2-user@nfs-1x1001 export1]$ ls -alh
total 92K
drwxr-xr-x  4 root root 4.0K May 20 21:30 .
dr-xr-xr-x 29 root root 4.0K May 20 21:24 ..
-rw-r--r--  1 root root  540 May 20 21:30 access-denied.php
```

```
-rw-r--r--  1 root root  267 May 20 21:30 auth.php
-rw-r--r--  1 root root  191 May 20 21:30 config.php
-rw-r--r--  1 root root   19 May 20 21:30 contributors.txt
drwxr-xr-x  2 root root   47 May 20 21:30 .ebextensions
drwxr-xr-x  8 root root  151 May 20 21:30 .git
-rw-r--r--  1 root root  108 May 20 21:30 .gitignore
-rw-r--r--  1 root root  435 May 20 21:30 index.php
-rw-r--r--  1 root root 2.4K May 20 21:30 license.txt
-rw-r--r--  1 root root 2.0K May 20 21:30 login-exec.php
-rw-r--r--  1 root root  536 May 20 21:30 login-failed.php
-rw-r--r--  1 root root  984 May 20 21:30 login-form.php
-rw-r--r--  1 root root  573 May 20 21:30 loginmodule.css
-rw-r--r--  1 root root  771 May 20 21:30 logout.php
-rw-r--r--  1 root root  632 May 20 21:30 member-index.php
-rw-r--r--  1 root root  550 May 20 21:30 member-profile.php
-rw-r--r--  1 root root  555 May 20 21:30 mysql.sql
-rw-r--r--  1 root root 2.1K May 20 21:30 readme.html
-rw-r--r--  1 root root  565 May 20 21:30 README.md
-rw-r--r--  1 root root 2.5K May 20 21:30 register-exec.php
-rw-r--r--  1 root root 1.7K May 20 21:30 register-form.php
-rw-r--r--  1 root root  501 May 20 21:30 register-success.php
-rw-r--r--  1 root root  209 May 20 21:30 users.config
[ec2-user@nfs-1x1001 export1]$ exit
Connection to 52.35.62.29 closed.
$
```

Now that we have some files, we want to simulate the situation where we have a new instance and we’ve tested software, and now want to serve up the same files from the new instance. Rather than copying the files, we’ll detach the resource, then reattach it to the new instance, with one caveat: we’ll need to remove the new instance’s /export1 directory because it gets created with it. AWS OpsWorks provides access to Resources via the Web UI and the command-line. The Web UI is more intuitive than the command-line. AWS can also allow access to the resources using the EC2’s EBS commands. In the Web UI, this can be found by selecting EC2 under Services, then clicking on EBS Volumes. However, when implementing OpsWorks, it’s best to control everything from OpsWorks, as one can inadvertently destroy a good resource.

NOTE: In the case that you have deleted an instance, but did not select “Delete with Resources”, and would still like to delete the volume (so you don’t get charged for it),then the volume will need to be deregistered from OpsWorks (under “Resources”) and deleted using the EC2 service’s volumes panel. First, we’ll create the second (new) instance:

```
$ aws opsworks --region us-east-1 create-instance \
--stack-id c9621867-084b-4244-81c7-0f65ac34af83 \
--layer-ids "07287239-1666-4f58-b266-038205d206e5" \
--instance-type t2.small \
--ssh-key-name awskeypair \
--os "Amazon Linux 2016.03" \
--root-device-type ebs \
--block-device-mappings '[
  {
        "DeviceName": "ROOT_DEVICE",
        "Ebs": {
          "VolumeSize": 100,
          "VolumeType": "gp2",
          "DeleteOnTermination": true
        }
  }
]'
{
        "InstanceId": "9d17cebb-6168-4f9f-ba57-da916423e856"
}
```

We won’t start the volume yet. If we go to our Web interface and click on the stack, then “Resources”, we’ll find a tab labeled “Volumes”. Under this tab, we’ll also see our first volume, attached to the nfs-1x1001 instance. Clicking on the name (Created for nfs-1x1001), displays the summary for the volume, where we can unassign the volume.

Deregistering a volume makes it available to be attached to another instance. Before we deregister the volume, let’s boot the instance to see the changes in the Resources panel. Click on Resources on the left-hand side and then “Volumes” again, then start the instance:

```
$ aws opsworks --region us-east-1 start-instance \
 --instance-id 9d17cebb-6168-4f9f-ba57-da916423e856
```

Now we’ll notice that the volume changes to an online state. We can SSH into the instance, do some tests, then stop the instance again, because we can’t unassign, change, or deregister the volume while the instance is still running.

```
$ aws opsworks --region us-east-1 stop-instance \
 --instance-id 9d17cebb-6168-4f9f-ba57-da916423e856
```

At this point, we have the option to either change the mount point for the new volume or unassign the volume. We want a clean system, so we’ll unassign the volume.

```
$ aws opsworks --region us-east-1 describe-volumes \
 --instance-id 9d17cebb-6168-4f9f-ba57-da916423e856 | grep -i Volu
               "VolumeId": "6338dcf7-6f5f-4e43-8b43-30a45497d1a0"
               "Ec2VolumeId": "vol-e3c9bc55",
$ aws opsworks --region us-east-1 unassign-volume \
 --volume-id 6338dcf7-6f5f-4e43-8b43-30a45497d1a0
```

Next, we’ll have to stop the primary NFS server and unassign its volume, before we can reassign it to the new NFS server. If you are running in a production environment, be sure you've scheduled your downtime or are operating within your maintenance window!

```
$ aws opsworks --region us-east-1 stop-instance \
 --instance-id dc85e1da-0269-468a-9261-442efe52f1bd
$ aws opsworks --region us-east-1 describe-instances \
 --stack-id c9621867-084b-4244-81c7-0f65ac34af83 | grep -i status
               "Status": "stopping",
               "Status": "stopped",
$ aws opsworks --region us-east-1 describe-instances  \
 --stack-id c9621867-084b-4244-81c7-0f65ac34af83 | grep -i status
               "Status": "stopped",
               "Status": "stopped",
```

Now that the instance is stopped, we can unassign it from the nfs-1x1001 server, then assign it to the nfs-1x1002 server.

```
$ aws opsworks --region us-east-1 describe-volumes \
 --instance-id dc85e1da-0269-468a-9261-442efe52f1bd | grep -i Volu
               "VolumeId": "1843dead-5740-4806-80d0-dc1fde5c1800",
               "Ec2VolumeId": "vol-7cd1a4ca",
$ aws opsworks --region us-east-1 unassign-volume \
 --volume-id 1843dead-5740-4806-80d0-dc1fde5c1800
$ aws opsworks --region us-east-1 assign-volume \
 --volume-id 1843dead-5740-4806-80d0-dc1fde5c1800 \
 --instance-id 9d17cebb-6168-4f9f-ba57-da916423e856
```

Now, if we look at the Resources tab in the Web interface, we'll see that the volume named "Created for nfs-1x1001" is assigned to nfs-1x1002. We wait a few minutes for the system to boot before logging in.

```
$ aws opsworks --region us-east-1 describe-instances \
 --instance-id 9d17cebb-6168-4f9f-ba57-da916423e856 | grep -i stat
```

```
                "Status": "online",
$ aws opsworks --region us-east-1 describe-instances \
 --instance-id 9d17cebb-6168-4f9f-ba57-da916423e856 | grep Ip
                "PrivateIp": "172.31.30.54",
                "PublicIp": "52.39.245.93"
$ ssh -i awskeypair.pem ec2-user@52.39.245.93
The authenticity of host '52.39.245.93 (52.39.245.93)' can't be est
ECDSA key fingerprint is ec:62:c3:1f:c8:aa:fa:cd:60:66:98:78:f1:ef
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.39.245.93' (ECDSA) to the list of kn
 This instance is managed with AWS OpsWorks.


   ######  OpsWorks Summary  ######
   Operating System: Amazon Linux AMI release 2016.03
   OpsWorks Instance: nfs-1x1002
   OpsWorks Instance ID: 9d17cebb-6168-4f9f-ba57-da916423e856
   OpsWorks Layers: NFS 1x100
   OpsWorks Stack: NFSv4 Stack
   EC2 Region: us-west-2
   EC2 Availability Zone: us-west-2a
   EC2 Instance ID: i-8b1bc84d
   Public IP: 52.39.245.93
   Private IP: 172.31.30.54
   VPC ID: vpc-7d0e6f18
   Subnet ID: subnet-e96c138c


 Visit http://aws.amazon.com/opsworks for more information.
[ec2-user@nfs-1x1002 ~]$ df -h /export1
Filesystem          Size  Used Avail Use% Mounted on
/dev/xvdi           100G   33M  100G   1% /export1
[ec2-user@nfs-1x1002 ~]$ cd /export1
[ec2-user@nfs-1x1002 export1]$ ls
access-denied.php  index.php            login-form.php        memb
xec.php
auth.php               license.txt          loginmodule.css   mysc
er-form.php
config.php             login-exec.php       logout.php
gister-success.php
contributors.txt   login-failed.php  member-index.php  README.md
[ec2-user@nfs-1x1002 export1]$
```

All of our files are there. So, now we have a standard configuration for an NFS server and an upgrade process that preserves our data. This is beneficial for testing patches, software additions, or EC2 size changes (t2.small to t2.medium, etc.). Unfortunately, this NFS server can't be used with Elasticbeanstalk, since Elasticbeanstalk uses its own directory path, albeit, we could do some advanced work and create an NFS mount point over a subdirectory in the /var/app/current directory, but this may void Elasticbeanstalk's benefits, and based on our recent experimentation with Elasticbeanstalk, we aren't going to go that route for our site.

Observations

We are able to create an EC2 instance with storage defined with our OpsWorks recipes. The storage we are able to keep and transfer to volumes to a new instance by simply attaching the storage volume to the new instance. Although we haven't tested it (yet), we expect the same behavior when proceeding with any resource within an EC2 instance controlled by OpsWorks. Another example is the Elastic IP. We would want our Elastic IP to be transferrable between instances.

Experiment 3: Adding Users

Experimentation

One of the largest differences between Elasticbeanstalk and OpsWorks is the control over the users and how development works. As we saw in Elasticbeanstalk, there is no need to log into the EC2 instance. Developers can develop locally, then do an "eb deploy" to deploy their code. OpsWorks works similarly, by using a Git repository as an interface, but deploying it to the instance is a manual process. In the OpsWorks Stack settings, under "Apps", you'll find that you can easily add and deploy apps. The advantage that OpsWorks has over Elasticbeanstalk is that it provides more control over what you want to do with your instance, and recipes can easily transfer, as long as you have your recipe repository configured correctly. In our environment, our developers use Git to push data to the git repository, but the repository is approximately 46G of data, and growing. Not only will I need to allow the developers to be able to SSH into the instance, but other admins will need access, as well, to provide monitoring and troubleshooting. Adding users to a stack is very simple, as long as you've created an IAM User for the person you wish to allow access (See Appendix A). This can be done easily through the Web UI. Appendix B contains the screenshot walkthrough using the Web UI.

The command-line form of this is at the top-level of ‘aws opsworks’: aws opsworks create-user-profile –iam-user-arn <user arn>, where the user’s arn can be found by the ‘aws iam list-users’ command, and permissions set by using ‘aws opsworks set-permissions’. Let’s start by creating a small NFS server instance. We’ll log in and look at the users. Then, we’ll add some users with the AWS CLI, and see if anything changes.

```
opsworks $ aws opsworks --region us-east-1 create-instance \
       --stack-id "c9621867-084b-4244-81c7-0f65ac34af83" \
       --layer-ids "07287239-1666-4f58-b266-038205d206e5" \
       --instance-type t2.small \
       --ssh-key-name awskeypair \
       --os "Amazon Linux 2016.03" \
       --root-device-type ebs \
       --block-device-mappings \
        '[
          {
             "DeviceName": "ROOT_DEVICE",
             "Ebs": {
             "VolumeSize": 20,
             "VolumeType": "gp2",
             "DeleteOnTermination": true
             }
          }
        ]'
{
    "InstanceId": "2bc475ed-5bd1-4191-9382-a4eb53ae9663"
}
opsworks $ aws opsworks --region us-east-1 start-instance \
       --instance-id "2bc475ed-5bd1-4191-9382-a4eb53ae9663"
opsworks $  aws opsworks --region us-east-1 describe-instances \
       --instance-id "2bc475ed-5bd1-4191-9382-a4eb53ae9663" | grep
            "PublicIp": "52.33.20.34"
```

Now that our instance is built, let’s log on and take a look at the current users.

```
opsworks $ ssh -i awskeypair.pem ec2-user@52.33.20.34
The authenticity of host '52.33.20.34 (52.33.20.34)' can't be esta
ECDSA key fingerprint is b6:84:26:a9:d9:20:34:92:e0:bd:54:cd:de:b9
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.33.20.34' (ECDSA) to the list of kn
 This instance is managed with AWS OpsWorks.

   ######  OpsWorks Summary  ######
   Operating System: Amazon Linux AMI release 2016.03
   OpsWorks Instance: nfs-1x1001
   OpsWorks Instance ID: 2bc475ed-5bd1-4191-9382-a4eb53ae9663
   OpsWorks Layers: NFS 1x100
   OpsWorks Stack: NFSv4 Stack
   EC2 Region: us-west-2
   EC2 Availability Zone: us-west-2a
   EC2 Instance ID: i-479e8deb
   Public IP: 52.33.20.34
   Private IP: 172.31.22.67
   VPC ID: vpc-7d0e6f18
   Subnet ID: subnet-e96c138c

 Visit http://aws.amazon.com/opsworks for more information.
[ec2-user@nfs-1x1001 ~]$ tail -4 /etc/passwd
ec2-user:x:500:500:EC2 Default User:/home/ec2-user:/bin/bash
aws:x:498:497::/opt/aws/opsworks/current:/bin/bash
alanducci:x:2002:501:OpsWorks user alanducci:/home/alanducci:/bin/
ganglia:x:497:496:Ganglia Monitoring System:/var/lib/ganglia:/sbin
[ec2-user@nfs-1x1001 ~]$
```

Let’s go back to our Command-line interface and import the Ashley Williams user. Ideally, this will be added to the system without having to rebuild it, unlike the properties we found in our Elasticbeanstalk experiments. As stated before, we’ll need to get the user’s arn, then use it with the create-user-profile subcommand.

```
[ec2-user@nfs-1x1001 ~]$ exit
logout
Connection to 52.33.20.34 closed.
opsworks $ aws iam list-users | egrep 'Name|arn'
            "UserName": "Ashley.J.Williams",
            "Arn": "arn:aws:iam::550807172100:user/Ashley.J.Willia
            "UserName": "alanducci",
            "Arn": "arn:aws:iam::550807172100:user/alanducci"
opsworks $ aws opsworks --region us-east-1 create-user-profile --ia
50807172100:user/Ashley.J.Williams"
{
    "IamUserArn": "arn:aws:iam::550807172100:user/Ashley.J.William
}
opsworks $ ssh -i awskeypair.pem ec2-user@52.33.20.34
 This instance is managed with AWS OpsWorks.


   ######  OpsWorks Summary  ######
   Operating System: Amazon Linux AMI release 2016.03
   OpsWorks Instance: nfs-1x1001
   OpsWorks Instance ID: 2bc475ed-5bd1-4191-9382-a4eb53ae9663
   OpsWorks Layers: NFS 1x100
   OpsWorks Stack: NFSv4 Stack
   EC2 Region: us-west-2
   EC2 Availability Zone: us-west-2a
   EC2 Instance ID: i-479e8deb
   Public IP: 52.33.20.34
   Private IP: 172.31.22.67
   VPC ID: vpc-7d0e6f18
   Subnet ID: subnet-e96c138c


 Visit http://aws.amazon.com/opsworks for more information.
[ec2-user@nfs-1x1001 ~]$ tail -4 /etc/passwd
ec2-user:x:500:500:EC2 Default User:/home/ec2-user:/bin/bash
aws:x:498:497::/opt/aws/opsworks/current:/bin/bash
alanducci:x:2002:501:OpsWorks user alanducci:/home/alanducci:/bin/
ganglia:x:497:496:Ganglia Monitoring System:/var/lib/ganglia:/sbin
[ec2-user@nfs-1x1001 ~]$ exit
```

So far, we haven't got any users.Let's take a look at our user profile, too. Notice that the SshUsername field is "ashleyjwilliams". If we wanted a different username, in the case that our site has standard usernames, we could change this at the command-line, with the –ssh-username switch for either the create-user-profile subcommand, or the update-user-profile subcommand.

```
opsworks $ aws opsworks --region us-east-1 describe-user-profiles -
::550807172100:user/Ashley.J.Williams"
{
    "UserProfiles": [
        {
            "IamUserArn": "arn:aws:iam::550807172100:user/Ashley.J
            "AllowSelfManagement": false,
            "Name": "Ashley.J.Williams",
            "SshUsername": "ashleyjwilliams"
        }
    ]
}
```

So, if we have an SshUsername, then why isn’t there an account? Well, if we explore the Web UI for the stack, we would find that the user is listed, but no permissions for SSH access was given. We can add that access either through the Web UI or the command-line. Creating the user profile did exactly what it was told to do: create the user profile so that it’s available to the stack. The default permission level is to use the levels explicitly stated in IAM; these can be overridden by the stack. We have to explicitly set the permissions to allow the user to SSH into the instances of the stack. Note that this applies to all instances within the stack, so you’ll want to create stacks based not only on function, but access, too.

```
opsworks $ aws opsworks --region us-east-1 \
set-permission \
--stack-id "c9621867-084b-4244-81c7-0f65ac34af83" \
--iam-user-arn "arn:aws:iam::550807172100:user/Ashley.J.Williams" `
--allow-ssh \
--no-allow-sudo
opsworks $ echo $?
0
opsworks $ ssh -i awskeypair.pem ec2-user@52.33.20.34
 This instance is managed with AWS OpsWorks.
```

```
    ######  OpsWorks Summary  ######
    Operating System: Amazon Linux AMI release 2016.03
    OpsWorks Instance: nfs-1x1001
    OpsWorks Instance ID: 2bc475ed-5bd1-4191-9382-a4eb53ae9663
    OpsWorks Layers: NFS 1x100
    OpsWorks Stack: NFSv4 Stack
    EC2 Region: us-west-2
    EC2 Availability Zone: us-west-2a
    EC2 Instance ID: i-479e8deb
    Public IP: 52.33.20.34
    Private IP: 172.31.22.67
    VPC ID: vpc-7d0e6f18
    Subnet ID: subnet-e96c138c


  Visit http://aws.amazon.com/opsworks for more information.
[ec2-user@nfs-1x1001 ~]$ tail -4 /etc/passwd
ec2-user:x:500:500:EC2 Default User:/home/ec2-user:/bin/bash
aws:x:498:497::/opt/aws/opsworks/current:/bin/bash
alanducci:x:2002:501:OpsWorks user alanducci:/home/alanducci:/bin/
ganglia:x:497:496:Ganglia Monitoring System:/var/lib/ganglia:/sbin
[ec2-user@nfs-1x1001 ~]$ exit
```

We notice in the Web UI that the excute_recipes recipe runs just after we created the permissions. We wait a few minutes before checking the passwd file for the new user, but he still isn’t in there. Further details shows that it ran the “ssh_users” recipe, which should have set up the user, but didn’t. Why not? Remember earlier when we said that AWS uses keys for everything? Well, it also uses keys for SSH into OpsWorks instances. We can generate new SSH keys or extract the public key from the key we created when we first created the Ashley J. Williams IAM account. To generate a new SSH key, OpsWorks supports bit lengths of 1024, 2048, or 4096. Since most other sites support 2048, we’ll use a 2048-bit RSA key.

```
opsworks $ ssh-keygen -t rsa -b 2048
Generating public/private rsa key pair.
```

```
Enter file in which to save the key (/home/alanducc/.ssh/id_rsa): /
ams
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/alanducc/ashleyjwilliar
Your public key has been saved in /home/alanducc/ashleyjwilliams.pu
The key fingerprint is:
3f:fe:06:bd:87:2b:df:2d:41:cc:ac:e3:4e:ac:3e:5c alanducci@necronom
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|                 |
|             +   |
|              =  |
|        S  . o   |
|         ...E .  |
|         .o+o+ . |
|         .++=.o. |
|         .+B=o...|
+-----------------+
opsworks $ cat /home/alanducc/ashleyjwilliams.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC4UcuShjyUinpEVpqIoTMkA6YT3V
+hz9ou58ulwKKWCi3od8lIPJnzj69wazNGdyDDPymHTN8kFHQIYhv6WUtEb2TYumKQ
oxFv1vnb2Nv4QURD3p0JAa6B/nVjWLnwu7PH92j+OmahpJh9U5xxMG7UX5ybhwYSoY
Gf+k0wHNzZ8L/iOTgT+rNrxvwyFl4wNaUPqPPzndvR6G2LRyTCjwW5yGvOn4cErEE3
ujZ3b6KF alanducci@necronomicon
opsworks $ aws opsworks --region us-east-1 update-user-profile --i
50807172100:user/Ashley.J.Williams" --ssh-public-key "ssh-rsa AAAA
4UcuShjyUinpEVpqIoTMkA6YT3VEeXe5dioX4gGi99kRhdngK/UO8M+hz9ou58ulwK
ymHTN8kFHQIYhv6WUtEb2TYumKQYEw4uBx6dpPsRcbku5A2lHS1YjvoxFv1vnb2Nv4
+OmahpJh9U5xxMG7UX5ybhwYSoYCn9gClUVyvorHoRiy0Sr74bxs+7Gf+k0wHNzZ8L
dvR6G2LRyTCjwW5yGvOn4cErEE3DJex8mRssPgLTixUlYdQuJJZXD2ujZ3b6KF"
```

Alternatively, we could have run “sshkeygen -y” and selected our previously generated PEM key. This gives us the public key. Let’s wait a few minutes for the recipe to run. This can be monitored by going to the browser and clicking on the instance, then scrolling down to the “Logs” section. From the CLI, this can be done using the “describe-commands” subcommand. This lists the Chef command log in reverse chronological order, so the first one is the most recent.

```
opsworks $ aws opsworks --region us-east-1 describe-commands \
       --instance-id "2bc475ed-5bd1-4191-9382-a4eb53ae9663"
{
    "Commands": [
        {
            "Status": "successful",
            "CompletedAt": "2016-06-25T22:42:51+00:00",
            "InstanceId": "2bc475ed-5bd1-4191-9382-a4eb53ae9663",
            "DeploymentId": "d19eed3e-dfa9-4dce-a79d-7434dcdcbf23",
            "AcknowledgedAt": "2016-06-25T22:42:45+00:00",
            "LogUrl": "https://s3.amazonaws.com/prod_stage-log/logs
c08442597d9?response-cache-control=private&response-content-encodi
pe=text%2Fplain&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=
0625%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20160625T225315Z&X-
edHeaders=host&X-Amz-Signature=9a6f7625500694d63ffb407345710f3f911
9f",
            "Type": "execute_recipes",
            "CommandId": "5a1450d2-be1b-41b0-a288-1c08442597d9",
            "CreatedAt": "2016-06-25T22:42:44+00:00",
            "ExitCode": 0
        },
        [...]
    ]
}
opsworks $ ssh -i awskeypair.pem ec2-user@52.33.20.34
 This instance is managed with AWS OpsWorks.


   ######  OpsWorks Summary  ######
   Operating System: Amazon Linux AMI release 2016.03
   OpsWorks Instance: nfs-1x1001
   OpsWorks Instance ID: 2bc475ed-5bd1-4191-9382-a4eb53ae9663
   OpsWorks Layers: NFS 1x100
   OpsWorks Stack: NFSv4 Stack
   EC2 Region: us-west-2
   EC2 Availability Zone: us-west-2a
   EC2 Instance ID: i-479e8deb
   Public IP: 52.33.20.34
   Private IP: 172.31.22.67
   VPC ID: vpc-7d0e6f18
   Subnet ID: subnet-e96c138c


 Visit http://aws.amazon.com/opsworks for more information.
[ec2-user@nfs-1x1001 ~]$ tail -4 /etc/passwd
aws:x:498:497::/opt/aws/opsworks/current:/bin/bash
alanducci:x:2002:501:OpsWorks user alanducc:/home/alanducc:/bin/ba
ganglia:x:497:496:Ganglia Monitoring System:/var/lib/ganglia:/sbin
```

```
ashleyjwilliams:x:2008:501:OpsWorks user ashleyjwilliams:/home/ash
[ec2-user@nfs-1x1001 ~]$ logout
Connection to 52.33.20.34 closed.
```

Ash’s account has been created on the instance without the need for any reboot, rebuild, or redeployment. This is successful, but we need to test a couple more things. First, let’s test Ash’s access using the new key we created.

```
opsworks $ ssh -i ~/ashleyjwilliams ashleyjwilliams@52.33.20.34
th AWS OpsWorks.

   ######  OpsWorks Summary  ######
   Operating System: Amazon Linux AMI release 2016.03
   OpsWorks Instance: nfs-1x1001
   OpsWorks Instance ID: 2bc475ed-5bd1-4191-9382-a4eb53ae9663
   OpsWorks Layers: NFS 1x100
   OpsWorks Stack: NFSv4 Stack
   EC2 Region: us-west-2
   EC2 Availability Zone: us-west-2a
   EC2 Instance ID: i-479e8deb
   Public IP: 52.33.20.34
   Private IP: 172.31.22.67
   VPC ID: vpc-7d0e6f18
   Subnet ID: subnet-e96c138c


 Visit http://aws.amazon.com/opsworks for more information.
[ashleyjwilliams@nfs-1x1001 ~]$ id
uid=2008(ashleyjwilliams) gid=501(opsworks) groups=501(opsworks)
[ashleyjwilliams@nfs-1x1001 ~]$ exit
```

Second, we’ll create another instance. Because we want everything to be the same across the platforms, we also want the userid to be the same. Otherwise, the account will lose access to the files it already created on the NFS share.

```
opsworks $ aws opsworks --region us-east-1 create-instance \
       --stack-id "c9621867-084b-4244-81c7-0f65ac34af83" \
       --layer-ids "07287239-1666-4f58-b266-038205d206e5" \
       --instance-type t2.small \
       --ssh-key-name awskeypair \
       --os "Amazon Linux 2016.03" \
       --root-device-type ebs \
       --block-device-mappings \
        '[
          {
             "DeviceName": "ROOT_DEVICE",
             "Ebs": {
             "VolumeSize": 20,
             "VolumeType": "gp2",
             "DeleteOnTermination": true
             }
          }
        ]'
{
    "InstanceId": "1bc89a85-6dc1-495c-b714-0ad29dcccfb5"
}
opsworks $ aws opsworks --region us-east-1 start-instance \
        --instance-id "1bc89a85-6dc1-495c-b714-0ad29dcccfb5"
opsworks $ aws opsworks --region us-east-1 describe-instances \
        --instance-id "1bc89a85-6dc1-495c-b714-0ad29dcccfb5" | gre
            "PublicIp": "52.40.188.8"
opsworks $ ssh -i /home/alanducci/ashleyjwilliams ashleyjwilliams@
The authenticity of host '52.40.188.8 (52.40.188.8)' can't be esta
ECDSA key fingerprint is a3:9f:d4:ea:55:0d:8b:43:9c:9f:bc:e9:17:a7
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.40.188.8' (ECDSA) to the list of kn
Permission denied (publickey).
opsworks $ ssh -i ~/ashleyjwilliams ashleyjwilliams@52.40.188.8
 This instance is managed with AWS OpsWorks.


   ######  OpsWorks Summary  ######
   Operating System: Amazon Linux AMI release 2016.03
   OpsWorks Instance: nfs-1x1002
   OpsWorks Instance ID: 1bc89a85-6dc1-495c-b714-0ad29dcccfb5
   OpsWorks Layers: NFS 1x100
   OpsWorks Stack: NFSv4 Stack
   EC2 Region: us-west-2
   EC2 Availability Zone: us-west-2a
   EC2 Instance ID: i-960a183a
   Public IP: 52.40.188.8
   Private IP: 172.31.23.243
   VPC ID: vpc-7d0e6f18
```

```
      Subnet ID: subnet-e96c138c


     Visit http://aws.amazon.com/opsworks for more information.
   [ashleyjwilliams@nfs-1x1002 ~]$ id
   uid=2008(ashleyjwilliams) gid=501(opsworks) groups=501(opsworks)
   [ashleyjwilliams@nfs-1x1002 ~]$ exit
   logout
   Connection to 52.40.188.8 closed.
   opsworks $
```

Finally, we may have users across multiple stacks. Ideally, the userid would be the same, but generally, we can try to keep ids within a stack if we really need to. To make things easy on us, let's clone our existing stack into a new one for our PHP services, and create a PHP layer. We'll create an instance on a this new stack, then give Ash his SSH access and permissions. Note that there is no need to re-enter Ash's SSH key for the other stack.

```
   opsworks $ aws opsworks --region us-east-1 clone-stack \
   >             --source-stack-id "c9621867-084b-4244-81c7-0f65ac34a
   >             --service-role-arn \
   >       "arn:aws:iam::550807172100:role/aws-opsworks-service-role" 
   >             --name "PHPApp Stack" \
   >             --hostname-theme Baked_Goods
   {
           "StackId": "5f8792f9-689d-4276-a0aa-3a0571cf1f2b"
   }
   opsworks $ aws opsworks --region us-east-1 create-layer \
           --stack-id "5f8792f9-689d-4276-a0aa-3a0571cf1f2b" \
           --type custom \
           --name "HTTPD-PHP" \
           --shortname httpd-php \
           --custom-security-group-ids \
             '[
                "sg-5f53823b",
                "sg-448e0820",
                "sg-478e0823"
             ]' \
           --volume-configurations \
             '[
              {
                "MountPoint": "/export1",
                "Size": 100,
                "VolumeType": "standard",
                "NumberOfDisks": 1
              }
           ]'
{
        "LayerId": "e93524f3-80b2-42a3-ae35-f38ccea0a992"
}
opsworks $ aws opsworks --region us-east-1 create-instance \
       --stack-id "5f8792f9-689d-4276-a0aa-3a0571cf1f2b" \
       --layer-ids "e93524f3-80b2-42a3-ae35-f38ccea0a992" \
       --instance-type t2.small \
       --ssh-key-name awskeypair \
       --os "Amazon Linux 2016.03" \
       --root-device-type ebs \
       --block-device-mappings \
      '[
          {
             "DeviceName": "ROOT_DEVICE",
             "Ebs": {
             "VolumeSize": 20,
             "VolumeType": "gp2",
             "DeleteOnTermination": true
             }
          }
       ]'
{
    "InstanceId": "0e10cad7-8601-4143-a28f-06162e0c5cb1"
}
opsworks $ aws opsworks --region us-east-1 set-permission \
        --stack-id "5f8792f9-689d-4276-a0aa-3a0571cf1f2b" \
        --iam-user-arn "arn:aws:iam::550807172100:user/Ashley.J.Wi
        --allow-ssh --no-allow-sudo
opsworks $ aws opsworks --region us-east-1 start-instance \
 --instance-id "0e10cad7-8601-4143-a28f-06162e0c5cb1"
opsworks $ aws opsworks --region us-east-1 describe-commands \
 --instance-id "0e10cad7-8601-4143-a28f-06162e0c5cb1"
{
    "Commands": []
}
opsworks $ aws opsworks --region us-east-1 describe-commands --ins
3-a28f-06162e0c5cb1"
{
    "Commands": [
        {
            "InstanceId": "0e10cad7-8601-4143-a28f-06162e0c5cb1",
            "Status": "pending",
            "Type": "setup",
            "CommandId": "fb85001d-b4d1-4c0f-81a9-9498d0db7d9f",
            "CreatedAt": "2016-06-25T23:39:16+00:00"
        }
    ]
}
opsworks $ aws opsworks --region us-east-1 describe-commands --ins
3-a28f-06162e0c5cb1"
{
    "Commands": [
        {
            "Status": "successful",
            "CompletedAt": "2016-06-25T23:44:50+00:00",
            "InstanceId": "0e10cad7-8601-4143-a28f-06162e0c5cb1",
            "AcknowledgedAt": "2016-06-25T23:44:28+00:00",
            "LogUrl": "https://s3.amazonaws.com/prod_stage-log/log
97804e90dd1?response-cache-control=private&response-content-encodi
pe=text%2Fplain&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=
0625%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20160625T000647Z&X-
edHeaders=host&X-Amz-Signature=1d33f80d03b87648e6d6c7d16b817688007
44",
            "Type": "configure",
            "CommandId": "22814678-f5ba-4275-bbae-e97804e90dd1",
            "CreatedAt": "2016-06-25T23:44:08+00:00",
            "ExitCode": 0
        },
        {
            "Status": "successful",
            "CompletedAt": "2016-06-25T23:43:58+00:00",
            "InstanceId": "0e10cad7-8601-4143-a28f-06162e0c5cb1",
            "AcknowledgedAt": "2016-06-25T23:42:22+00:00",
            "LogUrl": "https://s3.amazonaws.com/prod_stage-log/log
498d0db7d9f?response-cache-control=private&response-content-encodi
pe=text%2Fplain&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=
0625%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20160625T000647Z&X-
edHeaders=host&X-Amz-Signature=32e3e04ba79789a89fc1879d50ef0325850
ff",
            "Type": "setup",
            "CommandId": "fb85001d-b4d1-4c0f-81a9-9498d0db7d9f",
            "CreatedAt": "2016-06-25T23:39:16+00:00",
            "ExitCode": 0
        }
    ]
}
opsworks $ aws opsworks --region us-east-1 describe-instances \
        --instance-id "0e10cad7-8601-4143-a28f-06162e0c5cb1" | gre
            "PublicIp": "52.36.55.158"
opsworks $ ssh -i ~/ashleyjwilliams ashleyjwilliams@52.36.55.158
The authenticity of host '52.36.55.158 (52.36.55.158)' can't be es
ECDSA key fingerprint is 51:fd:93:a7:14:c3:92:28:60:ad:ec:8f:78:c1
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added '52.36.55.158' (ECDSA) to the list of k
 This instance is managed with AWS OpsWorks.


   ######  OpsWorks Summary  ######
   Operating System: Amazon Linux AMI release 2016.03
   OpsWorks Instance: nfs-1x1001
   OpsWorks Instance ID: 0e10cad7-8601-4143-a28f-06162e0c5cb1
   OpsWorks Layers: NFS 1x100
   OpsWorks Stack: AZDHS NFS4 Stack
   EC2 Region: us-west-2
   EC2 Availability Zone: us-west-2a
   EC2 Instance ID: i-5e3321f2
   Public IP: 52.36.55.158
   Private IP: 172.31.31.88
   VPC ID: vpc-7d0e6f18
   Subnet ID: subnet-e96c138c


 Visit http://aws.amazon.com/opsworks for more information.
[ashleyjwilliams@lemon-tart ~]$ id
uid=2008(ashleyjwilliams) gid=501(opsworks) groups=501(opsworks)
[ashleyjwilliams@lemon-tart ~]$ exit
```

This looks great! Not only does the userid persist within a layer, but it persists across stacks, as well. We can successfully maintain users and local accounts on the system. This integrates with our Developers' workflow.

Observations

When adding recipes to the stacks, we need to remember that order is important. When adding recipes to the OpsWorks layers, anything within a “deploy” or “configure” area is performed in order. If there is no dependency on AWS’s recipes, then the easiest preference is to use the in the order provided by OpsWorks (e.g., place in configure first, then place in deploy).

Experiment 4: PHP.INI configuration

Experimentation

In this experiment, we’re actually going to test two things at once. We’re going to test whether we can customize the PHP.INI configuration and if we can run recipes on demand. To propose a scenario, let’s suppose we need to increase the memory settings on all of our PHP applications within a stack. In our Chef cookbook, we can change different settings, but we don’t want to have to reboot or rebuild our instances, so we’ll want to run our recipe manually for now.

Since we’ve already created a new Stack with a new Layer and Instance by cloning our existing NFS stack in the previous experiment, we’ll use that instance for testing. Also, we’re going to use templates and see how they work with the profiles. If you want a simple refresher or introduction on templates and how OpsWorks uses them, point your browser to
http://docs.aws.amazon.com/opsworks/latest/userguide/workingcookbo
installingcustom-components-templates.html or
http://docs.aws.amazon.com/opsworks/latest/userguide/cookbooks-101-opsworks-attributes.html for a complete walkthrough. Go ahead. I’ll wait.

Let’s start by creating our PHP.INI template file. I have one that I’ve copied from an existing server, and I’ll modify a couple of lines to use templates, using AWS documentation as my guide.*

```
opsworks $ scp existingserver:/etc/php.ini \
  ./alanscookbook/templates/default/php.ini.erb
opsworks $ vi ./alanscookbook/templates/default/php.ini.erb
[...]
memory_limit = <%= node[:php][:memory_limit] %>
[...]
opsworks $
```

Templates are used in the recipes with the ‘template’ keyword, just as we did before, with our NFS scripts, so we’ll create a php recipe with the template for php.ini.

```
opsworks $ vi ./alanscookbook/recipes/php.rb
template '/etc/php.ini' do
  source 'php.ini.erb'
  group 'root'
  owner 'root'
  mode '0644'
end
```

So, we have our template and the rule to create the template, but what’s the funky code that’s in the template? We’ve basically converted our php.ini file into an embedded ruby file; the ruby code is embedded within the file itself, designated by the “<%=” area. This tells the system to read the information up until “%>” as ruby code. OpsWorks sees “node[:php][:memory_limit]” as information that can be replaced by specific attributes. To do this, we create attributes files for our default attributes and the overrides. Default attributes start with “default”, while the overrides start with “normal”. These attribute files are placed in the attributes subdirectory with a .rb Ruby extension.

```
opsworks $ vi alanscookbook/attributes/default.rb
# Default settings
default[:php][:memory_limit] = '128M'
opsworks $ vi alanscookbook/attributes/customize.rb
normal[:php][:memory_limit] = '256M'
```

In Chef recipes, default attributes get overridden by a file called “customize.rb” in the attributes subdirectory. AWS OpsWorks supports this functionality, although, if you were to compare Chef attributes versus OpsWorks attributes, many Chef recipes use a single quote designation for the bracketed values. That is, default[‘php’][‘memory_limit’] is more often seen in Chef than default[:php][:memory_limit].

Now that we’ve added the files, we need to upload them to our git repository, then execute the recipe “alanscookbook::php” on our instance.

```
opsworks $ git add .
opsworks $ git commit -m "Added PHP recipe, template, and attribut
[master 66009ed] Added PHP recipe, template, and attributes.
 4 files changed, 1816 insertions(+)
 create mode 100644 alanscookbook/attributes/customize.rb
 create mode 100644 alanscookbook/attributes/default.rb
 create mode 100644 alanscookbook/recipes/php.rb
 create mode 100644 alanscookbook/templates/default/php.ini.erb
opsworks $ git push
Counting objects: 16, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (11/11), 20.70 KiB | 0 bytes/s, done.
Total 11 (delta 0), reused 0 (delta 0)
To https://haxxon_hax@bitbucket.org/haxxon_hax/awsopsworks.git
```

```
   1fd8dd8..66009ed  master -> master
opsworks $
```

To run the command, we use the create-deployment subcommand of AWS OpsWorks CLI.

```
opsworks $ aws opsworks --region us-east-1 create-deployment \
>              --stack-id "5f8792f9-689d-4276-a0aa-3a0571cf1f2b" \
>              --instance-ids \
>                '[ "0e10cad7-8601-4143-a28f-06162e0c5cb1" ]' \
>              --command \
>                '{
>                   "Name": "execute_recipes",
>                   "Args": { "recipes" : [ "alanscookbook::php" ]
>                 }' \
>              --comment "Testing php memory_limit"
{
        "DeploymentId": "be5b25ed-e1e4-44c5-93be-68beb8cf8f2c"
}
opsworks $ aws opsworks --region us-east-1 describe-deployments \
        --stack-id "5f8792f9-689d-4276-a0aa-3a0571cf1f2b" \
         | egrep -i "status|command|alanscookbook|recipe"
               "Status": "failed",
               "Command": {
                       "recipes": [
                           "alanscookbook::php"
                   "Name": "execute_recipes"
opsworks $
```

Our recipe failed. Why? Taking a look at the log using the browser, it tells us “could not find recipe php for cookbook alanscookbook.” But we uploaded it. In our stack, there’s also a command that we can run called “update_custom_cookbooks”. This command synchronizes OpsWorks’s cookbooks with our git repository’s cookbooks.

```
opsworks $ aws opsworks --region us-east-1 create-deployment \
> --instance-ids \
>   '[ "0e10cad7-8601-4143-a28f-06162e0c5cb1" ]' \
> --stack-id "5f8792f9-689d-4276-a0aa-3a0571cf1f2b" \
> --command '{ "Name": "update_custom_cookbooks" }'
```

We wait for this to be successful, then run our recipe again.

```
opsworks $ aws opsworks --region us-east-1 create-deployment \
>               --stack-id "5f8792f9-689d-4276-a0aa-3a0571cf1f2b" \
>               --instance-ids \
>                 '[ "0e10cad7-8601-4143-a28f-06162e0c5cb1" ]' \
>               --command \
>                 '{
>                    "Name": "execute_recipes",
>                    "Args": { "recipes" : [ "alanscookbook::php" ]
>                  }' \
>               --comment "Testing php memory_limit"
[...]
opsworks $ aws opsworks --region us-east-1 describe-deployments \
            --stack-id "5f8792f9-689d-4276-a0aa-3a0571cf1f2b" \
             | egrep -i "status|command|alanscookbook|recipe"
                "Status": "successful",
                "Command": {
                        "recipes": [
                            "alanscookbook::php"
                    "Name": "execute_recipes"
                "Status": "successful",
                "Command": {
                "Status": "failed",
                "Command": {
                        "recipes": [
                            "alanscookbook::php"
                    "Name": "execute_recipes"
opsworks $ ssh -i ~/ashleyjwilliams ashleyjwilliams@52.36.55.158
 This instance is managed with AWS OpsWorks.


   ######  OpsWorks Summary  ######
   Operating System: Amazon Linux AMI release 2016.03
   OpsWorks Instance: lemon-tart
   OpsWorks Instance ID: 0e10cad7-8601-4143-a28f-06162e0c5cb1
```

```
     OpsWorks Layers: HTTPD-PHP
     OpsWorks Stack: PHPApp Stack
     EC2 Region: us-west-2
     EC2 Availability Zone: us-west-2a
     EC2 Instance ID: i-00170bac
     Public IP: 52.36.55.158
     Private IP: 172.31.31.252
     VPC ID: vpc-7d0e6f18
     Subnet ID: subnet-e96c138c


  Visit http://aws.amazon.com/opsworks for more information.
[ashleyjwilliams@lemon-tart ~]$ grep memory_limit /etc/php.ini
memory_limit = 256M
[ashleyjwilliams@lemon-tart ~]$ exit
opsworks $
```

Observations

Our PHP tests are successful! We are able to use the attributes to change the php.ini file for our environment. The next phase is to use the environments to our advantage. In a normal Chef configuration, environments are used to specify which attribute file to use in the templates. According to AWS[4], OpsWorks does not support Chef environments: “A common practice is to have multiple stacks that represent different environments.”[5]

Thus, if we want our environments to look similar, but be configured differently, we have to create separate, similar, cookbooks for each environment, and maintain everything within our chef configurations. In Linux, we can easily hard-link files together that we want to keep consistent, such as our templates; but our attribute directory will be different for each recipe. For example, we might call one cookbook “lampdev”, and another “lampprod”, with separate memory settings for PHP. There isn’t much testing for this; our cookbooks will look the same, except for a few changes in the templates. Next, we’ll take a look at automatically pulling our site down from the Git repository.

Experiment 5: Deploying the Site from Git Repository - Part I

Experimentation

One of our problems with Elasticbeanstalk was trying to get our 47G repository onto the beanstalk without running out of space or timing out the transfer. Because AWS OpsWorks is only an interface for building EC2 instances and creating recipes, we can expect better results for attempting to pull down our data from the repository. Pulling our site from Git now has several different options with OpsWorks. OpsWorks can manage application deployments (see http://docs.aws.amazon.com/opsworks/latest/userguide/gettingstarted-linux-explore-cookbook.html), but in this experiment, we’re going to configure our universal NFS server, which will have separate EBS volumes for each share. Later, we may look at creating several different apps that we can deploy on a single instance (under one EBS volume mounted at /srv). Another option is to have the NFS server built, but manage the deployments on the clients. From a deployment standpoint, this is ideal, but because we’re just testing deployments for now, we’re going to create our own deployment script and pull the code directly onto our NFS server; this also has less network overhead, since we’re going directly to the EBS volume on the NFS server, rather than through an NFS-mounted filesystem. We’ll start by copying our git script into the cookbook, and create the Ruby recipe file for distributing and executing it.

```
opsworks $ vi alanscookbook/templates/default/git_pull.sh
#!/bin/sh
if [ -d /export1 ]; then
  cd /export1
  git init
  git remote add origin http://172.31.8.217/azdhs/azdhs-gov.git
  git config core.sparseCheckout true
  git pull origin master
  done
fi
opsworks $ vi alanscookbook/recipes/run_git_service.rb
template '/tmp/git_pull.sh' do
  source 'git_pull.sh'
  group 'root'
  owner 'root'
  mode '0755'
end


execute 'git_pull' do
  user 'root'
  command           'nohup /tmp/git_pull.sh &>/tmp/git_pull.log &'
end
opsworks $ git add .
opsworks $ git commit -m "Added run_git_service.rb and git_pull.sh'
opsworks $ git push
```

If we have an existing instance that we want to run this on, we have to remember to update the AWS git cache for the custom cookbooks before we run the command. We also could add this into the deploy section of our layer's recipes; in my case, I chose the former because I'm building an NFS server and I don't want this git pull to be run on all of my NFS servers.[6]

All that's left is to run the cookbooks in our instances. Again, we first update the cookbooks so that OpsWorks knows about our changes. Then, we deploy the cookbook. If we had no instances, then we don't need to run the "Update Custom Cookbooks" deployment; in fact, the system won't let us:

```
opsworks $ aws opsworks --region us-east-1 create-deployment \
> --stack-id "5f8792f9-689d-4276-a0aa-3a0571cf1f2b" \
> --command '{ "Name": "update_custom_cookbooks" }'

A client error (ValidationException) occurred when calling the Crea
ease provide at least an instance ID of one running instance
opsworks $ aws opsworks --region us-east-1 create-deployment \
> --instance-ids \
>   '[ "0e10cad7-8601-4143-a28f-06162e0c5cb1" ]' \
> --stack-id "5f8792f9-689d-4276-a0aa-3a0571cf1f2b" \
> --command '{ "Name": "update_custom_cookbooks" }'
[...]
opsworks $ aws opsworks --region us-east-1 create-deployment \
>               --stack-id "5f8792f9-689d-4276-a0aa-3a0571cf1f2b" \
>               --instance-ids \
>                 '[ "0e10cad7-8601-4143-a28f-06162e0c5cb1" ]' \
>               --command \
>                 '{
>                    "Name": "execute_recipes",
>                    "Args": { "recipes" : [ "alanscookbook::run_gi
>                  }' \
>               --comment "Testing php memory_limit"
```

We won’t notice an immediate failure because I’ve configured the service to run in the background. The system itself assumes it was successful because all it wants to do is launch the service and leave. We can ssh into the instance and watch as the directory gets downloaded.

Observations

The deployment was successful! We're able to create a script that can handle our data, and create a recipe that will run it. Also, we've been able to use the templates directory to store our script. This is just one of the ways to handle the initial deployment (and further deployments. of the site, but it gives us some more information about the things we can do with OpsWorks when configuring a system. If we wanted to handle further deployments, we could also create a recipe that would call a script to pull down any changes [in the git repository] to the site. In the next section, we'll explore the deployment tool that's provided by AWS OpsWorks.

Experiment 6: Deploying the Site from Git Repository - Part II

Experimentation

In the second scenario, which is the preferred model for deployment, we'll look at using the built-in deployment tool that AWS uses. While this looks straightforward, this can become somewhat frustrating if you don't have all the terms defined. This deployment scheme has a couple of advantages. We can configure our stack to have several of the same type of deployments. For example, we may have a PHP application that supports our blog, and another PHP application that runs reports. The stack is basically the same, but the underlying deployed code is different.

The second advantage is that we can easily re-deploy code with one click (or a single command-line). Let's take a look at the structure of the Application Deployment feature of OpsWorks. To deploy an application, we first have to create an Application within OpsWorks. This is similar to how we first created recipes before we were able to run them on our instances.

The ‘help’ page of the ‘create-app’ subcommand gives us the usage:

```
opsworks $ aws opsworks --region us-west-1 create-app help | awk ',
SYNOPSIS
            create-app
          --stack-id <value>
          [--shortname <value>]
          --name <value>
          [--description <value>]
          [--data-sources <value>]
          --type <value>
          [--app-source <value>]
          [--domains <value>]
          [--enable-ssl | --no-enable-ssl]
          [--ssl-configuration <value>]
          [--attributes <value>]
          [--environment <value>]
          [--cli-input-json <value>]
          [--generate-cli-skeleton]


opsworks-cookbooks $
```

Let’s assume we already have our NFS server instance. We’ll need the NFS server’s internal IP address.

```
opsworks $ aws opsworks --region us-east-1 describe-instances \
> --stack-id "c9621867-084b-4244-81c7-0f65ac34af83" \
>  | grep -i privateip
                "PrivateIp": "172.31.21.116",
opsworks $
```

Now that we have the IP address, we’ll configure the layer in our PHP Application to overmount the default root directory for the application. By default, this is /srv.[7] We'll create a new cookbook for PHP and include a recipe for our NFS mounts. This recipe can go in our "Deploy" section of the cookbook, which will get run every time that the app is deployed or redeployed.

```
opsworks $ mkdir lamp
opsworks $ cd lamp/
lamp $ ls
lamp $ mkdir attributes recipes templates
lamp $ vi metadata.rb
name                   "lamp"
maintainer             "Alan Landucci-Ruiz"
maintainer_email  "landucci@something-magical.net"
license                "GNU"
description            "Configures and starts Alans LAMP Stacks"
version                "1.0.0"
lamp $ vi recipes/mount_nfs.rb
mount '/srv' do
  device '172.31.21.116:/export1'
  fstype 'nfs'
  options 'rw'
  action [:mount,:enable]
end
lamp $
lamp $ cd ..
opsworks $ git add .
opsworks $ git commit -m "Added LAMP cookbook and nfs mount recipe'
[master 55460ad] Added LAMP cookbook and nfs mount recipe
 3 files changed, 22 insertions(+)
 create mode 100644 lamp/metadata.rb
 create mode 100644 lamp/recipes/mount_nfs.rb
opsworks $ git push
Counting objects: 8, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 939 bytes | 0 bytes/s, done.
Total 7 (delta 0), reused 0 (delta 0)
To https://haxxon_hax@bitbucket.org/haxxon_hax/awsopsworks.git
   66009ed..55460ad  master -> master
opsworks $
```

Now that we've pushed this out to the git server, we'll add this recipe to our deploy section of the Custom Chef Recipes in our HTTPD-PHP layer.

```
opsworks $ aws opsworks --region us-east-1 update-layer \
> --layer-id "e93524f3-80b2-42a3-ae35-f38ccea0a992" \
> --custom-recipes '{
>                          "Deploy": ["lamp::mount_nfs"]
>                        }'
opsworks $
```

We don't want to create our instance yet. We want to add an application first. AWS help page for "create-app" tells us that the application type is associated with the layer type, which means a PHP type of application should be for a PHP stack. If we intend on implementing our own Deploy recipe (which we do), then for a type, we would select "Other." For fun, though, let's try to put a PHP application in a custom layer.

```
opsworks $ aws opsworks --region us-east-1 create-app \
> --stack-id "5f8792f9-689d-4276-a0aa-3a0571cf1f2b" \
> --type php \
> --name MyProdApp \
> --app-source '{
>                      "Type": "git",
>                      "Url" : "git@bitbucket.org:haxxon_hax/allsta
>                      "SshKey": "..."
>                    }'
{
        "AppId": "33e23d61-d191-4219-b2bf-8b0c65a5675d"
}
opsworks $
```

We opted to enter the SSH Key in the Command-Line, but it's much easier (and you run into less problems by pasting it into the browser. We have our Application ID. We can add another app, too, if we wanted, called "MyDevApp" for the development tier. Let's create an instance within the layer and try to deploy our production to it. Note that nowhere in the instance creation does it ask for the Application. We will try to deploy the application after creating the instance.

```
opsworks $ aws opsworks --region us-east-1 create-instance \
> --stack-id "5f8792f9-689d-4276-a0aa-3a0571cf1f2b" \
> --layer-ids "e93524f3-80b2-42a3-ae35-f38ccea0a992" \
> --ssh-key-name "awskeypair" \
> --instance-type "t2.small" \
> --root-device-type ebs \
>               --block-device-mappings \
>             '[
>               {
>                  "DeviceName": "ROOT_DEVICE",
>                  "Ebs": {
>                      "VolumeSize": 20,
>                      "VolumeType": "gp2",
>                      "DeleteOnTermination": true
>                  }
>               }
>             ]'
{
        "InstanceId": "a7f6d8c2-287d-4e86-8ff6-a4e12982127f"
}
opsworks $ aws opsworks --region us-east-1 start-instance \
> --instance-id "a7f6d8c2-287d-4e86-8ff6-a4e12982127f"
opsworks $ aws opsworks --region us-east-1 describe-instances \
>   --instance-id "a7f6d8c2-287d-4e86-8ff6-a4e12982127f" \
> | grep -i publicip
                "PublicIp": "52.39.120.227"
opsworks $ ssh -i ~/ashleyjwilliams ashleyjwilliams@52.39.120.227
 This instance is managed with AWS OpsWorks.

   ######  OpsWorks Summary  ######
   Operating System: Amazon Linux AMI release 2016.03
   OpsWorks Instance: apple-pie
   OpsWorks Instance ID: a7f6d8c2-287d-4e86-8ff6-a4e12982127f
   OpsWorks Layers: HTTPD-PHP
   OpsWorks Stack: PHPApp Stack
   EC2 Region: us-west-2
```

```
     EC2 Availability Zone: us-west-2a
     EC2 Instance ID: i-6ef816c1
     Public IP: 52.39.120.227
     Private IP: 172.31.24.63
     VPC ID: vpc-7d0e6f18
     Subnet ID: subnet-e96c138c

  Visit http://aws.amazon.com/opsworks for more information.
[ashleyjwilliams@apple-pie ~]$ df -h /srv
Filesystem                  Size  Used Avail Use% Mounted on
172.31.21.116:/export1  100G   32M  100G   1% /srv
[ashleyjwilliams@apple-pie ~]$ ls /srv
[ashleyjwilliams@apple-pie ~]$ ls /var/www
[ashleyjwilliams@apple-pie ~]$ exit
logout
Connection to 52.39.120.227 closed.
opsworks $
```

We have our instance and the NFS mount is working correctly, but there’s no content. This should be because we haven’t actually told the system to deploy the application. We’ll try to deploy the application using the create-deployment subcommand. We can either deploy to an entire layer, which will apply the deployment to all instances within the layer, or deploy to each instance by the instance ID.

```
opsworks $ aws opsworks --region us-east-1 create-deployment \
> --stack-id "5f8792f9-689d-4276-a0aa-3a0571cf1f2b" \
> --app-id "33e23d61-d191-4219-b2bf-8b0c65a5675d" \
> --instance-ids "a7f6d8c2-287d-4e86-8ff6-a4e12982127f" \
> --command '{
>                   "Name": "deploy"
>                 }'
{
        "DeploymentId": "98ce339f-35eb-4f0c-aa80-5714aa596425"
}
```

After a few minutes, we check the deployment log of the instance and see that nothing has been pulled down. If we log into the instance itself, there is no code. Why is this? Remember when we created this as a custom layer? Deployments of applications don’t work in this way. We can note, however, that the “Deploy” section of our custom layer was run when we deployed the application.

Let’s take a look at how AWS does this. AWS uses its own cookbooks, so let’s take a look at their “deploy::php” cookbook at https://github.com/aws/opsworks-cookbooks/blob/release-chef-11.4/deploy/recipes/php.rb.

```
#
# Cookbook Name:: deploy
# Recipe:: php
#

include_recipe 'deploy'
include_recipe "mod_php5_apache2"
include_recipe "mod_php5_apache2::php"

node[:deploy].each do |application, deploy|
  if deploy[:application_type] != 'php'
        Chef::Log.debug("Skipping deploy::php application #{applica
app")
        next
  end

  opsworks_deploy_dir do
        user deploy[:user]
        group deploy[:group]
        path deploy[:deploy_to]
  end

  opsworks_deploy do
        deploy_data deploy
        app application
  end
end
```

So, the recipe itself detects if the application is of type 'php', then will run the deployment. Let's try to directly add this recipe first.

```
opsworks $ aws opsworks --region us-east-1 update-layer \
> --layer-id "e93524f3-80b2-42a3-ae35-f38ccea0a992" \
> --custom-recipes '{
>                          "Deploy": ["lamp::mount_nfs", "deploy::pł
>                        }'
opsworks $ aws opsworks --region us-east-1 create-deployment \
> --stack-id "5f8792f9-689d-4276-a0aa-3a0571cf1f2b" \
> --app-id "33e23d61-d191-4219-b2bf-8b0c65a5675d" \
> --instance-ids "a7f6d8c2-287d-4e86-8ff6-a4e12982127f" \
> --command '{
>                   "Name": "deploy"
>                 }'
{
        "DeploymentId": "9cb545ff-30ca-431f-8f3f-b21c1ec2f349"
}
```

Now, when we look at the deployment log, we find an error: "InsufficientPermissions". Oops. Looks like we're sharing the directory in read-only mode; we did this before because we were running our deployments on the NFS server itself. We'll have to fix it on our NFS server, but we'll also fix it in our git repository.

```
opsworks $ vi alanscookbook/templates/default/create_nfs_exports.sł
#!/bin/sh

EXPORT_LIST=`/bin/df -h | /bin/grep '/export[0-9]' | /bin/awk  '{p:

for i in $EXPORT_LIST; do
   /bin/echo "$i            172.31.0.0/16(rw,no_root_squash)" >> /e
done

/usr/sbin/exportfs -a
opsworks $ git add .
opsworks $ git commit -m "Changed export to rw"
[master 5c6faec] Changed export to rw
```

```
 1 file changed, 1 insertion(+), 1 deletion(-)
opsworks $ git push
Counting objects: 11, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 554 bytes | 0 bytes/s, done.
Total 6 (delta 2), reused 0 (delta 0)
To https://haxxon_hax@bitbucket.org/haxxon_hax/awsopsworks.git
   4483a69..5c6faec  master -> master
opsworks $ ssh -i ~/ashleyjwilliams ashleyjwilliams@52.38.93.157
 This instance is managed with AWS OpsWorks.

   ######  OpsWorks Summary  ######
   Operating System: Amazon Linux AMI release 2016.03
   OpsWorks Instance: nfs-1x1002
   OpsWorks Instance ID: 070e97ce-642e-4381-825b-453e7c9e8012
   OpsWorks Layers: NFS 1x100
   OpsWorks Stack: NFSv4 Stack
   EC2 Region: us-west-2
   EC2 Availability Zone: us-west-2a
   EC2 Instance ID: i-09d937a6
   Public IP: 52.38.93.157
   Private IP: 172.31.21.116
   VPC ID: vpc-7d0e6f18
   Subnet ID: subnet-e96c138c

 Visit http://aws.amazon.com/opsworks for more information.
[ashleyjwilliams@nfs-1x1002 ~]$ cat /etc/exports
/export1            172.31.0.0/16(ro,no_root_squash)
[ashleyjwilliams@nfs-1x1002 ~]$ sudo rm /etc/exports
[ashleyjwilliams@nfs-1x1002 ~]$ sudo /tmp/create_nfs_exports.sh
[ashleyjwilliams@nfs-1x1002 ~]$ cat /etc/exports
/export1            172.31.0.0/16(rw,no_root_squash)
[ashleyjwilliams@nfs-1x1002 ~]$
```

Now that we have read/write privileges on the directory for the NFS mount, let’s run our deployment again.

```
opsworks $ aws opsworks --region us-east-1 create-deployment \
> --stack-id "5f8792f9-689d-4276-a0aa-3a0571cf1f2b" \
> --app-id "33e23d61-d191-4219-b2bf-8b0c65a5675d" \
> --instance-ids "a7f6d8c2-287d-4e86-8ff6-a4e12982127f" \
> --command '{
```

```
>                  "Name": "deploy"
>                }'
{
        "DeploymentId": "4b5a5cc3-4553-4439-aa87-43eac88a94a3"
}
opsworks $
```

This time, our deployment is successful after a few minutes. Let's login to our server and take a look at the contents of the directory.

```
opsworks $ ssh -i ~/ashleyjwilliams ashleyjwilliams@52.39.120.227
 This instance is managed with AWS OpsWorks.

   ######  OpsWorks Summary  ######
   Operating System: Amazon Linux AMI release 2016.03
   OpsWorks Instance: apple-pie
   OpsWorks Instance ID: a7f6d8c2-287d-4e86-8ff6-a4e12982127f
   OpsWorks Layers: HTTPD-PHP
   OpsWorks Stack: PHPApp Stack
   EC2 Region: us-west-2
   EC2 Availability Zone: us-west-2a
   EC2 Instance ID: i-6ef816c1
   Public IP: 52.39.120.227
   Private IP: 172.31.24.63
   VPC ID: vpc-7d0e6f18
   Subnet ID: subnet-e96c138c

 Visit http://aws.amazon.com/opsworks for more information.
[ashleyjwilliams@apple-pie ~]$ ls /srv/www/
myprodapp
[ashleyjwilliams@apple-pie ~]$ ls /srv/www/myprodapp/
current  releases  shared
[ashleyjwilliams@apple-pie ~]$ ls /srv/www/myprodapp/current/
access-denied.php  index.php            login-form.php          my
xec.php
auth.php              license.txt          loginmodule.css
orm.php
config                log                  logout.php
er-success.php
config.php            login-exec.php       member-index.php
contributors.txt   login-failed.php  member-profile.php  README.md
[ashleyjwilliams@apple-pie ~]$
```

Observations

We were able to deploy the application using AWS's recipe for PHP. We would like to add another instance for development, while making sure that it doesn't apply the deployment for "MyProdApp". This would leave us with a small problem, though. If we create a new instance, it's configured to mount the same NFS share. The alternative is to create a new layer that is built the same, but with a different NFS mount. When we have to do this for four different environments, this can become tedious. The recommendation from AWS is to use a seperate stack for each stage in an application lifecycle. For example, you would create a development stack that looks exactly like your production stack, but with different definitions on how the data is accessed. Later, in Chapter 2.4, we'll take a look at whether we can use the environment variable feature of Chef 11.10 to help us with this.

I'm going to leave this chapter with an exercise for you to do. Leaving the new Web instance running, remove the nfs mount from the layer's recipe, then create a new instance. Check to see if anything is in /srv/www. Check the first web instance and see if the directory is still mounted.

Conclusion

We started out wanting to build out a simple NFS server using custom recipes. AWS OpsWorks seamlessly integrates Chef recipes into the building out of the instances. We were able to create a stack for a custom application, and were able to use different methods to populate the web tier with the data to be presented. Separate stacks are recommended for each environment, but cloning and environment variables can help save us a lot of unnecessary duplicate coding.

We were also able to transfer resources, such as EBS volumes and Elastic IPs, to new instances. Creating users within a stack is easy to do using both the Web UI and the CLI; we have to wait a minute or two for the recipe to run. Finally, using templates, we were able to customize our application’s PHP settings. Using the information gathered from these experiments, we can create recipes for other applications.

Architecting

For our architecture, we're going to create five environments (Production, UAT, QA, Development, and Sandbox), supported by three NFS servers (one for Production data, one for UAT and QA, and another for Development and Sandbox). We've already created an NFS stack with one Layer that we can use to build an NFS server sharing a single 100G drive. This will be good for our production server, where we want to keep the files separate from the files for the other environments. We'll also create a second layer, called 2x100, which provides two 100G EBS volumes, shared out via NFS. From this, we'll create two instances. We'll create one instance from the 1x100 layer, as well.

We might need a couple of databases, so we'll create that layer, too. We could create a layer for an Amazon Relational Database Service (RDS), but for simplicity, I'll just create a layer for another EC2 instance configured for MySQL. AWS OpsWorks includes a preconfigured type called "MySQL" for just this purpose. As a bonus, if I ever want to spin up a simple one-instance LAMP setup, I can quickly create one instance under two layers (using the CLI).

Now that we have our architecture, we can split up the OpsWorks Stacks by function. We'll use one OpsWorks layer for the NFS, one OpsWorks layer for the Web tier, but we'll also include the database as part of that Web tier. So, we now have two stacks. For smaller environments, it may make sense to include the NFS server as part of your stack. In my case, I'll probably create more NFS servers similar to this one, supporting other departments.

Deployment

Here, we're going to go through a complete walkthrough of the deployment of our stacks, using the architecture that was discussed in the last chapter. We'll start by creating the NFS servers to support three environments (development, QA, and production). We'll then create the MySQL nodes and, finally, the the web nodes for all three environments.

Based on our architecture from chapter 3.3, here's a hierarchical view of our stacks, layers, instances, and applications:

- Stack - NFS Stack
    - Layer - NFS 1x100
        - Instance - lampnfsprod1
    - Layer - NFS 2x100
        - Instance - lampnfsuatqa1
        - Instance - lampnfsdevsb1
- Stack - LAMP Stack
    - Layer - PHP
        - Instance - lampprod1
        - Instance - lampdev1
        - Instance - lampqa1
        - Instance - lampuat1
        - Instance - lampsb1
    - Layer - Database

We haven't split out the stacks by their environment function because I want to take a look at whether or not we can use the environment variables feature of application deployment to separate out the environments.

Creating the NFS Stack

First, we create our NFS Stack and Layers:

```
opsworks $ alias aws='aws --region us-east-1'
opsworks $ aws opsworks create-stack \
> --name "NFS Stack" \
> --stack-region us-west-2 \
> --use-custom-cookbooks \
> --default-ssh-key-name awskeypair \
> --custom-cookbooks-source '{
> "Type": "git",
> "Url": "git@bitbucket.org:haxxon_hax/awsopsworks.git",
> "Username": "haxxon_hax",
> "Password": "MyPassword"
> }' \
> --service-role-arn \
> "arn:aws:iam::550807172100:role/aws-opsworks-service-role" \
> --default-instance-profile-arn \
> "arn:aws:iam::550807172100:instance-profile/aws-opsworks-ec2-rol
> --configuration-manager '{
> "Name": "Chef",
> "Version": "11.10"
> }'
{
        "StackId": "ce267764-4453-4b2f-957c-c5111d20e0f5"
}
opsworks $ aws opsworks create-layer \
> --stack-id "ce267764-4453-4b2f-957c-c5111d20e0f5" \
> --type custom \
> --name "NFS 1x100" \
> --shortname nfs_1x100 \
> --custom-security-group-ids \
>   '[
>          "sg-35b56751",
>          "sg-448e0820",
>          "sg-478e0823"
>       ]' \
> --custom-recipes \
>    '{
>       "Deploy": [ "alanscookbook::nfs" ]
>     }'\
> --volume-configurations \
>       '[
>          {
>            "MountPoint": "/export1",
>             "Size": 100,
>                    "VolumeType": "standard",
>                    "NumberOfDisks": 1
>          }
>       ]'
{
        "LayerId": "df80dd7a-1327-4100-98c7-cece7e41a6c2"
}
opsworks $ aws opsworks create-layer \
> --stack-id "ce267764-4453-4b2f-957c-c5111d20e0f5" \
> --type custom \
> --name "NFS 2x100" \
> --shortname nfs_2x100 \
> --custom-security-group-ids \
> '[
>   "sg-35b56751",
>   "sg-448e0820",
>   "sg-478e0823"
> ]' \
> --custom-recipes \
> '{
>        "Deploy": [ "alanscookbook::nfs" ]
> }' \
> --volume-configurations \
> '[
>   {
>          "MountPoint": "/export1",
>           "Size": 100,
>              "VolumeType": "standard",
>             "NumberOfDisks": 1
>        },
>        {
>        "MountPoint": "/export2",
>          "Size": 100,
>          "VolumeType": "standard",
>          "NumberOfDisks": 1
>        }
> ]'
{
        "LayerId": "4fb70db4-5b2e-438e-b10d-d5d0f6fdbaac"
}
opsworks $
```

If we don't want our repository public and prefer to use SSH for our git deployments, we'll also need to go to the Web UI and paste our BitBucket deployment Key in the stack settings. Before we can create our PHP layers, we’ll need to know the NFS instances’ IP addresses. This can be avoided using DNS, but… I’m cheap.

```
opsworks $ aws opsworks create-instance \
> --stack-id "ce267764-4453-4b2f-957c-c5111d20e0f5" \
> --layer-ids "df80dd7a-1327-4100-98c7-cece7e41a6c2" \
> --instance-type t2.small \
> --ssh-key-name awskeypair \
> --os "Amazon Linux 2016.03" \
> --root-device-type ebs \
> --block-device-mappings \
> '[
>   {
>     "DeviceName": "ROOT_DEVICE",
>     "Ebs": {
>       "VolumeSize": 20,
>       "VolumeType": "gp2",
>       "DeleteOnTermination": true
>     }
>   }
>  ]'
{
        "InstanceId": "7b4a0d9a-81d7-4307-a959-114cd5c57532"
}
opsworks $ aws opsworks start-instance \
> --instance-id "7b4a0d9a-81d7-4307-a959-114cd5c57532"
opsworks $ aws opsworks create-instance \
> --stack-id "ce267764-4453-4b2f-957c-c5111d20e0f5" \
> --layer-ids "4fb70db4-5b2e-438e-b10d-d5d0f6fdbaac" \
> --instance-type t2.small \
> --ssh-key-name awskeypair \
> --os "Amazon Linux 2016.03" \
> --root-device-type ebs \
> --block-device-mappings \
> '[
>   {
>         "DeviceName": "ROOT_DEVICE",
>         "Ebs": {
>           "VolumeSize": 20,
>           "VolumeType": "gp2",
>           "DeleteOnTermination": true
>         }
>   }
>  ]'
{
        "InstanceId": "d387cc81-2780-49ac-a0d0-604a2ccff898"
}
opsworks $ aws opsworks start-instance \
> --instance-id "d387cc81-2780-49ac-a0d0-604a2ccff898"
opsworks $ aws opsworks create-instance \
> --stack-id "ce267764-4453-4b2f-957c-c5111d20e0f5" \
> --layer-ids "4fb70db4-5b2e-438e-b10d-d5d0f6fdbaac" \
> --instance-type t2.small \
> --ssh-key-name awskeypair \
> --os "Amazon Linux 2016.03" \
> --root-device-type ebs \
> --block-device-mappings \
> '[
>   {
>          "DeviceName": "ROOT_DEVICE",
>          "Ebs": {
>            "VolumeSize": 20,
>            "VolumeType": "gp2",
>            "DeleteOnTermination": true
>          }
>   }
>  ]'
{
        "InstanceId": "e3c421ac-365f-475c-98d5-58956fe0287c"
}
opsworks $ aws opsworks start-instance \
> --instance-id "e3c421ac-365f-475c-98d5-58956fe0287c"
opsworks $ aws opsworks describe-instances \
> --stack-id "ce267764-4453-4b2f-957c-c5111d20e0f5" \
>  | egrep -i 'Hostname|PrivateIp'
                "Hostname": "nfs-1x1001",
                "PrivateIp": "172.31.23.15",
                "Hostname": "nfs-2x1001",
                "PrivateIp": "172.31.30.84",
                "Hostname": "nfs-2x1002",
                "PrivateIp": "172.31.22.140",
opsworks $
```

Now that we have our Private IPs, we can create our Chef recipes for the PHP applications. Because we're going to use Chef 11.10, we can now take advantage of the custom environment variables.

```
opsworks $ vi lamp/recipes/mount_nfs.rb
if node[:deploy]['lamp'][:environment_variables][:USE_ENVIRONMENT]
```

```
  mount '/srv/www' do
        device '172.31.30.84:/export1'
        fstype 'nfs'
        options 'rw'
        action [:mount,:enable]
  end
elsif node[:deploy]['lamp'][:environment_variables][:USE_ENVIRONMEN
  mount '/srv/www' do
        device '172.31.30.84:/export2'
        fstype 'nfs'
        options 'rw'
        action [:mount,:enable]
  end
elsif node[:deploy]['lamp'][:environment_variables][:USE_ENVIRONMEN
  mount '/srv/www' do
        device '172.31.22.140:/export1'
        fstype 'nfs'
        options 'rw'
        action [:mount,:enable]
  end
elsif node[:deploy]['lamp'][:environment_variables][:USE_ENVIRONMEN
  mount '/srv/www' do
        device '172.31.22.140:/export2'
        fstype 'nfs'
        options 'rw'
        action [:mount,:enable]
  end
elsif node[:deploy]['lamp'][:environment_variables][:USE_ENVIRONMEN
  mount '/srv/www' do
        device '172.31.23.15:/export1'
        fstype 'nfs'
        options 'rw'
        action [:mount,:enable]
  end
end
opsworks $ git add .
opsworks $ git commit -m "Added if..elsif for Environment support"
[master c95ff45] Added if..elsif for Environment support
 1 file changed, 36 insertions(+), 8 deletions(-)
 rewrite lamp/recipes/mount_nfs.rb (96%)
opsworks $ git push
Counting objects: 9, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 732 bytes | 0 bytes/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To https://haxxon_hax@bitbucket.org/haxxon_hax/awsopsworks.git
   c318e7d..c95ff45  master -> master
opsworks $
```

Of course, this is only the first part. We'll need to make sure we include AWS's deploy::php recipe after this recipe is run. We do that when we create the layers. We also want to make sure we get an elastic IP for each instance, so our developers will always access through the same IP; if we build replacement systems, then we won't have to communicate new IPs.

Creating the LAMP Stack

We have our IP addresses and repositories configured,

```
opsworks $ aws opsworks create-stack \
> --name "LAMP Stack" \
> --stack-region us-west-2 \
> --use-custom-cookbooks \
> --default-ssh-key-name awskeypair \
> --custom-cookbooks-source '{
> "Type": "git",
> "Url": "git@bitbucket.org:haxxon_hax/awsopsworks.git",
> "Username": "haxxon_hax",
> "Password": "abc12345"
> }' \
> --service-role-arn \
> "arn:aws:iam::550807172100:role/aws-opsworks-service-role" \
> --default-instance-profile-arn \
> "arn:aws:iam::550807172100:instance-profile/aws-opsworks-ec2-rol
> --configuration-manager '{
> "Name": "Chef",
> "Version": "11.10"
> }'
{
    A>     "StackId": "154b191f-9ad6-4571-99f1-6b207a9619e0"
}
opsworks $ aws opsworks create-layer \
> --stack-id "154b191f-9ad6-4571-99f1-6b207a9619e0" \
> --type custom \
> --name "PHP Layer" \
> --shortname "php_lamp" \
> --custom-security-group-ids \
> '[
>   "sg-448e0820",
> 	"sg-478e0823",
> 	"sg-5f53823b"
> ]' \
> --custom-recipes \
> '{
> 	"Deploy": [ "lamp::mount_nfs", "deploy::php" ]
> }' \
> --auto-assign-elastic-ips
{
	"LayerId": "ece241c2-57c8-41fc-b532-a8a12b1164cb"
}
opsworks $
```

To add a database layer in the Web UI, we have the options to set the MySQL root user password and whether or not the system should set the root user password on every instance. In the help document for the CLI, it doesn’t list these options. However, they are available via the the “–attributes” flag. AWS documentation doesn’t specify what “MysqlRootPasswordUbiquitous” means, but by logic, I can make a warranted assumption that this means to set the root user password on every instance.

```
opsworks $ aws opsworks create-layer \
> --stack-id "154b191f-9ad6-4571-99f1-6b207a9619e0" \
> --type db-master \
> --name "MySQL Layer" \
> --shortname "awsmysql" \
> --custom-security-group-ids \
> '[
> 	"sg-448e0820",
> 	"sg-478e0823",
> 	"sg-64f94700"
> ]' \
> --attributes \
> '{
> 		"MysqlRootPassword" : "This_isnt_a_very_str0ng_p@ssword.",
> 		"MysqlRootPasswordUbiquitous": "true"
> }'
```

Now that we’ve created the layer, we’ll find that we won’t be able to add another database layer. There’s not really any need for another database layer, anyway; it only serves one function. We can create our instances for the database and application. We’ll create two database instances.

```
opsworks $ aws opsworks create-instance \
> --stack-id "154b191f-9ad6-4571-99f1-6b207a9619e0" \
> --layer-ids "d8693dc3-9bfa-4438-9e10-e5dc6199623e" \
> --instance-type t2.small \
> --ssh-key-name awskeypair \
> --os "Amazon Linux 2016.03" \
> --root-device-type ebs \
> --block-device-mappings \
> '[
>   {
>         "DeviceName": "ROOT_DEVICE",
>         "Ebs": {
>           "VolumeSize": 20,
>           "VolumeType": "gp2",
>           "DeleteOnTermination": true
>         }
>   }
> ]'
{
        "InstanceId": "c90598fa-72a2-408b-9857-0b3f1a772ff5"
}
opsworks $ aws opsworks start-instance \
> --instance-id "c90598fa-72a2-408b-9857-0b3f1a772ff5"
opsworks $ aws opsworks create-instance \
> --stack-id "154b191f-9ad6-4571-99f1-6b207a9619e0" \
> --layer-ids "d8693dc3-9bfa-4438-9e10-e5dc6199623e" \
> --instance-type t2.small \
> --ssh-key-name awskeypair \
> --os "Amazon Linux 2016.03" \
> --root-device-type ebs \
> --block-device-mappings \
> '[
>   {
>         "DeviceName": "ROOT_DEVICE",
>         "Ebs": {
>           "VolumeSize": 20,
>           "VolumeType": "gp2",
>            "DeleteOnTermination": true
>          }
>    }
> ]'
{
        "InstanceId": "c9270ba7-7179-494a-9ee5-085a2c93eddf"
}
opsworks $ aws opsworks start-instance \
> --instance-id "c9270ba7-7179-494a-9ee5-085a2c93eddf"
opsworks $

After trying to start the instances, we find a failure in the log.
And we'll create five application instances.  We can run a quick "
 creating these.

opsworks $ for i in {1..5}; do
> aws opsworks create-instance \
> --stack-id "154b191f-9ad6-4571-99f1-6b207a9619e0" \
> --layer-ids "ece241c2-57c8-41fc-b532-a8a12b1164cb" \
> --instance-type t2.small \
> --ssh-key-name awskeypair \
> --os "Amazon Linux 2016.03" \
> --root-device-type ebs \
> --block-device-mappings \
> '[
>    {
>          "DeviceName": "ROOT_DEVICE",
>          "Ebs": {
>            "VolumeSize": 20,
>            "VolumeType": "gp2",
>            "DeleteOnTermination": true
>          }
>    }
> ]'
> done
{
        "InstanceId": "1a64d1e1-d2ad-4af3-ad5b-ac9df0d11020"
}
{
        "InstanceId": "c7d069a6-834f-4b57-8690-eff73e438105"
}
{
        "InstanceId": "1fd5dfe2-4928-487d-bec1-17bf470db11c"
}
{
        "InstanceId": "0a026b7c-3567-426c-ad6b-25bcfa1e09f8"
}
{
        "InstanceId": "7fbceefc-ad63-474b-a22e-2e52033ba6fe"
}
```

```
opsworks $ aws opsworks start-instance \
> --instance-id "1a64d1e1-d2ad-4af3-ad5b-ac9df0d11020"
opsworks $ aws opsworks start-instance \
> --instance-id "c7d069a6-834f-4b57-8690-eff73e438105"
opsworks $ aws opsworks start-instance \
> --instance-id "1fd5dfe2-4928-487d-bec1-17bf470db11c"
opsworks $ aws opsworks start-instance \
> --instance-id "0a026b7c-3567-426c-ad6b-25bcfa1e09f8"
opsworks $ aws opsworks start-instance \
> --instance-id "7fbceefc-ad63-474b-a22e-2e52033ba6fe"
opsworks $
```

My instances now show a failure in the log. After inspecting this, it gives a stacktrace to Chef for an undefined method ‘[]’. This is within the node attributes. Let’s try changing the format to a node[:deploy].each, and remove the appname.

```
opsworks $ vi lamp/recipes/mount_nfs.rb
node[:deploy].each do |app, deploy|
  if deploy[:environment_variables][:USE_ENVIRONMENT] == 'SANDBOX'
        mount '/srv/www' do
        device '172.31.30.84:/export1'
        fstype 'nfs'
        options 'rw'
        action [:mount,:enable]
      end
  elsif node[:deploy]['lamp'][:environment_variables][:USE_ENVIRONI
      mount '/srv/www' do
        device '172.31.30.84:/export2'
node[:deploy].each do |app, deploy|
  if deploy[:environment_variables][:USE_ENVIRONMENT] == 'SANDBOX'
        mount '/srv/www' do
        device '172.31.30.84:/export1'
        fstype 'nfs'
        options 'rw'
        action [:mount,:enable]
      end
  elsif deploy[:environment_variables][:USE_ENVIRONMENT] == 'DEV'
      mount '/srv/www' do
        device '172.31.30.84:/export2'
        fstype 'nfs'
        options 'rw'
```

```
            action [:mount,:enable]
          end
    elsif deploy[:environment_variables][:USE_ENVIRONMENT] == 'QA'
          mount '/srv/www' do
            device '172.31.22.140:/export1'
            fstype 'nfs'
            options 'rw'
            action [:mount,:enable]
          end
    elsif deploy[:environment_variables][:USE_ENVIRONMENT] == 'UAT'
          mount '/srv/www' do
            device '172.31.22.140:/export2'
            fstype 'nfs'
            options 'rw'
            action [:mount,:enable]
          end
    elsif deploy[:environment_variables][:USE_ENVIRONMENT] == 'PROD'
          mount '/srv/www' do
            device '172.31.23.15:/export1'
            fstype 'nfs'
            options 'rw'
            action [:mount,:enable]
          end
    end
  end
  opsworks $ git add .
  opsworks $ git commit -m "Changed deployment to a node.each"
  [master 5ee1499] Changed deployment to a node.each
   1 file changed, 38 insertions(+), 36 deletions(-)
   rewrite lamp/recipes/mount_nfs.rb (99%)
  opsworks $ git push
  Counting objects: 9, done.
  Delta compression using up to 4 threads.
  Compressing objects: 100% (5/5), done.
  Writing objects: 100% (5/5), 669 bytes | 0 bytes/s, done.
  Total 5 (delta 1), reused 0 (delta 0)
  To https://haxxon_hax@bitbucket.org/haxxon_hax/awsopsworks.git
     1af0593..5ee1499  master -> master
  opsworks $
  opsworks $ vi lamp/recipes/mount_nfs.rb
  node[:deploy].each do |app, deploy|
    if deploy[:environment_variables][:USE_ENVIRONMENT] == 'SANDBOX'
            mount '/srv/www' do
            device '172.31.30.84:/export1'
            fstype 'nfs'
            options 'rw'
            action [:mount,:enable]
          end
    elsif node[:deploy]['lamp'][:environment_variables][:USE_ENVIRON
          mount '/srv/www' do
            device '172.31.30.84:/export2'
  node[:deploy].each do |app, deploy|
    if deploy[:environment_variables][:USE_ENVIRONMENT] == 'SANDBOX'
            mount '/srv/www' do
            device '172.31.30.84:/export1'
            fstype 'nfs'
            options 'rw'
            action [:mount,:enable]
          end
    elsif deploy[:environment_variables][:USE_ENVIRONMENT] == 'DEV'
          mount '/srv/www' do
            device '172.31.30.84:/export2'
            fstype 'nfs'
            options 'rw'
            action [:mount,:enable]
          end
    elsif deploy[:environment_variables][:USE_ENVIRONMENT] == 'QA'
          mount '/srv/www' do
            device '172.31.22.140:/export1'
            fstype 'nfs'
            options 'rw'
            action [:mount,:enable]
          end
    elsif deploy[:environment_variables][:USE_ENVIRONMENT] == 'UAT'
          mount '/srv/www' do
            device '172.31.22.140:/export2'
            fstype 'nfs'
            options 'rw'
            action [:mount,:enable]
          end
    elsif deploy[:environment_variables][:USE_ENVIRONMENT] == 'PROD'
          mount '/srv/www' do
            device '172.31.23.15:/export1'
            fstype 'nfs'
            options 'rw'
            action [:mount,:enable]
          end
    end
  end
  opsworks $ git add .
  opsworks $ git commit -m "Changed deployment to a node.each"
  [master 5ee1499] Changed deployment to a node.each
   1 file changed, 38 insertions(+), 36 deletions(-)
   rewrite lamp/recipes/mount_nfs.rb (99%)
  opsworks $ git push
  Counting objects: 9, done.
  Delta compression using up to 4 threads.
  Compressing objects: 100% (5/5), done.
  Writing objects: 100% (5/5), 669 bytes | 0 bytes/s, done.
  Total 5 (delta 1), reused 0 (delta 0)
```

```
To https://haxxon_hax@bitbucket.org/haxxon_hax/awsopsworks.git
   1af0593..5ee1499  master -> master
opsworks $
```

Now, we’re able to bring the instances online, but we’ll want to now try to create an application with the environment variables. We’re going to use php-lamp5 as a production server, so we’ll create a production app and start there.

```
opsworks $ aws opsworks create-app \
> --stack-id "154b191f-9ad6-4571-99f1-6b207a9619e0" \
> --type php \
> --name MyProdApp \
> --app-source '{
>   "Type": "git",
>   "Url" : "git@bitbucket.org:haxxon_hax/allstars.git",
>   "SshKey": "`cat ~/.ssh/id_rsa`"
> }' \
> --environment '{
>   "Key": "USE_ENVIRONMENT",
>   "Value": "PROD",
>   "Secure": false
> }'
{
        "AppId": "94baf722-2698-44ed-bb1a-1d4dc531bc26"
}
opsworks $ aws opsworks create-deployment \
> --stack-id "154b191f-9ad6-4571-99f1-6b207a9619e0" \
> --app-id "94baf722-2698-44ed-bb1a-1d4dc531bc26" \
> --instance-ids "7fbceefc-ad63-474b-a22e-2e52033ba6fe" \
> --command '{
>   "Name": "deploy"
> }'
{
        "DeploymentId": "f7b8921d-4581-4518-b083-ec0612cee6d7"
}
opsworks $ aws opsworks describe-instances \
> --instance-ids "7fbceefc-ad63-474b-a22e-2e52033ba6fe" \
> | grep PublicIp
                "PublicIp": "52.3.7.23"
opsworks $
opsworks $ ssh -i ~/.ssh/awskeypair.pem ec2-user@52.3.7.23
```

```
The authenticity of host '52.3.7.23 (52.3.7.23)' can't be establisl
ECDSA key fingerprint is 93:2d:42:5a:06:e8:7c:32:bd:13:58:07:e5:d4
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.3.7.23' (ECDSA) to the list of know
 This instance is managed with AWS OpsWorks.


   ######  OpsWorks Summary  ######
   Operating System: Amazon Linux AMI release 2016.03
   OpsWorks Instance: php-lamp5
   OpsWorks Instance ID: 7fbceefc-ad63-474b-a22e-2e52033ba6fe
   OpsWorks Layers: PHP Layer
   OpsWorks Stack: LAMP Stack
   EC2 Region: us-west-2
   EC2 Availability Zone: us-west-2a
   EC2 Instance ID: i-684ca6c7
   Public IP: 52.3.7.23
   Private IP: 172.3.3.18
   VPC ID: vpc-7d0e6f18
   Subnet ID: subnet-e96c138c


 Visit http://aws.amazon.com/opsworks for more information.
[ec2-user@php-lamp5 ~]$ df -h
Filesystem                 Size  Used Avail Use% Mounted on
/dev/xvda1                  20G  2.0G   18G  10% /
devtmpfs                   994M   60K  994M   1% /dev
tmpfs                     1002M     0 1002M   0% /dev/shm
172.31.23.15:/export1  100G   33M  100G   1% /srv/www
[ec2-user@php-lamp5 ~]$ ls /srv/www/myprodapp/current
access-denied.php  index.php   login-form.php  mysql.sql  register
auth.php    license.txt     loginmodule.css opsworks.php  register
config      log             logout.php      public        register
config.php  login-exec.php  member-index.php  readme.html tmp
contributors.txt  login-failed.php member-profile.php README.md  u
[ec2-user@php-lamp5 ~]$
[ec2-user@php-lamp5 ~]$ exit
logout
Connection to 52.3.7.23 closed.
opsworks $
```

So, we’re now able to deploy the applications successfully in /srv/www. We can now create the application definitions for our sandbox, development, QA, and UAT servers. We also only need to do this if each uses a separate repository. In most normal environments, there would be 1 or 2 repositories used, and content promotion would be done through deployments. Let’s log into one of the other systems and take a look at its NFS mounts.

```
opsworks $ ssh -i ~/.ssh/awskeypair.pem ec2-user@52.41.49.144
The authenticity of host '52.41.49.144 (52.41.49.144)' can't be es
ECDSA key fingerprint is 8e:14:d9:3c:67:e0:e4:39:d4:67:f5:9b:b8:68
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.41.49.144' (ECDSA) to the list of k
 This instance is managed with AWS OpsWorks.


   ######  OpsWorks Summary  ######
   Operating System: Amazon Linux AMI release 2016.03
   OpsWorks Instance: php-lamp1
   OpsWorks Instance ID: 1a64d1e1-d2ad-4af3-ad5b-ac9df0d11020
   OpsWorks Layers: PHP Layer
   OpsWorks Stack: LAMP Stack
   EC2 Region: us-west-2
   EC2 Availability Zone: us-west-2a
   EC2 Instance ID: i-9d4da732
   Public IP: 52.41.49.144
   Private IP: 172.31.16.204
   VPC ID: vpc-7d0e6f18
   Subnet ID: subnet-e96c138c

 Visit http://aws.amazon.com/opsworks for more information.
[ec2-user@php-lamp1 ~]$ df -h
Filesystem                 Size  Used Avail Use% Mounted on
/dev/xvda1                  20G  2.1G   18G  11% /
devtmpfs                   994M   60K  994M   1% /dev
tmpfs                     1002M        0 1002M   0% /dev/shm
172.31.23.15:/export1  100G   34M  100G   1% /srv/www
[ec2-user@php-lamp1 ~]$
```

Oops. It looks like it’s mounting our production systems even if there’s no environment. This is because we didn’t specify an application name for our environment variables. Our application is the recipe name (lamp), so we need to change our mount_nfs.rb file.

```
opsworks $ vi lamp/recipes/mount_nfs.rb
if (defined?(node[:deploy]['php'])).nil?
  if node[:deploy]['lamp'][:environment_variables][:USE_ENVIRONMEN
        mount '/srv/www' do
          device '172.31.30.84:/export1'
          fstype 'nfs'
          options 'rw'
          action [:mount,:enable]
        end
  elsif node[:deploy]['lamp'][:environment_variables][:USE_ENVIRON
        mount '/srv/www' do
          device '172.31.30.84:/export2'
          fstype 'nfs'
          options 'rw'
          action [:mount,:enable]
        end
  elsif node[:deploy]['lamp'][:environment_variables][:USE_ENVIRON
        mount '/srv/www' do
          device '172.31.22.140:/export1'
          fstype 'nfs'
          options 'rw'
          action [:mount,:enable]
        end
  elsif node[:deploy]['lamp'][:environment_variables][:USE_ENVIRON
        mount '/srv/www' do
          device '172.31.22.140:/export2'
          fstype 'nfs'
          options 'rw'
          action [:mount,:enable]
        end
  elsif node[:deploy]['lamp'][:environment_variables][:USE_ENVIRON
        mount '/srv/www' do
          device '172.31.23.15:/export1'
          fstype 'nfs'
          options 'rw'
          action [:mount,:enable]
        end
  end
end
opsworks $ git add .
opsworks $ git commit -m "Added 'lamp'."
[master bb19b24] Added conditional if defined.
 1 file changed, 6 insertions(+), 6 deletions(-)
opsworks $ git push
Counting objects: 9, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 556 bytes | 0 bytes/s, done.
Total 5 (delta 1), reused 0 (delta 0)
```

```
To https://haxxon_hax@bitbucket.org/haxxon_hax/awsopsworks.git
   efd4e06..bb19b24  master -> master
opsworks $
```

When we try to start another instance, we get another failure, but this one is rather cryptic for an undefined method ‘[]’ for nil:NilClass. What’s this, then? Well, it appears that the node[:deploy][‘lamp’] doesn’t get defined until the application is created and set to deploy. We need to set a conditional if the method is defined.

```
opsworks $ vi lamp/recipes/mount_nfs.rb
if (defined?(node[:deploy]['lamp'])).nil?
  if node[:deploy]['lamp'][:environment_variables][:USE_ENVIRONMENT
        mount '/srv/www' do
          device '172.31.30.84:/export1'
          fstype 'nfs'
          options 'rw'
          action [:mount,:enable]
        end
  elsif node[:deploy]['lamp'][:environment_variables][:USE_ENVIRONM
        mount '/srv/www' do
          device '172.31.30.84:/export2'
          fstype 'nfs'
          options 'rw'
          action [:mount,:enable]
        end
  elsif node[:deploy]['lamp'][:environment_variables][:USE_ENVIRONM
        mount '/srv/www' do
          device '172.31.22.140:/export1'
          fstype 'nfs'
          options 'rw'
          action [:mount,:enable]
        end
  elsif node[:deploy]['lamp'][:environment_variables][:USE_ENVIRONM
        mount '/srv/www' do
          device '172.31.22.140:/export2'
          fstype 'nfs'
          options 'rw'
          action [:mount,:enable]
        end
  elsif node[:deploy]['lamp'][:environment_variables][:USE_ENVIRONM
        mount '/srv/www' do
```

```
          device '172.31.23.15:/export1'
          fstype 'nfs'
          options 'rw'
          action [:mount,:enable]
        end
  end
end
opsworks $ git add .
opsworks $ git commit -m "Added conditional if defined."
[master bb19b24] Added conditional if defined.
 1 file changed, 6 insertions(+), 6 deletions(-)
opsworks $ git push
Counting objects: 9, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 556 bytes | 0 bytes/s, done.
Total 5 (delta 1), reused 0 (delta 0)
To https://haxxon_hax@bitbucket.org/haxxon_hax/awsopsworks.git
   efd4e06..bb19b24  master -> master
opsworks $
```

Now that we've added the conditional, we're able to start up one of the instances. Let's log in to see if we're getting any NFS mounts.

```
opsworks $ ssh -i ~/.ssh/awskeypair.pem ec2-user@52.3.15.13
The authenticity of host '52.3.15.13 (52.3.15.13)' can't be establ
ECDSA key fingerprint is 8a:a3:f4:47:3d:65:7a:a6:80:bd:88:76:77:83
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.3.15.13' (ECDSA) to the list of kno
 This instance is managed with AWS OpsWorks.

   ######  OpsWorks Summary  ######
   Operating System: Amazon Linux AMI release 2016.03
   OpsWorks Instance: php-lamp2
   OpsWorks Instance ID: c7d069a6-834f-4b57-8690-eff73e438105
   OpsWorks Layers: PHP Layer
   OpsWorks Stack: LAMP Stack
   EC2 Region: us-west-2
   EC2 Availability Zone: us-west-2a
   EC2 Instance ID: i-064da7a9
   Public IP: 52.3.15.13
   Private IP: 172.3.2.25
   VPC ID: vpc-7d0e6f18
   Subnet ID: subnet-e96c138c

  Visit http://aws.amazon.com/opsworks for more information.
[ec2-user@php-lamp2 ~]$ df -h
Filesystem          Size  Used Avail Use% Mounted on
/dev/xvda1           20G  2.1G   18G  11% /
devtmpfs            994M   56K  994M   1% /dev
tmpfs              1002M     0 1002M   0% /dev/shm
[ec2-user@php-lamp2 ~]$
```

So far, it looks good. We'll create a new app for development deploy it, then log back in to see if it's mounted the NFS mounts correctly.

```
opsworks $ aws opsworks create-app \
> --stack-id "154b191f-9ad6-4571-99f1-6b207a9619e0" \
> --type php \
> --name MyDevApp \
> --app-source '{
>   "Type": "git",
>   "Url" : "https://haxxon_hax@bitbucket.org/haxxon_hax/awsopswor
>  }' \
> --environment '{
>        "Key": "USE_ENVIRONMENT",
>        "Value": "DEV",
>        "Secure": false
> }'
{
        "AppId": "9b25878a-afe0-4c07-a36b-6a044e4e7c5d"
}
opsworks $ aws opsworks create-deployment \
> --stack-id "154b191f-9ad6-4571-99f1-6b207a9619e0" \
> --app-id "94baf722-2698-44ed-bb1a-1d4dc531bc26" \
> --instance-ids "7fbceefc-ad63-474b-a22e-2e52033ba6fe" \
> --command '{
>   "Name": "deploy"
> }'
{
        "DeploymentId": "f7b8921d-4581-4518-b083-ec0612cee6d7"
}
opsworks $ ssh -i ~/.ssh/awskeypair.pem ec2-user@52.3.15.13
The authenticity of host '52.3.15.13 (52.3.15.13)' can't be establ
ECDSA key fingerprint is 8a:a3:f4:47:3d:65:7a:a6:80:bd:88:76:77:83
```

```
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.3.15.13' (ECDSA) to the list of kno
 This instance is managed with AWS OpsWorks.

   ######  OpsWorks Summary  ######
   Operating System: Amazon Linux AMI release 2016.03
   OpsWorks Instance: php-lamp2
   OpsWorks Instance ID: c7d069a6-834f-4b57-8690-eff73e438105
   OpsWorks Layers: PHP Layer
   OpsWorks Stack: LAMP Stack
   EC2 Region: us-west-2
   EC2 Availability Zone: us-west-2a
   EC2 Instance ID: i-064da7a9
   Public IP: 52.3.15.13
   Private IP: 172.3.2.25
   VPC ID: vpc-7d0e6f18
   Subnet ID: subnet-e96c138c

 Visit http://aws.amazon.com/opsworks for more information.
[ec2-user@php-lamp2 ~]$ df -h
Filesystem          Size  Used Avail Use% Mounted on
/dev/xvda1           20G  2.1G   18G  11% /
devtmpfs            994M   56K  994M   1% /dev
tmpfs              1002M        0 1002M   0% /dev/shm
[ec2-user@php-lamp2 ~]$
[ec2-user@php-lamp2 ~]$ df -h
Filesystem          Size  Used Avail Use% Mounted on
/dev/xvda1           20G  2.1G   18G  11% /
devtmpfs            994M   56K  994M   1% /dev
tmpfs              1002M        0 1002M   0% /dev/shm
[ec2-user@php-lamp2 ~]$ ls /srv/www/
mydevapp  myprodapp
[ec2-user@php-lamp2 ~]$
```

Not only did it not create our NFS mounts, but it also deployed both applications! If we log into the production node, will we find mydevapp on it?

```
opsworks $ ssh -i ~/.ssh/awskeypair.pem ec2-user@52.3.7.23
 This instance is managed with AWS OpsWorks.

   ######  OpsWorks Summary  ######
   Operating System: Amazon Linux AMI release 2016.03
```

```
     OpsWorks Instance: php-lamp5
     OpsWorks Instance ID: 7fbceefc-ad63-474b-a22e-2e52033ba6fe
     OpsWorks Layers: PHP Layer
     OpsWorks Stack: LAMP Stack
     EC2 Region: us-west-2
     EC2 Availability Zone: us-west-2a
     EC2 Instance ID: i-684ca6c7
     Public IP: 52.3.7.23
     Private IP: 172.3.3.18
     VPC ID: vpc-7d0e6f18
     Subnet ID: subnet-e96c138c

    Visit http://aws.amazon.com/opsworks for more information.
  [ec2-user@php-lamp5 ~]$ ls /var/
  [ec2-user@php-lamp5 ~]$ ls /srv/www/
  myprodapp
  [ec2-user@php-lamp5 ~]$
```

Well, that's good. We didn't deploy dev onto our production. There must be either (a) something wrong with our logic, or (b) a bad implementation of environment variables on AWS. Actually, the problem is (c) we're using the stacks wrong. According to http://docs.aws.amazon.com/opsworks/latest/userguide/workingstacks.h "A common practice is to have multiple stacks that represent different environments. A typical set of stacks consists of: * A development stack to be used by developers to add features, fix bugs, and perform other development and maintenance tasks. * A staging stack to verify updates or fixes before exposing them publicly. * A production stack, which is the public-facing version that handles incoming requests from users." A little later, we'll take a look at the best way to duplicate the stacks. But let's take a look at what our environment variables are doing. The following recipe is called by specifying "debug::deploy" as a deploy recipe. It creates a file called "app_data.json" in the application subdirectories 'shared/config'. The JSON.dump specification tells OpsWorks to dump the entire node[:deploy] structure into the file in JSON format. If this is a custom recipe, then it should come after the application deployment, or we’ll get an error, “Parent directory /srv/www/lamp/shared/config does not exist.”

```
$ vi debug/recipes/deploy.rb
node[:deploy].each do |app, deploy|
  file File.join(deploy[:deploy_to], 'shared', 'config', 'app_data
        content JSON.dump(node[:deploy])
  end
end
$ cat /srv/www/lamp/shared/config
{"lamp":
  { "deploy_to": "/srv/www/lamp",
        "chef_provider":"Timestamped",
        "keep_releases":5,
        "current_path":"/srv/www/lamp/current",
        "document_root":"",
        "ignore_bundler_groups":["test","development"],
        "absolute_document_root":"/srv/www/lamp/current/",
        "rake":"/usr/local/bin/rake",
        "migrate":false,
        "migrate_command":"if [ -f Gemfile ]; then echo 'OpsWorks:
gration with bundle exec' && /usr/local/bin/bundle exec /usr/local/
echo 'OpsWorks: no Gemfile - running plain migrations' && /usr/loca
,
        "rails_env":"production",
        "action":"deploy",
        "user":"deploy",
        "group":"apache",
        "shell":"/bin/bash",
        "home":"/home/deploy",
        "sleep_before_restart":0,
        "stack": {"needs_reload":true},
        "enable_submodules":true,
        "shallow_clone":false,
        "delete_cached_copy":true,
        "purge_before_symlink":["log", "tmp/pids", "public/system"]
        "create_dirs_before_symlink":["tmp", "public", "config"],
        "symlink_before_migrate":{"config/opsworks.php":"opsworks.p
        "symlinks":{"system":"public/system", "pids":"tmp/pids", "l
        "environment":{"RAILS_ENV":null, "RUBYOPT":"", "RACK_ENV":n
, "USE_ENVIRONMENT":"DEV"},
        "environment_variables":{"USE_ENVIRONMENT":"DEV"},
        "ssl_support":false,
        "auto_npm_install_on_deploy":true,
        "nodejs":{"restart_command":"monit restart node_web_app_lam
stop node_web_app_lamp", "port":80},
        "application":"lamp",
        "application_type":"php",
        "auto_bundle_on_deploy":true,
        "deploying_user":null,
        "domains":["lamp"],
        "mounted_at":null,
        "restart_command":"echo 'restarting app'",
        "ssl_certificate":null,
        "ssl_certificate_key":null,
        "ssl_certificate_ca":null,
        "scm":{"scm_type":"git","repository":"https://haxxon_hax@b
stars.git","revision":null,"ssh_key":null,"user":null,"password":n
ed":{"host":null,"port":11211}
  }
}
```

The node[:deploy] structure has a sub-selector that can be accessed by using [‘lamp’]: node[:deploy][‘lamp’]. In our recipe above, we can change the JSON.dump selection to node[:deploy][‘lamp’], and we’ll get the inner JSON (all the content within braces after “lamp”). This also gives us the knowledge that we can still use the environment variables, having the application’s “USE_ENVIRONMENT” set differently within each stack.

Re-creating the LAMP Stack Now that we’ve created our NFS Stack and Production LAMP stack, we can create the development, QA, and staging stack. There are different ways to do this. We can go back through all the steps we went through creating the LAMP stack, or we can use the nifty “clone-stack” subcommand of AWS OpsWorks. This command requires at least two switches: “–source-stack-id” and “–service-role-arn”.

```
opsworks $ aws opsworks clone-stack \
> --source-stack-id 154b191f-9ad6-4571-99f1-6b207a9619e0 \
> --name "LAMP Dev" \
> --service-role-arn "arn:aws:iam::550807172100:role/aws-opsworks-
{
    "StackId": "90ae5e0d-a56a-4fa6-801e-02ed850d723b"
```

```
}
opsworks $ aws opsworks describe-stacks \
> --stack-id "90ae5e0d-a56a-4fa6-801e-02ed850d723b"
{
    "Stacks": [
        {
            "StackId": "90ae5e0d-a56a-4fa6-801e-02ed850d723b",
            "ServiceRoleArn": "arn:aws:iam::550807172100:role/aws-
            "VpcId": "vpc-7d0e6f18",
            "DefaultRootDeviceType": "instance-store",
            "Name": "LAMP Dev",
            "HostnameTheme": "Layer_Dependent",
            "UseCustomCookbooks": true,
            "UseOpsworksSecurityGroups": true,
            "Region": "us-west-2",
            "DefaultAvailabilityZone": "us-west-2a",
            "CreatedAt": "2016-06-25T22:14:47+00:00",
            "CustomCookbooksSource": {
                "Url": "https://haxxon_hax@bitbucket.org/haxxon_ha
                "Username": "haxxon_hax",
                "Password": "*****FILTERED*****",
                "Type": "git"
            },
            "ConfigurationManager": {
                "Version": "11.10",
                "Name": "Chef"
            },
            "ChefConfiguration": {
                "BerkshelfVersion": "3.2.0",
                "ManageBerkshelf": false
            },
            "DefaultSubnetId": "subnet-e96c138c",
            "DefaultSshKeyName": "awskeypair",
            "DefaultInstanceProfileArn": "arn:aws:iam::55080717210
orks-ec2-role",
            "Attributes": {
                "Color": null
            },
            "DefaultOs": "Amazon Linux 2016.03",
            "Arn": "arn:aws:opsworks:us-west-2:550807172100:stack/
d850d723b/",
            "AgentVersion": "3438-20160625031426"
        }
    ]
}
opsworks $ aws opsworks describe-layers \
> --stack-id "90ae5e0d-a56a-4fa6-801e-02ed850d723b"
{
    "Layers": [
        {
            "StackId": "90ae5e0d-a56a-4fa6-801e-02ed850d723b",
            "LifecycleEventConfiguration": {
                "Shutdown": {
                    "DelayUntilElbConnectionsDrained": false,
                    "ExecutionTimeout": 120
                }
            },
            "Packages": [],
            "Name": "MySQL Layer",
            "CustomRecipes": {
                "Undeploy": [],
                "Setup": [],
                "Configure": [],
                "Shutdown": [],
                "Deploy": []
            },
            "CustomSecurityGroupIds": [
                "sg-448e0820",
                "sg-478e0823",
                "sg-64f94700"
            ],
            "DefaultRecipes": {
                "Undeploy": [],
                "Setup": [
                    "opsworks_initial_setup",
                    "ssh_host_keys",
                    "ssh_users",
                    "mysql::client",
                    "dependencies",
                    "ebs",
                    "opsworks_ganglia::client",
                    "mysql::server",
                    "deploy::mysql"
                ],
                "Configure": [
                    "opsworks_ganglia::configure-client",
                    "ssh_users",
                    "mysql::client",
                    "agent_version",
                    "deploy::mysql"
                ],
                "Shutdown": [
                    "opsworks_shutdown::default",
                    "mysql::stop"
                ],
                "Deploy": [
                    "deploy::default",
                    "deploy::mysql"
                ]
            },
            "VolumeConfigurations": [
                {
                    "MountPoint": "/vol/mysql",
                    "Size": 10,
                    "VolumeType": "standard",
                    "NumberOfDisks": 1
                }
            ],
            "AutoAssignElasticIps": false,
            "EnableAutoHealing": false,
            "AutoAssignPublicIps": true,
            "UseEbsOptimizedInstances": false,
            "LayerId": "e5799aa0-8bc3-425c-9f50-50758d3a44a6",
            "Attributes": {
                "JvmVersion": null,
                "RailsStack": null,
                "EnableHaproxyStats": null,
                "MysqlRootPasswordUbiquitous": "",
                "NodejsVersion": null,
                "HaproxyHealthCheckUrl": null,
                "GangliaPassword": null,
                "Jvm": null,
                "HaproxyHealthCheckMethod": null,
                "RubyVersion": null,
                "HaproxyStatsPassword": null,
                "MysqlRootPassword": "*****FILTERED*****",
                "JavaAppServer": null,
                "MemcachedMemory": null,
                "HaproxyStatsUrl": null,
                "BundlerVersion": null,
                "PassengerVersion": null,
                "ManageBundler": null,
                "JavaAppServerVersion": null,
                "HaproxyStatsUser": null,
                "GangliaUser": null,
                "JvmOptions": null,
                "EcsClusterArn": null,
                "RubygemsVersion": null,
                "GangliaUrl": null
            },
            "Shortname": "db-master",
            "DefaultSecurityGroupNames": [
                "AWS-OpsWorks-DB-Master-Server"
            ],
            "Type": "db-master",
            "CreatedAt": "2016-06-25T22:14:47+00:00"
        },
        {
            "StackId": "90ae5e0d-a56a-4fa6-801e-02ed850d723b",
            "LifecycleEventConfiguration": {
                    "Shutdown": {
                        "DelayUntilElbConnectionsDrained": false,
                        "ExecutionTimeout": 120
                    }
                },
                "Packages": [],
                "Name": "lamp",
                "CustomRecipes": {
                    "Undeploy": [],
                    "Setup": [],
                    "Configure": [],
                    "Shutdown": [],
                    "Deploy": [
                        "lamp::deploy",
                        "deploy::php"
                    ]
                },
                "CustomSecurityGroupIds": [],
                "DefaultRecipes": {
                    "Undeploy": [],
                    "Setup": [
                        "opsworks_initial_setup",
                        "ssh_host_keys",
                        "ssh_users",
                        "mysql::client",
                        "dependencies",
                        "ebs",
                        "opsworks_ganglia::client"
                    ],
                    "Configure": [
                        "opsworks_ganglia::configure-client",
                        "ssh_users",
                        "mysql::client",
                        "agent_version"
                    ],
                    "Shutdown": [
                        "opsworks_shutdown::default"
                    ],
                    "Deploy": [
                        "deploy::default"
                    ]
                },
                "VolumeConfigurations": [],
                "AutoAssignElasticIps": false,
                "EnableAutoHealing": true,
                "AutoAssignPublicIps": true,
                "UseEbsOptimizedInstances": false,
                "LayerId": "54794319-8ad3-47ac-8f6f-22bf9be50d60",
                "Attributes": {
                    "JvmVersion": null,
                    "RailsStack": null,
                    "EnableHaproxyStats": null,
                    "MysqlRootPasswordUbiquitous": null,
                    "NodejsVersion": null,
                    "HaproxyHealthCheckUrl": null,
                    "GangliaPassword": null,
                    "Jvm": null,
                    "HaproxyHealthCheckMethod": null,
                    "RubyVersion": null,
                    "HaproxyStatsPassword": null,
                    "MysqlRootPassword": null,
                    "JavaAppServer": null,
                    "MemcachedMemory": null,
                    "HaproxyStatsUrl": null,
                    "BundlerVersion": null,
                    "PassengerVersion": null,
                    "ManageBundler": null,
                    "JavaAppServerVersion": null,
                    "HaproxyStatsUser": null,
                    "GangliaUser": null,
                    "JvmOptions": null,
                    "EcsClusterArn": null,
                    "RubygemsVersion": null,
                    "GangliaUrl": null
                },
                "Shortname": "lamp",
                "DefaultSecurityGroupNames": [
                    "AWS-OpsWorks-Custom-Server"
                ],
                "Type": "custom",
                "CreatedAt": "2016-06-25T22:14:47+00:00"
            }
        ]
    }
    opsworks $ aws opsworks describe-apps --stack-id "90ae5e0d-a56a-4f
    {
        "Apps": []
    }
    opsworks $
```

The “clone-stack” subcommand copied our layers, the layer configurations, but didn’t copy the applications or the instances. The latter is fine, because we might want to size our instances differently, or only have one node instead of two or three. For the application, however, we can use the “–cli-input-json” switch, using the source of our first application. The sections preceded with “//” indicate what should be removed. The lines preceded with //+ indicate what gets changed:

```
 opsworks $ aws opsworks describe-apps --app-id "36f99fbf-c2ad-4del
/myapp.json
 opsworks $ vi /tmp/myapp.json
 opsworks $ cat !$
 cat /tmp/myapp.json
 //{
 //    "Apps": [
         {
 //+            "StackId": "154b191f-9ad6-4571-99f1-6b207a9619e0",
             "StackId": "90ae5e0d-a56a-4fa6-801e-02ed850d723b",
             "Environment": [
                 {
                     "Value": "DEV",
                     "Key": "USE_ENVIRONMENT",
                     "Secure": false
                 }
             ],
             "AppSource": {
                 "Url": "https://haxxon_hax@bitbucket.org/haxxon_ha
                 "Type": "git"
             },
             "DataSources": [],
             "EnableSsl": false,
             "SslConfiguration": {},
 //            "AppId": "36f99fbf-c2ad-4deb-bbdd-8c3328d93614",
             "Attributes": {
 //                "RailsEnv": null,
                 "AutoBundleOnDeploy": "true",
 //                "AwsFlowRubySettings": null,
 //                "DocumentRoot": null
             },
             "Shortname": "lamp",
             "Type": "php",
 //            "CreatedAt": "2019-20-10T18:40:26+00:00",
             "Name": "lamp"
         }
 //    ]
 //}
 opsworks $ aws opsworks create-app \
 > --cli-input-json '{
```

```
 >     "StackId": "90ae5e0d-a56a-4fa6-801e-02ed850d723b",
 >     "Environment": [
 >         {
 >             "Value": "DEV",
 >             "Key": "USE_ENVIRONMENT",
 >             "Secure": false
 >         }
 >     ],
 >     "AppSource": {
 >         "Url": "https://haxxon_hax@bitbucket.org/haxxon_hax/alls
 >         "Type": "git"
 >     },
 >     "DataSources": [],
 >     "EnableSsl": false,
 >     "SslConfiguration": {},
 >     "Attributes": {
 >        "AutoBundleOnDeploy": "true"
 >     },
 >     "Shortname": "lamp",
 }
 >     "Type": "php",
 >     "Name": "lamp"
 > }
 > '

 Parameter validation failed:
 Missing required parameter in SslConfiguration: "Certificate"
 Missing required parameter in SslConfiguration: "PrivateKey"
 Invalid type for parameter Attributes.RailsEnv, value: None, type
types: <type 'basestring'>
 Invalid type for parameter Attributes.AwsFlowRubySettings, value:
e'>, valid types: <type 'basestring'>
 Invalid type for parameter Attributes.DocumentRoot, value: None, t
lid types: <type 'basestring'>
 opsworks $
```

Strangely, it asks for SslConfiguration's "Certificate" and "PrivateKey", even though we have EnableSsl set as false. We remove the SSL specifications and try again.

```
opsworks $ aws opsworks create-app --cli-input-json '{
    "StackId": "90ae5e0d-a56a-4fa6-801e-02ed850d723b",
    "Environment": [
        {
            "Value": "DEV",
            "Key": "USE_ENVIRONMENT",
            "Secure": false
        }
    ],
    "AppSource": {
        "Url": "https://haxxon_hax@bitbucket.org/haxxon_hax/allsta
        "Type": "git"
    },
    "DataSources": [],
    "Attributes": {
        "AutoBundleOnDeploy": "true"
    },
    "Shortname": "lamp",
    "Type": "php",
    "Name": "lamp"
}
'
{
    "AppId": "41885b6c-6f7d-4040-85d2-95b90c2e9ff7"
}
opsworks $ aws opsworks describe-apps \
> --app-id "41885b6c-6f7d-4040-85d2-95b90c2e9ff7"
{
    "Apps": [
        {
            "StackId": "90ae5e0d-a56a-4fa6-801e-02ed850d723b",
            "AppSource": {
                "Url": "https://haxxon_hax@bitbucket.org/haxxon_ha
                "Type": "git"
            },
            "DataSources": [],
            "Environment": [
                {
                    "Value": "DEV",
                    "Key": "USE_ENVIRONMENT",
                    "Secure": false
                }
            ],
            "SslConfiguration": {},
            "AppId": "41885b6c-6f7d-4040-85d2-95b90c2e9ff7",
            "Attributes": {
                "RailsEnv": null,
                "AutoBundleOnDeploy": "true",
                "AwsFlowRubySettings": null,
                "DocumentRoot": null
            },
            "Shortname": "lamp",
            "Type": "php",
            "CreatedAt": "2016-06-25T23:02:30+00:00",
            "Name": "lamp"
        }
    ]
}
```

We now have an application within our DEV stack, with the same shortname. We can use this to our advantage with the USE_ENVIRONMENT with our NFS mounts, or we can create a layer that includes our NFS server and programmatically mount the NFS server within the stack.

Conclusion

We were able to create one NFS stack. For our web tier (our LAMP stack), we weren’t able to use the environment variables to separate our apps within a stack, and it appears that all apps within a stack get deployed to an instance. We were, however, able to clone a stack to create the other environment. With some trial and error, we learned how environment variables worked in the applications. We can use these environment variables after cloning a stack and its application.

Epilogue

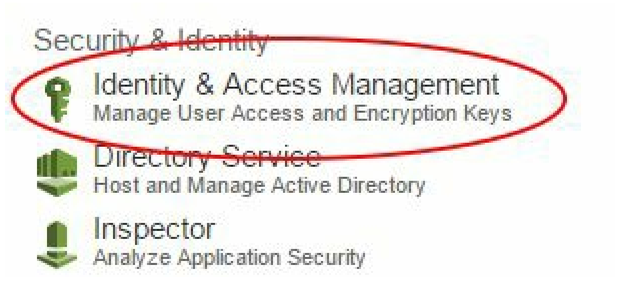
We explored the DevOps of Elasticbeanstalk and AWS OpsWorks. For smaller sites, Elasticbeanstalk is simple, easy to start, and easy to deploy. If we integrate Git for tracking changes to the site, deploying to the Elasticbeanstalk requires us to commit changes before deploying. With .ebextensions, we have some customizability of the Elasticbeanstalk instances, but this customization has its limitations. Elasticbeanstalk also serves one purpose: deploying an app. For greater customization or custom instances, AWS OpsWorks gave us greater flexibility.

With AWS OpsWorks, we were able to build out entire stacks of applications, including NFS servers, MySQL systems, and Apache web servers with PHP support. Adding and removing users to the OpsWorks is quite different than the Elasticbeanstalk, but has its advantages, allowing us to use AWS's Identity and Access Management. We are also able to give the users sudo access.

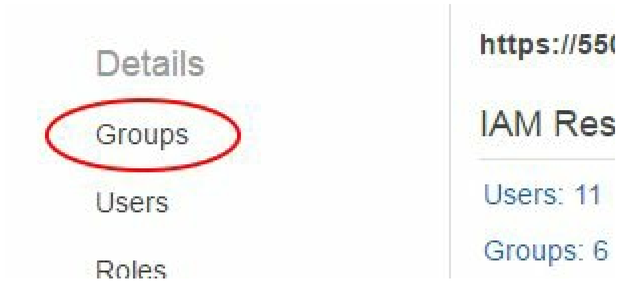
This book has been quite a learning process and I hope that I have included enough information in here to keep you from making a lot of the same mistakes that I have. While I didn't explore every facet of Opsworks and Elasticbeanstalk, I believe that the information here is a great resource to use as a model for other features, such as adding packages or software to an instance. In the next book, I hope to explore more of the DevOps features, such as CloudFormation, CodeDeploy, and CodeCommit.

Appendix A: Creating IAM Credentials

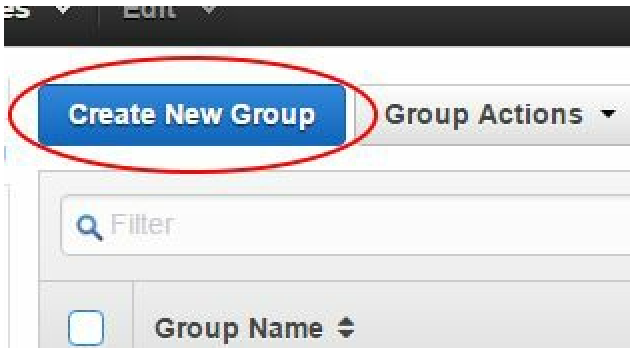
Create the Group


Security & Identity
Identity & Access Management
Manage User Access and Encryption Keys
Directory Service
Host and Manage Active Directory
Inspector
Analyze Application Security

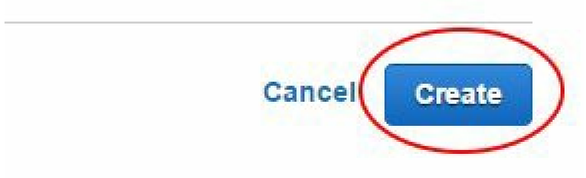
A-1



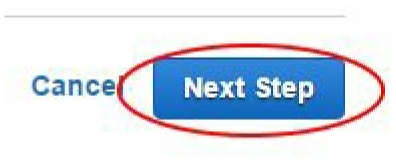
A-2



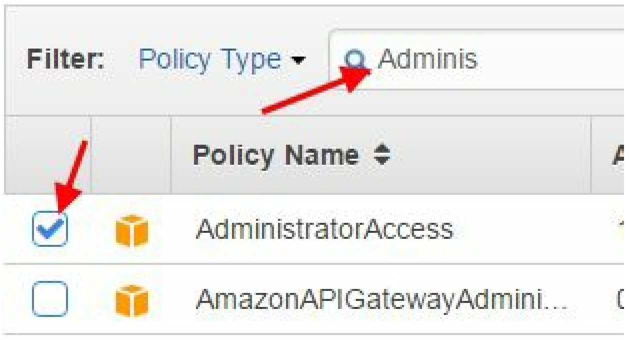
A-3



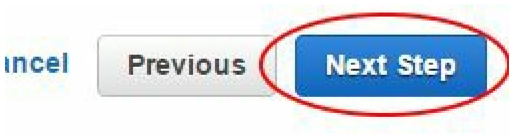
A-4



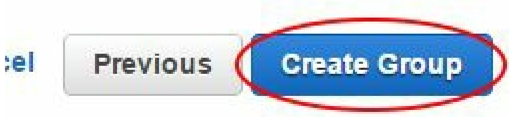
A-5



A-6

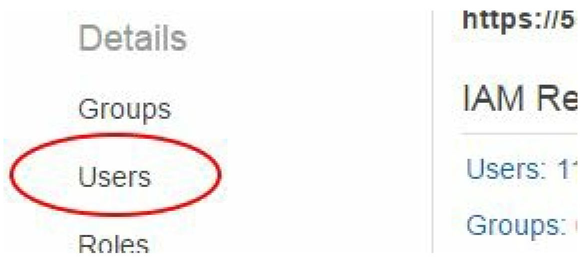


A-7

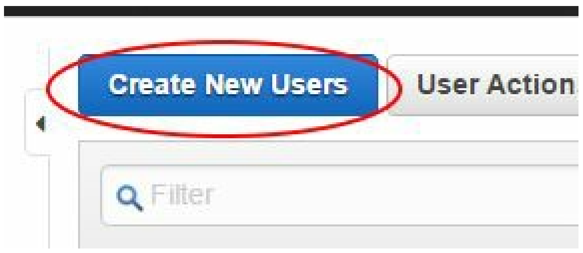


A-8

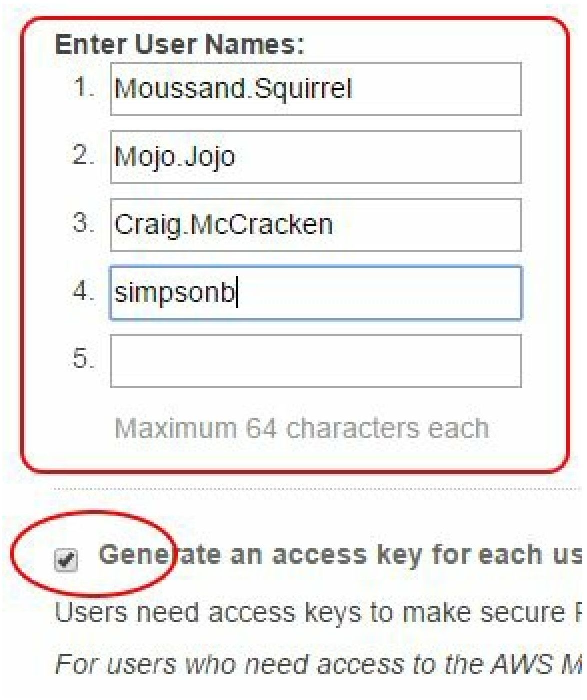
Create the User



A-9



A-10


Enter User Names:

1. Moussand.Squirrel
2. Mojo.Jojo
3. Craig.McCracken
4. simpsonb
5.

Maximum 64 characters each

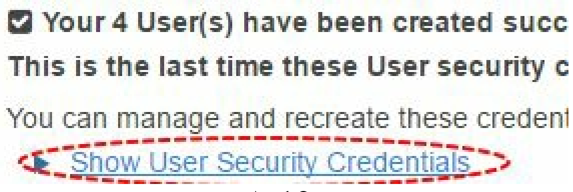
✓ Generate an access key for each us

Users need access keys to make secure F

For users who need access to the AWS M

A-11

You can choose to download the credentials or click on "Show User Security Credentials," where you can copy/paste them.

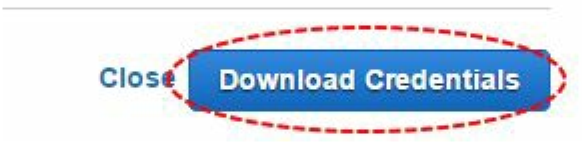

Your 4 User(s) have been created succ

This is the last time these User security c

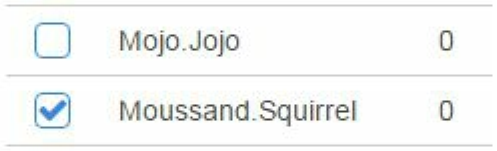
You can manage and recreate these credent

Show User Security Credentials

A-12

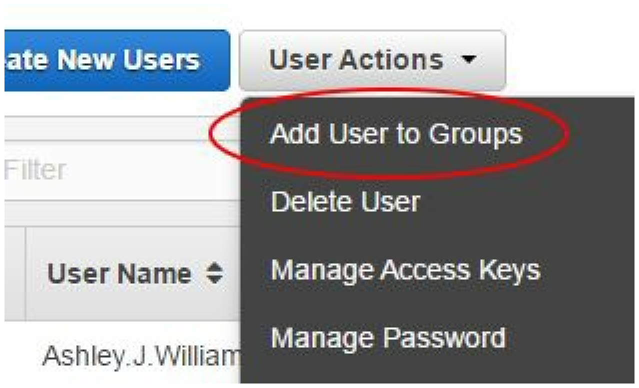


A-13

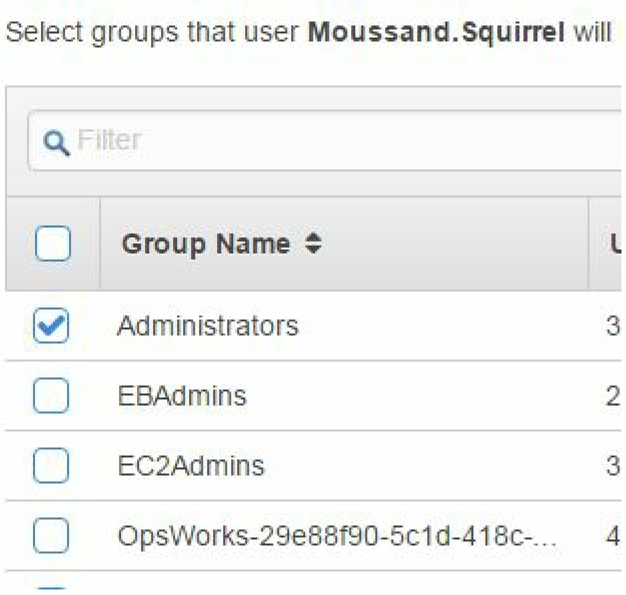


☐	Mojo.Jojo	0
☑	Moussand.Squirrel	0

A-14

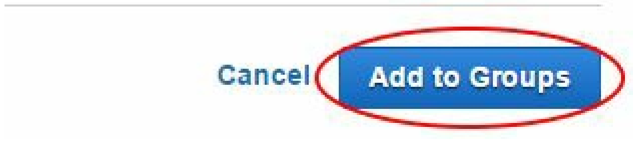


A-15

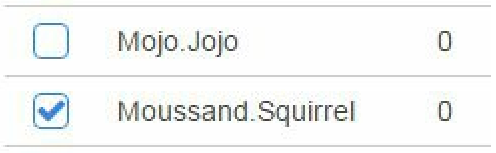

Select groups that user **Moussand.Squirrel** will

	Group Name	
✓	Administrators	3
	EBAdmins	2
	EC2Admins	3
	OpsWorks-29e88f90-5c1d-418c-...	4

A-16

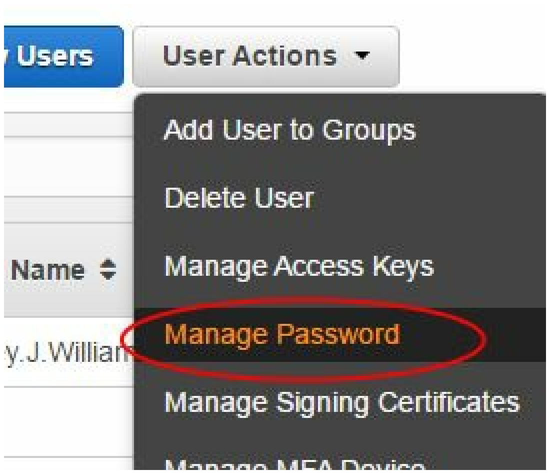


A-17



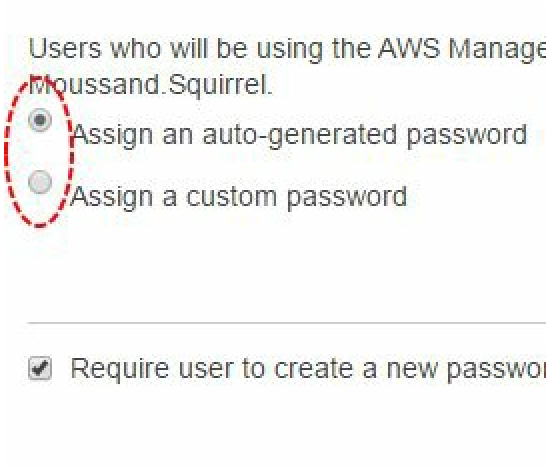
	Mojo.Jojo	0
✓	Moussand.Squirrel	0

A-18



A-19

You can choose to have AWS generate a password or create your own. It’s best to require the user to create a new password after they first login.

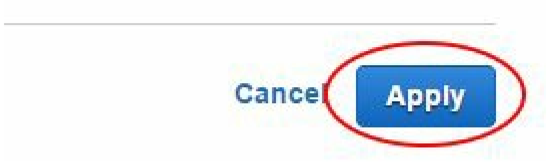

Users who will be using the AWS Manage
Moussand.Squirrel.

[✓ Assign an auto-generated password]

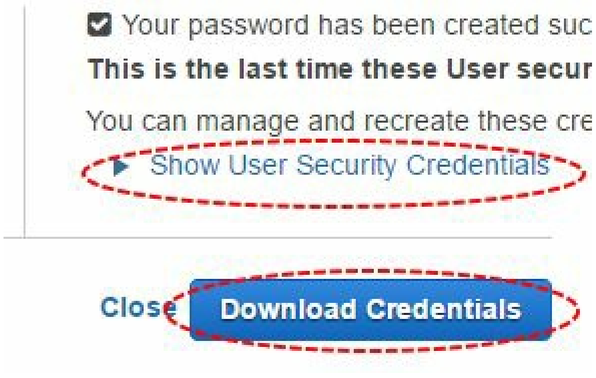
Assign a custom password

✓ Require user to create a new passwor

A-20



A-21



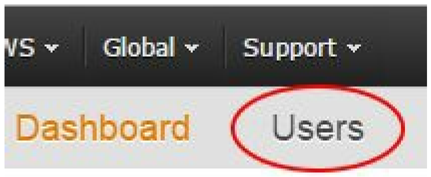
Your password has been created suc

This is the last time these User secur

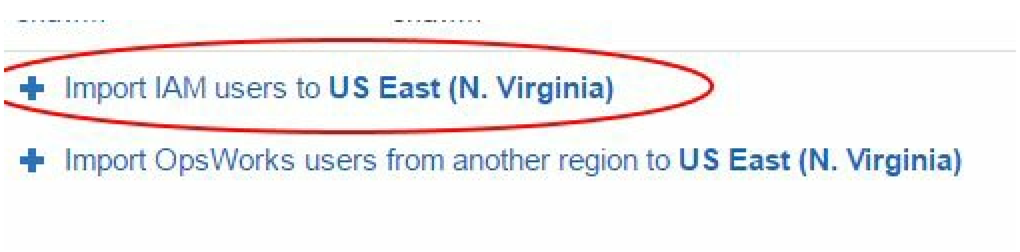
You can manage and recreate these cre

A-22

Appendix B: Adding Users to a Stack



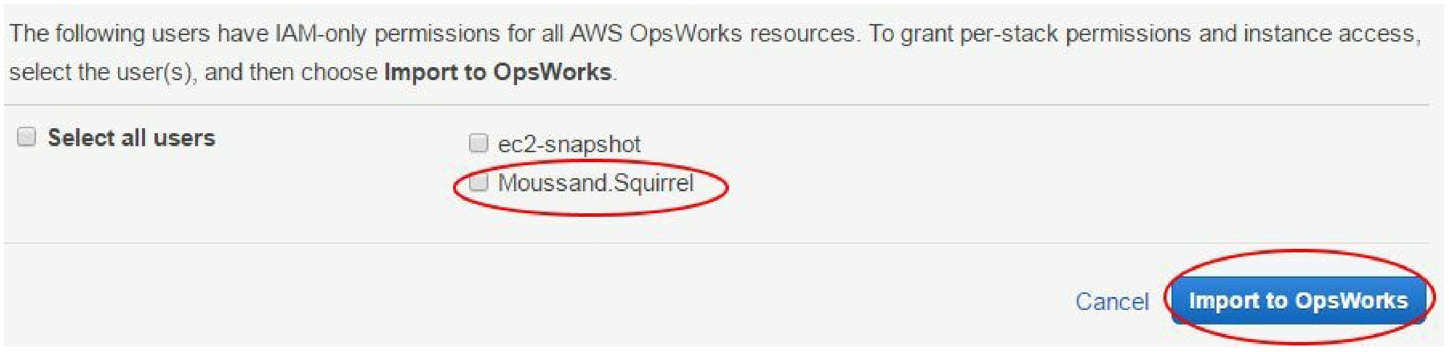
B-1



+ Import IAM users to **US East (N. Virginia)**

+ Import OpsWorks users from another region to **US East (N. Virginia)**

B-2



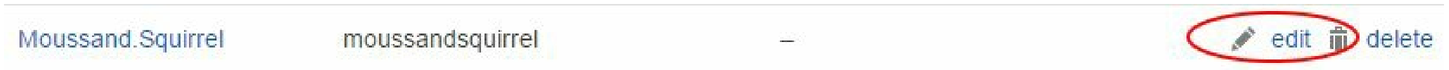
The following users have IAM-only permissions for all AWS OpsWorks resources. To grant per-stack permissions and instance access, select the user(s), and then choose **Import to OpsWorks**.

☐ **Select all users**

☐ ec2-snapshot
☐ Moussand.Squirrel

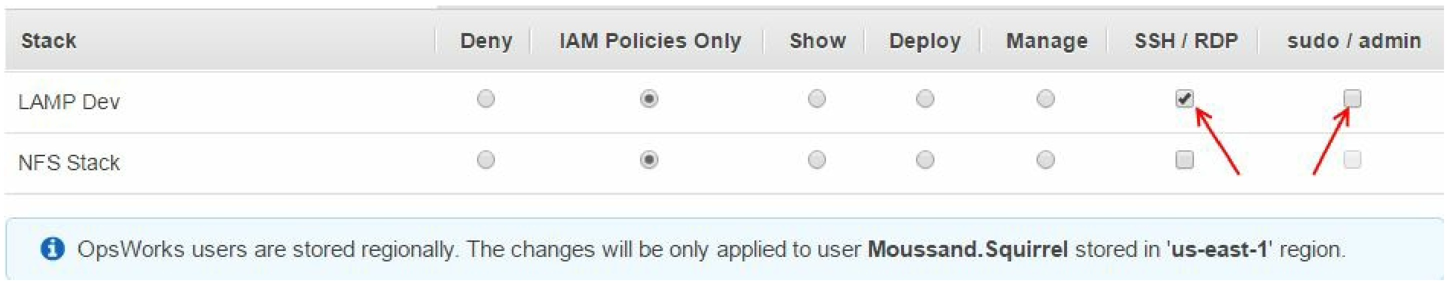
Cancel **Import to OpsWorks**

B-3



Moussand.Squirrel	moussandsquirrel	–	edit delete

B-4



Stack	Deny	IAM Policies Only	Show	Deploy	Manage	SSH / RDP	sudo / admin
LAMP Dev	○	●	○	○	○	✓	☐
NFS Stack	○	●	○	○	○	☐	☐

OpsWorks users are stored regionally. The changes will be only applied to user **Moussand.Squirrel** stored in '**us-east-1**' region.

B-5

Glossary

AWS: Amazon Web Services.
CLI: command-line interface.
CNAME: an alias for the web address to which one can browse. Used in Elasticbeanstalk for ease of navigation and testing.
EC2 container: a virtual machine running an Operating System in the AWS cloud.
git: a versioning control and publishing program that uses a repository to store changes.
Instance: Amazon Web Services's virtual machine that runs in the cloud.
IAM: Identity Access Management.
LAMP: Description of a stack with Linux, Apache, MySQL, and PHP.
stack: A group of programs or systems that collectively combine for a specific goal, such as an application.
UI: user interface.

Notes

1 EC2 containers are similar to ESX's Virtual Machines.↩

2 Users need access keys to make secure REST or Query protocol requests to AWS service APIs.↩

3 See https://aws.amazon.com/govcloud-us/pricing/s3/↩

4 http://docs.aws.amazon.com/opsworks/latest/userguide/best-practices-server-migrate.html#best-practices-server-migrate-environments ↩

5 See http://docs.aws.amazon.com/opsworks/latest/userguide/workingstacks.h

6 We can also take advantage of AWS OpsWorks' environment variables feature and define an environment variable that indicates what our instance is for. Then, in our git_pull.sh, we can detect the environment variable and run different git commands based on the definition. See http://docs.aws.amazon.com/opsworks/latest/userguide/apps-environment-var.html.
This only works with Chef >= 11.10. We're using Chef 11.4. The example below shows how to use the environment variable: {lang="text"} lamp $ vi recipes/mount_nfs.rb if node[:deploy]['lamp'][:environment_variables][:USE_ENVIRONMENT] == 'DEV' mount '/srv' do device '172.31.21.116:/export1' fstype 'nfs' options 'rw' action [:mount,:enable] end end lamp $↩

7 This directory can be changed using the Chef definitions of opsworks_deploy_dir and opsworks_deploy. For more information, visit http://docs.aws.amazon.com/opsworks/latest/userguide/create-custom-deploy.html.↩

Содержание