



Taking AWS Operations to the Next Level

Table of Contents

Introduction	3
Pre-lab	3
Get our credentials to the console and Key Pair setup.	3
Pre-lab step #1 Log into the Console via QwikLab	4
Pre-lab step #2 Set up PuTTY with a Key Pair	4
Lab #1	8
1.1 Launching our base VPC via CloudFormation	8
1.2 Adding resources to our VPC via new stacks	14
1.3 Adding a NAT Instance to our infrastructure	20
1.4 Modifying our VPC stack	21
Lab 1 Closing	23
Lab #2	24
2.1 Launch a Multi-AZ RDS MySQL Instance in our VPC.....	24
2.2 Use OpsWorks to Deploy our Application	26
2.3 Use CloudFormation and OpsWorks to Update Our Infrastructure and Application.....	35
Lab 2 Closing	40
Lab #3	42
3.1 Setup Amazon Simple Workflow Service	42
3.2 Setup SNS+ SQS + infrastructure helper instance.	43
3.3 Re-launch NAT instance as part of an Auto Scaling group	47
Lab 3 Closing	48

Introduction

A lot of IT work is doing the stuff that a lot of us wouldn't consider very challenging or exciting. From setting up basic networks, to building out instances, to installing basic software across a fleet of instances, there's a lot of low hanging activities that we all do day to day. Then there are the pages. Those things that wake up the person on-call in the middle of the night and add stress to our lives. Often these things are trivial once the issue is recognized, but how often do we make the right changes in our infrastructure to remove these basic problems? Throughout today's labs we'll see how treating your infrastructure like code and using simple automation and workflow services can benefit you in many ways.

First we'll explore provisioning and setting up our infrastructure resources via AWS CloudFormation. You can think of AWS CloudFormation as a living blueprint of your infrastructure, everything from Amazon Virtual Private Cloud (VPC) subnets, to security groups, to individual instance configuration. If you were told you needed to setup a datacenter in a day how would you even start? Well with AWS CloudFormation it is very easy to set up an advanced logical network to later deploy servers and services into. We will lay out an initial Amazon VPC using AWS CloudFormation and then build on top of it as we go the rest of the day. Think of it like building the walls, getting the equipment, racking and stacking, installing an operating system (OS), and configuring with software all wrapped into one powerful text based tool.

After the basic layout and some instances for our infrastructure are deployed, we'll go back and add in some configuration management, via Chef and AWS OpsWorks, to help with the deployment of software and instance configuration. Using Chef we'll see how we can modify instances' OS and software after they are up and running, as well as make it so that instances can be created and bootstrapped with everything they need to be ready for real use with little to no manual intervention. Chef is an incredibly powerful tool, and extremely flexible and we'll only be scraping the surface with it today. AWS OpsWorks is a DevOps solution for managing applications of any scale or complexity on the AWS cloud. Through OpsWorks you can easily manage deployments, scaling, and application lifecycle in your infrastructure.

Lastly, tying in Auto Scaling, notifications to Amazon SQS, Amazon SWF and various AWS APIs we'll show some example tricks that can be used to build out a self-healing and automated infrastructure. Auto Scaling doesn't always have to involve a growing and shrinking of a pool of instances, and we'll use it today to show how a single instance can be made to recover quickly in case of a failure. Beyond that there are often times when we need to tie in many different parts of our infrastructure around changes post-failure, and we'll work through using Amazon SWF and very basic code around AWS APIs to automate these things.

One big key take away from the labs today is that none of this needs to be done in a green-field brand-new, project or infrastructure. While applying all of these tools and methodologies will definitely help you on many fronts, a lot of this can be added in on top of your current infrastructure on AWS today, piece by piece. The other thing to remember, is that these are just examples of how these technologies can work together. The possibilities for how these tools can help you in your day-to-day operations are nearly endless and so try to think about how you could expand upon this and take even what you are learning today to the next level.

Pre-lab

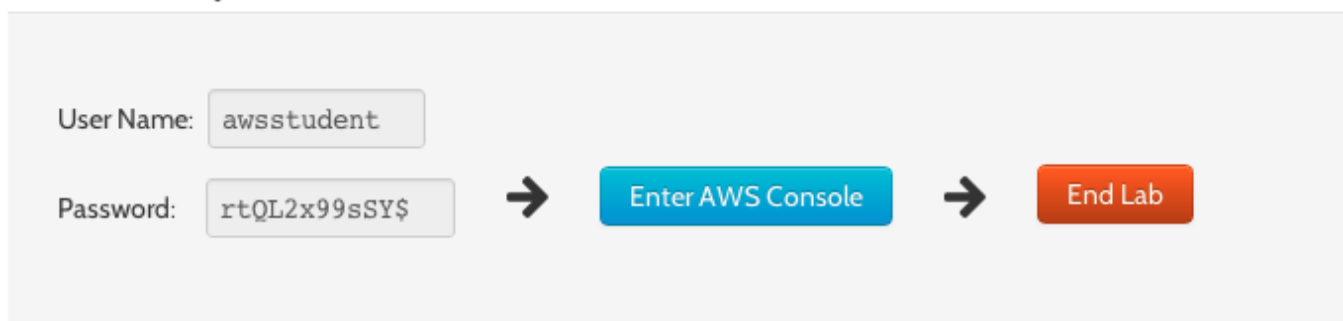
Get our credentials to the console and Key Pair setup.

We will be doing most of our work today in the Amazon Web Services Console. Typically found here: <https://console.aws.amazon.com>, the AWS Console is the primary tool used to interface with the different AWS services. Today we're going to be using a 3rd party tool called QwikLab to help provision access to the console, and therefore the AWS services within. If you haven't done so yet, we're going to go through QwikLab, login, and start today's labs. If you've already gone through this, feel free to get right on with Lab #1.

Pre-lab step #1 Log into the Console via QwikLab

1. Open your browser and go to <http://aws.qwiklab.com/>
2. Create a new Account if you don't yet have one, or if you do, sign in as a member.
3. Once logged in, find today's class and click on the name "Workshop 9 – Taking AWS Operations to the Next Level"
4. Click on Start Lab.
5. You should now be presented with the 'Workshop 9 – Set up Student Credentials' page. On it you will see some information about the duration of this lab, how long the setup time is, the creator's name, and creation date. Below that is what we are really looking for here. The credentials we'll use to access the console. Copy the User Name, and Password to Notepad, or write it down, as you might need it from time to time today.

Lab Setup: Review the Instructions and follow the lab setup sequence below.



The diagram illustrates the lab setup sequence. It starts with two input fields: 'User Name:' containing 'awsstudent' and 'Password:' containing 'rtQL2x99sSY\$'. An arrow points from the password field to a blue button labeled 'Enter AWS Console'. Another arrow points from this button to an orange button labeled 'End Lab'.

6. Click on Enter AWS Console.
7. The Console will open up in a new tab. Go over to it, and fill in the User Name and Password with the information from above.

At this point you are logged into the console and ready to move on to the next step!

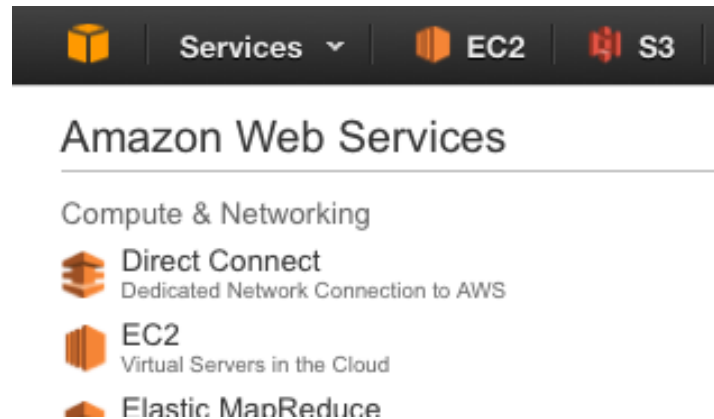
Pre-lab step #2 Set up PuTTY with a Key Pair

QwikLab creates a Key Pair for you when you first start the lab, but it is good to get familiar with how to set up Key Pairs, as it is a very important part of accessing instances in AWS (up until you get another authentication method implemented such as Active Directory, LDAP, or local user accounts on instances). We're going to ignore the premade one (you can even delete it if you would like, it is called "generic-qwiklab"). Let's go on and create our own Key Pair, which will be used through out the rest of the day.

It is very important to note, that you need to keep track of where you download this file! Do not delete it! Make sure you know where you downloaded it!

It is also very important that you make sure you are in the correct region when you create your Key Pair.

1. Now, from the main dashboard in the Console, find the EC2 link. Sometimes you'll find it in two places when you first sign in. First at the top bar, and also in the middle of the screen. You can see an example here:

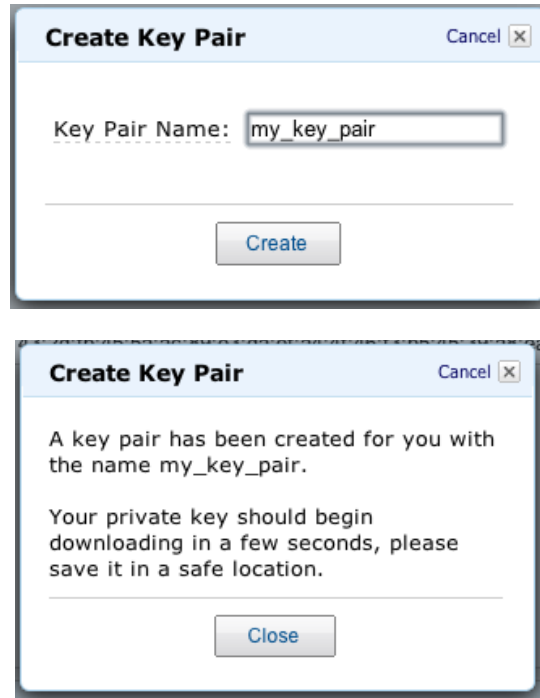


You can click on either one. They will both take you to the same place.

2. Let's confirm we are in the right region. Everyone should be launching in the US-West-2 region today. If you are not, go to the top right corner of the site, and click the region drop down and select **US West (Oregon)**:

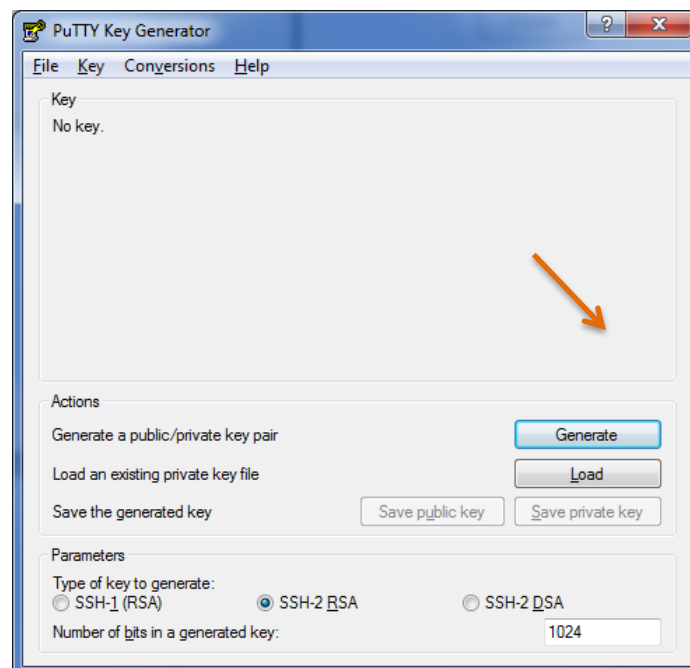


3. Once you are finally at the Amazon EC2 Console Dashboard, look for the Key Pair link. This link also exists at least twice on this page, once on the far left column near the bottom. The other in the right panel labeled "My Resources". Click on the link to go to the Key Pairs page.
4. Click on "Create Key Pair". Give your Key Pair a simple name that you can easily remember. You'll need this key several times today.
5. Download the file to your desktop, or some other location you will remember. You will need this file again later in the lab.

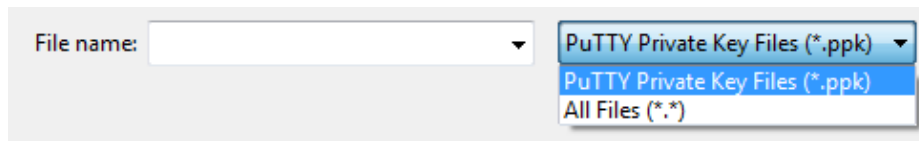


Windows Users will need to follow the next steps to make the Key Pair we just created usable with PuTTY. Ignore this if PuTTY is not your SSH tool, as you will just be able to use the Pem file as it is. You are all done with any pre-Lab setup if you don't need to follow these steps.

1. Now we need to convert the .PEM file we just downloaded into a format that can be used by our SSH tool PuTTY.
 - 1) Start **PuTTYgen** (for example, from the Start menu, click All Programs > PuTTY > PuTTYgen).
 - 2) Click Load and browse to the location of the private key file that you want to convert (e.g., my_key_pair.pem).



- 3) By default, PuTTYgen displays only files with extension .ppk; you'll need to change that to display files of all types in order to see your .pem key file. The private key file must end with a newline character or PuTTYgen cannot load it correctly.

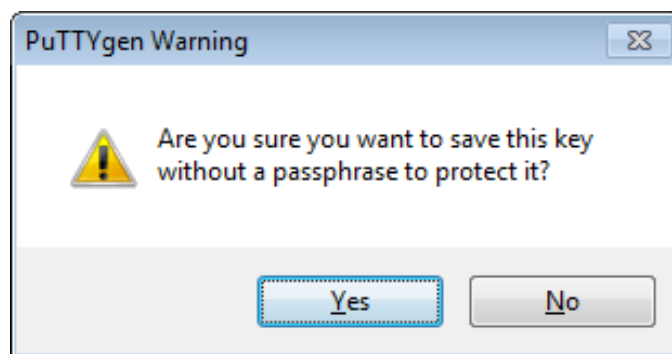


- 4) Select your .pem key file and click Open. PuTTYgen displays the following message.



When you click OK, PuTTYgen displays a dialog box with information about the key you loaded, such as the public key and the fingerprint. The keys that Amazon EC2 generates are 1024-bit SSH-2 RSA keys.

- 5) Click **Save private key** to save the key in PuTTY's format. PuTTYgen asks if you want to save the key without a passphrase. Click **"Yes"**. Normally you would want to make sure that the keys used to access your instances were as secure as possible, but for today's lab we'll skip the extra complication.



- 6) Use the same name for the key that you used for the Key Pair (for example, **"my_key_pair"**). PuTTY automatically adds the .ppk file extension. Close PuTTYgen.

Your private key is now in the correct format for use with PuTTY. You can now connect to your instance using PuTTY's SSH client. Later today when you start firing up instances, you'll use PuTTY to connect to the ones that are Internet accessible. We will look at how to do this in PuTTY a little later. That's it! We should now be all set up and ready to go on with the rest of the lab.

Lab #1

1.1 Launching our base VPC via CloudFormation

We're going to be creating a very basic VPC for our infrastructure to live in for the rest of the day. This VPC will look very similar to the one you would create if you had followed the Wizard in the console for the Public + Private subnet options. One of the biggest differences is that this infrastructure will be Multi-AZ from the start. In this first part of the lab we'll just be creating the public subnets, and later we'll take care of the private subnets.

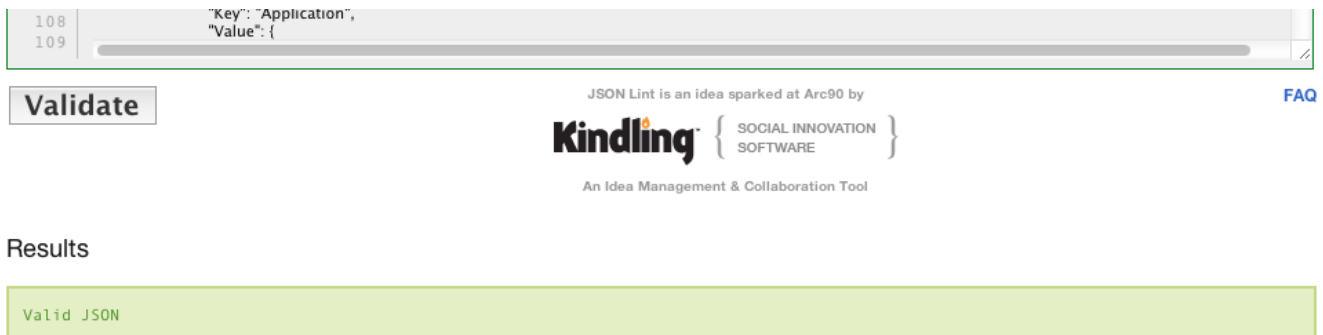
These are the components that this initial stack will create:

- A VPC with a CIDR Block of 192.168.0.0/16. This would give us ~65,534 usable IPs if it were one large subnet.
- Three subnets that will be our "Public" subnets with CIDR blocks: 192.168.10.0/24, 192.168.11.0/24, and 192.168.12.0/24.
- An Internet Gateway attached to our VPC. This is needed to allow public Internet access to any instances in our VPC. This is a routing construct for VPC and not an instance.
- Routes and Route tables for our three subnets so instances in them can communicate.
- Default Network ACL's to allow all communication inside of our VPC.

Let's walk through launching this first basic stack.

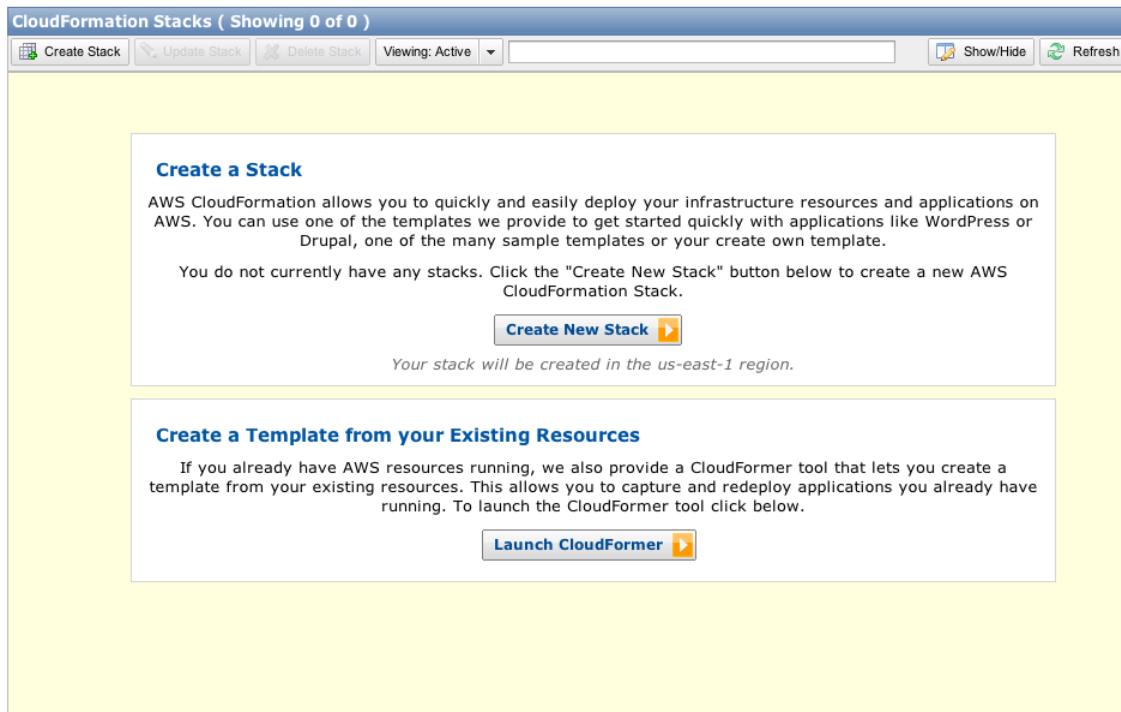
1. Go and download the first template from https://s3-us-west-2.amazonaws.com/munns-bootcamps/reinvent2013/Lab1-main_VPC_stack.template. Open this file in a text editor and look through it. You'll see that this template file only contains 4 of the 5 sections that an AWS CloudFormation template can have: Header, Mappings, Resources, and Outputs. We'll see the results of these when we launch this stack.
2. As mentioned earlier, it can be really useful to use a linter or similar tool to check the syntax of our template files. There just happens to be a great free one based on an open source project hosted on GitHub called JSONLint. Lets open it up in a web browser and check out our initial template for any errors:
 - 1) Open up <http://jsonlint.com> in a browser.
 - 2) Select all of the text from the template in the text editor. Then copy from the text editor and paste the text into the main field in the jsonlint.com page.
 - 3) Click on the "Validate" button.

If all goes well, you should see a green bar that says "Valid JSON" at the bottom of the page. All together it should look at little like this:

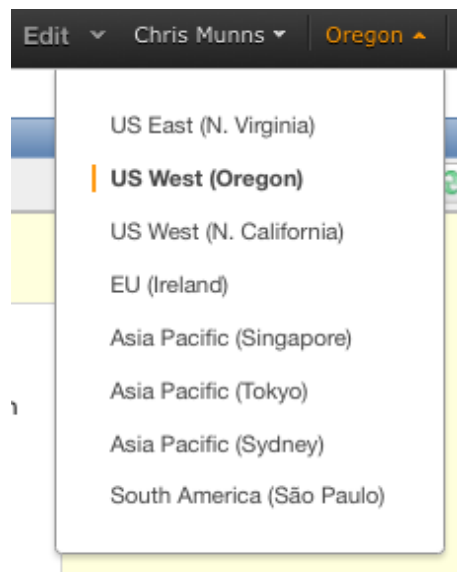


Remember this tool. You'll need to use it later as you modify some templates by hand.

- Return to the AWS Console, click on the orange AWS cube in the top right and then find the CloudFormation link under **“Deployment & Management”**, in the right most column in the center. This will take us to the CloudFormation main page in the console.



- Now we're going to create a new stack, but first let's confirm we are in the right region. Everyone should be launching in the US-West-2 region today. If you are not, go to the top right corner of the site, and click the region drop down and select **US West (Oregon)**:



If this is good, now we can go ahead click on **“Create New Stack”**.

- You should now see the **“Create Stack”** wizard in front of you. Let's go ahead and give this stack a meaningful name. I am calling mine **“masterVPC”** as this is my master VPC that my infrastructure will live in. Once you've done that, we're next going to select **“Upload a Template File”**. Do this and

choose the file that we just downloaded (“Lab1-main_VPC_stack.template”). The wizard should now look like this for you:

Create Stack [Cancel]

SELECT TEMPLATE | SPECIFY PARAMETERS | ADD TAGS | REVIEW

AWS CloudFormation gives you an easier way to create a collection of related AWS resources (a stack) by describing your requirements in a template. To create a stack, fill in the name for your stack and select a template. You may choose one of the sample templates to get started quickly, or one of your own templates stored in S3 or on your local hard drive.

Stack Name:
masterVPC

Template:

☐ Use a sample template

☒ Upload a Template File
Browse... Lab1-main_VPC_stack.template

☐ Provide a Template URL

☐ Show Advanced Options

Continue

We’re going to leave everything under “Show Advanced Options” alone for now, but if you wanted to, you could have it send notifications via Amazon SNS about the status of the creation of this stack. If all looks well, click **Continue**.

- We now see we have the option now of adding “Tags” to our environment. Tags provide a way to simplify management of the resources in your infrastructure by letting you specify key/value pairs containing information you consider important about this stack. I am adding in three tags, one for the name, one for the environment, and one for the project.

Create Stack [Cancel]

SELECT TEMPLATE | SPECIFY PARAMETERS | ADD TAGS | REVIEW

Add tags to your stack to simplify the administration of your infrastructure. A tag consists of a key/value pair and will flow to resources inside your stack. You can add up to 10 unique keys to each stack along with an optional value for each key. For more information, go to [Tagging a Stack](#) in the CloudFormation User Guide.

Key (127 characters maximum)	Value (255 characters maximum)	Remove
Name	masterVPC	✖
Environment	Production	✖
Project	Bootcamp	✖

[Add another Tag. \(Maximum of 10\)](#)

< Back Continue

Once you have added any tags you wish to, click **Continue**.

- On this next screen we'll see a summary of what we are creating including the "Stack Description" that is taken from the template. The "Template" field contains the S3 bucket location that this AWS CloudFormation template file will be uploaded to as part of creating this. Clicking on the word "Cost" next to "Estimated Cost" would take us to the AWS Monthly Web Calculator and could give us a break down of any costs caused by the creation of this stack's resources. This first stack actually would cost us nothing to turn up right now, so the calculator won't show any monthly cost.

This should be all good to go, so let's click on **"Continue"** once more to start creating this stack.

- This last window in the wizard just lets us know that we can track the progress of our stack creation by selecting it in the AWS CloudFormation console, and following the Events tab. Once everything is done we should see **"CREATE_COMPLETE"** at the end of the events, as well as on the Description tab. Let's go take a look at the console and see what is going on:

CloudFormation Stacks (Showing 1 of 1)				
Create Stack Update Stack Delete Stack Viewing: Active				
	Name	Created	Status	Description
<input checked="" type="checkbox"/>	masterVPC	2013-10-09 13:23:39 UTC-4	CREATE_IN_PROGRESS	Builds a

Stack: masterVPC	
Description	Outputs Resources Events Template Parameters Tags
Stack Name:	masterVPC
Stack ID:	arn:aws:cloudformation:us-west-2:102901597367:stack/masterVPC/8971de40-3107-11e3-b37d-500160d4da18
Status:	CREATE_IN_PROGRESS
Status (Reason):	User Initiated
Created:	2013-10-09 13:23:39 UTC-4
Description:	Builds a VPC w/ INET Gateway and 3 public subnets. **WARNING** This template creates Amazon EC2 instance(s). You for the AWS resources used if you create a stack from this template.

Taking AWS Operations to the Next Level

As we can see here, the stack's creation status is still "CREATE_IN_PROGRESS". If we click over to the Events tab, we can see just what is happening. Starting from the bottom and going up, we see the initialization of the stack:

Name	Created	Status	Description
<input checked="" type="checkbox"/> masterVPC	2013-10-09 13:23:39 UTC-4	CREATE_IN_PROGRESS	Builds a VPC w/ INET Gateway and 3 public subnets. **WA...

Stack: masterVPC

Description Outputs Resources **Events** Template Parameters Tags

Stack Events Refresh

Time	Type	Logical ID	Physical ID	Status	Reason
2013-10-09 13:24:06 UTC-4	AWS::EC2::Subnet	PublicSubnetA		CREATE_IN_PROGRESS	
2013-10-09 13:24:06 UTC-4	AWS::EC2::RouteTable	PublicRouteTable		CREATE_IN_PROGRESS	
2013-10-09 13:24:04 UTC-4	AWS::EC2::InternetGateway	InternetGateway	igw-03f1b468	CREATE_COMPLETE	
2013-10-09 13:24:03 UTC-4	AWS::EC2::VPC	VPC	vpc-08f1b463	CREATE_COMPLETE	
2013-10-09 13:23:44 UTC-4	AWS::EC2::InternetGateway	InternetGateway	igw-03f1b468	CREATE_IN_PROGRESS	Resource creation Initiated
2013-10-09 13:23:44 UTC-4	AWS::EC2::VPC	VPC	vpc-08f1b463	CREATE_IN_PROGRESS	Resource creation Initiated
2013-10-09 13:23:43 UTC-4	AWS::EC2::InternetGateway	InternetGateway		CREATE_IN_PROGRESS	
2013-10-09 13:23:43 UTC-4	AWS::EC2::VPC	VPC		CREATE_IN_PROGRESS	
2013-10-09 13:23:39 UTC-4	AWS::CloudFormation::Stack	masterVPC	arn:aws:cloudformation:us-west-2:102901597367:stack/masterVPC/8971de40-3107-11e3-b37d-500160d4da18	CREATE_IN_PROGRESS	User Initiated

What we see here is each bit of the VPC being built out piece by piece. Each individual component is first seen as "CREATE_IN_PROGRESS" and then if it succeeds as "CREATE_COMPLETE". We can see (going from the bottom up), the VPC being created, the Internet Gateway (IGW), the public route table, and then the start of the subnets. AWS CloudFormation takes care of the proper order and making the proper requests to the different service APIs under the hood to do what the template tells it to. It will also try and build out as many components as possible in parallel. If the stack is successful in completing all the events required, the very last step at the top should look like this

Stack: masterVPC


Description Outputs Resources **Events** Template Parameters Tags

Stack Events

Time	Type	Logical ID	Physical ID	Status	Reason
2013-10-09 13:24:49 UTC-4	AWS::CloudFormation::Stack	masterVPC	arn:aws:cloudformation:us-west-2:102901597367:stack/masterVPC/8971de40-3107-11e3-b37d-500160d4da18	CREATE_COMPLETE	
2013-10-09 13:24:48 UTC-4	AWS::EC2::SubnetRouteTable	PublicSubnetRouteTable	rtbassoc-8bf1b4e0	CREATE_COMPLETE	
2013-10-09 13:24:46 UTC-4	AWS::EC2::Route	PublicRoute	maste-Publi-FX832CNKO678	CREATE_COMPLETE	

Now going back to the Description tab, we should see a status of "**CREATE_COMPLETE**". If for any reason you see something else, such as an error or a failure to create the stack, please ask a lab assistant for help.









- Let's check out some other things about this stack. Click on the Outputs tab to see what we find:

	Name	Created	Status
<input checked="" type="checkbox"/>	masterVPC	2013-10-09 14:42:07 UTC-4	 CREATE_COMPLETE

Stack: masterVPC		
Description	Outputs	Resources Events Template Parameters Tags
Stack Outputs <i>Output values may have been specified by the template author and will be available when stack creation is complete.</i>		
Key	Value	Description
VpcId	vpc-30286c5b	VPC ID of newly created VPC
PublicSubnetA	subnet-da286cb1	Public Subnet in AZ A
PublicSubnetB	subnet-d2286cb9	Public Subnet in AZ B
PublicSubnetC	subnet-df286cb4	Public Subnet in AZ C

You should see the same 4 Outputs as shown here: **VpcId**, **PublicSubnetA**, **PublicSubnetB**, and **PublicSubnetC**. These are important things to know, as we'll need them later as we launch more resources into this infrastructure. These values are the actual values from the API specific to our environment.

- Now let's click over to the Resources tab. From here we can see all the different resources that were created for this environment as well as any calls that would modify it (such as the AttachGateway request). We can see the LogicalID from the CloudFormation template, the PhysicalID from the API, and the type, as well as its current status:

Stack: masterVPC			
Description	Outputs	Resources	Events Template Parameters Tags
Stack Resources			
Logical ID	Physical ID	Type	Status
InternetGateway	igw-31286c5a	AWS::EC2::InternetGateway	 CREATE_COMPLETE
VPC	vpc-30286c5b	AWS::EC2::VPC	 CREATE_COMPLETE
PublicSubnetA	subnet-da286cb1	AWS::EC2::Subnet	 CREATE_COMPLETE
PublicSubnetC	subnet-df286cb4	AWS::EC2::Subnet	 CREATE_COMPLETE
AttachGateway	maste-Attac-K2UL52A7KKDH	AWS::EC2::VPCGatewayAttachment	 CREATE_COMPLETE
PublicRouteTable	rtb-dd286cb6	AWS::EC2::RouteTable	 CREATE_COMPLETE
PublicSubnetB	subnet-d2286cb9	AWS::EC2::Subnet	 CREATE_COMPLETE
PublicSubnetRouteTableAssociationA	rtbassoc-8e286ce5	AWS::EC2::SubnetRouteTableAssociation	 CREATE_COMPLETE

- The last 3 tabs we have are all pretty predictable: Template, Parameters and Tags. Since we have no parameters in this initial stack, that tab will be blank. For Template we should see the exact same template as we uploaded, and for Tags we should see the tags we've created.

We are now ready to put more resources into this infrastructure that we just stood up via AWS CloudFormation. If you want to real quick, take a second to explore the Amazon VPC Console Dashboard by clicking on the Services drop down at the top of the page and selecting VPC. After you should see the main dashboard page for VPC showing us the resources we've created so far:



You'll notice here that it mentions a 2nd Route Table, a Network ACL, and a Security group that we did not specify in our template file. These were created by default as part of the creation of the VPC. We can ignore these. Ok, now it is time to add some more to this VPC!


1.2 Adding resources to our VPC via new stacks

Now that we have our base VPC built, and a way for our instances that we might launch into it to be able to get out to the Internet, let's launch a few instances. Instead of modifying our current stack, let's launch new resources using new CloudFormation templates into our VPC. Using the outputs detailing the resources from the main VPC stack, we can add things into our VPC individually instead of having to modify the main template. This allows us to really compartmentalize and separate out any components of our infrastructure. You can picture how in a large infrastructure, with many different services, you could have many different stacks inside of larger ones, so that individual teams can work with each other on each of those components without there being one huge unruly file.

We need an instance inside of this infrastructure to use as a bastion, or leap box, to the rest of the infrastructure. In your own environment, you might setup a VPN instead so that you can just access instances as if they were part of your existing network, but for today's class, we're going to go the bastion/leap instance route. Once this instance is up we're going to create another Key Pair on this host to use to access the rest of the instances we are going to create in this environment. Keeping this key safe will be paramount to being able to access any other instance we create, you should save the key file's content to a local file via a text editor. As with the previous step in this lab, we're going to use AWS CloudFormation to set up this host.

Let's start:

1. First thing we need to do is make sure we know what subnet in our previously created VPC we are going to put this host. If you go back to the stack that our VPC is part of in the CloudFormation console, click on the stack, and then in the window below click on the **Outputs tab**. **Copy this information to Notepad** or somewhere similar to save for later (NOTE: your IDs will be different than what you see below). We can use any one of the "PublicSubnets" we see listed here for this instance. If you don't see any outputs, check the stack status to make sure it has completed.

 **Stack:** masterVPC

Description	Outputs	Resources	Events	Template	Parameters	Tags
-------------	----------------	-----------	--------	----------	------------	------

Stack Outputs

Output values may have been specified by the template author and will be available when stack creation

Key	Value	Description
VpcId	vpc-30286c5b	VPC ID of newly created VPC
PublicSubnetA	subnet-da286cb1	Public Subnet in AZ A
PublicSubnetB	subnet-d2286cb9	Public Subnet in AZ B
PublicSubnetC	subnet-df286cb4	Public Subnet in AZ C

2. Next let's go and download the CloudFormation template we'll need for this step. You can find it here https://s3-us-west-2.amazonaws.com/munns-bootcamps/reinvent2013/Lab1-leap_stack.template. Open this template up in your text editor and take a look around. Quite a lot going on here compared to the previous change we made to our stack:
 - 1) We create an IAM user that can find out information about our stack, and has permissions to create Key Pairs for us.
 - 2) We use CloudFormation Init to install packages, create files, and run commands on this instance. We also take the credentials created for the IAM user and have them set up to be used by our scripts.
 - 3) We use the UserData to run the cfn-init command that actually does the above via a bash script.
 - 4) The completion of the instance is dependent on the scripts running properly, if they fail, the stack formation will error out and fail.
3. Go to the CloudFormation console and create a new stack using this template file. Name it something representative of what it is (I named mine "**vpc-leaphost**"), and select the template downloaded from the previous step. On the next step now in the wizard we are faced with a few parameters we need in order for this host to launch. In no particular order:
 - 1) "VpcId" – This is the ID of the VPC we are launching this instance into. From step 1 of this section of the lab you should have this copied somewhere.
 - 2) "SubnetId" – This is the ID of the "Public" subnet inside our VPC. Again, you should have this copied somewhere.
 - 3) "KeyName" – This is our existing Key Pair name that we will use to remotely access this host (i.e. "**my_key_pair**").
 - 4) "BastionKeyName" – This is the name of the Key Pair we will create by launching this instance, and use to access the other instances inside our infrastructure. You can name this something like "**bastkey1**".
 - 5) Lastly, you'll need to check the box next to "I acknowledge that this template may create IAM resources"

When you have this all filed out, it should look like this:

Taking AWS Operations to the Next Level

Create Stack [Cancel]

SELECT TEMPLATE | **SPECIFY PARAMETERS** | ADD TAGS | REVIEW

Stack Description: Add a bastion host to an existing VPC. VPC must have an internet gateway already. ****WARNING**** This template creates an Amazon EC2 instance. You will be billed for the AWS resources used if you create a stack from this template.

Specify Parameters
Below are the parameters associated with your CloudFormation template. You may review and proceed with the default parameters or make customizations as needed below.

SubnetId
SubnetId of an existing Public facing subnet in your Virtual Private Cloud (VPC)

BastionKeyName
Name of the EC2 KeyPair we will create internally to access instances in our VPC

VpcId
VpcId of your existing Virtual Private Cloud (VPC)

KeyName
Name of an existing EC2 KeyPair to enable SSH access to the instances

☒ **I acknowledge that this template may create IAM resources**

< Back Continue >

- Continue on, if you would like to add a tag to this instance, feel free to. I am going to make mine similar to the first stack:

Create Stack [Cancel]

SELECT TEMPLATE | SPECIFY PARAMETERS | **ADD TAGS** | REVIEW

Add tags to your stack to simplify the administration of your infrastructure. A tag consists of a key/value pair and will flow to resources inside your stack. You can add up to 10 unique keys to each stack along with an optional value for each key. For more information, go to [Tagging a Stack](#) in the CloudFormation User Guide.

Key (127 characters maximum)	Value (255 characters maximum)	Remove
<input type="text" value="Name"/>	<input type="text" value="vpc-leap"/>	<input checked="" type="checkbox"/>
<input type="text" value="Environment"/>	<input type="text" value="Production"/>	<input checked="" type="checkbox"/>
<input type="text" value="Project"/>	<input type="text" value="bootcamp"/>	<input checked="" type="checkbox"/>

[Add another Tag.](#) (Maximum of 10)

< Back Continue >

- Continue on again, and create the stack.
- Going back now to the CloudFormation dashboard, let's check on the status of this latest stack creation.

	Name	Created	Status	Description
<input checked="" type="checkbox"/>	vpc-leaphost	2013-10-09 14:50:18 UTC-4	CREATE_IN_PROGRESS	Add a bastion host to
<input type="checkbox"/>	masterVPC	2013-10-09 14:42:07 UTC-4	CREATE_COMPLETE	Builds a VPC w/ INET

- Click on this new leap instance stack we just made, and go to the **Events** tab. Here you can watch all the different Events needed to create this stack. If something goes wrong, you'll see an Event fail,

and a rollback be initiated by CloudFormation. If that happens, the far right column of an Event will explain what failed and why.

Stack: vpc-leaphost

Description Outputs Resources **Events** Template Parameters Tags

Stack Events

Time	Type	Logical ID	Physical ID	Status
2013-10-09 14:50:54 UTC-4	AWS::EC2::Instance	BastionHost	i-63c2dc57	CREATE_IN_PROGRESS
2013-10-09 14:50:53 UTC-4	AWS::EC2::Instance	BastionHost		CREATE_IN_PROGRESS
2013-10-09 14:50:50 UTC-4	AWS::EC2::SecurityGroup	BastionSecurityGroup	sg-34a0515b	CREATE_COMPLETE
2013-10-09 14:50:49 UTC-4	AWS::EC2::SecurityGroup	BastionSecurityGroup	sg-34a0515b	CREATE_IN_PROGRESS

- Once this stack has its status set to “CREATE_COMPLETE”, let’s go and log on to the instance we just created. If you click on the **Outputs** tab for this stack, you’ll see the field “IPAddress”. This is how we will access this instance below. You may want to copy this IPAddress to notepad.

	Name	Created	Status
<input checked="" type="checkbox"/>	vpc-leaphost	2013-10-09 14:50:18 UTC-4	CREATE_COMPLETE
<input type="checkbox"/>	masterVPC	2013-10-09 14:42:07 UTC-4	CREATE_COMPLETE

Stack: vpc-leaphost

Description **Outputs** Resources Events Template Parameters Tags

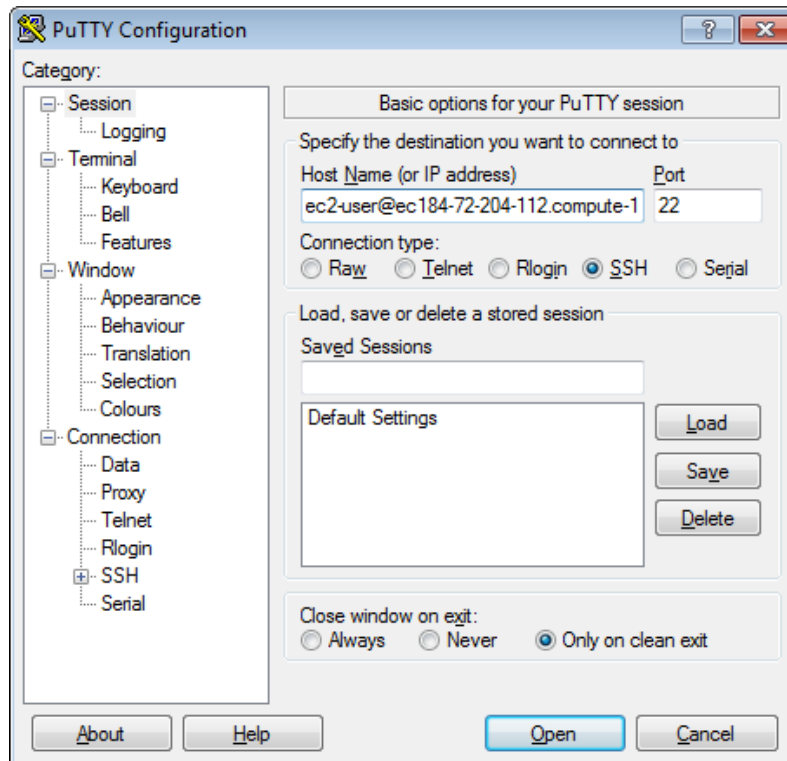
Stack Outputs

Output values may have been specified by the template author and will be available when stack creation is complete.

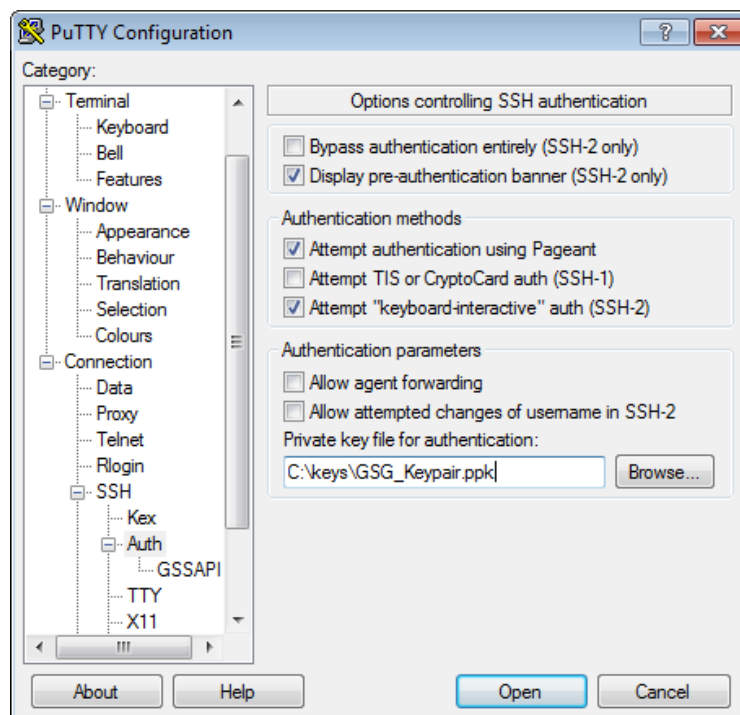
Key	Value	Description
InstanceID	i-63c2dc57	Bastion Instance ID
IPAddress	54.200.81.186	Public IP address of instance
BastionKeyName	bastkey1	The internal bastion KeyPair name

Now we want to connect to this instance and confirm that it is running right and that it created the bastion Key Pair that the CloudFormation script attempts to create. These are the instructions for PuTTY users. For non-PuTTY users your connections steps might differ slightly.

- Now go and open up **PuTTY** and connect to this instance using the IPAddress from the Outputs tab, the existing Key Pair we converted earlier and the user name “ec2-user”.
- Start **PuTTY** (from the Start menu, click All Programs > PuTTY > PuTTY). A dialog box opens with a Category menu on the left side. On the right side, the basic options for your PuTTY session are displayed. In the Host Name field, you will enter the public DNS name or public IP address of your instance. You can optionally prefix the DNS name or public IP address with ec2-user@ to automatically log in with the ec2-user, which we will be using today for all instances we launch.



In the Category menu, under Connection, click **SSH**, and then Auth. The options controlling SSH authentication are displayed. Click Browse and navigate to the PuTTY private key file you generated at the beginning of this workshop (i.e. "my_key_pair.ppk").



Click Open. An SSH session window opens and PuTTY displays a security alert asking if you trust the host you're connecting to. Click Yes. In the SSH session window, log in as ec2-user if you didn't as part of starting the SSH session. Note: If you specified a passphrase when you

converted your private key to PuTTY's format, you must provide that passphrase when you log in to the instance.

After you have connected, you should see something that looks like this (my SSH client is just OSX terminal, if you are using PuTTY client your output should be fairly similar):

```

munns — ec2-user@ip-192-168-11-238:~ — ssh — 80x24
[munns@3c075420d22f ~]$ ssh -i my_key_pair.pem ec2-user@54.200.81.186
The authenticity of host '54.200.81.186 (54.200.81.186)' can't be established.
RSA key fingerprint is c5:3a:1a:28:73:9b:9b:7a:c3:24:7b:61:c6:82:5e:f9.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '54.200.81.186' (RSA) to the list of known hosts.

  _| _| _| _|
  _| ( _| _| /   Amazon Linux AMI
  _|\_| _| _| _|

https://aws.amazon.com/amazon-linux-ami/2013.09-release-notes/
[ec2-user@ip-192-168-11-238 ~]$

```

The last thing we need to do in this part of the lab is confirm that the Bastion Key Pair we created by creating this stack exists. From this shell session, type in “ls -la .ssh” to confirm that the .pem file was created:

```

  _| _| _| _|
  _| ( _| _| /   Amazon Linux AMI
  _|\_| _| _| _|

https://aws.amazon.com/amazon-linux-ami/2013.09-release-notes/
[ec2-user@ip-192-168-11-238 ~]$ ls -la .ssh
total 16
drwx----- 2 ec2-user ec2-user 4096 Oct  9 18:52 .
drwx----- 3 ec2-user ec2-user 4096 Oct  9 18:52 ..
-rw----- 1 ec2-user ec2-user 393 Oct  9 18:52 authorized_keys
-rw----- 1 ec2-user ec2-user 1674 Oct  9 18:52 bastkey1.pem
[ec2-user@ip-192-168-11-238 ~]$

```

Since I see the “bastkey1.pem” file here. I know I am good to go! If not, try running the “create-keypair” command by hand from the ec2-user home directory (cd \$HOME; sh create-keypair) and seeing what error it throws. Ask a lab assistant for help! The last thing you should do is copy the contents of the “bastkey1.pem” file to another location. In the off chance something happens to your leap instance, it would be good to not be locked out of our other instances!

1. “cat” out the file from the terminal, and copy the results to a text editor on your laptop. Type in “cat .ssh/bastkey1.pem” to get the contents of the file:

```

[ec2-user@ip-192-168-11-238 ~]$ cat .ssh/bastkey1.pem
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAxvi4Vja1orbUiBhKM0Rj5UVs9z3lSeQvLvMtx5PR8jQ52Ibu3hkUN/CgGvx+
7sZVle0oLoXiB71d7XZgXNOKG6oqY+hx0zYnM0S1tdWC9qR00eLg3BN6I/UDTh2t29ZDAxws81XM
Ga56pYTwoIiVjxMDGBqUIeBPQFpDqUHa28VsLYwu5QtgK9J7pZnGn6APrkefuDsYz1N29J9SikXK
F5A0eelPorRHoamuW09P4qnzPVe37E7H13cx0D0GXv70QE1r0J0Fp03/LqhM40WBG90d0LX1JxP5
PvcuQlh1+w7bTyTxipCL/NC78lIJBU4YLwxD3r2HkLno+BEPuHBodQIDAQABAoIBAQCg7gY72PAF
/wxTmn0/FQLVa0ZDKENEf6yLnzeNXy4Bn08Zai+0h7dRVCmPA2mkzPufJ3pWyk4Ex6NZtGwWpxHC
PvxuMiQPbmLY/qBnRfqA/fcfhj7Mn5y1o6LxJ2l0apeZE0rvPWCijdL4ShXQF6+sJc0MXIHWE0JL

```

1.3 Adding a NAT Instance to our infrastructure

Our initial VPC has been built, and we've added a leap/bastion host, but looks like we missed something that we will certainly need. Now we are going to add a NAT instance into another of the "Public" subnets, so that future private instances can use the NAT instance for communication outside of our VPC to the Internet. We'll do this in the exact same way we built out the leap/bastion instance, by adding another stack to our infrastructure, built on top of our existing VPC stack. We'll also make use of the Key Pair we created in the last step as the Key Pair for our NAT instance. This way we don't have to expose the instance externally for management and we could use the leap instances for its intended purpose.

Let's start:

1. Just like in the last exercise we need to figure out what PublicSubnet we will put this instance in. Refer back to the Outputs tab of the main VPC stack or your notes. It doesn't matter which one we choose here.
2. Next let's go and download the CloudFormation template we'll need for this step. You can find it here https://s3-us-west-2.amazonaws.com/munns-bootcamps/reinvent2013/Lab1-nat_stack.template. Open this template up in your text editor and take a look around. This template file has a bit less going on then the last one:
 - 1) An Elastic IP address (EIP) for the NAT instance
 - 2) A Security Group for the NAT instance
 - 3) The NAT instance resource itself
 - 4) A Private Route table
 - 5) A Private route using the NAT instance as the default route for "0.0.0.0/0"(all traffic)
3. Go to the CloudFormation console and create a new stack using this template file. Name it something representative of what it is (I named mine "**vpc-NAT**"), and select the template downloaded from the previous step. On the next step now in the wizard we are faced with a few parameters we need in order for this host to launch. In no particular order:
 - 1) "VpcId" – This is the ID of the VPC we are launching this instance into. From step 1 of this section of the lab you should have this copied somewhere.
 - 2) "SubnetId" – This is the ID of the "Public" subnet inside our VPC. Again, you should have this copied somewhere.
 - 3) "KeyName" – This is the Key Pair we created on the leap/bastion host (i.e. "bastkey1").

When you have this all filled out, it should look like this:

Create Stack [Cancel]

SELECT TEMPLATE | **SPECIFY PARAMETERS** | ADD TAGS | REVIEW

Stack Description: Builds a NAT host. ****WARNING**** This template creates Amazon EC2 instance(s). You will be billed for the AWS resources used if you create a stack from this template.

Specify Parameters
Below are the parameters associated with your CloudFormation template. You may review and proceed with the default parameters or make customizations as needed below.

SubnetId
SubnetId of an existing Public facing subnet in your Virtual Private Cloud (VPC)

VpcId
VpcId of your existing Virtual Private Cloud (VPC)

KeyName
Name of an existing EC2 KeyPair to enable SSH access to the instances

< Back Continue >

- Click Continue; feel free to add in Tags like before, and then continue on to launch this stack.
- Much like before we'll need to wait for this stack to completely launch before we can move forward. Keep an eye on the Events tab to see if anything goes wrong.
- Once the stack creation is complete, take a look at the Outputs tab and record the "PrivateRouteTableId" you see there, as we will need it later today:

	Name	Created	Status
<input checked="" type="checkbox"/>	vpc-NAT	2013-10-09 16:34:19 UTC-4	● CREATE_COMPLETE
<input type="checkbox"/>	vpc-leaphost	2013-10-09 14:50:18 UTC-4	● CREATE_COMPLETE
<input type="checkbox"/>	masterVPC	2013-10-09 14:42:07 UTC-4	● CREATE_COMPLETE

Stack: vpc-NAT

Description
Outputs
Resources
Events
Template
Parameters
Tags

Stack Outputs

Output values may have been specified by the template author and will be available when stack creation is complete.

Key	Value	Description
PrivateRouteTableId	rtb-bf9cd8d4	Private Route Table ID

1.4 Modifying our VPC stack

We've got a lot of the components we'll need for our infrastructure to talk through minus one last piece, an Elastic Load Balancer (ELB). Later today we will actually build out the application's infrastructure, but we're going to add the ELB now to the main VPC now for a number of reasons. One is that it is going to live in our public subnet infrastructure so that users out on the Internet can access it via our IGW. We're also adding this to our VPC stack because we want to think of our ELB as an infrastructure component compared to an application component. It also isn't a component that is worth having in its own stack as opposed to being part of another one.

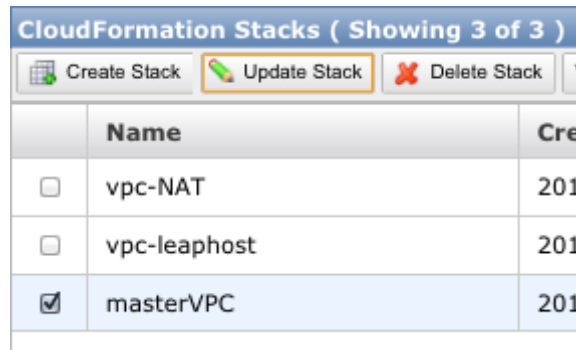
We'll be updating the already running main VPC stack we first launched earlier. We'll do that by adding resources to our existing template file, and then applying the update via the console. Note that so far today you haven't had to write any JSON manually. We'll save you the trouble of writing it from scratch totally, but you shouldn't underestimate what is involved in adding in the resources you are about to. Remember to run your modified stack through JSONLint to clear up any potential errors.

Let's start:

- Download the template snippet containing the resources that we need to add to the current environment. It can be found <https://s3-us-west-2.amazonaws.com/munns-bootcamps/reinvent2013/Lab1-addELB.template>. Lets open this up in our text editor and take a look:
 - You'll notice that this isn't a complete CloudFormation template. It is missing Parameters and Mappings, which we won't need to set.
 - There are two resources: A security group, and the ELB itself
 - There are two Outputs that we will need after the stack has been updated for later today.
- We need to merge the contents of this template file and the original one, into one file. Open up the original template file again ("Lab1-main_VPC_stack.template") in your text editor, and copy in the parts from this new one ("Lab1-addELB.template") into the relevant places. Note that order doesn't matter, just so long as object types are in the right section of the template. For instance, the Resources we are adding should be copied into the existing Resource block. You should be able to

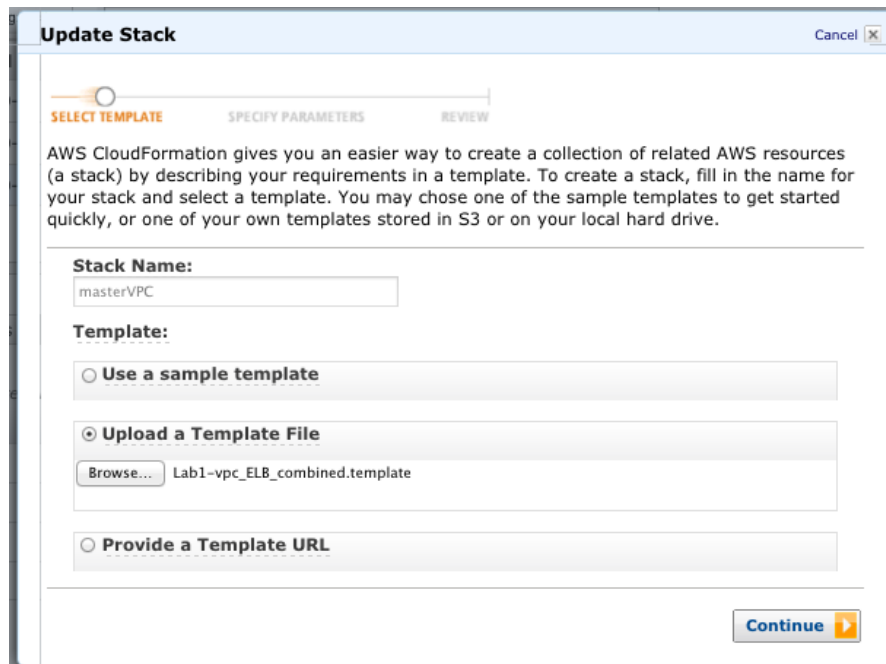
just copy and paste the different objects right in to their original template sections (Resources, Outputs).

3. Once you've gotten everything copied over, let's check our linter again (jsonlint.com) to confirm that there are no errors in our script:
 - 1) Open up <http://jsonlint.com> in a browser.
 - 2) Select all of the text from the template in the text editor. Use CTRL + A to do this. Then paste the text into the main field in the jsonlint.com page.
 - 3) Click on the "Validate" button.
 - 4) You should again see the green bar at the bottom with "Valid JSON". If you see any thing otherwise, look for missing or misplaced commas or brackets. JSONLint should give you a good idea what is wrong. Once JSONLint says we are all good, save this modified file under a different name, such as "**Lab1-vpc_ELB_combined.template**". Now with this modified template file, let's modify our running stack.
4. Go back to the AWS CloudFormation console again. Select the stack which created the main VPC and click on the "Update Stack" button on the top. If your Update Stack button is not active, check the status of your stack. If it says it is still in the process of being created, click the "**Refresh**" button on the upper right of the AWS Management Console:



CloudFormation Stacks (Showing 3 of 3)		
	Name	Created
<input type="checkbox"/>	vpc-NAT	201
<input type="checkbox"/>	vpc-leaphost	201
<input checked="" type="checkbox"/>	masterVPC	201

5. On the Update Stack window, select "Upload a Template File" and find the file we just created:



Update Stack Cancel

SELECT TEMPLATE

SPECIFY PARAMETERS

REVIEW

AWS CloudFormation gives you an easier way to create a collection of related AWS resources (a stack) by describing your requirements in a template. To create a stack, fill in the name for your stack and select a template. You may chose one of the sample templates to get started quickly, or one of your own templates stored in S3 or on your local hard drive.

Stack Name:

Template:

☐ Use a sample template
 ☒ Upload a Template File

Lab1-vpc_ELB_combined.template

☐ Provide a Template URL

Continue

6. There aren't any parameters to change here, so just Continue on through and start the update.
7. If you look now at the Events tab for your master VPC stack, you should see it making the changes we've asked it to:

<input checked="" type="checkbox"/>	masterVPC	2013-10-09 14:42:07 UTC-4	UPDATE_IN_PROGRESS	Builds a VPC w/ INET Gateway and 3 public subnets. **WARNING
-------------------------------------	-----------	---------------------------	--------------------	--

Stack: masterVPC														
Description	Outputs	Resources	Events	Template										
<table> <tr> <th>Time</th><th>Type</th><th>Logical ID</th><th>Physical ID</th><th>Status</th></tr> <tr> <td>2013-10-09 17:36:36 UTC-4</td><td>AWS::CloudFormation::Stack</td><td>masterVPC</td><td>arn:aws:cloudformation:us-west-2:102901597367:stack/masterVPC/7f8e4750-3112-11e3-9c8e-500150b352e0</td><td>UPDATE_IN_PROGRESS</td></tr> </table>					Time	Type	Logical ID	Physical ID	Status	2013-10-09 17:36:36 UTC-4	AWS::CloudFormation::Stack	masterVPC	arn:aws:cloudformation:us-west-2:102901597367:stack/masterVPC/7f8e4750-3112-11e3-9c8e-500150b352e0	UPDATE_IN_PROGRESS
Time	Type	Logical ID	Physical ID	Status										
2013-10-09 17:36:36 UTC-4	AWS::CloudFormation::Stack	masterVPC	arn:aws:cloudformation:us-west-2:102901597367:stack/masterVPC/7f8e4750-3112-11e3-9c8e-500150b352e0	UPDATE_IN_PROGRESS										

8. Once the update is complete, go back to the Outputs tab and notice the new resources listed:

<input checked="" type="checkbox"/>	masterVPC	2013-10-09 14:42:07 UTC-4	UPDATE_COMPLETE
-------------------------------------	-----------	---------------------------	-----------------

Stack: masterVPC																							
Description	Outputs	Resources																					
<table> <tr> <th>Key</th><th>Value</th><th>Description</th></tr> <tr> <td>VpcId</td><td>vpc-30286c5b</td><td>VPC ID of newly created VPC</td></tr> <tr> <td>PublicSubnetA</td><td>subnet-da286cb1</td><td>Public Subnet in AZ A</td></tr> <tr> <td>PublicSubnetB</td><td>subnet-d2286cb9</td><td>Public Subnet in AZ B</td></tr> <tr> <td>PublicSubnetC</td><td>subnet-df286cb4</td><td>Public Subnet in AZ C</td></tr> <tr> <td>ProdWebElasticLoadBalancer</td><td>masterVPC-ProdWebE-DE1FEJ2MXHN2</td><td>ELB to Assign to Prod Web Instances</td></tr> <tr> <td>ExternalElbUrl</td><td>http://masterVPC-ProdWebE-DE1FEJ2MXHN2-585410477.us-west-2.elb.amazonaws.com</td><td>URL of the website</td></tr> </table>			Key	Value	Description	VpcId	vpc-30286c5b	VPC ID of newly created VPC	PublicSubnetA	subnet-da286cb1	Public Subnet in AZ A	PublicSubnetB	subnet-d2286cb9	Public Subnet in AZ B	PublicSubnetC	subnet-df286cb4	Public Subnet in AZ C	ProdWebElasticLoadBalancer	masterVPC-ProdWebE-DE1FEJ2MXHN2	ELB to Assign to Prod Web Instances	ExternalElbUrl	http://masterVPC-ProdWebE-DE1FEJ2MXHN2-585410477.us-west-2.elb.amazonaws.com	URL of the website
Key	Value	Description																					
VpcId	vpc-30286c5b	VPC ID of newly created VPC																					
PublicSubnetA	subnet-da286cb1	Public Subnet in AZ A																					
PublicSubnetB	subnet-d2286cb9	Public Subnet in AZ B																					
PublicSubnetC	subnet-df286cb4	Public Subnet in AZ C																					
ProdWebElasticLoadBalancer	masterVPC-ProdWebE-DE1FEJ2MXHN2	ELB to Assign to Prod Web Instances																					
ExternalElbUrl	http://masterVPC-ProdWebE-DE1FEJ2MXHN2-585410477.us-west-2.elb.amazonaws.com	URL of the website																					

You should go ahead and copy down the two new Key's listed for your ELB, as we will need both later today.

Lab 1 Closing

You've now completed the first lab of the day. Hopefully you've gotten a feel for how to work with AWS CloudFormation and some of the power it has. We now have a basic infrastructure inside of an Amazon VPC that contains methods for instances both public and private to access the outside Internet, an instance to interact with the rest of the network from, and an ELB we will eventually access our application through today. This entire infrastructure is in JSON based documents that can be easily modified, version tracked, and tested against to track change in your infrastructure. We only really scraped the surface here of the full power of CloudFormation, and if you have some free time here at the end of this lab and before we start the next part of today's presentation I invite you to look deeper at the templates and think about what parts of your own infrastructure on AWS could benefit from what you just did now.

Lab #2

2.1 Launch a Multi-AZ RDS MySQL Instance in our VPC

We're almost ready to jump into using OpsWorks and Chef to deploy our web/application instances and our application itself, but we need one last component in our infrastructure before we can; a database. Let's use CloudFormation again to define and deploy our database using Relational Database Service (RDS). As part of deploying this RDS instance, we're going to create the private subnets inside our VPC to deploy it into. The private subnets exist without having direct access to the Internet via an IGW, and instead would need to rely on our NAT instance to reach the outside world. In the next lab step we'll actually add our web/app instances into these same private subnets.

Let's start:

1. Go ahead and download the CloudFormation template file we'll use for this stack: https://s3-us-west-2.amazonaws.com/munns-bootcamps/reinvent2013/Lab2-rds_stack.template.
2. If you look inside this file, you'll see that there's nothing too different about it from what we've seen before. Much like the first stack we launched, there is a number of VPC Subnet resources, some route table attachments, and then our resources for our RDS instance, namely the DBSubnetGroup, DBSecurityGroup, and the RDS instance itself. In the stack template there are already settings for the instance size, the amount of storage, and the engine and its version (MySQL 5.6), as well setting Multi-AZ to true.
3. There are a few parameters we'll need to fill in for this RDS instance:
 - 1) Our "VpcId"
 - 2) The "PrivateRouteTableID" that we created as part of our NAT stack in the last lab
 - 3) Three parameters we will set for our database: the name of the default database, and the username/password that we'll have our application use. You can make these anything you want, but be sure to remember the password you choose. You won't be able to see it again in CloudFormation. Note that you could create an admin account, and then log in manually and create a user for our application with appropriate grants, but for the sake of time and trying to minimize complexity we won't do that in this lab.
4. Go to the CloudFormation console and create this stack as a new one. When you fill out the parameters with the above information, it should look like this:

Create Stack

Stack Description: AWS CloudFormation Sample Template VPC_RDS_DB_Instance: Sample template showing how to create an RDS DBInstance in an existing Virtual Private Cloud (VPC); **WARNING** This template creates an Amazon Relational Database Service database instance. You will be billed for the AWS resources used if you create a stack from this template.

Specify Parameters

Below are the parameters associated with your CloudFormation template. You may review and proceed with the default parameters or make customizations as needed below.

PrivateRouteTableID	rtb-bf9cd8d4
ID of the private route table in your VPC that will use the NAT instance as default path to 0.0.0.0/0	
DBPassword	*****
The database admin account password	
DBUsername	reinvent
The database admin account username	
DBName	reinventdb
The database name	
VpcId	vpc-30286c5b
VpcId of your existing Virtual Private Cloud (VPC)	

< Back Continue

5. Click on through and launch the stack. Again it is up to you if you want to add any tags.
6. RDS takes a bit longer to launch than the rest of our CloudFormation stack based resources have so far. Behind the scenes RDS is building out the two EC2 instances, their storage, and then installing the software needed to make RDS work. Based on the defaults that are set for RDS, it will also take an initial backup of the database to make future backups quicker, so this adds to the time as well.
7. Once the create is complete, go and take a look at the Outputs tab for this stack. You'll want to make note of information we see here, as we will need it in the next step.

	Name	Created	Status
<input checked="" type="checkbox"/>	vpc-RDS	2013-10-10 14:24:54 UTC-4	CREATE_COMPLETE
<input type="checkbox"/>	vpc-NAT	2013-10-09 16:34:19 UTC-4	CREATE_COMPLETE
<input type="checkbox"/>	vpc-leaphost	2013-10-09 14:50:18 UTC-4	CREATE_COMPLETE
<input type="checkbox"/>	masterVPC	2013-10-09 14:42:07 UTC-4	UPDATE_COMPLETE

Stack: vpc-RDS

Description **Outputs** Resources Events Template Parameters Tags

Stack Outputs
Output values may have been specified by the template author and will be available when stack creation is complete.

Key	Value	Description
DBConnectionName	vmx4u32ugv6c4d.cgmaymugwpkn.us-west-2.rds.amazonaws.com	DNS name for this RDS instance.
DatabaseName	reinventdb	Name of pre-created database
PrivateSubnetA	subnet-fa89c891	Public Subnet in AZ A
PrivateSubnetB	subnet-ff89c894	Public Subnet in AZ B
PrivateSubnetC	subnet-e589c88e	Public Subnet in AZ C

8. The last thing we should do is confirm that our database is up and accessible to our infrastructure. We'll do this from our leap/bastion host that we set up earlier.
 - 1) Log back into our bastion host: "ssh -i my_key_pair.pem ec2-user@54.200.81.186"
 - 2) Let's see if we have the mysql client already installed. Just type in mysql and hit enter. There is a good chance that the default Amazon Linux instance we launched earlier doesn't have mysql installed yet:

```
[ec2-user@ip-192-168-11-238 ~]$ mysql
-bash: mysql: command not found
```

- 3) Let's quickly install the mysql client to help us here. Type in "sudo yum install -y mysql" and hit enter. A bunch of output from the yum command will scroll by, but at the end you should see this:

```
Installed:
  mysql.noarch 0:5.5-1.3.amzn1

Dependency Installed:
  mysql55.x86_64 0:5.5.33-1.37.amzn1
  mysql55-common.x86_64 0:5.5.33-1.37.amzn1
  mysql55-libs.x86_64 0:5.5.33-1.37.amzn1

Complete!
[ec2-user@ip-192-168-11-238 ~]$
```

- 4) With mysql installed, we can now test connectivity to our database instance. This is where we will need the information from both our CloudFormation Outputs tab, plus the information we input into our CloudFormation stack parameters when we launched this stack. With that in hand, we can access mysql with the following command from the leap/bastion host:

```
mysql -u <DBUsername> -p -h <DBConnectionName>
```

Then you will be prompted for the password you set. It should all look like this:

```
[ec2-user@ip-192-168-11-238 ~]$ mysql -u reinvent -p -h vmx4u32ugv6c4d.cgmaymugw
pkn.us-west-2.rds.amazonaws.com
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 26
Server version: 5.6.13-log MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

9. If you've reached this point then your database is all set up and you are ready for the next part.

2.2 Use OpsWorks to Deploy our Application

With our VPC, ELB, and RDS database all set up and running, now we need an actual application to live in our infrastructure. We're going to be setting up WordPress (<http://wordpress.org/>); an open source PHP based content management system as our demo application today. WordPress is a great piece of software to use because it can be very easy and basic to set up, but is extremely customizable and powerful. To deploy and update WordPress we're going to step away from CloudFormation for a bit, and use AWS OpsWorks to create a number of instances and deploy our application to. We'll make use of OpsWork's built in implementation of Opscode Chef, a configuration management tool to help.

There are a number of pieces to OpsWorks that need to be put in place to get our application up and running. OpsWorks has a concept of having a "Stack", which is defined as:

"A stack is a **set of Amazon EC2 instances**, AWS resources, and configurations that serves a single purpose. For example, one stack might be used for your **staging environment** and another for **production**."

Stacks are made up of layers, which are in turn made up of EC2 instances. A layer is defined as:







"A layer is a **blueprint** for setting up and configuring a set of instances and related resources such as **volumes** and **Elastic IPs**. OpsWorks provides a set of built-in layers and makes it easy to create custom layers."

Once you have a Stack created, with layers added, made up of instances, then you can easily deploy your applications and manage them as they run. With Chef built in, this means you can customize and configure pretty much every aspect of your EC2 instances, their OS, the software running on them, and your application code, in an automated and efficient fashion. In fact, once we have everything in OpsWorks configured, we shouldn't even have a need to log into the EC2 instances that will be running WordPress for us. Ideally we'd be centralizing all our logs and monitoring/metrics collection, though for today's lab we won't go through all of that.

One thing we will be doing is deploying custom Chef cookbooks on our instances to handle some of the custom configuration behind our application. One of the powers of OpsWorks is that even the built in Chef cookbooks can be largely modified and overwritten without too much worry.

Let's get started with setting up OpsWorks for our WordPress site:

1. Go to the OpsWorks console: <https://console.aws.amazon.com/opsworks/> or select it from the Services dropdown, under the Deployment & Management heading:

All AWS Services	 CloudFormation	 CloudWatch	 Data Pipeline
Compute & Networking	AWS CloudFormation lets you create and update a collection of related AWS resources in a predictable fashion.	Amazon CloudWatch provides monitoring for resources and applications.	AWS Data Pipeline is a lightweight orchestration service for periodic, data-driven workflows.
Storage & Content Delivery			
Database			
Deployment & Management >	 Elastic Beanstalk	 IAM	 OpsWorks
App Services	AWS Elastic Beanstalk is an application container for deploying and managing applications.	AWS Identity and Access Management (IAM) lets you securely control access to AWS services and resources.	AWS OpsWorks is a DevOps platform for managing applications of any scale or complexity on the AWS cloud.

2. Next click on "Add Your First Stack":

Welcome to AWS OpsWorks

AWS OpsWorks is a flexible application management solution with automation tools that enable you to model and control the complete lifecycle of your applications and the infrastructure on which they run. OpsWorks simplifies operations management and application deployment but also lets you orchestrate complex systems of applications.

[Add Your First Stack](#)

3. From here we are going to fill out some of the most basic settings for our stack. Some key ones to be aware of and set properly:
 - 1) Name: Give this stack a name meaningful to what will run in it. I am calling mine WordPress
 - 2) Region: **IMPORTANT!** Make sure you select **US West (Oregon)**
 - 3) Select from the dropdown the VPC we created earlier. You should only see one listed
 - 4) Select a subnet that is in the 192.168.100+ range. (101, 102, or 103), as these are the right subnets that are considered "private" and will allow us to use our ELB properly
 - 5) Leave the default OS as Amazon Linux
 - 6) You could leave the Default Root Device set to Instance Store, but EBS Backed is slightly faster to get running
 - 7) Leave the IAM role the default for now
 - 8) Set the "Default SSH key" to the one we created on our leap/bastion host: i.e. "bastkey1"
 - 9) Leave the Default IAM instance profile the default
 - 10) You can select any Hostname theme you want or leave it as the default.
 - 11) Feel free to select any color!
 - 12) Set nothing under the Advanced Settings

It should look similar to this when you are done:

Add Stack

Name	WordPress
Region	US West (Oregon)
VPC NEW	vpc-30286c5b - masterVPC
Default subnet NEW	192.168.101.0/24 - us-west-2a
Default operating system	Amazon Linux
Default root device type	<input type="radio"/> Instance store <input checked="" type="radio"/> EBS backed
IAM role	aws-opsworks-service-role
Default SSH key	bastkey1
Default IAM instance profile	aws-opsworks-ec2-role
Hostname theme	European Cities
Stack color	<input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
Advanced »	

- Click on “Add Stack”
- After that you should now be viewing the main dashboard for your stack:

WordPress ▾

Stack

Layers

Instances

Time-based

Load-based

Apps

Deployments

Monitoring


Resources NEW

Permissions

WordPress

A stack represents a collection of EC2 instances and related AWS collectively. Within a stack, you use layers to define the configurati deploy. [Learn more.](#)

1




Add your first layer

A layer is a blueprint for a set of EC2 profiles and security groups.

[Add a layer](#)

2



Add your first instance

An instance represents an EC2 insta

- Now we’re ready to follow through and build out our first layer. Click either on the Layers button in the menu to the left, or on the “Add a layer” link you see in the main dashboard page.
- On this page we are given a number of options based on the default “Layer type” selected. Change that to be “PHP App Server” and you’ll see it simplified quite a bit.

8. In the “Elastic Load Balancer” drop down, select the ELB we created earlier as part of our VPC stack. It should be the only option for you visible. Once done, your “Add Layer” page should look just like this:

Add Layer

Layer type PHP App Server

The PHP Application Server layer is a blueprint for instances that function as PHP application servers. By default PHP 5.3 and Apache 2.2 are installed. [Learn more.](#)

Elastic Load Balancer masterVPC-ProdWebE-DE1FEJ2M

[Cancel](#) [Add Layer](#)

9. Click on “Add Layer” to create this PHP layer. Any instances we launch as part of this layer will get attached to the ELB we selected as part of their bootstrap and configure process, there will be no further work. Once the instances are healthy to the ELB, they would be available to serve traffic to our end users.
10. Before we can go and add instances to this layer, we need to do some more configuration of it. Click on the “edit” action found in the Layers page:

Layer Name	Instances	Elastic Load Balancer	Actions
PHP App Server	Add an instance	masterVPC-ProdWebE- DE1FEJ2MXHN2	edit delete

11. Find the section on the Layer page entitled “Custom Chef Recipes” and click on “configure cookbooks”:

Custom Chef Recipes ⓘ

If you want to use Custom Chef recipes you need to [configure cookbooks](#) first.

12. This will actually take us back to our Stack settings page, scroll down and find the section labeled “Configuration Management”.
13. Toggle the button next to “Use custom Chef cookbooks” from “No” to “Yes”.
14. Select Git from the drop down as “Repository type”, and then enter the following URL for “Repository URL”: <https://github.com/teknogeek0/reinvent2013-cookbooks.git>
15. Leave “Repository SSH” key and “Branch/Revision” blank.
16. Next we need to enter some custom Chef JSON reflecting the information about our database that we just set up. Download this base template for our attributes to your computer: <https://s3-us-west-2.amazonaws.com/munns-bootcamps/reinvent2013/Lab2-attributes.json>. Once you have it, open it up and copy the entire contents of this file into the text box by “Custom Chef JSON” in OpsWorks.
17. You'll need to fill in every field here with the information that is for your infrastructure today, including:
 - 1) “dbhost”: is the “DBConnectionName” output from our RDS CloudFormation stack
 - 2) “user” & “password”: is what we set in our RDS CloudFormation stack for the parameters with the same name.

- 3) “database”: is the “DatabaseName” output from our CloudFormation stack

Once filled out, your JSON should look like this:

Custom Chef JSON

```
{
  "wordpress" :
  {
    "dbhost" : "vmx4u32ugv6c4d.cgmavmugwkn.us-west-2.rds.amazonaws.com",
    "db" : {
      "user" : "reinvent",
      "password" : "reinvent",
      "database" : "reinventdb"
    }
  }
}
```

Enter custom JSON that is passed to your Chef recipes for all instances in your stack. You can use this to override and customize built-in recipes or pass variables to your own recipes. [Learn more.](#)

Click on “Save”.

18. With our custom Chef settings in place, we can now go back to our Layer configuration and continue on. Click back to the “Layers” page using the link in the left hand menu.
19. Once there, again click on the “edit” action to the right of our ELB.
20. You’ll notice that the “Custom Chef Recipes” section looks different than it did before. It now gives us the ability to set some cookbook::recipe entries for the different steps that Chef can run in for OpsWorks.
21. We’re going to add in “wordpress::default” to the 3rd step, “Deploy”, and then click on the plus icon to the right of the field to add it:

Custom Chef Recipes ⓘ

Repository URL	https://github.com/teknogeek0/reinvent2013-cookbooks.git (change)	
0 Setup	myrecipe::default, myrecipe	+
0 Configure	myrecipe::default, myrecipe	+
1 Deploy	myrecipe::default, myrecipe	+ Add recipes to the Deploy lifecycle event.
	wordpress::default	✖

22. With that done, scroll down to the bottom and click on “Save”. We’re now ready to add instances to this layer and test our Chef cookbooks out.
23. Click on “Instances” on the left hand menu.
24. From this window, click on “Add an instance” under our layer title “PHP App Server”:

Instances

An instance represents an EC2 instance. Each instance belongs to a layer that defines the instance's settings, resources, installed packages, profiles and security groups. When you start the instance, OpsWorks uses the associated layer's blueprint to create and configure a corresponding EC2 instance. [Learn more](#).

PHP App Server

Using ELB: [masterVPC-ProdWebE-DE1FEJ2MXHN2](#)

No instances. [Add an instance](#).

You can [add more layers](#) to this stack.

25. This now will drop down and ask if we want to use a New or Existing instance. Leave it selected as "New". If you selected to have your hosts named after a certain scheme, you'll notice that the "Hostname" field is already populated. As for instances size, let's actually change that down to "Medium (m1.medium)" as we don't need a lot to make this application work. Lastly, leave the default subnet set to what it is, which should again be in the 192.168.(101-103).0/24 range, and in US-West-2(a,b,c). We won't change anything under the Advanced options for this instance, so leave everything set to default there:

26. Click on "Add Instance" to the right hand side. You'll see the page refresh and change a bit now that we have an instance set up:

Hostname	Status	Size	Type	AZ	Public IP	Actions
saint-petersburg	stopped	m1.medium	24/7	us-west-2a		start delete

[+ Instance](#)

27. Our instance hasn't been started yet, but its configuration is all ready to be started. Click on the "start" button under "Actions". Behind the scenes OpsWorks will call EC2 to start the instance, passing into the instance all of the things we've configured such as the instance class, AZ, VPC, subnet, our Stack and Layer configuration, and then the information it will need to access our Chef cookbooks and recipes. Quite a lot is going on under the cover. It is going to take a few minutes for this first instance to launch, so feel free to get up and stretch a bit! You'll notice that the instance has moved from the 'Stopped' state, to the "launching" state. Down below next to the instance you'll see it go into the "requested", "pending", "booting" state, followed by "running_setup" and then eventually "online".
28. We need to create an Application to deploy out to our "PHP App Server" layer. Click over to the "Apps" page via the left hand menu. From here it is just a few steps to get us set up:
- 1) Name your app. I am going with "MyApp".
 - 2) Select PHP from the "App type" dropdown.
 - 3) Select "Http Archive" from the "Repository type" dropdown.
 - 4) Enter the location of our WordPress zip file, which is here: <https://s3-us-west-2.amazonaws.com/munns-bootcamps/reinvent2013/wordpress-3.6.1.zip>
 - 5) Leave all the other settings blank or default. Your Add App page should now look like this:

Settings	
Name	MyApp
App type	PHP
Document root	Optional

Application Source	
Repository type	Http Archive
Repository URL	https://s3-us-west-2.amazonaws.com/munns-bootcamps/reinvent2013/wordpress-3.6.1.zip
User name	Optional
Password	Optional

- 6) If this is all good click on "Add App".
- 7) Now we see some details about the application we just created. Up in the top right hand corner is the "Deploy App" button. Click that so that we can deploy this app to our instance.
- 8) Leave all the settings default here on this page, and click on Deploy.
- 9) Our app is now being deployed out to our instance. The Chef recipe we listed in the "Deploy" section of our Layer setup ("wordpress::default") will also be run now at this time. When this is all done OpsWorks will have taken our application zip file, deployed it out on our instance, and then with the above recipe configured the wp-config.php file that WordPress needs for basic database connectivity.
- 10) The deploy should take a minute or two. Once it is done you can view the logs of what happened during the deploy to get a little bit better idea of what OpsWorks and Chef are doing for us.

Deployment **MyApp - deploy**[Repeat](#)Status **successful** User **root**

Created at 2013-10-30 19:00:07 UTC

Completed at 2013-10-30 19:01:06 UTC

Duration 00:00:59

Hostname	SSH	Layers	Duration	Log
✔ dortmund	ssh	PHP App Server	00:00:59	show

If everything went ok, then you are good to move onto the next step!

29. With our instance startup and setup complete, and our application deployed it is time to confirm that it is healthy to our ELB. Click on the ELB name next to “Using ELB:” to go to the ELB page in OpsWorks:

Using ELB: **masterVPC-ProdWebE-DE1FEJ2MXHN2** [🔗](#)

AZ	Public IP	Actions
us-west-2a		stop ssh

30. On this page we can see that our ELB is showing a single instance as green, with a checkmark next to it letting us know that the instance has passed health checks, and is now healthy as far as the ELB is concerned. If we click on the “DNS Name” of our ELB, it will take us to the main page of our new WordPress site. Let’s go through now and do the basic configuration just to test against later:

- 1) Fill out the “Information needed”, making note of the Username and Password you enter:

Information needed

Please provide the following information. Don't worry, you can always change these settings later.

Site Title

Username

Usenames can have only alphanumeric characters, spaces, underscores, hyphens, periods and the @ symbol.

Password, twice

A password will be automatically generated for you if you leave this blank.

Strong

Hint: The password should be at least seven characters long. To make it stronger, use upper and lower case letters, numbers and symbols like ! * ? \$ % ^ &).

Your E-mail

Double-check your email address before continuing.

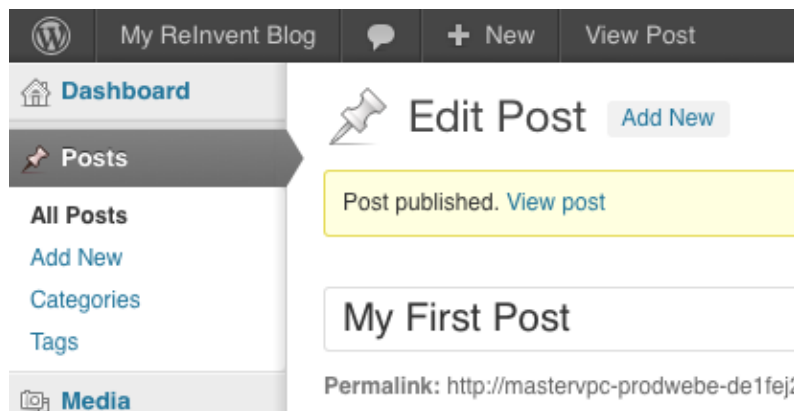
Privacy ☒ Allow search engines to index this site.

[Install WordPress](#)

- 2) Click on “Install WordPress”, on through the next screen, and then log into your WordPress dashboard using the credentials you just created.
- 3) While we could spend some time customizing our blog further, let’s instead just write a simple first post and get back to the infrastructure stuff we care about today. Click on the “+ New” button up in the top left to start a new post:



- 4) Enter in a Title and something into the actual Post text field, and then click on the blue “Publish” button to the far right.
- 5) You’ll now see that your Post has been published. Click on the “View post” link to get to what our outside users would see of this post:



My First Post

🕒 October 11, 2013 📁 Uncategorized ✎ Edit

This is my first post and will be used as a demo.

31. Now that we have our site up and running, and configured to work with our database, let’s go back to OpsWorks and configure two more instances to serve up this application and offer us some multiple AZ redundancy. Go back to the OpsWorks console, and the Instances page.
32. Add two more instances to our “PHP App Server” layer, putting one in each of the AZ’s we currently are not running in. For instance, if you are currently just running in “us-west-2a”, then add the two

instances to the 192.168.10(1,2,3).0/24 subnets in us-west-2b and 2c. Make both instances use the same instance class as the first one we launched, and don't modify any of the other Advanced options.

33. Once you have them configured, start up both of them. Again, it will take a little while for them to start up and bootstrap to completion. Once they are in the "online" state, click back over your ELB linked in above your instances:

PHP App Server

Using ELB: [masterVPC-ProdWebE-DE1FEJ2MXHN2](#)

Hostname	Status	Size	Type	AZ	Public IP	Actions
dortmund	online	m1.medium	24/7	us-west-2a		stop ssh
cardiff	online	m1.medium	24/7	us-west-2b		stop ssh
tyneside	online	m1.medium	24/7	us-west-2c		stop ssh

34. A few minutes after your instances go into the "online" state, you should see that all of your instances are registered and considered healthy by the ELB.

Layer	PHP App Server	
DNS Name	masterVPC-ProdWebE-DE1FEJ2MXHN2-585410477.us-west-2.elb.amazonaws.com	
Region	US West (Oregon)	
Attached availability zones	us-west-2a, us-west-2c, us-west-2b	
Attached subnets	192.168.10.0/24 - us-west-2a - masterVPC 192.168.11.0/24 - us-west-2b - masterVPC 192.168.12.0/24 - us-west-2c - masterVPC	
	192.168.102.0/24 - us-west-2b - VPC-RDS cardiff ● ✔ 1	192.168.101.0/24 - us-west-2a - VPC-RDS dortmund ● ✔ 1
	192.168.103.0/24 - us-west-2c - VPC-RDS tyneside ● ✔ 1	

If you've got this, then you are all done with this step!

2.3 Use CloudFormation and OpsWorks to Update Our Infrastructure and Application

Now that we have our application hosts all up and running, and our initial WordPress site up and configured, let's try pushing out some new functionality to our application. WordPress is an extremely versatile application with many plugins and customizations possible. One really popular plugin is the Memcached plugin (<http://wordpress.org/plugins/memcached/>), which adds caching for objects that would normally require database queries be made per page load. It can sometimes drastically speed up WordPress. It is a very easy plugin to install, just requiring the plugin library be put into a directory, and the wp-config file be updated.

From our last lab, we have wp-config updated via Chef as part of our wordpress::default recipe, and the full WordPress application controlled via our Layer App deployment. We'll need to update both of those in order to push out our changes here. Before that though, we'll need Memcached running somewhere. Luckily for us AWS has Amazon ElastiCache, a hosted cache platform supporting both Memcached and Redis. With ElastiCache, much like RDS, we can deploy managed resources that make it very painless to operate and scale with little day-to-day management. Much like we did earlier in the lab, we're going to make use of CloudFormation to deploy these resources into our existing VPC.

Let's start:

1. Go ahead and download the CloudFormation template file we'll use for this stack: <https://s3-us-west-2.amazonaws.com/munns-bootcamps/reinvent2013/Lab2-addMemcached.template>.
2. If you look inside this file, you'll see that there's nothing too different about it from what we've seen before. Much like the leap/bastion host, and NAT instance we launched earlier, we'll be launching our Memcached cluster into our VPC, into the private subnets we created earlier. One thing you will notice though, is that there are no Mappings or Outputs in his template file. We don't really have a need to have any mappings here, and in terms of Outputs; ElastiCache works a bit differently than other services in giving you back an endpoint that you can use in your applications. We'll need to get the endpoints from the ElastiCache console.
3. There are a few parameters we'll need to fill in for this ElastiCache instance:
 - 1) Our "VpcId"
 - 2) The "CacheSubnets" which are our private subnets created as part of our RDS stack.
4. Go to the CloudFormation console and create a new stack. Go and select the template file we downloaded for this section and again name it something relevant. I am calling mine "vpc-Cache". With everything entered, my "Create Stack" review page looks like this:

Create Stack Cancel

Please review the information below, then click Continue to create the stack.

Stack Information Edit Stack

Stack Name: vpc-Cache

Stack Description: Add ElastiCache Memcached to our VPC. ****WARNING**** This template creates an Amazon Ec2 Instance and an Amazon ElastiCache Cluster. You will be billed for the AWS resources used if you create a stack from this template.

Template: https://cf-templates-1djd9q44m7l2r-us-west-2.s3.amazonaws.com/2013285kCs-Lab2-addMemcached.template

IAM Acknowledgement: false

Parameters Edit Parameters

CacheSubnets subnet-fa89c891, subnet-ff89c894, subnet-e589c88e

VpcId vpc-30286c5b

Notification Edit Notification

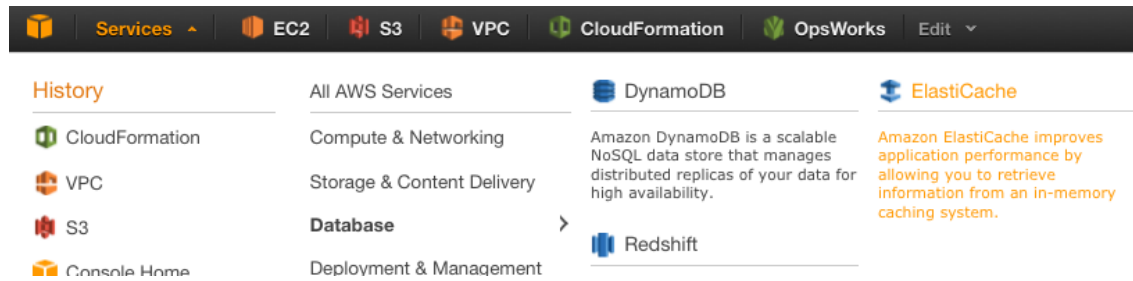
Notification: none

Creation Timeout (minutes): none

Rollback on Failure: true

< Back Continue

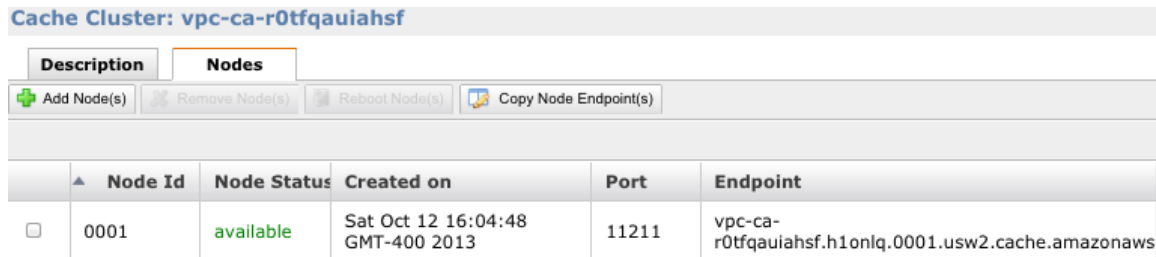
5. Click "Continue" and go on to start creating the stack. It will take a bit of time to complete, so keep your eye on the "Events" tab in the CloudFormation console.
6. Once your new stack is complete, navigate over to the ElastiCache console page, via the Services drop down, and the "Database" heading:



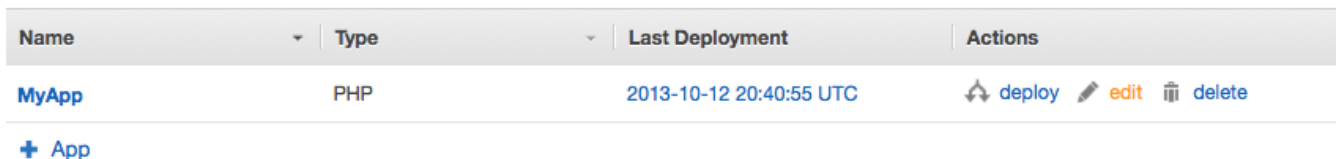
- On the ElastiCache dashboard, you should see to the right the “My Cache Clusters” section, with the cache cluster we just created:



- Click on either the “1 nodes” link, or on the “Configuration Endpoint” and you should see the “Endpoint” listed that we’ll need to use with our application. This is what it looks like when I click over to the nodes link, note the “Endpoint” column, that is what I am looking for:



- Copy the full name of the Endpoint and save it for later.
- Going back to the OpsWorks console, and then the Apps page.
- You’ll see your app listed here, click on the “edit” button:



- I’ve gone ahead and created an updated WordPress zip file with the Memcached plugin already in place. Think of me having done the work of another developer on your team, and now made the code ready to be deployed. Change the ‘Repository URL’ to: “<https://s3-us-west-2.amazonaws.com/munns-bootcamps/reinvent2013/wordpress-3.6.1-2.zip>” (notice the -2 at the end of the version), and then click “Save” at the bottom.
- Next we need to update our custom Chef JSON to reflect the new ElastiCache node we have. There has always been an attribute for it in our recipe, but normally you would have gone and added this to the recipe yourself, updating the attributes file, recipe, and template file. I did this for you today already; we just haven’t had the value set to anything but the default that was there. Click on the Stack button on the left hand menu, and then the “Stack Settings” button to the right:

Stack

Layers

Instances

Time-based

WordPress

A stack represents a collection of EC2 instances and related AWS resources that have a common purpose and that you want to manage collectively. Within a stack, you use layers to define the configuration of your instances and use apps to specify the code you want to deploy. [Learn more.](#)

Run Command

Stack Settings

Delete Stack

- Click on the Edit button to the right, and scroll down to the “Custom Chef JSON” text field. We’re going to update it from what it was earlier, adding in a new attribute called “cachecode”. You can see an example of the updated template file here: <https://s3-us-west-2.amazonaws.com/munns-bootcamps/reinvent2013/Lab2-3-attributes.json>. Once updated, your “Custom Chef JSON” text

Custom Chef JSON

```
{
  "wordpress" :
  {
    "dbhost" : "vmx4u32ugv6c4d.cgmaymugwpkn.us-west-2.rds.amazonaws.com",
    "db" : {
      "user" : "reinvent",
      "password" : "reinvent",
      "database" : "reinventdb"
    },
    "cachecode" : "vpc-ca-r0tfqauiahsf.h1on1q.0001.usw2.cache.amazonaws.com"
  }
}
```

- Click on the Save button, and then go back to our Apps page.
- On the right of our app is the “deploy” Action link, click on that. There isn’t anything that we need to change here, so go ahead and click on Deploy.
- You’ll now see the deployment happening across our instances:

Deployment MyApp - deploy

Repeat

Status running

User root

Created at 2013-10-12 21:48:08 UTC

Completed

at

Duration

Hostname	SSH	Layers	Duration	Log
tyneside	ssh	PHP App Server	-	
cardiff	ssh	PHP App Server	-	
dortmund	ssh	PHP App Server	-	

- It shouldn’t take much more than a minute for this deploy to happen. After it has completed you’ll see the duration of the deploy, as well as have access to a link to show you the logs:

Deployment **MyApp - deploy**[Repeat](#)Status **successful** User **root**

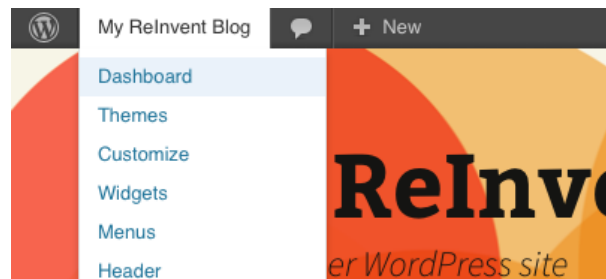
Created at 2013-10-12 21:48:08 UTC

Completed at 2013-10-12 21:49:45 UTC

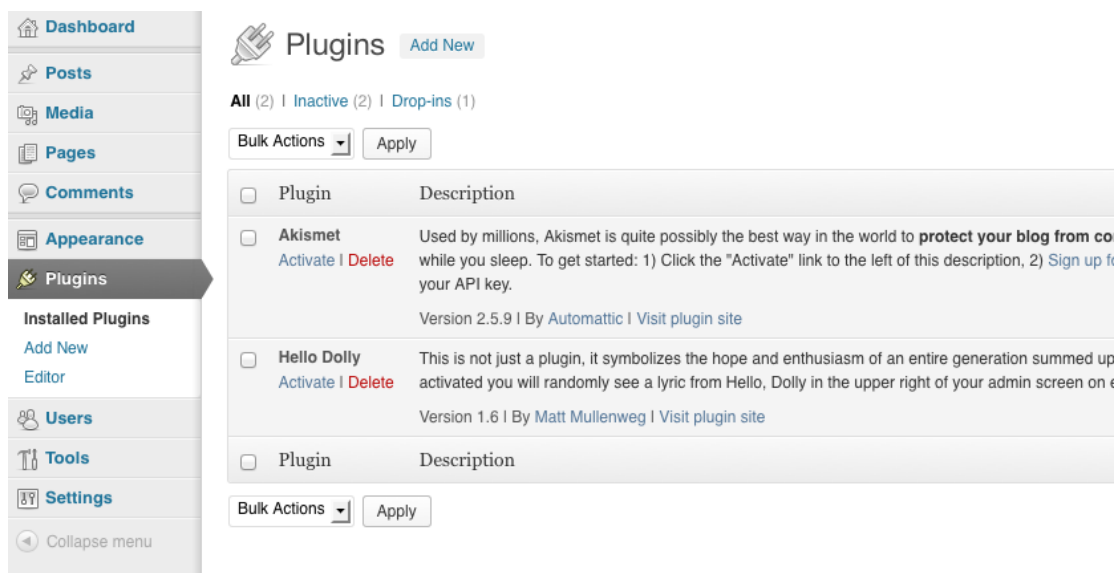
Duration 00:01:37

Hostname	SSH	Layers	Duration	Log
✓ tyneside	ssh	PHP App Server	00:01:36	show
✓ cardiff	ssh	PHP App Server	00:01:18	show
✓ dortmund	ssh	PHP App Server	00:00:37	show

19. If this was successful, then we should now have our WordPress application using Memcached on our ElastiCache node to cache objects for us. Let's confirm that WordPress is set up correctly for this by going back to our WordPress blog, and to the Admin Dashboard, which you can get to via the top bar on the blog when logged in as an Admin:

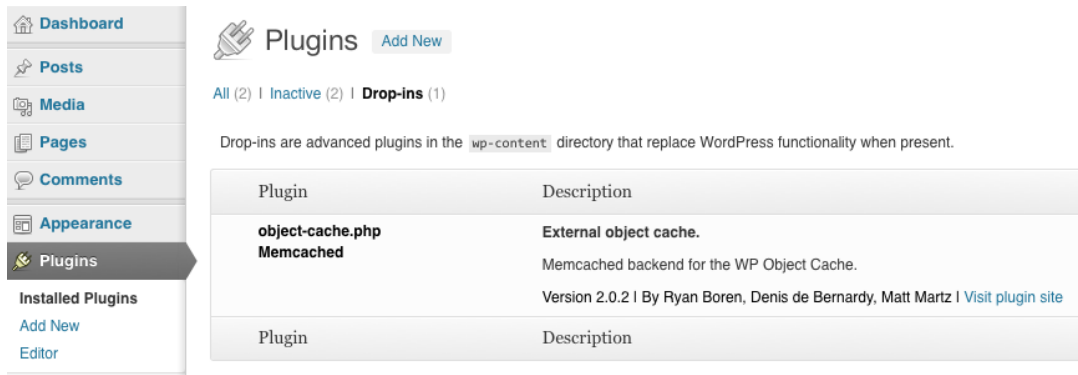


20. Now go and click on the "Plugins" link down in the left hand menu. From here we can see two Plugins that are available, but not activated. Note however that we don't see our new plugin installed here:



21. Turns out that the plugin we installed behaves a bit differently than some other WordPress plugins. It is what is known as a "Drop-in" plugin, one that once put into the proper directory will just start

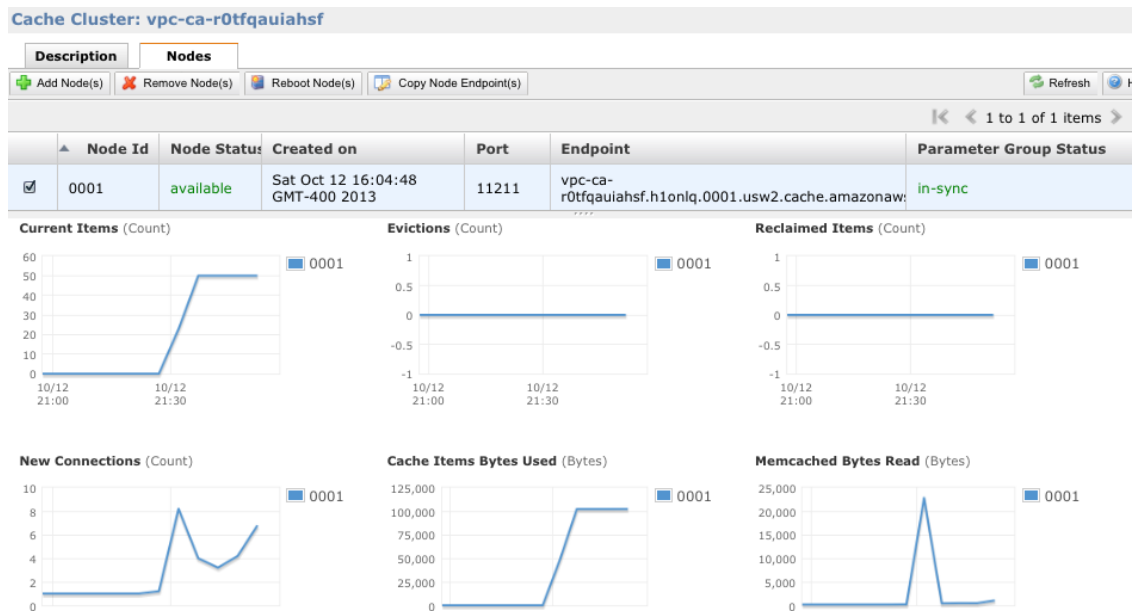
working without needing to be enabled. Clicking on the “Drop-ins” link on the Plugins page confirms this for us:



The screenshot shows the WordPress dashboard with the 'Plugins' menu item selected. The 'Drop-ins' tab is active, showing a table of installed drop-ins. The table has two columns: 'Plugin' and 'Description'. The first entry is 'object-cache.php' with the description 'Memcached'.

Plugin	Description
object-cache.php	Memcached

22. With the Memcached Plugin installed, the next thing to do is to actually confirm it is configured properly and caching objects for us. Go back to the Amazon ElastiCache console, click on the name of the cache cluster we created, and then on the “Nodes” tab. Next, select the node you see available. You should see several graphs below, and if you scroll through, you should see a slight increase in “Current Items”, “New Connections”, or “Cache Items Bytes Used”:



If you don't try refreshing the main page of the WordPress blog and clicking around a bit, then check back. This should generate some traffic that would let the Memcached plugin do its thing and cache some of our site's objects. Once you've confirmed that you are seeing some movement in the graphs here then you are done with this lab step.

Lab 2 Closing

You've just had a quick run through of using AWS OpsWorks to deploy, update, and manage an application on AWS. The instances we set up didn't require us to once log in directly to them in order to deploy or update our software. Opscode Chef played a big part in allowing us to push out configurations to our application in a controlled and standardized way. The instances are deployed in a Multi-AZ fashion and attached to our forward facing Elastic Load Balancer, providing us with a high level of redundancy and fault tolerance. We could have gone further and set up the Time or Load based scaling that OpsWorks has and given our infrastructure the

ability to scale as far as we could ever want it to. We also once again made use of AWS CloudFormation to deploy two different resources to our infrastructure, first a database, and then memcached nodes. CloudFormation made it easy to set up these managed resources, preventing us from having to touch a single command line, or click through the console to deploy them. We only scratched the surface of what is possible with OpsWorks. How do you see it being able to help you in your day-to-day tasks? Feel free to grab an AWS bootcamp assistant to discuss further.

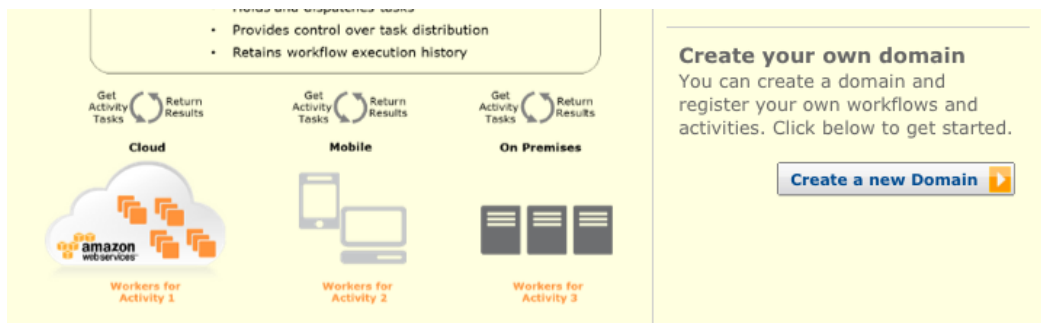
Lab #3

3.1 Setup Amazon Simple Workflow Service

In the last two labs we built out first our base infrastructure; a VPC, including subnets that were both “public” and “private”, a NAT instance, a leap/bastion host, a Multi-AZ RDS instance, an ELB, an ElastiCache Cluster, and then our application in a Multi-AZ setup. The application, database, caching, and load balancing tiers in our application are all made to quickly recover from any failure that they come up against and keep our application up and running with as little potential downtime as possible. There is however one component we haven’t yet made HA that plays a key role, our NAT instance. The NAT instance is needed so that our OpsWorks controlled application instances can talk back to the control plane. Without that communication path, the instances can’t be reached out to have updates or modifications applied. We should rectify this by making our NAT instance handle failure more gracefully. That is what this lab is all about.

The first bits in the big picture here are getting Simple Workflow Service setup. Unfortunately SWF isn’t yet supported in CloudFormation, so we’ll do it manually via the console. Again make sure you are in the right region for today’s lab (Oregon - US-West-2).

1. Go to the Simple Workflow console: <https://console.aws.amazon.com/swf/home?region=us-west-2>
2. Click on “Create a new Domain”:



From here, fill out the “Register Domain” dialog:

- 1) Name it something relevant for today, perhaps “InfraHelper01”
- 2) Set the “Workflow Execution Retention Period” to 1 day.
- 3) Fill in any sort of description you want. Click Register.
- 4) Close the page after this, as we don’t want to deprecate our new Domain!

Register Domain
Cancel

Provide the details of your new Domain below, then click **Register**

Name: * InfraHelper01

Workflow Execution Retention Period: * 1 Days

Description: A domain to help our infrastructure be better automated.

☒ Domain successfully registered.

Close


This is actually it for this part of the lab. We're going to rely on some pre-made code in the next step to create the Workflow and Activity types for this Domain. The code will run when the instance first bootstraps, and then shouldn't need to be run again. This all could be done by hand, but if you can put it in code, why not! Spend a moment or two digging around the SWF console to get familiar a bit with it, especially the Workflow and Activity Type pages.

3.2 Setup SNS+ SQS + infrastructure helper instance.

Next let's start getting the things we'll need in place for our automation stack to work. Before we can go and start having Auto Scaling generating triggers we'll need the place where they are going to go to exist. This is going to require us to create an SNS Topic, an SQS queue and have something that can talk to and poll them. We're going to actually set up another instance for this, which will live inside our VPC. We're going to launch it with OpsWorks with "Auto Healing" enabled such that if it were the instance were to become unavailable, it will just re-stand itself back up and keep on running the processes we need it to. We'll need this host to be able to talk to the AWS APIs too, and won't want that ability potentially blocked because of another dependent resource being down (such as a NAT instance). Because of this, we'll give this instance its own public IP address instead of having it go through the NAT. This instance is going to pull in some software we have in an outside repo to handle our SWF, SNS, SQS, and other API interactions.

Let's get started building out our automation infrastructure:

1. The first thing we're going to do is build out the SNS topic, SQS queue, and some other resources we'll need for our instance, namely its security group, EC2 role, and IAM policy to talk to all the services it needs to. Go and grab the template file found here <https://s3-us-west-2.amazonaws.com/munns-bootcamps/reinvent2013/Lab3-infrahelpersqs.template>.
2. If you look inside this file, you'll see some new resource types that we haven't come across before. You'll also see some resources that are typically tied to an instance in a CloudFormation template, but we're going to instead reference them in our OpsWorks Stack in a little bit.
3. There is only one parameter we'll need to fill in for this template:
 - 1) Our "VpcId" which our Security Group will need
4. Go to the CloudFormation console, and launch this template as a new stack.
5. Once the stack has reached the "CREATE_COMPLETE" state, take a look at the Outputs created. We'll need all of these values as we move forward into the rest of this lab:

 **Stack:** vpc-InfraHelper

Description	Outputs	Resources	Events	Template	Parameters	Tags
Stack Outputs						
<i>Output values may have been specified by the template author and will be available when stack creation is complete.</i>						
Key	Value	Description				
InfraHelperSecurityGroup	sg-530afb3c	Name of InfraHelper SecurityGroup				
QueueURL	https://sqs.us-west-2.amazonaws.com/102901597367/vpc-InfraHelper-InfraHelperQueue-8RVAA31AKKTL	ARN of SQS Queue				
TopicArn	arn:aws:sns:us-west-2:102901597367:vpc-InfraHelper-InfraHelperSNS-1M49W9BANJCWA	ARN of SNS Topic				

6. The next thing we need to do is to create the helper instance that will take our notifications out of our queue and pass them into SWF. It will also run all of the worker and decider processes that then interfaces with SWF and the AWS APIs. Go back to the OpsWorks console, and click on the Layers link.

7. Click on the “+ Layer” link to add a new layer to our stack. Set the “Layer type” to “Custom” and then give the layer a “Name” and “Short name” relevant to its use. I am going with “InfraHelper” and “ih” for these values (note: the “Short name” can’t contain capital letters):

Add Layer

Layer type Custom

The Custom layer allows you to create a fully customized layer. Standard recipes handle basic setup and configuration for the layer instances, and you implement custom Chef recipes to install and configure any required software. You can create as many custom layers as you require. [Learn more.](#)

Name InfraHelper

Short name ih

[Cancel](#) [Add Layer](#)

8. We have some further customization to do of this layer, so click on the “edit” link under “Actions”:
 - 1) There is a custom Chef recipe that is going to run to install all of our application code, and some cron jobs to run the Worker and Decider processes. Add this to the “Deploy” step: “InfraHelper::default”.

1 Deploy myrecipe::default, myrecipe [+](#) Add recipes to the Deploy lifecycle event.

InfraHelper::default [✖](#)

- 2) Under “Automatically Assign IP Addresses” slide the slider from “No” to “Yes” next to “Public IP addresses”

Automatically Assign IP Addresses ⓘ

Public IP addresses Yes

Elastic IP addresses No

- 3) Next, since this is a custom layer, there aren’t any language specific OS packages that are installed by default. The SWF processes we are going to run later are all written in PHP, and make use of the Amazon PHP SDK, so we’ll need to add those under “OS Packages”. You need to first add “php”, click the “+” button, and then do the same for “php-amazon-sdk”:

OS Packages ⓘ

Package name Enter name [+](#)

php [✖](#) php-amazon-sdk [✖](#)


- 4) We need to apply the Security Group that created in the CloudFormation stack we just created before. Select it from the drop down next to “Additional groups” and click the plus button:

Security Groups

Default groups

AWS-OpsWorks-Custom-Server

Additional groups

Select a security group ... 



vpc-InfraHelper-InfraHelperSecurityGroup-KS2VA1RCMF16 


- 5) Next you'll need to change the "Layer profile" to the one we also just created in our CloudFormation stack:

IAM Instance Profile

Stack default

aws-opsworks-ec2-role

Layer profile

vpc-InfraHelper-SWFInstancePro 

- 6) Lastly, we'll leave "Auto healing enabled":

Auto Healing

Auto healing enabled

Yes ☒

- 7) If you have all of this set, then click on "Save"
9. With our layer configured, there is one last bit of pre-deploy configuration we need to do before we can create our instance. The InfraHelper default Chef recipe we added just now to our layer requires some configuration attributes to be set, so let's go and set those in our Stack's custom Chef JSON.
- 1) Click on the "Stack" link in the left hand menu.
 - 2) Click on "Stack Settings" and then "Edit" in the top right.
 - 3) Go down to "Custom Chef JSON". We need to add to what we already have here. Download the attribute template file found here: <https://s3-us-west-2.amazonaws.com/munns-bootcamps/reinvent2013/Lab3-attributes.json>. We just need to modify the existing JSON by just adding in the new section. You can copy and paste it, and don't forget the comma at the end of the first section after the "cachecode". Fill in the various fields of this new JSON with the outputs, including the SQS URL from the outputs of the CloudFormation stack we just created, the SWF Domain we created in the first part of this lab, and the region that we'll be working in today (us-west-2). When you are done it should look like this:

Custom Chef JSON

```
{
  "wordpress" :
  {
    "dbhost" : "vmx4u32ugv6c4d.cgmaymugwpkn.us-west-2.rds.amazonaws.com",
    "db" : {
      "user" : "reinvent",
      "password" : "reinvent",
      "database" : "reinventdb"
    },
    "cachecnode" : "vpc-ca-r0tfqauiahsf.h1onlq.0001.usw2.cache.amazonaws.com"
  },
  "InfraHelper" :
  {
    "IH_queue" : "https://sqs.us-west-2.amazonaws.com/102901597367/vpc-InfraHelper-InfraHelperQueue-8RVAA31AKKTL",
    "IHswf_domain" : "InfraHelper01",
    "SWF_Region" : "swf.us-west-2.amazonaws.com",
    "EC2_Region" : "ec2.us-west-2.amazonaws.com"
  }
}
```

4) Click on “Save”.

10. With our Layer and Stack JSON properties set properly, we can now go and create the instance we’ll use as our Infrastructure Helper. Click on the “Instances” link on the left hand menu.
11. Under instances, scroll down to the new layer we created and click on “Add an instance”. Set the size to an m1.small, and select a subnet that is in our “public” range (192.168.1(1,2,3).0/24. Don’t modify anything under “Advanced”:

12. Click on “Add Instance”. Then once you see it listed, click on the “start” action.
13. Once the instance has the status of “online” we need to confirm that during its bootstrap it created the various activities in SWF that it was supposed to. Go back to the SWF console, and click on the “Activity Types” link in the left hand menu. You should now see a few different activities listed:

Activity Actions: Register New Deprecate

	Name	Version	Created On
<input type="checkbox"/>	VPCRouteMapper	1.0	Sun Oct 13 17:25:08 GMT-400 2013
<input type="checkbox"/>	SrcDestCheckSet	1.0	Sun Oct 13 17:25:07 GMT-400 2013
<input type="checkbox"/>	EIPMapper	1.0	Sun Oct 13 17:25:06 GMT-400 2013

We’re now done with this step in the lab. There isn’t a whole lot you had to do, but behind the scenes quite a bit is going on. On this infrastructure helper host are several cron jobs running scripts that are interacting with SQS, SWF, and other AWS APIs. They will fire up every minute and look to see if they have work from

various sources. First the IHQueueWatcher script runs to see if there is any new messages for it in the SQS queue that was just created. That queue, for the sake of this lab, only gets populated by messages coming in via SNS. Once the IHQueueWatcher finds a message, it then parses it, and depending on what the message is, goes on and creates Simple WorkFlow “Workflow Executions”. Another script then acts as a “Decider” on the workflow and handles the full process of managing what activities need to happen and in what order. Lastly, other scripts acting as “Activity Workers” will also run every minute looking for work, and take action on that work that the Decider script has deemed needed.

The code involved in all of these scripts, while not the most complicated, is also not trivial. We won't go into the actual code in this lab because doing so would probably take up more time than we have today, and it is more important to understand the pieces how each bit of the pieces work. The scripts that are running right now were written using the AWS PHP SDK, and you can find an example of writing code for the Simple WorkFlow Service using the SDK here: <https://github.com/amazonwebservices/aws-sdk-for-php/tree/master/samples/AmazonSimpleWorkflow/cron>. For information on working with SQS, you can refer to the general SDK documentation, found here: <http://docs.amazonwebservices.com/AWSSDKforPHP/latest/index.html>

Since PHP might not be your language of choice, there are other SDKs, in languages including Java, .Net, Ruby and others available on the main documentation page: <http://aws.amazon.com/documentation/>.

3.3 Re-launch NAT instance as part of an Auto Scaling group

Auto Scaling groups are an important part of building both a scalable infrastructure, but also a self-healing one. Let's modify our NAT CloudFormation stack to add an Auto-Scaling configuration and group to our instance. After it is configured, we'll re-launch for the changes to take effect. Unfortunately there's no way today to put Auto Scaling around running instances. Luckily enough that's one of the scenarios we were going to test in the first place! Code and smarts to make this all happen was deployed in the previous lab step when we set up our InfraHelper instance.

Ok, let's update the main NAT CloudFormation template to update our NAT instance to put it in an Auto Scaling group, and also to not have CloudFormation try to attach our EIP to the instance that gets created as part of Auto Scaling doing its thing.

Let's get started:

1. Instead of making you go and try to merge together two different CloudFormation templates, I have one created here already for you here <https://s3-us-west-2.amazonaws.com/munns-bootcamps/reinvent2013/Lab3-autoscalingnat.template>. We're going to update the main NAT stack that we have with this template, so do an “Update Stack” using this new file.
2. You do need two bits of information that you haven't needed before:
 1. The SNS Topic ARN that Auto Scaling will use for triggers. You can find this in the Outputs of the InfraHelper stack we created in the previous part of this lab. It looks like this:

`arn:aws:sns:us-west-2:102901597367:vpc-InfraHelper-InfraHelperSNS-1M49W9BANJCWA`
 2. The next thing we'll need are the “public” subnets that our NAT will launch into. You can find this in the Outputs of the main VPC stack.
3. With this information, go ahead and do an “Update Stack” on your original NAT CloudFormation stack. Your parameter's with the above information should look like this:

Update Stack

Cancel

SELECT TEMPLATE

SPECIFY PARAMETERS

REVIEW

Stack Description: Builds a NAT host. ****WARNING**** This template creates Amazon EC2 instance(s). You will be billed for the AWS resources used if you create a stack from this template.

Specify Parameters
Below are the parameters associated with your CloudFormation template. You may review and proceed with the default parameters or make customizations as needed below.

PublicSubnets

subnet-da286cb1,subnet-d

The list of Public SubnetIds for the NAT instance, you need at least 2 subnets in different availability zones

VpcId

vpc-30286c5b

VpcId of your existing Virtual Private Cloud (VPC)

InfraHelperSNS

arn:aws:sns:us-west-2:102

SNS Topic ARN for Auto Scaling to use for triggers.

KeyName

bastkey1

Name of an existing EC2 KeyPair to enable SSH access to the instances

< Back

Continue

4. This CloudFormation template is going to make a bunch of changes to our infrastructure as it runs. Check the Events tab and watch what happens. It should do the following:
 - 1) Disassociate the EIP from the existing NAT instance
 - 2) Remove and disassociate any route that the NAT instance was a destination for
 - 3) Create our new Auto Scaling configuration and launch group
 - 4) Start Auto Scaling so a new NAT instance is created
 - 5) Terminate the old running NAT instance

The successful creation of the instance in the Auto Scaling group will trigger a notification to SNS, which will in turn kick off the previously discussed flow to finish the task.

Feel free to play with this setup a bit more. Try terminating the NAT instance via the EC2 console. Watch it come back and be in action within a few minutes. While it is not a 100% perfect HA solution, it does allow for quick recovery in case of failure.

Lab 3 Closing

Even though this was a brief and not terribly deep example of these tools, hopefully the bigger picture is apparent. AWS CloudFormation and AWS OpsWorks can help you get from 0 to 60 in terms of provisioning, creating, configuring and managing resources. By making use of further automation tools on AWS we can start to build complicated frame works and work flows that do the kind of tasks that we would be doing manually in any other environment. The examples here also just scrape the surface of what is possible. What types of operations tasks are you doing every day? What types of things are you doing when the on-call pager goes off at 4am? Are there a few that are fairly routine? What about multi-step processes that are only manual, because up till now you haven't had a good flow control system? By working with Simple WorkFlow Service and some of the of the other "glue" services that AWS has to offer, such as Amazon SQS, Amazon SNS and AWS Auto Scaling, you can build really flexible frameworks in your infrastructure, in turn removing some of those day to day headaches, and letting you focus more your business's true goals.

Taking AWS Operations to the Next Level

The last thing we need to do today is to end this lab. Go back to aws.qwiklab.com. Click on this class “Workshop 9 – Taking AWS Operations to the Next Level”, the current “Class Lab” and then finally:

A rectangular button with a gradient from dark orange to light orange, containing the text "End Lab" in white.