



# **Working with AWS CloudFormation (for Linux)**

---

## Contents

Introduction .....	4
Create a JSON Document.....	4
What is JSON?.....	4
JSON for AWS CloudFormation Templates .....	5
Template Format Version .....	6
Description .....	6
Resources .....	6
Start your qwikLAB™.....	7
Select the AWS CloudFormation Service.....	9
Confirm your AWS Region .....	9
Your First AWS CloudFormation Template .....	10
Your Second AWS CloudFormation Template.....	14
Switch Regions .....	14
Create a Key Pair.....	14
Parameters.....	16
Add a Security Group.....	17
Outputs.....	18
Mappings.....	19
Launch .....	19
Inspect Output.....	20
Log into the Server .....	20
Your Third AWS CloudFormation Template.....	22
Define a Script in User Data .....	22
Set up a Wait Condition .....	23
Launch .....	23
Log Files.....	23
What's Next?.....	24
End Lab .....	24
Appendix A - How Do Scripts, Packages, and Templates Work Together? .....	25
Bootstrap Applications from a AWS CloudFormation Template.....	25
Helper Scripts.....	25
Cloudinit.....	25
Cloud-init Runs as a RC Package.....	25
Appendix B – SSH to an EC2 Instance .....	28

Connect to your EC2 Instance via SSH (Windows) .....	28
Download PuTTY .....	28
Download your EC2 Key Pair private key file .....	28
Connect to the EC2 Instance using SSH and PuTTY.....	29
Connect to your EC2 Instance via SSH (OS X and Linux).....	31
Download your EC2 Key Pair private key file .....	31
Connect to the EC2 Instance using the OpenSSH CLI client.....	31
End Lab .....	31

Copyright © 2013 Amazon Web Services, Inc. or its affiliates. All rights reserved.  
This work may not be reproduced or redistributed, in whole or in part,  
without prior written permission from Amazon Web Services, Inc.  
Commercial copying, lending, or selling is prohibited.

## Introduction

The purpose of this lab is to show you how to launch Amazon EC2 instances via automation, using AWS CloudFormation. We will "start simple" and then add more features to our template.

Imagine a "document driven data center". That's what you are about to create: a self-documented data center where everything is driven via automation. It's a rigorous approach that reduces errors by creating a set of repeatable processes. Because this lab is only an hour long, we are actually going to create a small subset of a datacenter...

AWS CloudFormation is a powerful way to launch a collection of AWS resources, known as a "stack" that consists in our case of multiple Amazon EC2 instances and other resources. One of the reasons that this is so powerful is that if anything fails to launch, AWS CloudFormation can roll the entire stack back. In other words, it will tear down every other resource in the stack. Coupled with other components such as Amazon Simple Workflow, Chef, and Puppet, Amazon CloudFormation can deal with complicated scenarios, such as servers that need to use a database connection string that won't be known until that database server is up and running.

In this exercise we will create an environment consisting of Web servers, an Elastic Load Balancer to manage inbound traffic, a database tier, and of course the ancillary things such as security groups and EBS volumes. There are several important caveats built in to this use case:

- We plan to launch a Linux environment, although Windows works in much the same way.
- This exercise uses Amazon Linux specifically, because the distro is tuned for AWS. You can substitute any other Linux distribution that you want – Amazon Linux just happens to already have some tools built in that we would otherwise need to install.
- We will launch a base AMI; and then run a *yum update* and install the software appropriate for our environment. By always applying updates, we avoid having to maintain (and update!) a library of custom AMIs. You may prefer to maintain "gold master" AMIs in your own operation.
- There are some other considerations that you may need to consider. For example, if you plan to use a database in your environment (you will in this lab), you would need to determine if you want to use one with a known connection string or launch one and then determine the connection string. The answer to that question would determine whether you would need to add some intelligent scripting.

Our examples in this exercise will be deliberately simplistic because most people start a template of their own from a downloaded sample such as the ones at:

<http://aws.amazon.com/cloudformation/aws-cloudformation-templates/>.

It's usually easier to modify existing templates than to create a template from scratch.

## Create a JSON Document

A bit of reading is in order before we get to the hands on part of this exercise.

### What is JSON?

According to Wikipedia, JSON (an acronym for JavaScript Object Notation) is a lightweight text-based open standard designed for human-readable data interchange. It is derived from the JavaScript programming language for representing simple data structures and associative arrays, called objects. Despite its relationship to JavaScript, it is language-independent, with parsers available for virtually every programming language. JSON Documents are intended for machine-to-machine interchanges, and accordingly are both terse and don't support comments.

## JSON for AWS CloudFormation Templates

An AWS CloudFormation template is structured similar to the following example:

```
{
  "AWSTemplateFormatVersion" : "version date",

  "Description" : "Valid JSON strings up to 4K",

  "Parameters" :
  {
    set of parameters
  },

  "Mappings" :
  {
    set of mappings
  },

  "Resources" :
  {
    set of resources
  },

  "Outputs" :
  {
    set of outputs
  }
}
```

Let's look at some key characteristics of AWS CloudFormation templates:

- The entire template (document) is based on key/value pairs, separated by colons.
- Note that usually everything in a template is a string, and accordingly needs to be enclosed in quotes. There are cases, such as parameters, where values can be a string, a number, or a comma-delimited list. You still enclose these values in quotes, though.
- Look closely and you'll note that there are also commas that mark the close of each object, except for the final one.
- If there are "subcomponents," then use curly brackets to denote where these subcomponents begin and end. This is just like JavaScript and many other languages.

There are six main objects in a CloudFormation JSON template:

1. Template format version
2. Description
3. Parameters
4. Mappings
5. Resources
6. Outputs

You can use some or all of these to create a valid AWS CloudFormation template. Objects can appear in any order in the document, but each object can only appear once.

### Template Format Version

The version must exactly match one of the release dates for AWS CloudFormation template specifications. If you omit this object altogether, AWS CloudFormation will assume that you are using the latest version, and “2010-09-09” is currently the only supported version.

```
"AWSTemplateFormatVersion": "2010-09-09",
```

### Description

This is free-form field which allows you to describe the template.

```
"Description": "Template to launch an Amazon Linux instance.",
```

### Resources

“Resources” contains a list of items that compose the AWS CloudFormation stack. AWS CloudFormation requires at least one resource to launch a stack, and each resource is listed as a type namespaced type such as AWS::EC2::Instance, AWS::S3::Bucket, etc.

The following example is a complete template which creates an Amazon EC2 instance from an AMI in the US East (N. Virginia) region. Later we’ll improve the template to work in all regions.

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "Template to launch an Amazon Linux instance.",
  "Resources" :
  {
    "Ec2Instance" :
    {
      "Type" : "AWS::EC2::Instance",
      "Properties" :
      {
        "ImageId" : "ami-3275ee5b",
        "InstanceType" : "m1.small"
      }
    }
  }
}
```

Download and use the completed template:

<http://us-east-1-aws-training.s3.amazonaws.com/self-paced-lab-14/Template1.json>

## Start your qwikLAB™

### 1. Start your qwikLAB™

Use the 'Start Lab' button to start your lab.

(Hint: If you are prompted for a token, please use one you've been given or have purchased.)



You will see the lab creation in progress.

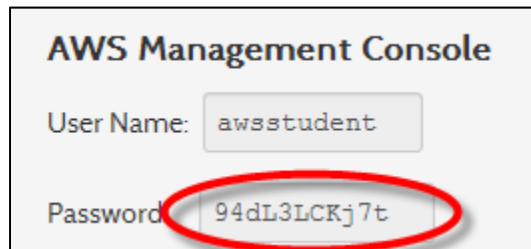


### 2. Note a few properties of the lab.

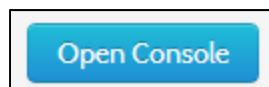
- a. **Duration** - The time the lab will run for before shutting itself down.
- b. **Setup Time** - The estimated lab creation time on starting the lab.
- c. **AWS Region** - The AWS Region the lab resources are being created in.

### 3. Copy the Password provided.

- d. Hint: selecting the value shown and using Ctrl+C works best



### 4. Click the 'Open Console' button.



### 5. Make sure that you are not logged into any other instances of the AWS console (in a student account or your own account), as this may cause conflicts when you open the console and log in below for this lab.

6. Login to the AWS Management Console

Enter the **User Name** 'awsstudent' and paste the password you copied from the lab details in qwikLAB™ into the **Password** field.

Click on the 'Sign in using our secure server' button.

In this step you logged into the AWS Management Console using login credentials for a user provisioned via AWS Identity Access Management in an AWS account by qwikLAB™.

**Amazon Web Services Sign In**

Please enter the AWS Identity & Access Management (IAM) User name and password assigned by your system administrator to sign in.

**AWS Account: 832809622232**

**User Name:**

**Password:**

[Sign in using our secure server](#)

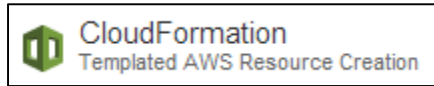
Please contact your system administrator if you have forgotten your user credentials.

[Sign in using AWS Account credentials](#)



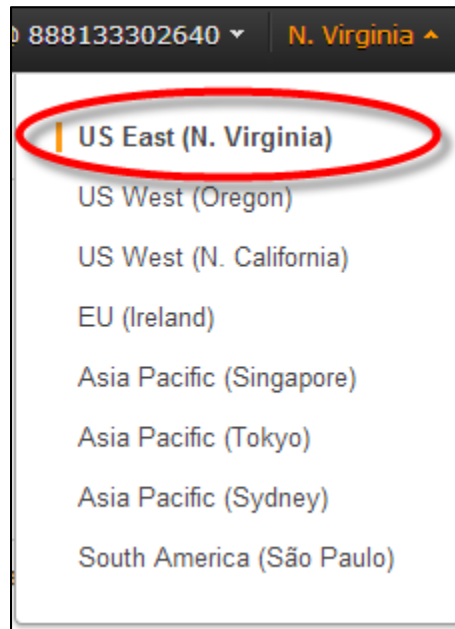
## Select the AWS CloudFormation Service

1. Select "CloudFormation" from the Console Home.



## Confirm your AWS Region

2. Set your AWS Region to **US East (N. Virginia)**



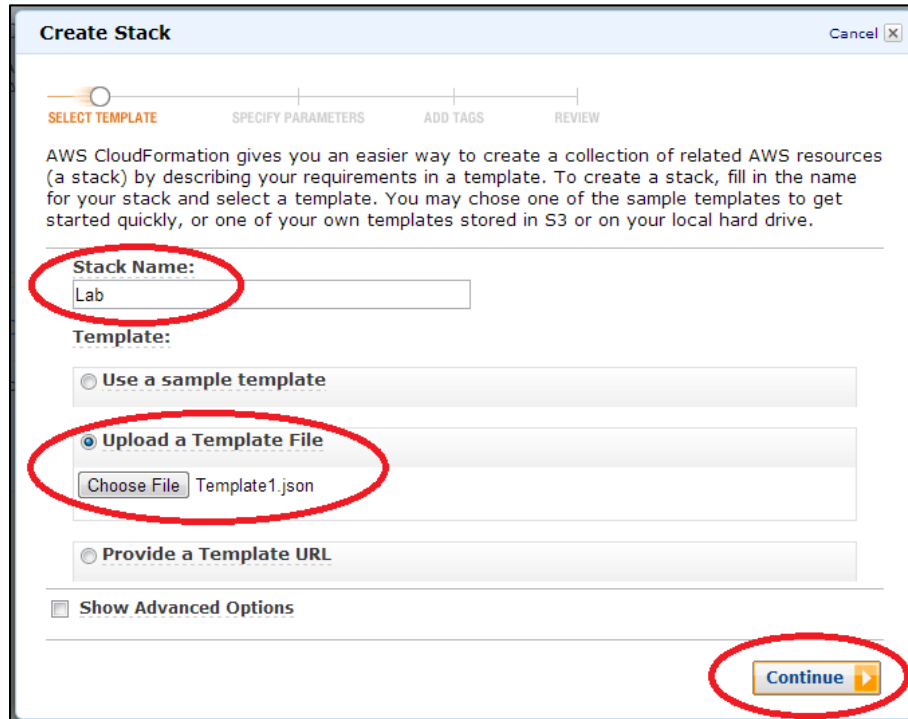
We will in a later evolution of this template be able to use any AWS Region you choose.

## Your First AWS CloudFormation Template

1. Click on "Create New Stack".



2. Give your stack a name, choose the sample file created above, and click Continue.

A screenshot of the "Create Stack" dialog box in the AWS Management Console. The dialog has a title bar with "Create Stack" and a "Cancel" button. Below the title bar is a progress bar with four steps: "SELECT TEMPLATE" (highlighted with an orange circle), "SPECIFY PARAMETERS", "ADD TAGS", and "REVIEW". The main text explains that AWS CloudFormation allows creating a stack of resources by describing requirements in a template. There are three input sections: "Stack Name:" with a text box containing "Lab"; "Template:" with three radio buttons: "Use a sample template", "Upload a Template File" (selected), and "Provide a Template URL". The "Upload a Template File" section has a "Choose File" button and the filename "Template1.json". At the bottom, there is a checkbox for "Show Advanced Options" and a "Continue" button with a right-pointing orange arrow icon. Red circles are drawn around the "Stack Name" text box, the "Upload a Template File" radio button, the "Choose File" button and filename, and the "Continue" button.

3. The next screen allows you to tag your AWS CloudFormation resources. We'll skip this step for now, and ASW CloudFormation will automatically tag resources that support tagging with stack name and id, so click Continue.

**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"/>	<input type="text"/>	<span>✖</span>

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

[< Back](#) **Continue**

- The final screen allows you to verify AWS CloudFormation settings before clicking Continue to create the stack.

**Create Stack** Cancel

SELECT TEMPLATE SPECIFY PARAMETERS ADD TAGS **REVIEW**

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

**Stack Information** Edit Stack

**Stack Name:** Lab

**Stack Description:** Template to launch an Amazon Linux instance.

**Template:** <https://cf-templates-1g3633by4f9n3-us-east-1.s3.amazonaws.com/2013109oRe-Template1.json>

**IAM Acknowledgement:** false

**Estimated Cost:** Cost

**Notification** Edit Notification

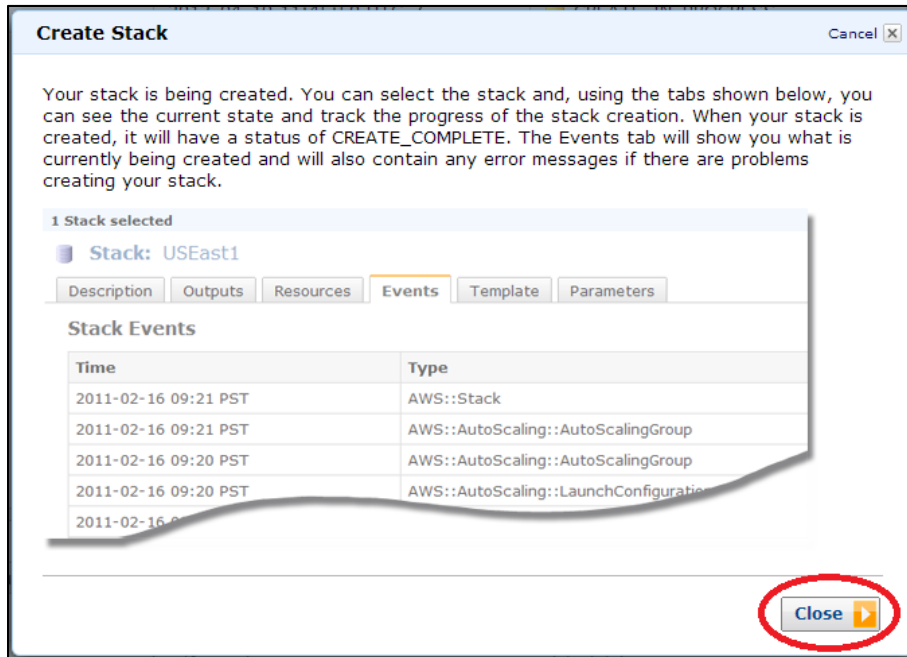
**Notification:** none

**Creation Timeout (minutes):** none

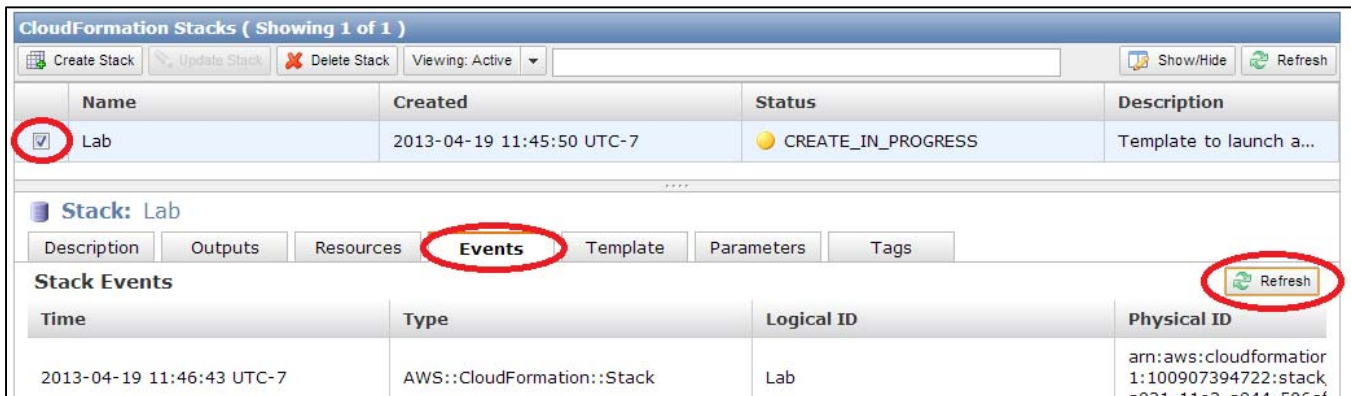
**Rollback on Failure:** false

[< Back](#) **Continue**

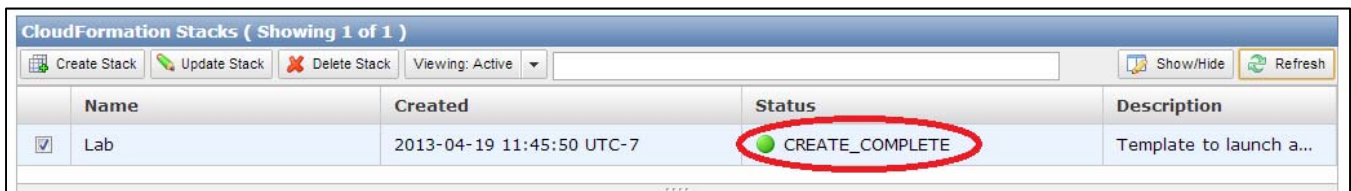
- The stack is launching. Click Close.



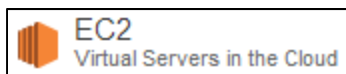
- You can watch the progress of stack creation by selecting the stack, clicking on the events tab, and refreshing occasionally.



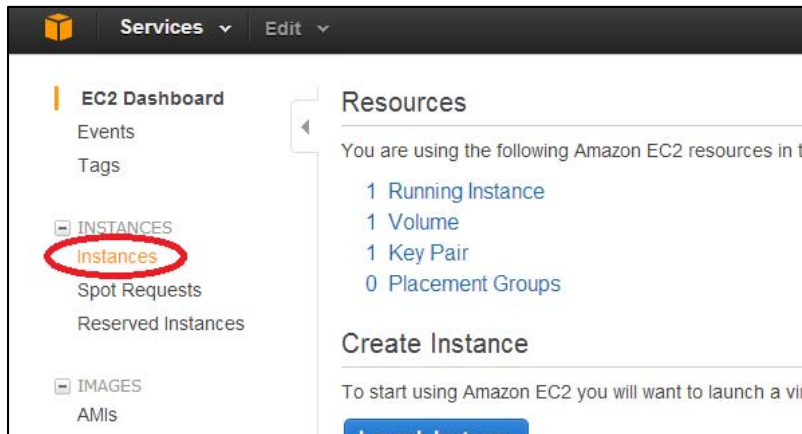
When AWS CloudFormation is finished creating the stack, the status will show CREATE\_COMPLETE.



- Now you can inspect the instance created by AWS CloudFormation by switching the Management Console to the EC2 Service.



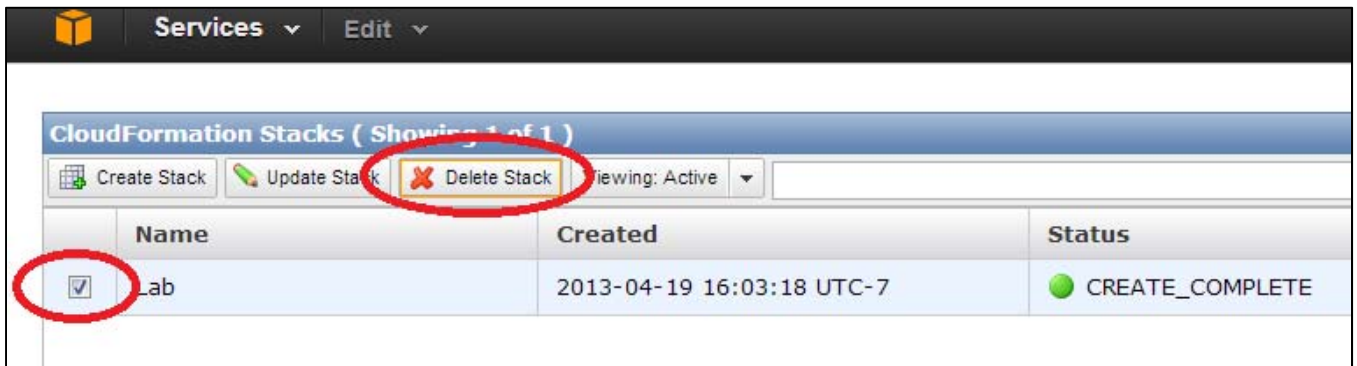
8. Click on Instances to see the new instance created by AWS CloudFormation.



AWS CloudFormation launched one instance, but it is not very useful in its current state. For example, since we did not specify an EC2 Key Pair there is no way to log in to the server.

Viewing: All Instances   All Instance Types   Search									
	Name	Instance	AMI ID	Root Device	Type	State	Status Checks	Alarm Status	Monitor
<input checked="" type="checkbox"/>	empty	i-e956ec8b	ami-3275ee5b	ebs	m1.small	running	2/2 checks passed	none	basic

9. Switch the Management Console back to the AWS CloudFormation service, select the stack, and choose Delete Stack. This will terminate the server and any other stack resources.

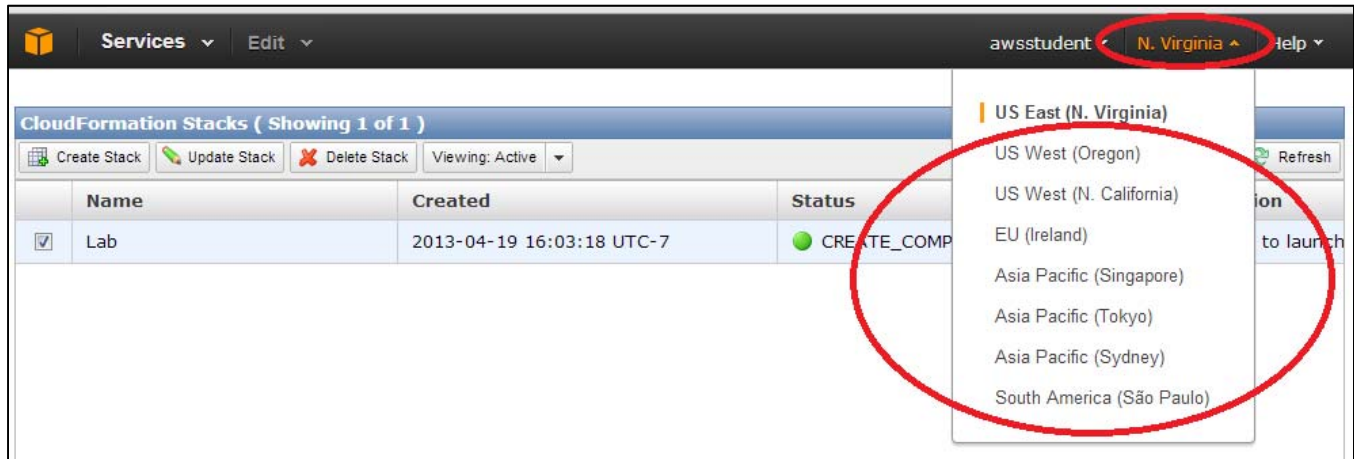


## Your Second AWS CloudFormation Template

The first template launched an instance in the US East region and there was no way to log in to that instance afterwards. In this exercise, you will improve the template to allow its use in any region, allow you to log in to the instance using SSH in order to start a web server, and connect to the web server in a browser.

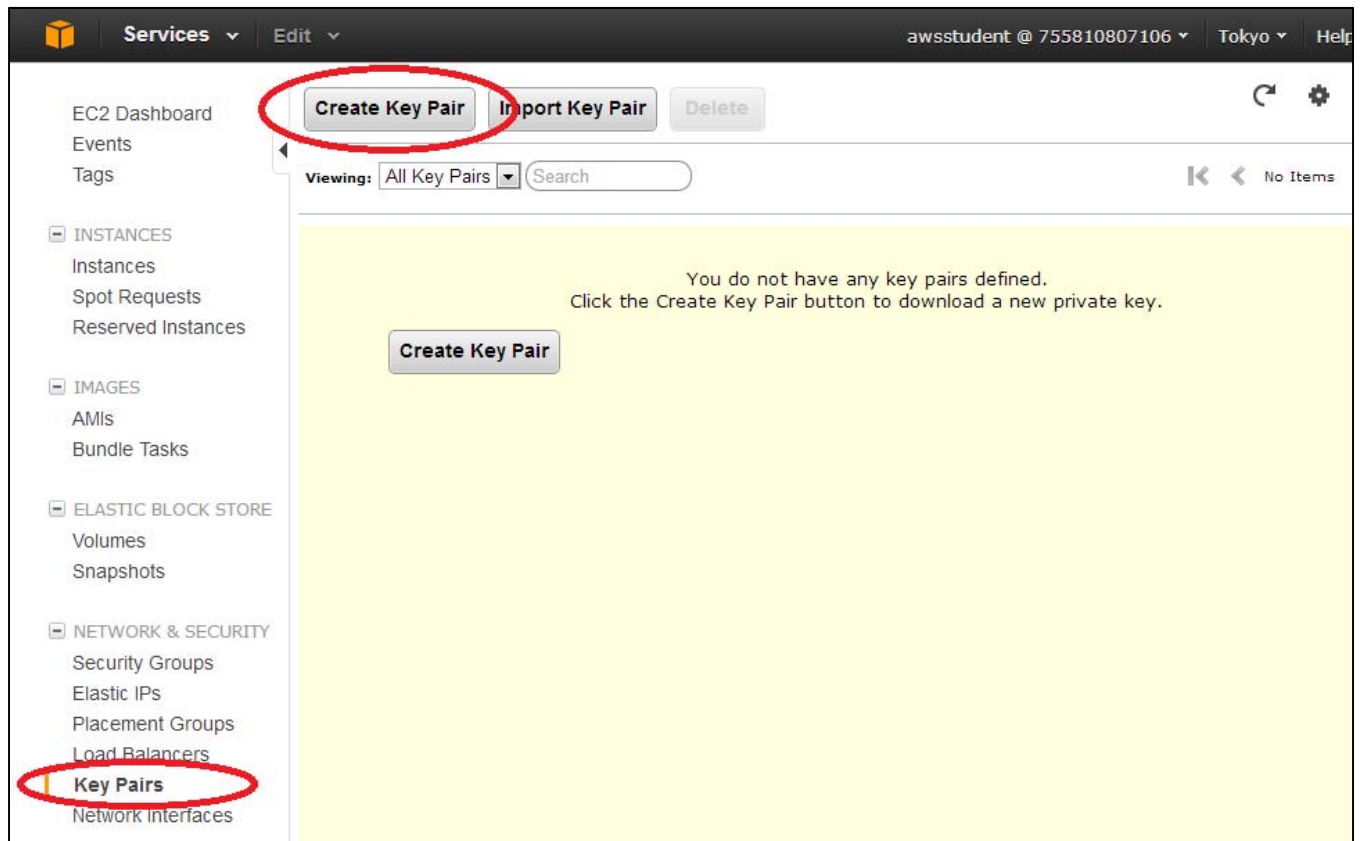
### Switch Regions

1. This exercise will adapt the template for use in any regions, so switch the Management Console into a different region than US East. Choose the one nearest you!

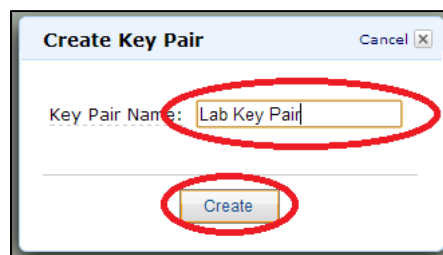


### Create a Key Pair

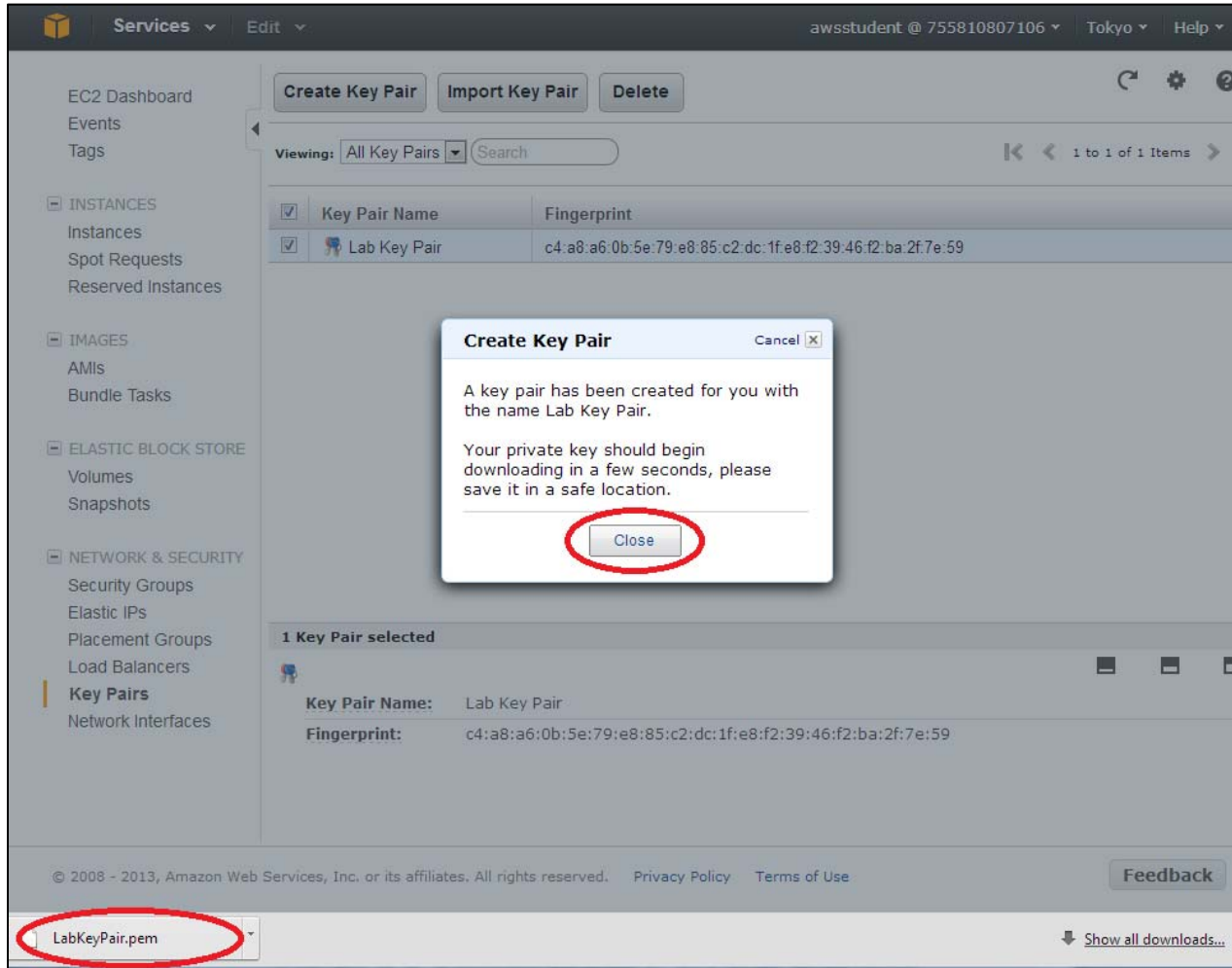
2. Amazon Linux is configured to allow logging in with an EC2 Key Pair instead of plain text passwords. To create a key pair for use in this exercise, switch the Management Console to the EC2 service and click Key Pairs, then Create Key Pair.



3. Name the key pair "Lab Key Pair" and click Create.



- Click Close in the dialog. Your browser will download your half of the key pair. Because AWS does not keep the private key, it's critical that you know where your browser saves the file. You may be prompted for a location, or your browser may automatically download the file to your Downloads folder or some other location depending on your browser settings.



## Parameters

Parameters are values that you supply when you launch a stack. You can specify default values in the template.

Note: You can download the completed file at the end of this section.

We will set up Instance Type as a parameter, and make m1.small the default. Note that there are many more instance types than listed here, but we want to restrict the choices to a short list that is appropriate for our scenario.

We will also add in the KeyPair so that we can successfully log in to the instance. Of course, no default is likely to exist for this parameter, because each user will likely generate their own. (Ignoring team key pairs, etc...)

Don't forget that the final input parameter does not have a comma after it; however all others need one.



5. Add the following Parameters section to your template.

```
"Parameters" :
{
  "InstanceType" :
  {
    "Type" : "String",
    "Default" : "m1.small",
    "AllowedValues" : ["m1.small", "m1.large", "m1.xlarge"],
    "Description" : "Enter m1.small, m1.large, or m1.xlarge. Default is
m1.small. "
  },
  "KeyName" :
  {
    "Type" : "String",
    "Description" : "Enter the key pair name that you want associated with this
instance. Note: Name is required"
  }
}
```

6. Adjust the existing Amazon EC2 Instance resource to add the KeyName property and use the Ref keyword to refer to the user-specified parameter values for KeyName and InstanceType.

```
"Resources" :
{
  "Ec2Instance" :
  {
    "Type" : "AWS::EC2::Instance",
    "Properties" :
    {
      "ImageId" : "ami-3275ee5b",
      "InstanceType" : { "Ref" : "InstanceType" },
      "KeyName" : { "Ref" : "KeyName" }
    }
  }
}
```

### Add a Security Group

Without a security group, the instance will launch into a special group named "Default", but we want to open ports 22 and 80 for SSH and HTTP access.

7. Add the following SecurityGroup definition to the Resources section of your template.

```
"WebSecurityGroup":
{
  "Type": "AWS::EC2::SecurityGroup",
  "Properties":
  {
    "GroupDescription": "Enable tcp access for Web and SSH traffic
from outside",
    "SecurityGroupIngress": [
      {
        "IpProtocol": "tcp",
        "FromPort": "80",
        "ToPort": "80",
        "CidrIp": "0.0.0.0/0"
      }
    ]
  }
}
```

```

    },
    {
        "IpProtocol": "tcp",
        "FromPort": "22",
        "ToPort": "22",
        "CidrIp": "0.0.0.0/0"
    }
]
}

```

8. Adjust the existing EC2 Instance resource definition to utilize the new Security Group.

```

"Ec2Instance" :
{
    "Type" : "AWS::EC2::Instance",
    "Properties" :
    {
        "ImageId" : "ami-3275ee5b",
        "InstanceType" : { "Ref" : "InstanceType" },
        "KeyName" : { "Ref" : "KeyName" },
        "SecurityGroups" : [ { "Ref" : "WebSecurityGroup" } ]
    }
}

```

## Outputs

Upon successful creation of a AWS CloudFormation stack, Outputs can be used to expose useful information about the stack. Often it's the DNS name associated with a just-launched instance, along with the instance ID and other relevant information.

9. Add the following Outputs section as a top-level object in your template.

```

"Outputs" :
{
    "InstanceId" :
    {
        "Description" : "Instance ID of the Web Server",
        "Value" : { "Ref" : "Ec2Instance" }
    },
    "AZ" :
    {
        "Description" : "Instances is running in Availability Zone ",
        "Value" : { "Fn::GetAtt" : ["Ec2Instance", "AvailabilityZone"] }
    },
    "PublicIP" :
    {
        "Description" : "Public IP",
        "Value" : { "Fn::GetAtt" : ["Ec2Instance", "PublicIp"] }
    },
    "PublicDNS" :
    {
        "Description" : "Instance Public DNS Name",
        "Value" : { "Fn::GetAtt" : ["Ec2Instance", "PublicDnsName"] }
    }
}

```

The example above introduces a new concept: executable code. `Fn::GetAtt`. We can't know these values before the stack launches, so this is how we retrieve them.

## Mappings

The final object, Mappings, makes templates work across regions. Many resource names such as AMI names and key pairs are unique to a single region. Mappings are lookup tables.

This example is for a 64-bit Amazon Linux AMI:

```
"Mappings" :
{
  "RegionMap" :
  {
    "us-east-1"      : { "AMI" : "ami-3275ee5b" },
    "us-west-1"      : { "AMI" : "ami-66d1fc23" },
    "us-west-2"      : { "AMI" : "ami-ecbe2adc" },
    "eu-west-1"      : { "AMI" : "ami-44939930" },
    "sa-east-1"      : { "AMI" : "ami-dd6bb0c0" },
    "ap-southeast-1" : { "AMI" : "ami-aa9ed2f8" },
    "ap-southeast-2" : { "AMI" : "ami-363eaf0c" },
    "ap-northeast-1" : { "AMI" : "ami-173fbf16" }
  }
},
```

- Adjust the existing Amazon EC2 Instance resource definition to replace the hard-coded ImageId with the AMI value from the RegionMap corresponding to the current region.

```
"Ec2Instance" :
{
  "Type" : "AWS::EC2::Instance",
  "Properties" :
  {
    "ImageId" : { "Fn::FindInMap" : [ "RegionMap", { "Ref" : "AWS::Region" }, "AMI" ] },
    "InstanceType" : { "Ref" : "InstanceType" },
    "KeyName" : { "Ref" : "KeyName" },
    "SecurityGroups" : [ { "Ref" : "WebSecurityGroup" } ]
  }
}
```

Download and use the completed template:

<http://us-east-1-aws-training.s3.amazonaws.com/self-paced-lab-14/Template2.json>

## Launch

Using the AWS Management Console, create a stack from this new template in the region you selected before creating the Key Pair. Since the new template has Parameters, you will see a screen in the wizard allowing you to provide parameter values.

- Enter the name of the Key Pair ("Lab Key Pair") and click Continue, then proceed as before.

**Create Stack** Cancel

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

**Stack Description:** Template to launch an Amazon Linux instance.

**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.

**InstanceType**   
Enter m1.small, m1.large, or m1.xlarge. Default is m1.small.

**KeyName**   
Enter the key pair name that you want associated with this instance. Note: Name is required

[< Back](#) [Continue](#)

### Inspect Output

12. Once the Management Console shows CREATE\_COMPLETE, click on the Outputs tab to view the Stack Outputs.

**Stack: Lab**

Description **Outputs** Resources Events Template Parameters Tags Refresh

**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-c76ea1c5	Instance ID of the Web Server
AZ	ap-northeast-1a	Instances is running in Availability Zone
PublicIP	54.248.145.97	Public IP
PublicDNS	ec2-54-248-145-97.ap-northeast-1.compute.amazonaws.com	Instance Public DNS Name

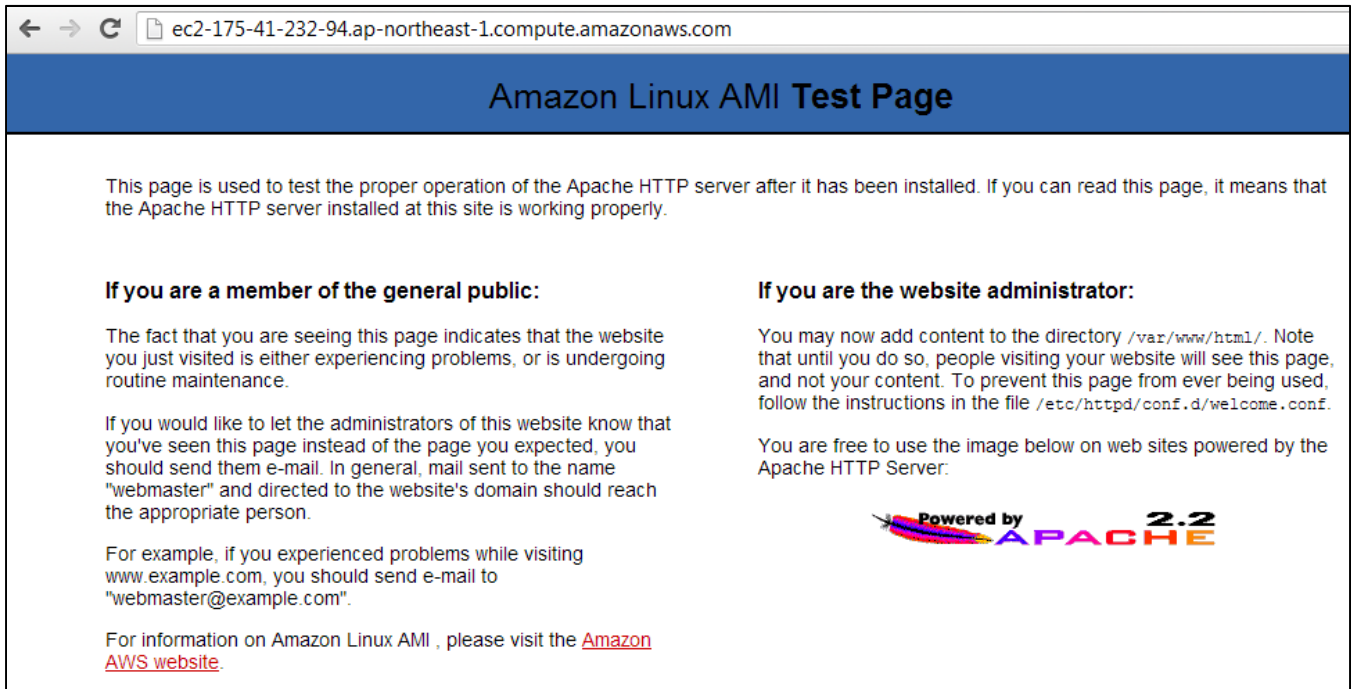
## Log into the Server

See Appendix B for instructions on how to log in to your instance via SSH. Use the LabKeyPair.pem file already downloaded by your browser and the PublicDNS host name provided in the CloudFormation Outputs.

13. Install and start a web server by issuing the following commands:

```
sudo yum install -y httpd && sudo service httpd start
```

14. Using a web browser to navigate to the PublicDNS host name, you should see the Amazon Linux AMI Test Page, shown below.



15. It is a good practice to delete a stack when it is no longer needed, so delete this stack before moving on to the next exercise.

## Your Third AWS CloudFormation Template

Manual configuration procedures such as those from the last exercise are not scalable especially when AWS CloudFormation is used to launch many instances at once. Since it would be time consuming to log in to each instance individually, in this exercise we will configure AWS CloudFormation to automatically bootstrap the same configuration without manual intervention.

AWS CloudFormation provides a number of features that can help with automated instance configuration. AWS CloudFormation can:

- Perform updates and install packages
- Write files such as scripts and configuration files to disk and chmod them as needed.
- Run a series of steps in order, e.g. to install dependencies before installing a package
- Pass parameters from the template that act as input for scripts
- Use other automation packages such as Chef or Puppet. (This feature is beyond the scope of this lab exercise)

After the instance boots, but before SSH starts, a program named cloud-init runs to perform the tasks that you instruct it to. Details about cloud-init can be found in Appendix A. We will pass configuration information to cloud-init using the UserData property and also utilize a WaitCondition so AWS CloudFormation pauses after launching the instance until the configuration scripts are completed.

### Define a Script in User Data

User Data is passed to the instance on startup. It can be arbitrary data that your instance knows how to process, but if the User Data starts with `#!` it is automatically executed as a script.

The following template modifications to the existing EC2 Instance configuration replicate the previous exercise's manual commands to install and start the web server.

```
"Ec2Instance" :
{
  "Type" : "AWS::EC2::Instance",
  "Properties" :
  {
    "ImageId" : { "Fn::FindInMap" : [ "RegionMap", { "Ref" : "AWS::Region" } ],
    "AMI" ] },
    "InstanceType" : { "Ref" : "InstanceType" },
    "KeyName" : { "Ref" : "KeyName" },
    "SecurityGroups" : [ { "Ref" : "WebSecurityGroup" } ],
    "UserData" :
    {
      "Fn::Base64" : { "Fn::Join" : [ "", [
        "#!/bin/bash -v\n",
        "# Make certain that cfn itself is up to date\n",
        "yum update -y aws-cfn-bootstrap\n",
        "\n",
        "# Helper function\n",
        "function error_exit\n",
        "{\n",
```

```

        " /opt/aws/bin/cfn-signal -e 1 -r \"\$1\" ' ", { "Ref" : "WaitHandle" },
        "'\n",
        " exit 1\n",
        "}\n",

        "# install and start httpd \n",
        "yum install -y httpd || error_exit 'Failed to install Apache' \n",
        "/sbin/service httpd start || error_exit 'Failed to start Apache' \n",

        "# We got here without issues (except as signaled), so signal success\n",
        "/opt/aws/bin/cfn-signal -e 0 -r \"User data script complete\" ' ", {
"Ref" : "WaitHandle" }, "'\n"
    ]}]
  }
},

```

### Set up a Wait Condition

A WaitCondition tells the stack to wait for some future signal. Those signals can be above seen in the success and failure calls to cfn-signal which reference a WWaitConditionHandle called WaitHandle.

In order for this to work, the following resources must be added to the template.

```

"WaitHandle" : {
  "Type" : "AWS::CloudFormation::WaitConditionHandle"
},

"WaitCondition" : {
  "Type" : "AWS::CloudFormation::WaitCondition",
  "DependsOn" : "Ec2Instance",
  "Properties" : {
    "Handle" : { "Ref" : "WaitHandle" },
    "Timeout" : "300"
  }
}

```

The Timeout setting is the number of seconds before the Wait Condition assumes that something went wrong and reports an error. In this case after 5 minutes if no OK signal is received, AWS CloudFormation will report that it failed to create the stack.

### Launch

1. Download and use the completed template:

<http://us-east-1-aws-training.s3.amazonaws.com/self-paced-lab-14/Template3.json>.

Launch this template using the same process from the previous exercise.

Try connecting to the new PublicDNS host name from this stack's Outputs. You should see the same test page without ever having logged in to configure the instance.

### Log Files

AWS CloudFormation and cloud-init create log files which can help diagnose issues when you begin to work with more advanced scenarios.

The two log files of interest are `cfn-init.log` and `cloud-init.log` located in the `/var/log` directory. To examine these log files, SSH to the new instance and execute the following command:

```
cat /var/log/cfn-init.log /var/log/cloud-init.log | more
```

## What's Next?

There are so many other things that AWS CloudFormation can do! For example, you can pre-install software, files, and even download and automatically inflate .gz and .zip archives. The online docs at <http://aws.amazon.com/cloudformation> have a wealth of examples.

You can also launch the stack using a temporary IAM user. This is one way to restrict what the server is capable of doing, among other benefits.

This is the end of the interactive lab; however we invite you to continue reading Appendix A to learn more about how AWS CloudFormation and cloud-init work under the hood.

## End Lab

Sign-out of the AWS Management Console.

Click the End Lab button in *qwikLAB*<sup>™</sup>.



Give the lab a thumbs-up/down, or enter a comment and click Submit

A feedback form interface. At the top, there are three icons: a thumbs-up, a thumbs-down, and a square box, all enclosed in a red oval. Below these icons is a large, empty rectangular text area for comments. To the left of this area is the label "Comment". At the bottom right of the form, there is a "Submit" button, also enclosed in a red oval.

Any errors in the lab instructions can be reported to [aws-course-feedback@amazon.com](mailto:aws-course-feedback@amazon.com).



## Appendix A - How Do Scripts, Packages, and Templates Work Together?

AWS CloudFormation is an orchestrated experience that combines scripts with a service named Cloudinit that runs on the instance. Most Amazon Linux, Ubuntu, and Windows AMIs have this software already installed to run as a service. From the point of view of AWS CloudFormation, both Linux and Windows behave exactly the same; except that (of course) Linux can't install Windows software and vice-versa.

### Bootstrap Applications from a AWS CloudFormation Template

A three-way combination allows AWS CloudFormation and an AMI to interact with each other.

1. You can specify helper scripts from the AWS CloudFormation template.
2. There is a package known as *cloudinit* that runs on the AMI
3. Cloudinit runs at startup as a rc script: e.g. *K25cloud-init* in the *rc6.d* directory.

### Helper Scripts

AWS CloudFormation provides the following helpers to allow you to deploy your application code or application and OS configuration at the time you launch your EC2 instances:

**cfn-init:** Used to retrieve and interpret the resource metadata, installing packages, creating files and starting services.

**cfn-signal:** A simple wrapper to signal an AWS CloudFormation WaitCondition allowing you to synchronize other resources in the stack with the application, once it's ready.

**cfn-get-metadata:** A wrapper script making it easy to retrieve either all metadata defined for a resource or path to a specific key or sub tree of the resource metadata.

**cfn-hup:** A daemon to check for updates to metadata and execute custom hooks when changes are detected.

These scripts are installed by default on Amazon Linux AMIs in */opt/aws/bin*. They are also available in the Amazon Linux AMI yum repository, and via RPM for other Linux/Unix distributions. The scripts are also pre-installed on most Microsoft Windows AMIs, along with Python for Windows.

By the way, AWS CloudFormation documentation mentions that these scripts can also run on your local computer. We want to be clear that in production the scripts run on the instance, not on your local machine. The only reason to run them locally is to test something – and even then it probably makes more sense to test on an AMI.

### Cloudinit

Cloudinit is an open-source package written by Canonical (the authors of Ubuntu) that runs under Linux to facilitate early startup scripts such as those in an AWS CloudFormation template. Of course your Linux distro needs this installed in order for the feature to work. Amazon Linux, and most Ubuntu AMIs, and most recent Windows AMIs have the equivalent to *cloudinit* installed. You'll need to check other Linux distributions, and install if needed. We are not going to discuss how to install the package here.

Cloud-init, cloudinit, and cloudinit.d are three ways of referring to exactly the same thing: *cloudinit*.

### Cloud-init Runs as a RC Package

*Cloudinit* is similar in name, and in concept, to *init.d*. It's a Python package that, at least in Amazon Linux, called from a script in the *rc.6* directory, and that usually runs just before SSH starts.

Here's the RC script, which is named *K25cloud-init* in Amazon Linux:

```
#!/bin/bash
#
# Init file for cloud-init
#
# chkconfig: 2345 25 25
# description: cloud-init is the distribution-agnostic package that handles early
#               initialization of a cloud instance.
#
# Some of the things it configures are:
#   - setting a default locale
#   - setting hostname
#   - generate ssh private keys
#   - adding ssh keys to user's .ssh/authorized_keys so they can log in
#   - setting up ephemeral mount points
#   - preparing package repositories
# and performs a variety of at-boot customization actions based on user-data

# config: /etc/sysconfig/cloudinit

# source function library
. /etc/rc.d/init.d/functions

RETVAL=0

LOGFILE=/var/log/cloud-init.log

start() {
    echo -n "Running cloud-init"
    if [ -f /etc/sysconfig/cloudinit ]; then
        . /etc/sysconfig/cloudinit
    else
        echo "Unable to load /etc/sysconfig/cloudinit"
        failure
    fi
    echo

    if [ -x /usr/bin/cloud-init ]; then
        echo -n "cloud-init:  initialization"
        /usr/bin/cloud-init start-local >>$LOGFILE
        /usr/bin/cloud-init start >>$LOGFILE && success || failure
        echo
    fi

    if [ "${CONFIG_LOCALE}" = "yes" ]; then
        echo -n "cloud-init:  locale"
        /usr/bin/cloud-init-cfg locale >>$LOGFILE && success || failure
        echo
    fi

    if [ "${CONFIG_SSH}" = "yes" ]; then
        echo -n "cloud-init:  ssh"
        /usr/bin/cloud-init-cfg ssh >>$LOGFILE && success || failure
        echo
    fi

    if [ "${CONFIG_MOUNTS}" = "yes" ]; then
```

```

        echo -n $"cloud-init:  mounts"
        /usr/bin/cloud-init-cfg mounts >>$LOGFILE && success || failure
        echo
    fi
    if [ "${PACKAGE_SETUP}" = "yes" ]; then
        echo -n $"cloud-init:  package-setup"
        /usr/bin/cloud-init-cfg package-setup >>$LOGFILE && success || failure
        echo
    fi
    if [ "${RUNCMD}" = "yes" ]; then
        echo -n $"cloud-init:  runcmd"
        /usr/bin/cloud-init-cfg runcmd >>$LOGFILE && success || failure
        echo
    fi
    # Note that user-scripts are run at the end of the boot process (99), not
here
}

stop () {
    # May add cleanup tasks here in the future...for now, no op
    echo -n $"Stopping cloud-init (cleanup): " && success
    echo
}

case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    restart)
        stop
        start
        ;;
    *)
        echo $"Usage: $0 {start|stop|restart}"
        RETVAL=1
esac
exit $RETVAL

```

## Appendix B – SSH to an EC2 Instance

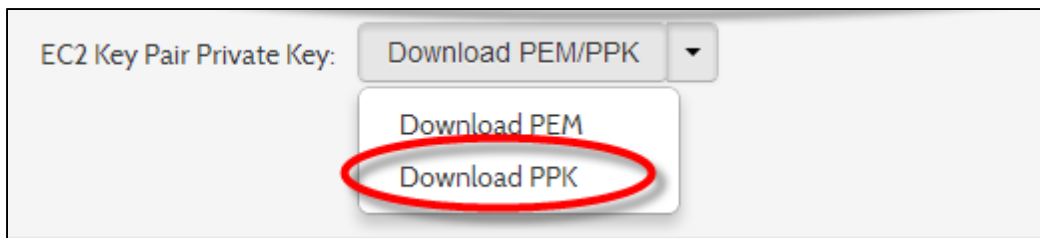
### Connect to your EC2 Instance via SSH (Windows)

#### Download PuTTY

1. Download PuTTY to a location of your choice unless you already have PuTTY.  
<http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe>

#### Download your EC2 Key Pair private key file

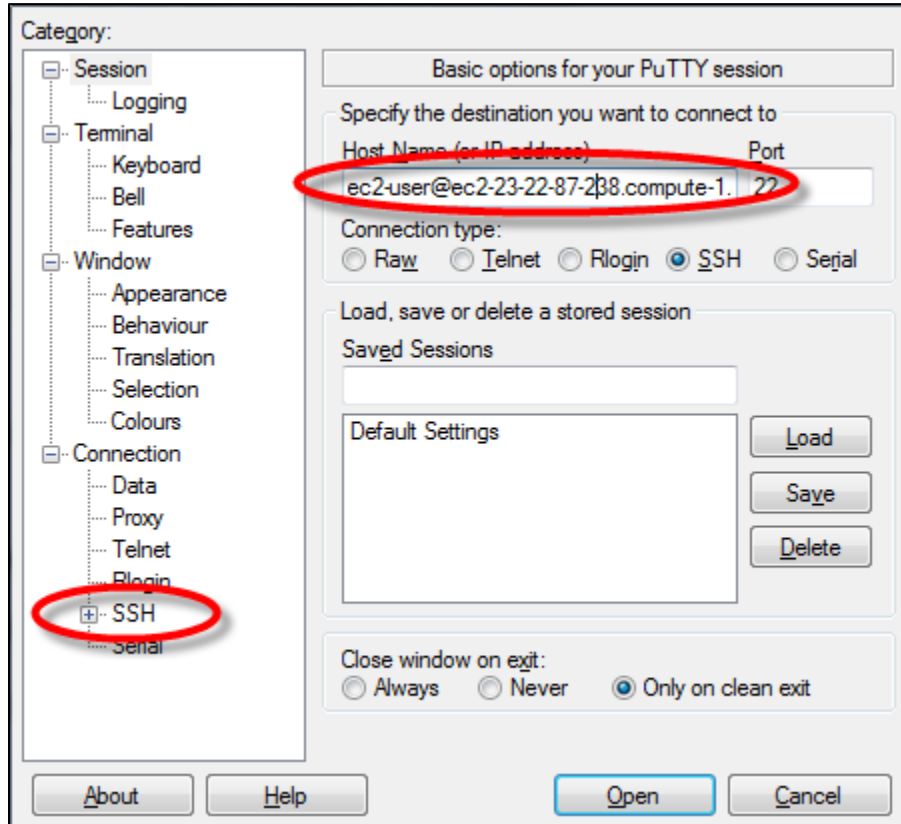
2. Go back to your lab in *qwikLAB*™.
3. Download the *qwikLAB*™ provided EC2 Key Pair private key file in the PuTTY compatible PPK format by clicking on the Download PPK option in the “Download PEM/PPK” drop-down.



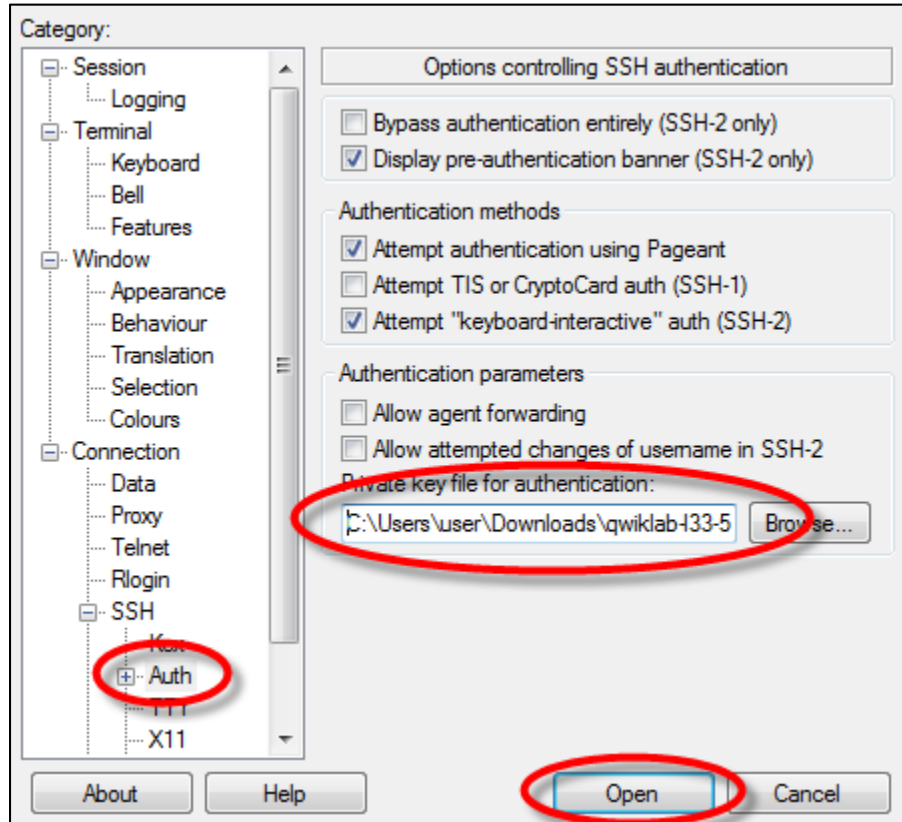
4. Save the file to your Downloads directory (or some other directory of your choice.)

### Connect to the EC2 Instance using SSH and PuTTY.

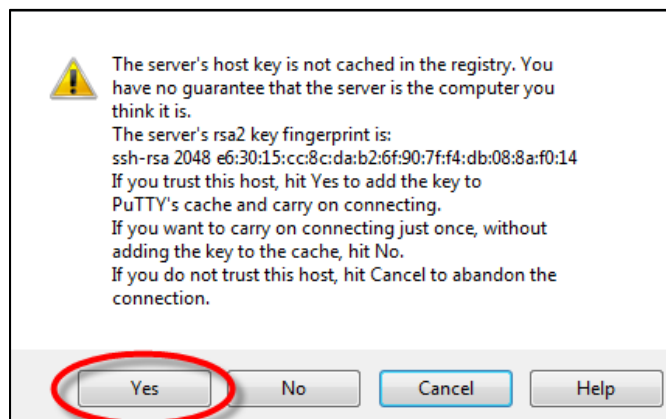
1. Open the putty.exe you downloaded or already had.
2. Enter ec2-user@<your EC2 hostname> into the Host Name input in Putty (Ctrl+v).
3. Expand the SSH category by clicking on it.



4. Select the Auth category by clicking on it (not the + symbol next to it).
5. Click Browse and locate the PPK file (ending in .ppk) in your Downloads directory or whatever other location you chose.
6. Click Open



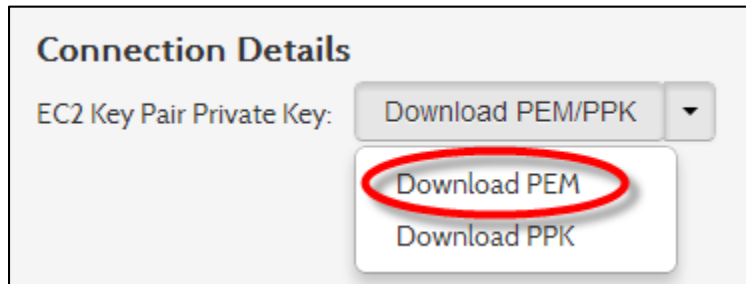
Click Yes when prompted to allow a first connection to this remote SSH server.



## Connect to your EC2 Instance via SSH (OS X and Linux)

### Download your EC2 Key Pair private key file

1. Go back to your lab in *qwikLAB™*.
2. Download the *qwikLAB™* provided EC2 Key Pair private key file in the PEM format by clicking on the Download PEM option in the “Download PEM/PPK” drop-down.



3. Save the file to your Downloads directory (or some other directory of your choice.)

### Connect to the EC2 Instance using the OpenSSH CLI client

1. Open the Terminal application.
2. Enter the below commands substituting the path/filename for the .pem file you downloaded from *qwikLAB™* and pasting `ec2-user@<your EC2 hostname>` to substitute the example below.

```
chmod 600 ~/Downloads/qwiklab-l33-5018.pem  
ssh -i ~/Downloads/qwiklab-l33-5018.pem ec2-user@ec2-23-22-87-238.compute-1.amazonaws.com
```

## End Lab

Sign-out of the AWS Management Console.

Click the End Lab button in *qwikLAB™*.



Give the lab a thumbs-up/down, or enter a comment and click Submit