14 – Working with CloudFormation (for Linux)

Contents

Overview

The purpose of this lab is to demonstrate launching AWS EC2 instances using Amazon CloudFormation. We will "start simple" and then add more features to our template.

This lab focuses on the bootstrapping features specific to Linux and especially to Amazon Linux.

Imagine a “document driven data center.” That is what you are about to create: a self-documented data center where everything is driven via automation. It is 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.

CloudFormation is a powerful way to launch a collection of AWS resources, known as a “stack”; that consists, in our case, of multiple EC2 instances and other resources. One of the reasons that this is so powerful is that if anything fails to launch, CloudFormation can roll the entire stack back: that is, it will tear down every other resource in the stack. Coupled with other components such as Amazon Simple Workflow, Chef, and Puppet, CloudFormation can deal with complicated scenarios, such as servers that need to use a database connection string that will not 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 footnotes, or 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 are a gray area. For example, if you plan to use a database in your environment (you will in this lab), is it an existing one with a known connection string, or do you need to launch one and then determine the connection string? The answer to this will determine whether you 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 easier to modify than to create a template from scratch. It’s also easy to get lost in a long template.

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 CloudFormation Templates

A 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 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 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 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 Amazon CloudFormation template specifications. If you omit this object altogether, 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 CloudFormation stack. CloudFormation requires at least one resource to launch a stack, and each resource is listed as a namespaced type such as AWS::EC2::Instance.

The following example is a complete template which creates an EC2 instance from an AMI in the US East (N. Virginia) region. Notice that we have some code that allows the template to run in any AWS region. We'll describe this code in more detail later.

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "Template to launch an Amazon Linux instance.",
  "Mappings" :
  {
    "RegionMap" : {
      "us-east-1"      : { "AMI" : "ami-35792c5c" },
      "us-west-2"      : { "AMI" : "ami-d03ea1e0" },
      "us-west-1"      : { "AMI" : "ami-687b4f2d" },
      "eu-west-1"      : { "AMI" : "ami-149f7863" },
      "ap-southeast-1" : { "AMI" : "ami-14f2b946" },
      "ap-southeast-2" : { "AMI" : "ami-9f6ec982" },
      "ap-northeast-1" : { "AMI" : "ami-3561fe34" },
      "sa-east-1"      : { "AMI" : "ami-0439e619" }
    }
  },

  "Resources" :
  {
    "Ec2Instance" :
    {
      "Type" : "AWS::EC2::Instance",
      "Properties" :
      {
        "ImageId" : { "Fn::FindInMap" : [ "RegionMap", { "Ref" : "AWS::Region" }, "AMI"
]},
        "InstanceType" : "m1.small"
      }
    },
}
```

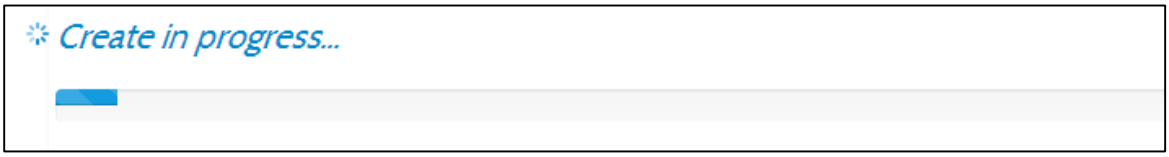
Download and use the completed template:
http://us-east-1-aws-training.s3.amazonaws.com/self-paced-lab-14/Template1.json

Accessing the qwikLAB™ Lab Environment

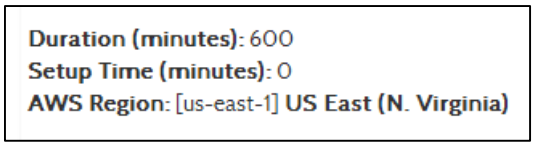
To access your lab environment in *qwikLAB™*:

1) To the right of the lab title, click the **Start Lab** button to launch your *qwikLAB™*. If you are prompted for a token, use the one distributed to you (or the token you purchased).

Note: A status bar shows the progress of the lab environment creation process. The AWS Management Console is accessible during lab resource creation, but your AWS resources may not be fully available until the process is complete.

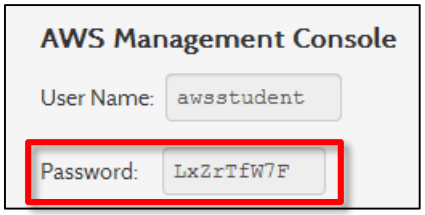


2) On the lab details page, notice the lab properties.
   1. **Duration -** The time the lab will run before automatically shutting down.
   2. **Setup Time -** The estimated time to set up the lab environment.
   3. **AWS Region** - The AWS Region in which the lab resources are created.



Note: The AWS Region for your lab will differ depending on your location and the lab setup.

3) In the AWS Management Console section of the *qwikLAB™* page, copy the **Password** to the clipboard.



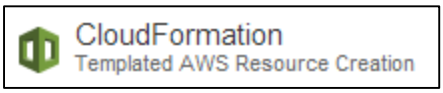
4) Click the **Open Console** button.

5) Log into the AWS Management Console using the following steps.
   1. In the **User Name** field type **awsstudent**.
   2. In the **Password** field, paste the password copied from the lab details page.
   3. Click **Sign in using our secure server**.

Note: The AWS account is automatically generated by *qwikLAB*™. Also, the login credentials for the awsstudent account are provisioned by *qwikLAB*™ using AWS Identity Access Management.

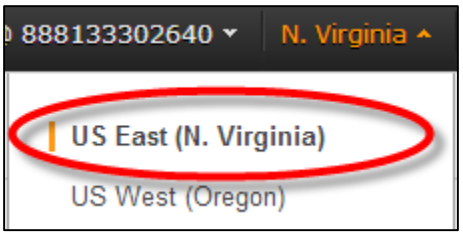
Select the CloudFormation Service

Click **CloudFormation** on the Console home page.



Confirm your AWS Region

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

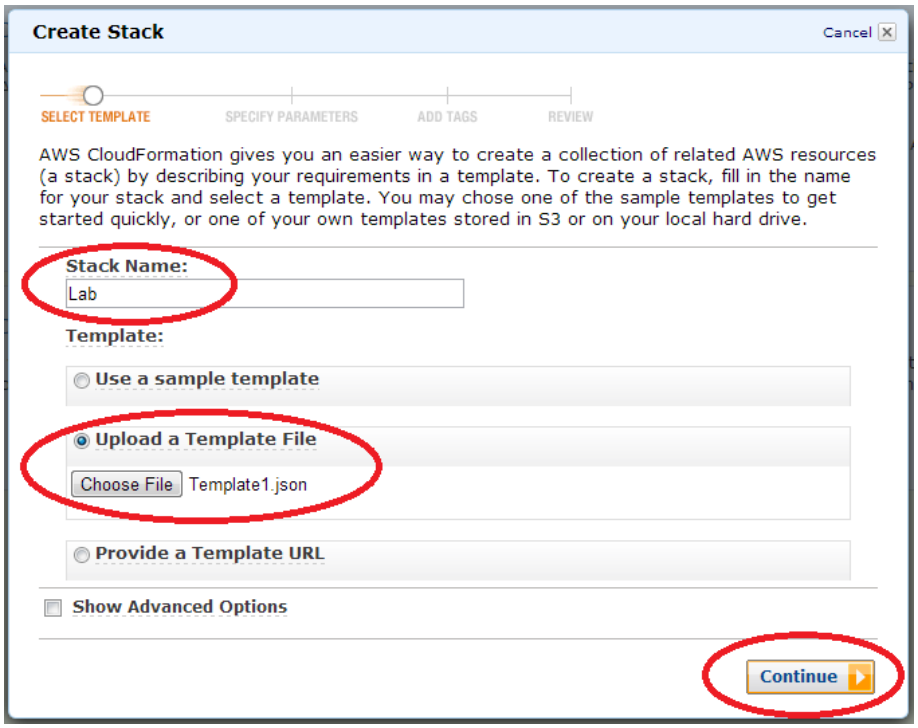


We will (in a later evolution of this template) be able to use any AWS Region.

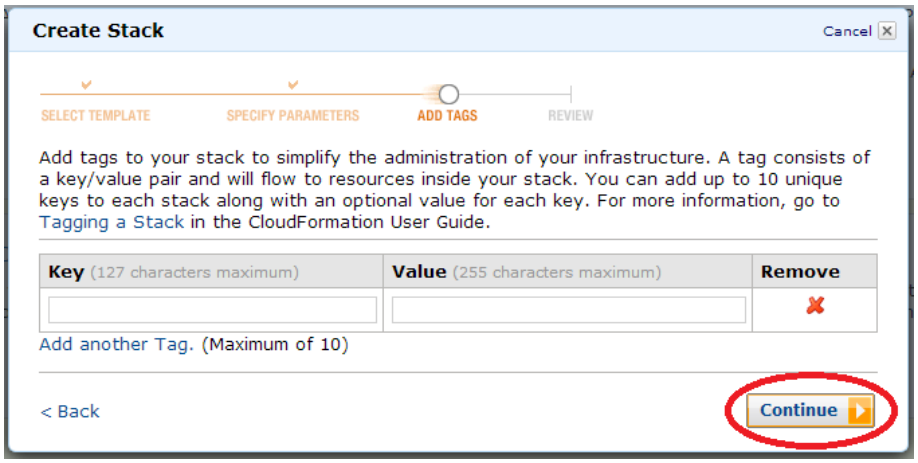
Your First CloudFormation Template

Click **Create New Stack**.

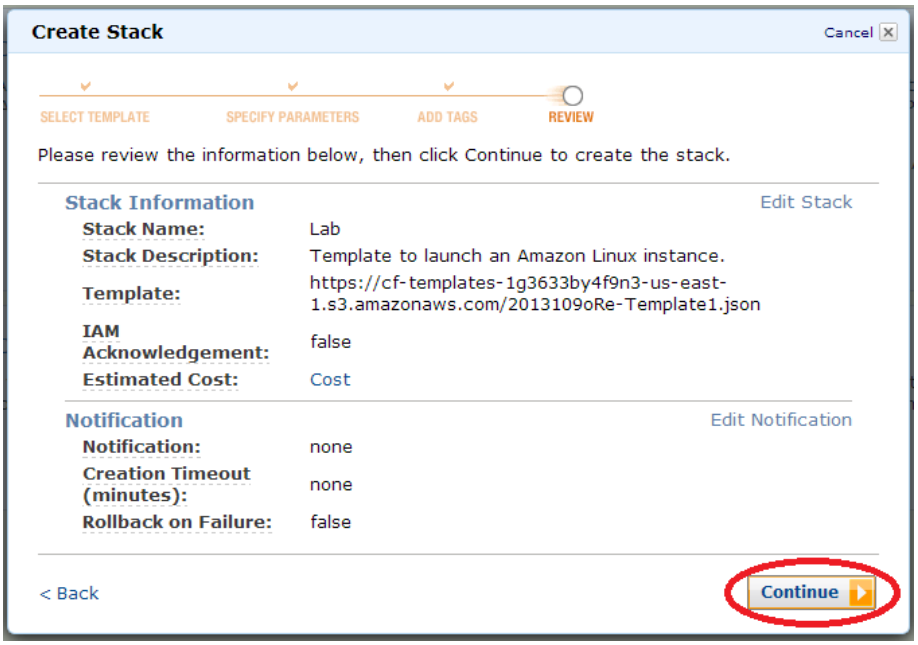
Give your stack a name, choose the sample file created previously, and click **Continue**.



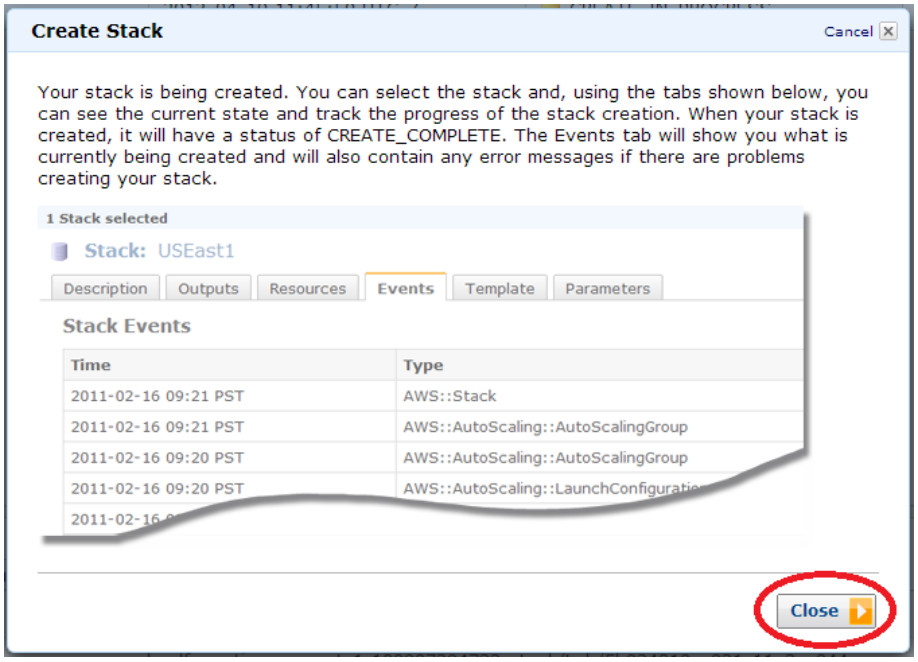
The next screen allows you to tag your CloudFormation resources. Skip this step for now, and CloudFormation will automatically tag resources that support tagging with stack name and id. Click **Continue**.



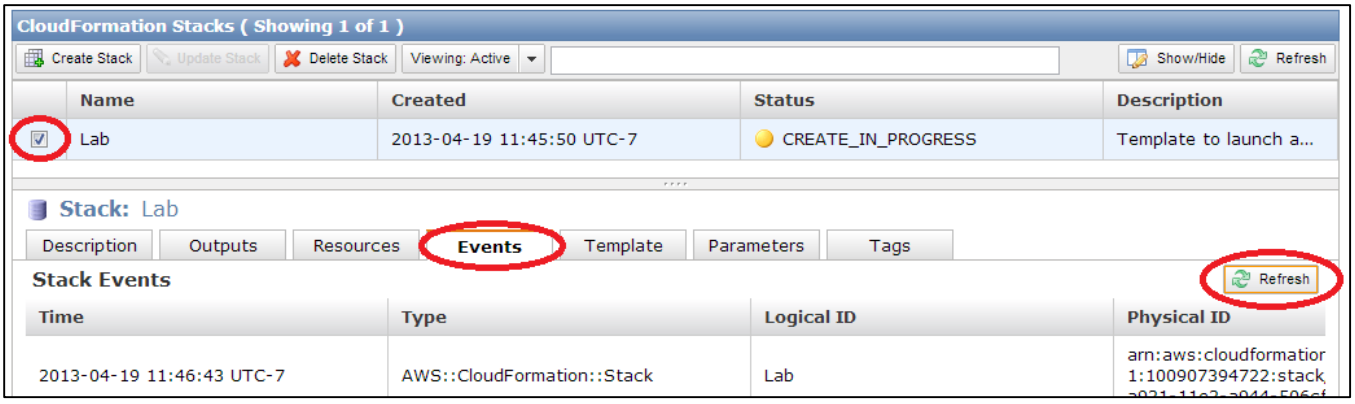
The final screen allows you to verify CloudFormation settings before clicking **Continue** to create the stack.



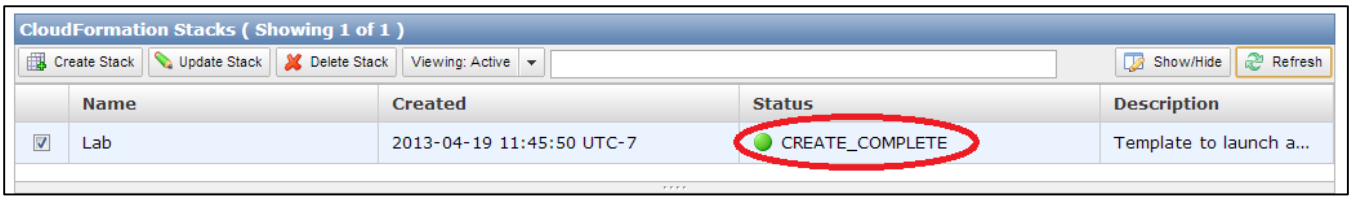
The stack is launching. Click **Close**.



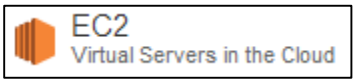
You can watch the progress of stack creation by selecting the stack, clicking the **Events** tab, and refreshing occasionally.



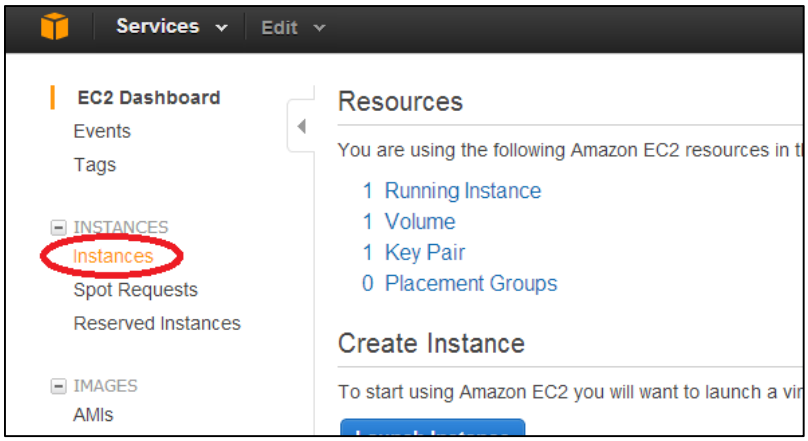
When CloudFormation is finished creating the stack, the status will show CREATE_COMPLETE.



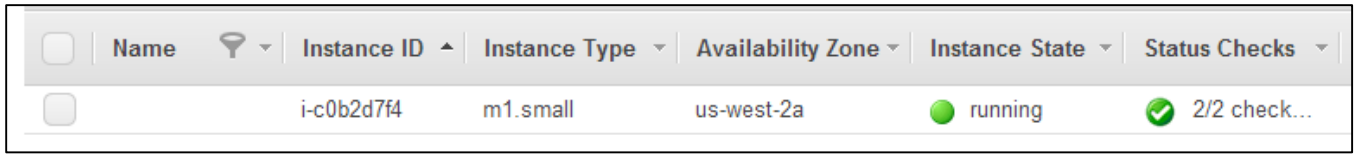
Now you can inspect the instance created by CloudFormation by switching the Management Console to the EC2 Service. Click **EC2** from the console home page or click **Services > All AWS Services > EC2**.



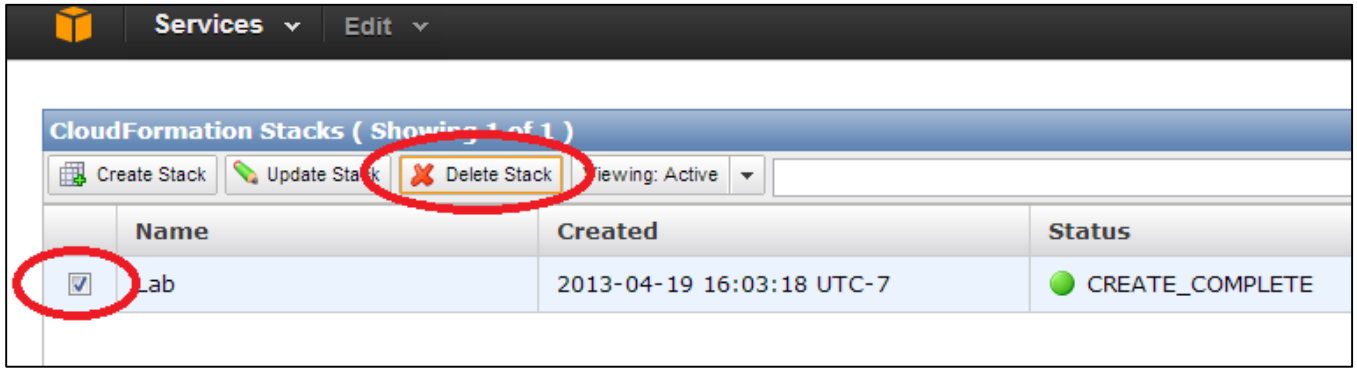
Click **Instances** to see the new instance created by CloudFormation.



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.



Switch to the CloudFormation console, select the stack, and click **Delete Stack**. This will terminate the server and any other stack resources.

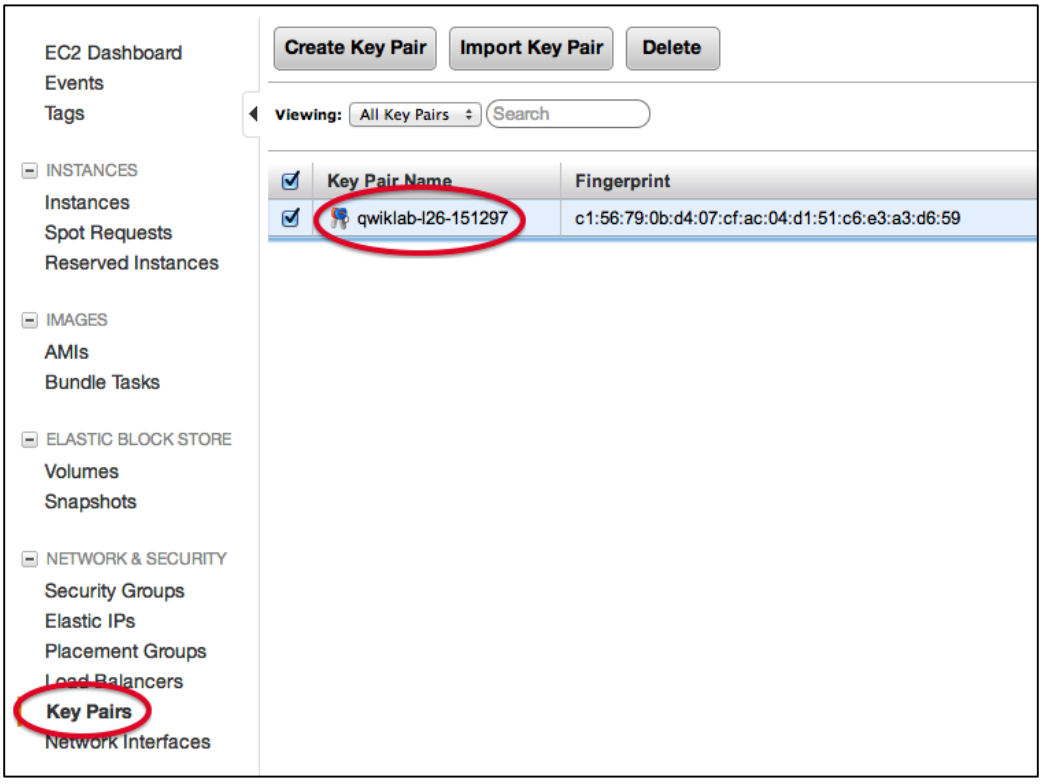


Your Second CloudFormation Template

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.

Employ a Key Pair

qwikLAB™ creates an EC2 Key Pair that you can reference with EC2 Instances that you run, including those run via CloudFormation. The private key can be downloaded from qwikLAB when you want to connect to the Instance that references the Key Pair. You can view the Key Pair Name by switching to the EC2 console and clicking Key Pairs. The name of the Key Pair begins with 'qwiklab'. Note the name for later use.



Parameters

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

Let us 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.

Amazon Linux is configured to allow you to log in with an EC2 Key Pair instead of a password. Let us use one so we can log into the instance via SSH. We will not specify a default, as a user is likely to generate an EC2 Key Pair.

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

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"
    }
}
```

Adjust the existing EC2 Instance resource to add the KeyName property and use the Ref function 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. 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"
          }
        ]
      }
    }
```

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 CloudFormation stack, Outputs can be used to expose useful information about the stack. Often it is the DNS name associated with a newly launched instance, along with the instance ID and other relevant information.

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 cannot 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. Your original template included a Mappings section similar to the following for 64-bit Amazon Linux AMIs:

```
"Mappings" : {
    "RegionMap" : {
      "us-east-1"      : { "AMI" : "ami-35792c5c" },
      "us-west-1"      : { "AMI" : "ami-687b4f2d" },
      "us-west-2"      : { "AMI" : "ami-d03ea1e0" },
      "eu-west-1"      : { "AMI" : "ami-149f7863" },
      "sa-east-1"      : { "AMI" : "ami-9f6ec982" },
      "ap-southeast-1" : { "AMI" : "ami-14f2b946" },
      "ap-southeast-2" : { "AMI" : "ami-a148d59b" },
      "ap-northeast-1" : { "AMI" : "ami-3561fe34" }
    }
},
```

Your EC2 instance access this region map through with the highlighted code.

```
            "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:

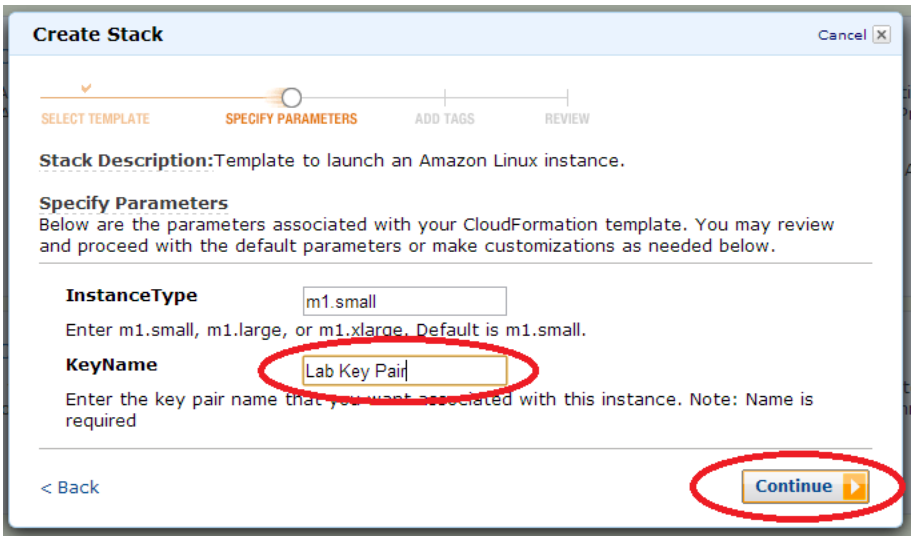
http://us-east-1-aws-training.s3.amazonaws.com/self-paced-lab-14/Template2.json

Note this template uses a Fn::Join function to select an AMI for whatever region you run it in.

Launch the Stack

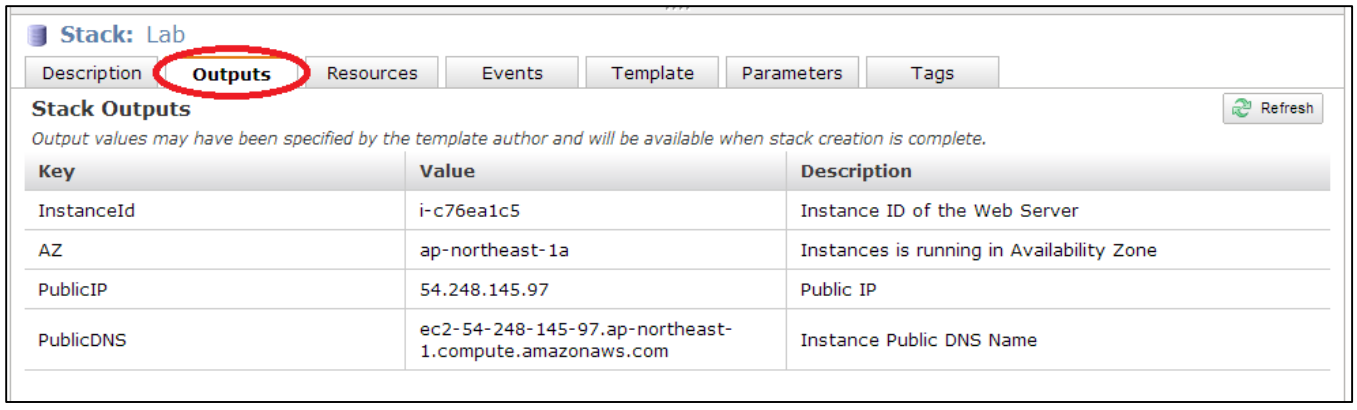
Using the AWS Management Console, create a stack from this new template in the region you selected. Since the new template has Parameters, you will see a screen in the wizard allowing you to provide parameter values.

Enter the name of the Key Pair qwikLAB™ created for you and click **Continue**, then proceed as before.



Inspect Output

Once the Management Console shows CREATE_COMPLETE, click the **Outputs** tab to view the Stack Outputs.



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

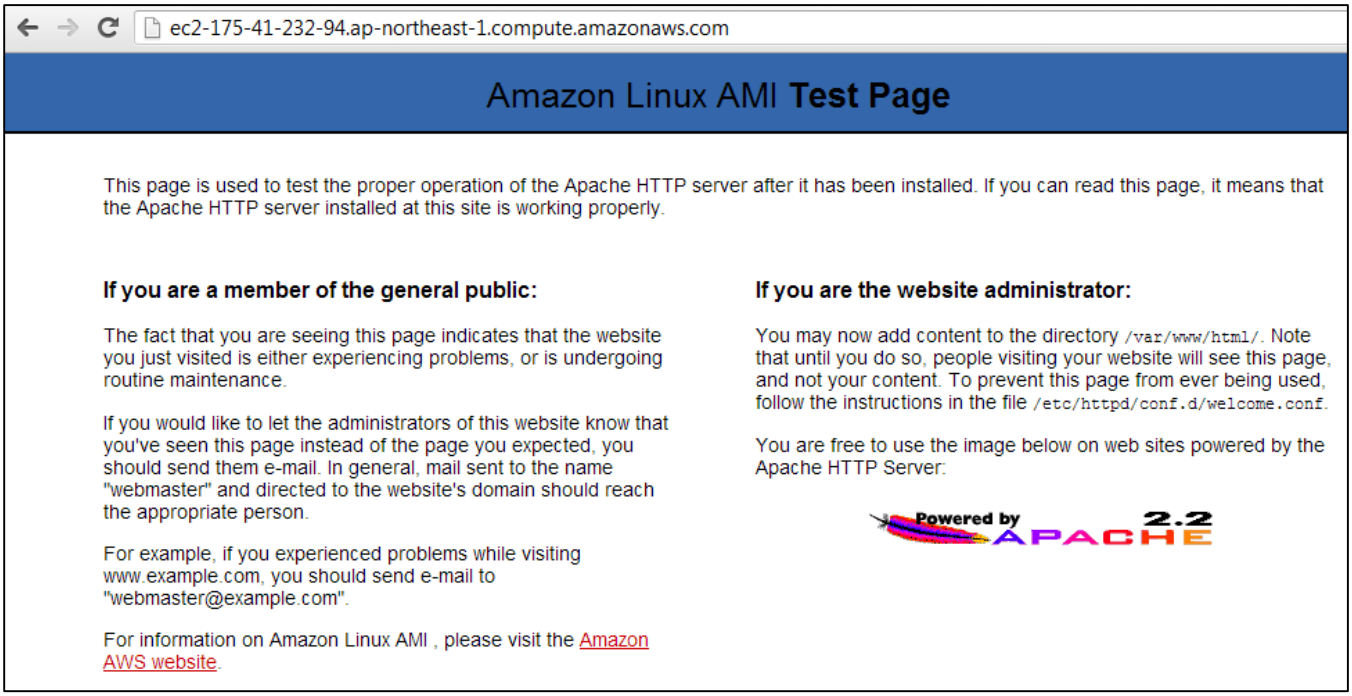
Log into the Server

See Appendix B for instructions on how to log in to your instance via SSH. Downlaod the KeyPair file (either the PPK file for Windows, or the PEM file for Linux/Mac) from the qwikLAB tab in your browser. Then locate the PublicDNS host name for your instance in the CloudFormation Outputs.

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

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

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



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 section.

Your Third CloudFormation Template

Manual configuration procedures such as those from the last exercise are not scalable especially when 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 CloudFormation to automatically bootstrap the same configuration without manual intervention.

CloudFormation provides a number of features that can help with automated instance configuration. 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 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",

                  "# 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, CloudFormation will report that it failed to create the stack.

Launch

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

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
```

Lab Summary

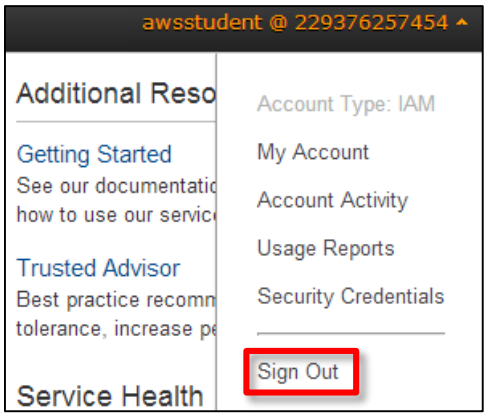
There are so many other things that CloudFormation can do! For example, you can pre-install software, files, and even download and automatically inflate .gz and .zip archives. The online docs at http://aws.amazon.com/cloudformation have a wealth of examples.

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 CloudFormation and cloud-init work under the hood.

Ending the Lab

1. To log out of the AWS Management Console, from the menu, click **awsstudent @ [YourAccountNumber]** and choose **Sign out** (where [YourAccountNumber] is the AWS account generated by *qwikLAB*™).



2. Close any active SSH client sessions or remote desktop sessions.
3. Click the **End Lab** button on the *qwikLAB*™ lab details page.

4. When prompted for confirmation, click **OK**.
5. For **My Rating**, rate the lab (using the applicable number of stars), optionally type a **Comment**, and click **Submit**.

Note: The number of stars indicates the following: 1 star = very dissatisfied, 2 stars = dissatisfied, 3 stars = neutral, 4 stars = satisfied, and 5 stars = very satisfied. Also, you may close the dialog if you do not wish to provide feedback.

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

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 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 CloudFormation Template

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

1. You can specify helper scripts from the 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 a 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, 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 a 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
```

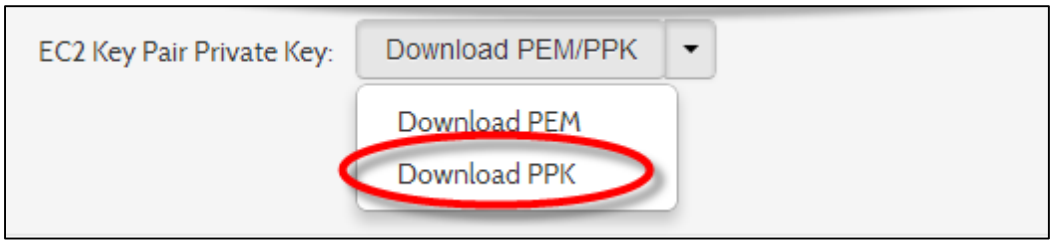
Appendix B – SSH to an EC2 Instance
Connect to your EC2 Instance via SSH (Windows)

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

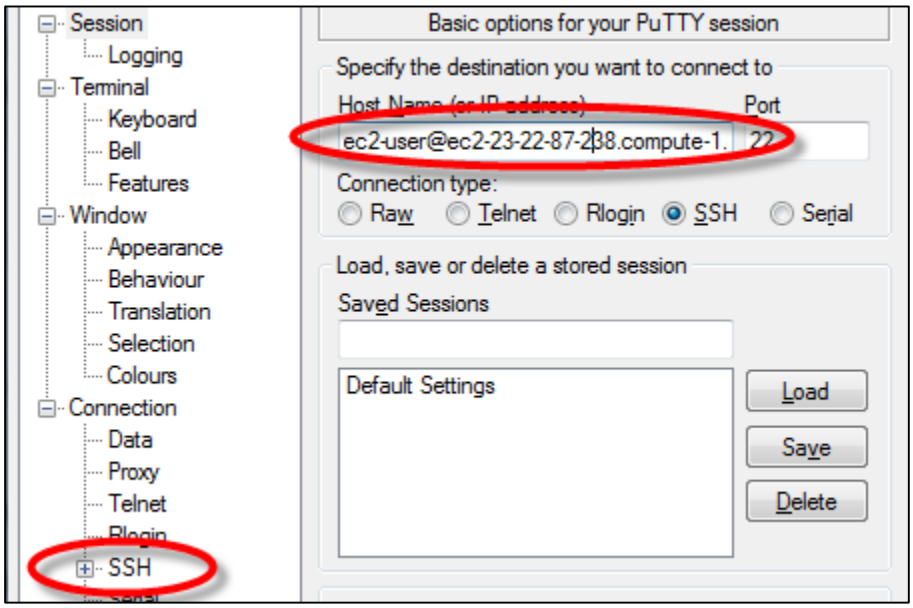
2. Switch to the *qwikLAB*™ tab in your browser.
3. Download the *qwikLAB*™ provided EC2 Key Pair private key file in the PuTTY compatible PPK format by clicking the **Download PPK** option in the **Download PEM/PPK** drop-down.



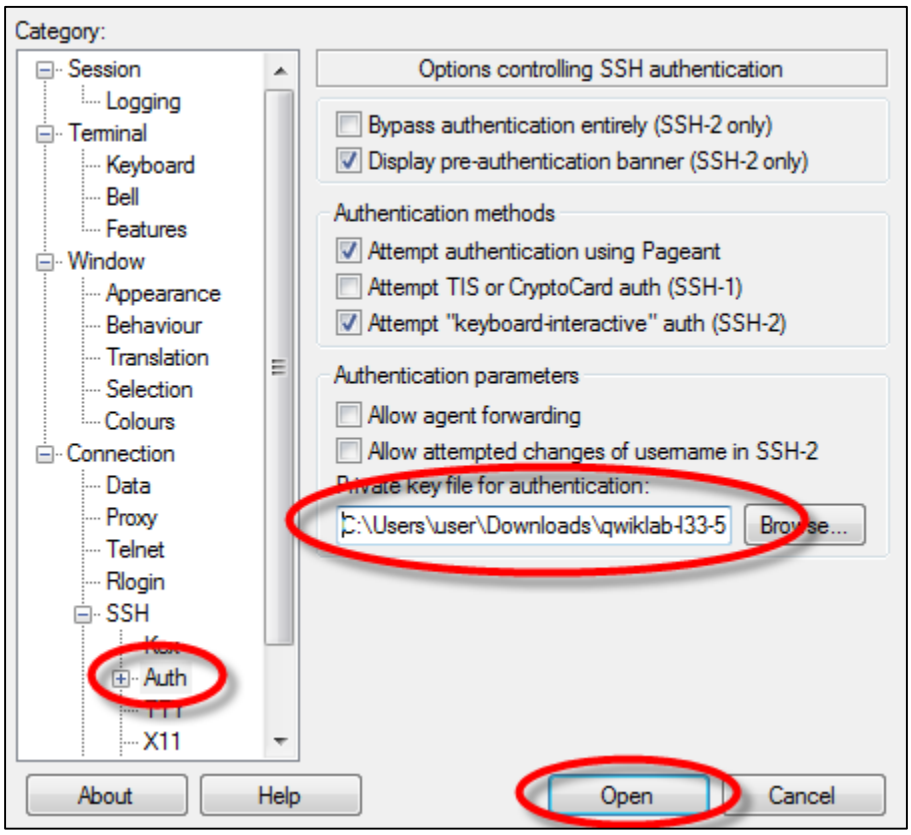
4. Save the file to your **Downloads** directory (or some other directory of your choice.)

Connect to the EC2 Instance using SSH and PuTTY.

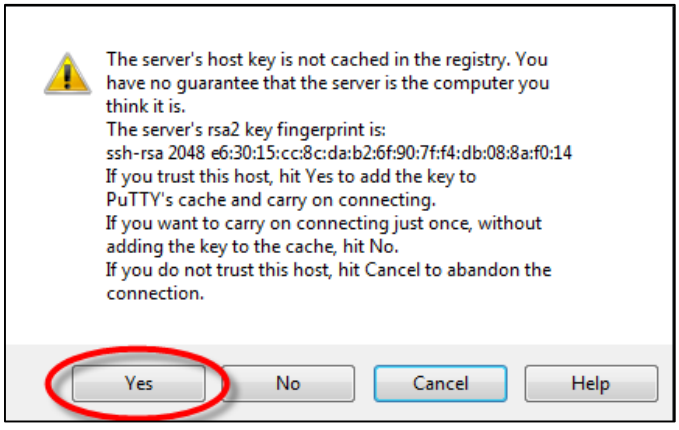
1. Launch **`putty.exe`**.
2. Enter **ec2-user@<your EC2 hostname>** into the **Host Name** input in Putty (Ctrl+v).
3. Expand the **SSH** category by double-clicking it.



4. Select the **Auth** category by clicking 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**.



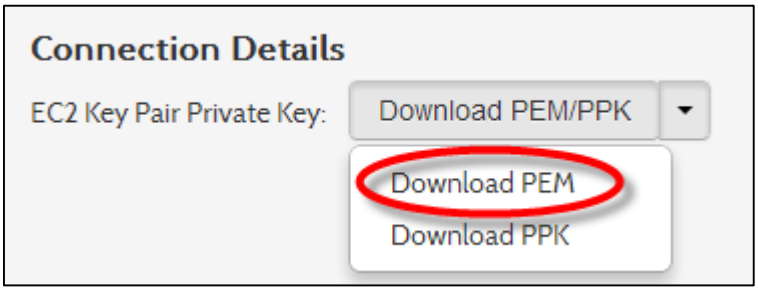
7. Click **Yes** when prompted to cache the server's key.



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

Download your EC2 Key Pair private key file

1. Switch to the *qwikLAB*™ tab in your browser.
2. Download the *qwikLAB*™ provided EC2 Key Pair private key file in the PEM format by clicking 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 following commands substituting the path/filename for the .pem file you downloaded from *qiwk*LAB™ and substituting ec2-user@<your EC2 hostname> for the host name.

```
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
```

