Introduction to AWS CloudFormation

Overview

This lab introduces you to AWS CloudFormation. You will use a pre-configured CloudFormation template that creates an Amazon EC2 instance and installs WordPress with a local MySQL database for storage. You will then clean-up your resources by deleting the stack.

Topics Covered

By the end of this lab, you will be able to:

- Create a stack using an AWS CloudFormation template
- Monitor the progress of stack creation
- Explore CloudFormation parameters, resources, mappings, and outputs
- Use the stack resources
- Clean-up when the stack is no longer required

What is AWS CloudFormation?

AWS CloudFormation gives developers and systems administrators an easy way to create and manage a collection of related AWS resources, provisioning and updating them in an orderly and predictable fashion.

You can use AWS CloudFormation's sample templates or create your own templates to describe the AWS resources, and any associated dependencies or runtime parameters, required to run your application. You don't need to figure out the order for provisioning AWS services or the subtleties of making those dependencies work. CloudFormation takes care of this for you. After the AWS resources are deployed, you can modify and update them in a controlled and predictable way, in effect applying version control to your AWS infrastructure the same way you do with your software.

You can deploy and update a template and its associated collection of resources (called a stack) by using the AWS Management Console, AWS Command Line Interface, or APIs. CloudFormation is available at no additional charge, and you pay only for the AWS resources needed to run your applications.

Start Lab

Task 1: Create a CloudFormation Stack

In this task, you will:

- Create a stack from a pre-defined AWS CloudFormation template. The stack will deploy an Amazon EC2 instance within a VPC. It will also configure it to run WordPress and MySQL.
- Explore CloudFormation parameters

Create a CloudFormation Stack

Before you begin, you must download a CloudFormation template.

- Right-click this link and download the file to your computer: wordpress.template

The template contains a definition of resources that will create a WordPress blog.

- In the **AWS Management Console**, on the **Services** menu, click **CloudFormation**.
- Click **Create stack**
- Click **With new resources (standard)**
- Click **Upload a template file**

- Click Choose file
- Browse to and select the *wordpress.template* file you downloaded.
- Click **Next**

Explore CloudFormation Parameters

AWS CloudFormation has the ability to specify **Parameters**. Values entered as parameters are passed to the stack and can be used to customize the creation of resources, such as configuring a Username and Password.

The following parameters are defined in the template:

- **KeyName:** Name of a KeyPair for SSH access
- **InstanceType:** The EC2 instance type to use. It also specifies which instance types are allowed.
- **SSHLocation:** The IP address range that can be used to SSH to the EC2 instances
- **DBName:** Database name
- **DBUser:** Database user to use
- **DBPassword:** Database password to use
- **DBRootPassword:** Root database password to use
- **VPCCIDR:** CIDR Block for the Lab VPC that you will create
- **PublicSubnet1Param:** Values allowed in a public subnet
- **PublicSubnet2Param:** Values allowed in a public subnet
- On the **Create stack** page, configure:
- **Stack name:** WordPres
- Look at the default value for **DBName**.
- Open your template with a text editor and look at the **DBName** parameter.

```
  DBName:
    Default: wordpressdb
    Description: The WordPress database name
    Type: String
    MinLength: '1'
    MaxLength: '64'
    AllowedPattern: '[a-zA-Z][a-zA-Z0-9]*'
    ConstraintDescription: must begin with a letter and contain only
alphanumeric
      characters.
```

Notice that the default name for **DBName** is *wordpressdb*. Because the template is using the default property for **DBName**, *wordpressdb* is the default text that appears in the parameters section when launching the template. Also, notice that the description for **DBName** appears just above the entry field.

- In the **CloudFormation Management Console**, look at the default values for **DBPassword**, **DBRootPassword**, and **DBUser**.

As you can see, there are none.

- In your text editor, locate the same parameters.

```
  DBUser:
    NoEcho: 'true'
    Description: The WordPress database admin account username
    Type: String
    MinLength: '1'
    MaxLength: '16'
    AllowedPattern: '[a-zA-Z][a-zA-Z0-9]*'
    ConstraintDescription: must begin with a letter and contain only
alphanumeric
      characters.
  DBPassword:
    NoEcho: 'true'
    Description: The WordPress database admin account password
    Type: String
    MinLength: '8'
    MaxLength: '41'
    AllowedPattern: '[a-zA-Z0-9]*'
    ConstraintDescription: must contain only alphanumeric characters.
  DBRootPassword:
    NoEcho: 'true'
    Description: MySQL root password
    Type: String
    MinLength: '8'
    MaxLength: '41'
    AllowedPattern: '[a-zA-Z0-9]*'
    ConstraintDescription: must contain only alphanumeric characters.
```

Notice that all three parameters do not have a **default** property. This is why the field is empty. However, all of these parameters have constraints (*MinLength*, *MaxLength*, *AllowedPattern*). These constraints are also properties of the parameters.

- In the **CloudFormation Management Console**, configure:
- **DBPassword:** passworc
- **DBRootPassword:** passworc
- **DBUser:** admi
- Look at the **InstanceType** parameter.

Notice that **t2.micro** is selected by default. However, if you click the drop-down you can also select *t1.micro* and *t2.nano*.

- In your text editor, locate the **InstanceType** parameter.

```
  InstanceType:
    Description: WebServer EC2 instance type
    Type: String
    Default: t2.micro
    AllowedValues:
      - t1.micro
      - t2.nano
      - t2.micro
    ConstraintDescription: must be a valid EC2 instance type.
content_copy
```

Notice that the **InstanceType** parameter has a default value of **t2.micro** but it also allows you to select *t1.micro* and *t2.nano* instance types.

- In the **CloudFormation Management Console**, for **InstanceType** keep *t2.micro* selected.

- For **KeyName**, select the key pair that you see.

This key pair was already present in your AWS Account.

- In your text editor, locate the **KeyName** parameter.

```
  KeyName:
    Description: Name of an existing EC2 KeyPair to enable SSH access to the
instances
    Type: AWS::EC2::KeyPair::KeyName
    ConstraintDescription: must be the name of an existing EC2 KeyPair.
```

Notice that it has a **Type** of *AWS::EC2::KeyPair::KeyName*.

When you use AWS-specific parameter types, anyone who uses your template to create or update a stack must specify existing AWS values that are in their account and in the region for the current stack. AWS-specific parameter types help ensure that input values for these types exist and are correct before AWS CloudFormation creates or updates any resources. For example, if you use the AWS::EC2::KeyPair::KeyName parameter type, AWS CloudFormation validates the input value against users' existing key pair names before it creates any resources, such as Amazon EC2 instances.

- In the **CloudFormation Management Console**, compare the rest of the parameters to the parameters defined in the template, then click **Next**

Explore CloudFormation Options

The **Options** page allows configuration of tags, roles, and advanced settings.

1. **Tags:** Tags are arbitrary key-value pairs that can be used to identify your stack for purposes such as cost allocation.
2. **Permissions:** An existing AWS Identity and Access Management (IAM) service role that AWS CloudFormation can assume.
3. **Stack policy:** Defines the resources that you want to protect from unintentional updates during a stack update. By default, all resources can be updated during a stack update.
4. Click **Stack creation options**
5. **Rollback on failure:** Specifies whether the stack should be rolled back if stack creation fails. Typically, you want to accept the default value of Yes. Select No if you want the stack's state retained even if creation fails, such as when you are debugging a stack template.
6. **Timeout:** Specifies the amount of time, in minutes, that CloudFormation should allot before timing out stack creation operations. If CloudFormation cannot create the entire stack in the time allotted, it fails the stack creation due to timeout and rolls back the stack. By default, there is no timeout for stack creation. However, individual resources may have their own timeouts based on the nature of the service they implement. For example, if an individual resource in your stack times out, stack creation also times out even if the timeout you specified for stack creation has not yet been reached.
7. **Enable termination protection:** Prevents a stack from being accidently deleted. If a user attempts to delete a stack with termination protection enabled, the deletion fails and the stack--including its status--remains unchanged.

In this lab, you will keep the default values.

- Click **Next**

- On the **Review** page:

- Review the configuration
- Click **Create stack**

AWS CloudFormation will create the resources in the template.

Task 2: Monitor CloudFormation

In this task, you will monitor the CloudFormation stack creation.

- Click the **Template** tab.

The **Template** tab, displays the contents of your template.

If you do not see the **Template** tab, it might be due to your window width. If this is the case, you should be able to widen your window so that the template appears.

- Click the **Parameters** tab.

The **Parameters** tab, displays the parameters and parameter values that are part of your template.

- Click the **Events** tab.

The **Events** tab displays each major event in the creation of the stack sorted by the time of each event, with the latest events on top. You can see different events and their status, such
as CREATE_IN_PROGRESS or CREATE_COMPLETE.

Explore CloudFormation Resources

- Click the **Resources** tab.

The **Resources** tab, displays the resources that are part of the stack. Initially, you may only see one or two resources. Once the stack has been created, you will see the rest of your resources.

For each resource, it displays the resource:

- **Logical ID:** The logical ID must be alphanumeric (A-Za-z0-9) and unique within the template. Use the logical name to reference the resource in other parts of the template. For example, if you want to map an Amazon Elastic Block Store volume to an Amazon EC2 instance, you reference the logical IDs to associate the block stores with the instance.
- **Physical ID:** In addition to the logical ID, certain resources also have a physical ID, which is the actual assigned name for that resource, such as an EC2 instance ID or an S3 bucket name. Use the physical IDs to identify resources outside of AWS CloudFormation templates, but only after the

resources have been created. For example, you might give an EC2 instance resource a logical ID of MyEC2Instance; but when AWS CloudFormation creates the instance, AWS CloudFormation automatically generates and assigns a physical ID (such as i-28f9ba55) to the instance. You can use this physical ID to identify the instance and view its properties (such as the DNS name) by using the Amazon EC2 console.

- **Type:** The resource type identifies the type of resource that you are declaring. For example, AWS::EC2::Instance declares an EC2 instance.
- **Status:** Displays status codes the resources that are being created in your stack such as: *CREATE_IN_PROGRESS*, *CREATE_COMPLETE*, as well as *CREATE_FAILED*.

The following resources are defined within the template that you are deploying.

- **WebServerSecurityGroup:** A security group that enables access via port 80 (HTTP) and port 22 (SSH).
- **VPC:** Virtual Private Cloud used for your other resources
- **InternetGateway:** A gateway to access the Internet
- **AttachGateway:** A way to attach to your Internet Gateway
- **PublicSubnet1:** Your first public subnet
- **PublicSubnet2:** Your second public subnet
- **PublicRouteTable:** A route table for your VPC
- **PublicRoute:** A public route for your route table
- **PublicSubnet1RouteTableAssociation:** A way to associate your first subnet with your public route table
- **PublicSubnet2RouteTableAssociation:** A way to associate your second subnet with your public route table
- **WebServer:** Amazon EC2 instance that is created as a WordPress server
- In your text editor, locate the **WebServerSecurityGroup** resource.

```
  WebServerSecurityGroup:
    DependsOn: AttachGateway
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Enable HTTP access via port 80 locked down to the
load balancer
        + SSH access
      VpcId: !Ref 'VPC'
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: '80'
          ToPort: '80'
          CidrIp: 0.0.0.0/0
        - IpProtocol: tcp
          FromPort: '22'
          ToPort: '22'
          CidrIp: !Ref 'SSHLocation'
```

This code defines a security group, which permits HTTP traffic (port 80) from the Internet (0.0.0.0/0) and SSH connections (port 22) from the IP range that was specified as the **SSHLocation** in the parameters section.

- In your text editor, locate the **WebServer** resource.

```
  WebServer:
    Type: AWS::EC2::Instance
    DependsOn: WebServerSecurityGroup
    Metadata:
      AWS::CloudFormation::Init:
        install_wordpress:
          packages:
            yum:
              php: []
              php-mysql: []
              mysql: []
              mysql-server: []
              mysql-devel: []
              mysql-libs: []
              httpd: []
```

This code defines an Amazon EC2 instance with steps to install WordPress. This involves installing the PHP scripting language, a MySQL database and an HTTP web server. The above code is truncated from the original.

- Locate the **Properties** section of your **WebServer** resource.

```
    Properties:
       ImageId: !FindInMap
         - AWSRegionArch2AMI
         - !Ref 'AWS::Region'
         - !FindInMap
           - AWSInstanceType2Arch
           - !Ref 'InstanceType'
           - Arch
```

Notice that there is an **ImageId** property. Your template creates an Amazon EC2 instance from a predefined image.

The **ImageId** property provides the unique ID of the Amazon Machine Image (AMI) that gets assigned during registration. Notice that the **ImageId** is located by using a function (!FindInMap). The function looks up your **ImageId** using two maps in the template **AWSRegionArch2AMI** and **AWSInstanceType2Arch**. These maps are in the **Mappings** section of your template.

Explore CloudFormation Mappings

- In your text editor, locate the **Mappings** section.

The optional Mappings section matches a key to a corresponding set of named values. For example, if you want to set values based on a region, you can create a mapping that uses the region name as a key and contains the values you want to specify for each specific region. You use the Fn::FindInMap intrinsic function to retrieve values in a map.

- Locate the mapping for **AWSInstanceType2Arch**.

```
Mappings:
  AWSInstanceType2Arch:
    t1.micro:
      Arch: PV64
    t2.nano:
      Arch: HVM64
    t2.micro:
      Arch: HVM64
```

- What is the value if the instance type used is **t2.micro**?

It should be **HVM64**.

- Locate the mapping for **AWSRegionArch2AMI**.

```
  AWSRegionArch2AMI:
    us-east-1:
      PV64: ami-2a69aa47
      HVM64: ami-6869aa05
      HVMG2: ami-2e5e9c43
    us-west-2:
      PV64: ami-7f77b31f
      HVM64: ami-7172b611
      HVMG2: ami-83b770e3
```

- What is the value for **us-west-2**?

It should be **ami-7172b611**.

Explore CloudFormation Outputs

- In your text editor, locate the **Outputs** section of the template.

```
Outputs:
  WebsiteURL:
    Value: !Sub 'http://${WebServer.PublicDnsName}/wordpress'
    Description: WordPress Website
  EC2IPAddress:
    Value: !GetAtt WebServer.PublicIp
```

The optional **Outputs** section declares output values that you can import into other stacks (to create cross-stack references), return in response (to describe stack calls), or view on the AWS CloudFormation console.

The **Outputs** code above provides a link to the WordPress website. It also provides the public IP Address of the EC2 instance used.

- In the **CloudFormation Management Console**, click the **Outputs** tab.

When a stack has been deployed, it can display output information. This stack provides a link to the WordPress website.

- Copy the value for the **WebsiteURL** and **EC2IPAddress** to your text editor.

Connect to your WordPress site using your web browser

- Open a new web browser tab and enter the value of WebsiteURL, which connects you to your new WordPress website.

You should be presented with the WordPress configuration page that looks similar to this:

- Configure the following:
- **Site Title:** Enter any title you wish
- **Username:** student
- **Password:** student12
- **Confirm Password:** Confirm use of weak password
- **Your Email:** student@example.c
- **Search Engine Visibility:** Do not change
- Click **Install WordPress**

You should be presented with a *Success* screen:

- Click **Log In** and login with these credentials:
- **Username:** student
- **Password:** student12

You will be taken to the WordPress Dashboard.

You can now create a blog post to add information to your website.

Connect to your EC2 Instance via SSH

Because you also opened port 22 to your EC2 instance, you should also be able to connect to your instance.

Windows Users: Using SSH to Connect

These instructions are for Windows users only.

If you are using Mac or Linux, skip to the next section.

- To the left of the instructions you are currently reading, click **Download PPK**.
- Save the file to the directory of your choice.

You will use PuTTY to SSH to Amazon EC2 instances.

If you do not have PuTTY installed on your computer, download it here.

- Open PuTTY.exe
- Configure the PuTTY to not timeout:
- Click **Connection**
- Set **Seconds between keepalives** to `30`

This allows you to keep the PuTTY session open for a longer period of time.

- Configure your PuTTY session:
- Click **Session**
- **Host Name(or IP address):** Copy and paste the **EC2IPAddress** that you copied to your text editor
- In the **Connection** list, expand **SSH**
- Click **Auth** (don't expand it)
- Click **Browse**
- Browse to and select the PPK file that you downloaded
- Click **Open** to select it
- Click **Open**
- Click **Yes**, to trust the host and connect to it.
- When prompted **login as**, enter `ec2-use`

This will connect to your EC2 instance.

- Windows Users: Click here to skip ahead to the next task.

Mac and Linux Users

These instructions are for Mac/Linux users only. If you are a Windows user, skip ahead to the next task.

- To the left of the instructions you are currently reading, click **Download PEM**.
- Save the file to the directory of your choice.
- Copy this command to a text editor:

```
chmod 400 KEYPAIR.pem

ssh -i KEYPAIR.pem ec2-user@EC2IPAddress
```

- Replace **KEYPAIR.pem** with the path to the PEM file you downloaded.
- Replace **EC2IPAddress** with the IP address that you copied to your text editor
- In a Terminal, cd to the directory that you downloaded the *.PEM* file to.
- Paste the updated command into the Terminal window and run it.
- Type `yes` when prompted to allow a first connection to this remote SSH server.

Because you are using a key pair for authentication, you will not be prompted for a password.

- Check the processes that are running on the system by entering `ps -ef`
- Close your SSH session.

Task 3: Delete the Stack

In this section, you will delete the stack. This will automatically delete all of the resources created by the stack.

- In the AWS Management console, on the **Services** menu, click **CloudFormation**.
- Select **WordPress**.
- Click the **Delete** button.
    1. Click on **Edit termination protection**
    2. Select **Disabled**
    3. Click **Save**
- Click the **Delete** button.
- Click **Delete stack**

The status for the stack changes to DELETE_IN_PROGRESS. In the same way you monitored the creation of the stack, you can monitor its deletion by using the **Events** tab. When AWS CloudFormation completes the deletion of the stack, it removes the stack from the list.

Conclusion

Congratulations! You now have successfully:

- Created a stack using an AWS CloudFormation template
- Monitored the progress of stack creation
- Explored CloudFormation parameters, resources, mappings, and outputs
- Used the stack resources
- Cleaned up the stack