Linux Academy

Live! Lab

Introduction to CloudFormation

Contents

Lab Connection Information

- Labs may take up to five minutes to build
- Access to an AWS Console is provided on the Live! Lab page, along with your login credentials
- Ensure you are using the N. Virginia region
- Labs will automatically end once the alloted amount of time finishes

Related Videos

CloudFormation Concepts

CloudFormation Essentials

Validating CloudFormation Templates

CloudFormation Hands On

Need Help?

Linux Academy Community

... and you can always send in a support ticket on our website to talk to an instructor!

CloudFormation allows AWS users to create templates for generating "stacks" using AWS resources. This allows developers, engineers, and admins to generate identical environments from which to work and otherwise. CloudFormation, like Puppet and Chef, allows users to create "infrastructure as code."

Create a Template

Navigate to the CloudFormation dashboard. You are prompted to select between creating a new stack, designing a template, or creating a template from existing resources using CloudFormer (currently in beta).

Select **Design template**. This sends you to a screen where you can type in your template (bottom), or use the upper portion of the screen to design your template using drag-and-drop. For this Live! Lab, we instead copy and paste in the template provided at https://linuxacademy.com/cp/guides/download/refsheets/guides/refsheets/example-template-json_1485040032.zip.

The template is also provided below:

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
  "Description" : "Introduction to CloudFormation",
  "Mappings" : {
    "SubnetConfig" : {
      "VPC"     : { "CIDR" : "10.0.0.0/16" },
      "Public"  : { "CIDR" : "10.0.0.0/24" }
    }
  },
  "Resources" : {
    "VPC" : {
      "Type" : "AWS::EC2::VPC",
      "Properties" : {
        "EnableDnsSupport" : "true",
        "EnableDnsHostnames" : "true",
        "CidrBlock" : { "Fn::FindInMap" : [ "SubnetConfig", "VPC",
"CIDR" ]},
        "Tags" : [
          { "Key" : "Application", "Value" : { "Ref" : "AWS::StackName"
} },
          { "Key" : "Name", "Value" : "LinuxAcademy" },
          { "Key" : "Network", "Value" : "Public" }
        ]
      }
    },
    "PublicSubnet" : {
      "Type" : "AWS::EC2::Subnet",
      "Properties" : {
        "VpcId" : { "Ref" : "VPC" },
        "AvailabilityZone": { "Fn::Select": [ "0", { "Fn::GetAZs": "" }
] },
        "CidrBlock" : { "Fn::FindInMap" : [ "SubnetConfig", "Public",
"CIDR" ]},
        "Tags" : [
          { "Key" : "Application", "Value" : { "Ref" : "AWS::StackName"
```

```
} },
          { "Key" : "Network", "Value" : "Public" }
        ]
      }
    },
    "InternetGateway" : {
      "Type" : "AWS::EC2::InternetGateway",
      "Properties" : {
        "Tags" : [
          { "Key" : "Application", "Value" : { "Ref" : "AWS::StackName"
} },
          { "Key" : "Network", "Value" : "Public" }
        ]
      }
    },
    "GatewayToInternet" : {
      "Type" : "AWS::EC2::VPCGatewayAttachment",
      "Properties" : {
        "VpcId" : { "Ref" : "VPC" },
        "InternetGatewayId" : { "Ref" : "InternetGateway" }
      }
    },
    "PublicRouteTable" : {
      "Type" : "AWS::EC2::RouteTable",
      "Properties" : {
        "VpcId" : { "Ref" : "VPC" },
        "Tags" : [
          { "Key" : "Application", "Value" : { "Ref" : "AWS::StackName"
} },
          { "Key" : "Network", "Value" : "Public" }
        ]
      }
    },
    "PublicRoute" : {
      "Type" : "AWS::EC2::Route",
      "DependsOn" : "GatewayToInternet",
      "Properties" : {
        "RouteTableId" : { "Ref" : "PublicRouteTable" },
        "DestinationCidrBlock" : "0.0.0.0/0",
        "GatewayId" : { "Ref" : "InternetGateway" }
      }
    },
    "PublicSubnetRouteTableAssociation" : {
      "Type" : "AWS::EC2::SubnetRouteTableAssociation",
      "Properties" : {
        "SubnetId" : { "Ref" : "PublicSubnet" },
        "RouteTableId" : { "Ref" : "PublicRouteTable" }
      }
    },
    "PublicNetworkAcl" : {
      "Type" : "AWS::EC2::NetworkAcl",
      "Properties" : {
        "VpcId" : { "Ref" : "VPC" },
        "Tags" : [
          { "Key" : "Application", "Value" : { "Ref" : "AWS::StackName"
} },
```

```
            { "Key" : "Network", "Value" : "Public" }
          ]
        }
      },
      "InboundHTTPPublicNetworkAclEntry" : {
        "Type" : "AWS::EC2::NetworkAclEntry",
        "Properties" : {
          "NetworkAclId" : { "Ref" : "PublicNetworkAcl" },
          "RuleNumber" : "100",
          "Protocol" : "6",
          "RuleAction" : "allow",
          "Egress" : "false",
          "CidrBlock" : "0.0.0.0/0",
          "PortRange" : { "From" : "80", "To" : "80" }
        }
      },
      "InboundSSHPublicNetworkAclEntry" : {
        "Type" : "AWS::EC2::NetworkAclEntry",
        "Properties" : {
          "NetworkAclId" : { "Ref" : "PublicNetworkAcl" },
          "RuleNumber" : "102",
          "Protocol" : "6",
          "RuleAction" : "allow",
          "Egress" : "false",
          "CidrBlock" : "0.0.0.0/0",
          "PortRange" : { "From" : "22", "To" : "22" }
        }
      },
      "OutboundPublicNetworkAclEntry" : {
        "Type" : "AWS::EC2::NetworkAclEntry",
        "Properties" : {
          "NetworkAclId" : { "Ref" : "PublicNetworkAcl" },
          "RuleNumber" : "100",
          "Protocol" : "6",
          "RuleAction" : "allow",
          "Egress" : "true",
          "CidrBlock" : "0.0.0.0/0",
          "PortRange" : { "From" : "0", "To" : "65535" }
        }
      },
      "PublicSubnetNetworkAclAssociation" : {
        "Type" : "AWS::EC2::SubnetNetworkAclAssociation",
        "Properties" : {
          "SubnetId" : { "Ref" : "PublicSubnet" },
          "NetworkAclId" : { "Ref" : "PublicNetworkAcl" }
        }
      },
      "EC2SecurityGroup" : {
        "Type" : "AWS::EC2::SecurityGroup",
        "Properties" : {
          "GroupDescription" : "Enable access to the EC2 host",
          "VpcId" : { "Ref" : "VPC" },
          "SecurityGroupIngress" : [
            { "IpProtocol" : "tcp", "FromPort" : "22",  "ToPort" : "22",
"CidrIp" : "0.0.0.0/0" },
            { "IpProtocol" : "tcp", "FromPort" : "80",  "ToPort" : "80",
```

```
        "CidrIp" : "0.0.0.0/0" }
          ]
        }
      },
      "PublicInstance" : {
        "Type" : "AWS::EC2::Instance",
        "DependsOn" : "GatewayToInternet",
        "Properties" : {
          "InstanceType" : "t2.micro",
          "ImageId"  : "ami-9be6f38c",
          "NetworkInterfaces" : [{
            "GroupSet"                   : [{ "Ref" : "EC2SecurityGroup" }],
            "AssociatePublicIpAddress" : "true",
            "DeviceIndex"                : "0",
            "DeleteOnTermination"      : "true",
            "SubnetId"                   : { "Ref" : "PublicSubnet" }
          }]
        }
      }
    }
}
```

Reviewing the Template

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
  "Description" : "Introduction to CloudFormation",
  "Mappings" : {
    "SubnetConfig" : {
      "VPC"     : { "CIDR" : "10.0.0.0/16" },
      "Public"  : { "CIDR" : "10.0.0.0/24" }
    }
  },
```

This opening segment states that we are create a template using the defined version, provides a basic description, and make note of the subnets we wish to use in our stack. The `SubnetConfig` section works as a reference point and is explained later in the template review.

Next, we have the "resource" settings, wherein we really begin to flesh out what AWS resources we want to use in our final stack. First, we define the VPC:

```
  "Resources" : {
    "VPC" : {
      "Type" : "AWS::EC2::VPC",
      "Properties" : {
        "EnableDnsSupport" : "true",
        "EnableDnsHostnames" : "true",
        "CidrBlock" : { "Fn::FindInMap" : [ "SubnetConfig", "VPC",
"CIDR" ]},
        "Tags" : [
          { "Key" : "Application", "Value" : { "Ref" : "AWS::StackName"
```

```
} },
          { "Key" : "Name", "Value" : "LinuxAcademy" },
          { "Key" : "Network", "Value" : "Public" }
        ]
      }
    },
```

Here, `VPC` acts as a naming convention, and AWS uses `"Type" : "AWS::EC2::VPC"` to know that it is creating an actual VPC. We define which properties we want the VPC to have, including the CIDR block, which is created using an intrinsic function, `FindInMap`. This allows us to build more module functions by allowing it to grab the CIDR range from the *VPC* key, under our *SubnetConfig* in the first section of the template. The `Tags` section is optional but sets a number of key-value pairs for use. Note that some of these are not hard-coded but instead use a function to grab the value.

We now create the public subnet:

```
    "PublicSubnet" : {
      "Type" : "AWS::EC2::Subnet",
      "Properties" : {
        "VpcId" : { "Ref" : "VPC" },
        "AvailabilityZone": { "Fn::Select": [ "0", { "Fn::GetAZs": "" }
] },
        "CidrBlock" : { "Fn::FindInMap" : [ "SubnetConfig", "Public",
"CIDR" ]},
        "Tags" : [
          { "Key" : "Application", "Value" : { "Ref" : "AWS::StackName"
} },
          { "Key" : "Network", "Value" : "Public" }
        ]
      }
    },
```

Notice how the `VpcId` uses a reference function to call back to the previous stanza, since this is a subnet created for the defined VPC. A `Select` function is used to choose an Availability Zone; this allows the template to be used across a number of AWS accounts to determine the appropriate availability zone for the account.

The next number of segments, up to the `EC2SecurityGroup` resource, further define the properties of the stack's network, including defining ACL rules. These sections use similar functions and references as those used in the above sections. Review them to see how each functions within the whole of the template.

```
    "EC2SecurityGroup" : {
      "Type" : "AWS::EC2::SecurityGroup",
      "Properties" : {
        "GroupDescription" : "Enable access to the EC2 host",
        "VpcId" : { "Ref" : "VPC" },
        "SecurityGroupIngress" : [
          { "IpProtocol" : "tcp", "FromPort" : "22",  "ToPort" : "22",
"CidrIp" : "0.0.0.0/0" },
          { "IpProtocol" : "tcp", "FromPort" : "80",  "ToPort" : "80",
```

```
        "CidrIp" : "0.0.0.0/0" }
          ]
        }
      },
```

The EC2SecurityGroup resource itself does not do anything special – it creates the security group for the EC2 instance and ensures that both ports 22 and 80 are accessible. Note that the placement of this does not matter in the template, but can be defined to depend on another resource's creation. This is used in the next stanza, for the `PublicInstance`:

```
      "PublicInstance" : {
        "Type" : "AWS::EC2::Instance",
        "DependsOn" : "GatewayToInternet",
        "Properties" : {
          "InstanceType" : "t2.micro",
          "ImageId"  : "ami-9be6f38c",
          "NetworkInterfaces" : [{
            "GroupSet"                   : [{ "Ref" : "EC2SecurityGroup" }],
            "AssociatePublicIpAddress" : "true",
            "DeviceIndex"                : "0",
            "DeleteOnTermination"      : "true",
            "SubnetId"                   : { "Ref" : "PublicSubnet" }
          }]
        }
      }
    }
}
```

Here we see that the instance's creation depends on the creation of the `GatewayToInternet` resource. It also defines the instance type, the image ID, and which network interfaces we want to use, including defining the security group resource above.

Validate and Use the Template

Copy the template into the console at the bottom, ensuring that the **Template** tab is selected (*not* Components). Press the check box at the top of the screen to validate the template.

Press the **Create stack** cloud button on the upper left of the dashboard to create the template. This drops up in the **Create Stack** page, where we can specify a template URL or otherwise when selecting a template. Ensure *Specify an Amazon S3 template URL* is selected, and press **Next**.

Give the stack a name. We named ours *first-stack*. Press **Next**.

We can now define tags, change permissions, set notifications, and more on this screen. We do not want to change anything on this page for this lab. Press **Next**, confirm the options on the review page, then select **Create**.

Refresh the page to see that the stack is now listed as *CREATE_IN_PROGRESS*. Using the **Resources** and

Events tabs on the bottom the dashboard, we can watch as various parts of the template are created. Wait until the statue reads *CREATE_COMPELTE*. If we view the **Events** tab, we can see what happened during the creation process.

We can further verify that everything has been created by going to the EC2 Dashboard and the VPC Dashboard. Here we can see the create EC2 instance and the created VPC.

Return to the CloudFormation Dashboard, and delete the stack. When the stack is deleted, all associated resources will be removed.