CloudFormation Features

Intrinsic Functions

Built-in functions that help you manage your stacks

CloudFormation has over 10 built-in functions which help us manage our stack.

Let's see it in action. We will use our simple template which we used to provision an EC2 instance and this time use an intrinsic function.

Join - Appends a set of values into a single value. Join is a function that appears in many programming languages as well.

Here is the JSON syntax for a join function:

{“Fn::Join”:[“delimiter”,[comma-delimited list of values]]}

We call Fn::Join and pass it an array where the first item will be the delimiter to join the list of values together with, and the second item will be the comma delimited list of values. An example of this would be the following:

{“Fn::Join” : [“:”, [“a”, “b”, “c”]]} = “a:b:c”

Here we are passing in a colon as a delimiter and a, b, c as the list of values. The output of this would be “a:b:c”

Here is the YAML syntax for a join function:

!Join [delimiter,[comma-delimited list of values]]

We call the Join function with an exclamation mark before the word Join. Then we pass in the delimiter as the first item in the array and then a list of values. An example of this would be the following:

!Join [“:”, [a, b, c]] = “a:b:c”

We call the Join function by using an exclamation mark beforehand. Here we have the colon as the first item and a, b, c as the array of the values for the second item. The result will be “a:b:c”

So the JSON and YAML examples above are equivalent. The AWS documentation has a list of CloudFormation functions at the following web address:

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/intrinsic-function-reference.html

Intro to Intrinsic Functions Lab: Utilizing the Join Function

In this lab we will use the Join function.

Here is a simple YAML formatted template:

★★★Be advised that this template utilizes an AMI from the Sydney region so make sure you set your region to Sydney in the AWS console or it will not work.

```
Resources:
 Ec2Instance:
  Type: AWS::EC2::Instance
  Properties:
   InstanceType: t2.micro
   ImageId: ami-43874721 # Amazon Linux AMI in Sydney
   Tags:
    - Key: Name
     Value: A simple example
```

On line 3 of the above template, we are creating a resource of Type that is an 'AWS::EC2::Instance'

On line 5 the InstanceType is a t2.micro.

On line 6 we have specified the AMI ID. This AMI is in the Sydney region so we will be creating this stack in the Sydney region.

On line 7,8, and 9 we created a tag. The Key is "Name" and the Value is "Simple Example"

<u>Now let's utilize the join function</u>

We are going to utilize the join function in the Tags section. Change the Value of the tag on line 9 of the above YAML template to be the following:

```
Value: !Join [ " ", [ EC2, Instance, with, Fn, Join ] ]
```

↑FYI there is a space in between the quotation marks

For our Join function, the space is our join parameter and then we gave it an array of strings and we expect all of these comma delimited values to then be joined into a single value with a space in between them. So we expect the name of the EC2 instance that we create to be EC2 Instance with Fn Join.

So now our template looks like the following:

```
Resources:
 Ec2Instance:
  Type: "AWS::EC2::Instance"
  Properties:
   InstanceType: t2.micro
   ImageId: ami-43874721 # Amazon Linux AMI in Sydney
   Tags:
    - Key: "Name"
     Value: !Join [ " ", [ EC2, Instance, with, Fn, Join ] ]
```

★★★Copy and paste the above 9 lines of YAML code into a text editor (such as Visual Studio Code) and save it as IntrinsicFunctions.yaml

Let’s go to the AWS management console and we will utilize this template in CloudFormation.

★★★At the top right of the Management console make sure to change your region to the Sydney region since we will be using an AMI from the Sydney region.

Click **Services** and then click **CloudFormation**

Click **Create stack**

Prepare template: ✓Template is ready 🡐 select Template is ready

Template source: ✓Upload a template file

Click the **Choose file** button

And choose the IntrinsicFunctions.yaml file that you created

Then click **Next**

Stack Name: SimpleStack ← give it a name like so

Click **Next**

This will direct you to the Configure stack options page. Just leave everything as is and click **Next**.

This will direct you to the Review page. Just leave everything as is and click the **Create stack** button at the bottom of the page.

Status will say CREATE_IN_PROGRESS

Give it a minute or two. Then go to the EC2 console. So click **Services** and then click **EC2** and then click **Running instances**.

Right away you will see the instance with the name of EC2 Instance with Fn Join

Mission accomplished!

CloudFormation properly executed the join function that we placed in our template. It joined the comma delimited list of values together with a space in between each. Do not delete SimpleStack just yet as we will be utilizing it in the next lab/tutorial.

CloudFormation Features: Adding Multiple Resources to a Template

In this lesson we create a template that has an EC2 instance with a security group open to Port 22

The problem we have with multiple resources that depend on each other is that they need to be created in a specific order!!!

For example, the EC2 instance needs to specifiy it's security groups when it is being provisioned. This cannot be done if the security group doesn't exist. So the order of the creation will need to be: first the security group, which we will define with port 22 open…And then secondly we will create the EC2 instance which can then refer to the security group. Hence the dependencies and the order of creation is very important. However, CloudFormation handles this for us!

Lab: Multiple Resources

Per the last tutorial/lab. We used the following 9 line YAML template:

```
Resources:
  Ec2Instance:
    Type: "AWS::EC2::Instance"
  Properties:
    InstanceType: t2.micro
    ImageId: ami-43874721 # Amazon Linux AMI in Sydney
    Tags:
      - Key: "Name"
        Value: !Join [ " ", [ EC2, Instance, with, Fn, Join ] ]
```

Now we will extend this template to include another resource, which is the security group for our EC2 instance. Here is the YAML code that we will utilize:

```
Resources:
  Ec2Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      InstanceType: t2.micro
      ImageId: ami-43874721 # Amazon Linux AMI in Sydney
      Tags:
        - Key: "Name"
          Value: !Join [ " ", [ EC2, Instance, with, Fn, Join ] ]
      SecurityGroups:
        - !Ref MySecurityGroup
  MySecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: Enable SSH access via port 22
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: '22'
          ToPort: '22'
          CidrIp: 0.0.0.0/0
```

★★★Take the above 20 lines of YAML code and paste it into a blank code editor file. Then save the file as MultipleResources.yaml

The first 9 lines are exactly the same as before. However, we have added from line 10 onwards all that pertains to the security group.

Starting at line 12 we have created a new resource in the resources section called MySecurityGroup

The Type on line 13 is 'AWS::EC2::SecurityGroup'

The Properties set up a GroupDescription called Enable SSH access via port 22

We have then defined the SecurityGroupIngress (rule)

We defined the tcp protocol for port 22 from all IPs

★★★On line 10, SecurityGroups is an EC2 instance resource and on line 11 we have linked the security group to the EC2 instance by utilizing the Ref instrinsic function. Here the Ref intrinsic function is referring to the MySecurityGroup logical name for the security group on line 12.

So we have created the security group and we've asked the EC2 instance to refer to that security group. So we have brought them together. CloudFormation is going to figure out the dependencies automatically and create them in the correct order.

Let's update the stack we created in the previous lab/tutorial called SimpleStack. If you have since deleted it then just recreate it again per the last lab/tutorial.

In the AWS console make sure you are in the Sydney Region.

Click **Services** and then click **CloudFormation**

✓SimpleStack ← select the stack that we created in the last lab/tutorial called SimpleStack

Then click the **Update** button

Prepare template: ✓Replace current template ← select Replace current template

Template Source: ✓Upload a template file ←

Click the **Choose file** button

Choose the file that you saved called MultipleResources.yaml

Then click **Next**

This will direct you to the Specify stack details page which you can leave as is

Click **Next**

This will direct you to the Configure stack options page which you can leave as is

Click **Next**

This will direct you to the Review page which you can leave as is

Scroll down and click the **Update stack** button at the bottom of the page

Status will be CREATE_IN_PROGRESS

Wait a minute and then click the **refresh** icon in the Events section of the CloudFormation console.

You will notice that it is creating a security group but you will also notice that it is creating a brand new EC2 instance. Once that EC2 instance is created, it will delete our original EC2 instance. Go to the EC2 console to see that this is indeed what occurred. So click **Services** and click **EC2** and then click **Running instances**. You will see the new instance and you will also see the original instance has been terminated.

✓Select the new instance and then scroll down to view the description. You will see that this new instance has the security group called SimpleStack-MySecurityGroup as per the template we created. Click the **view inbound rules** button and you will see it is open on Port 22

CloudFormation automatically figured out the dependencies and calculated the order that the resources needed to be created in. So it created the security group first. Then created the new EC2 instance and deleted the original. Mission accomplished!

CloudFormation Features: Pseudo Parameters

Parameters that are predefined by CloudFormation and do **not** need to be declared in your template.

They are similar to Environment variables that you can rely upon to exist and be set correctly

To reference pseudo parameters, you need to use the Ref Instrinsic Function

Here is how it will look in JSON format:

```
{
  “Resources”:{
    “MySecurityGroup”:{
      “Type”:”AWS::EC2::SecurityGroup”,
      “Properties”:{
        “GroupDescription”:{
          “Ref”:”AWS::Region”
        }
      }
    }
  }
}
```

Here you can see the definition of the security group. We are setting the “GroupDescription” to call the intrinsic function “Ref” and the pseudo parameter “AWS::Region”

Here is how it looks in YAML format:

```
Resources:
  MySecurityGroup:
    Type:
      ‘AWS::EC2::SecurityGroup’
    Properties:
      GroupDescription:
        !Ref AWS::Region
```

Similar to the JSON format, the YAML format has the GroupDescription set to the AWS::Region using the Ref intrinsic function.

There are plenty of pseudo parameters available in CloudFormation. Let's cover the most common ones here.

AWS::AccountId
    Returns the AWS account ID of the account

AWS::NotificationARNs
    Returns the list of notification ARNs for the current stack

AWS::StackID
    Returns the ID of the stack

AWS::StackName
    Returns the name of the stack

AWS::Region
    Returns a string representing the AWS Region in which the resource is being created

The AWS documentation has a list of CloudFormation pseudo parameters at the following web address:

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/pseudo-parameter-reference.html

---

Lab: Pseudo Parameters

Per the last tutorial/lab our template, which we saved as MultipleResources.yaml, looked like the following:

```
Resources:

  Ec2Instance:

    Type: 'AWS::EC2::Instance'

    Properties:

      InstanceType: t2.micro

      ImageId: ami-43874721 # Amazon Linux AMI in Sydney

      Tags:

        - Key: "Name"

          Value: !Join [ " ", [ EC2, Instance, with, Fn, Join ] ]

      SecurityGroups:

        - !Ref MySecurityGroup
```

```
MySecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: Enable SSH access via port 22
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: '22'
        ToPort: '22'
        CidrIp: 0.0.0.0/0
```

In the last lab/tutorial we updated our stack called SimpleStack with the above template. Now let’s incorporate a pseudo parameter into our YAML code. See the below 23 lines of YAML code:

```
Resources:
  Ec2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      InstanceType: t2.micro
      ImageId: ami-43874721 # Amazon Linux AMI in Sydney
      SecurityGroups:
        - !Ref MySecurityGroup
      Tags:
        - Key: "Name"
          Value: !Join
            - ""
            - - "EC2 Instance for "
              - !Ref AWS::Region
  MySecurityGroup:
    Type: "AWS::EC2::SecurityGroup"
    Properties:
      GroupDescription: Enable SSH access via port 22
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: "22"
          ToPort: "22"
          CidrIp: 0.0.0.0/0
```

★★★Copy and paste the above 23 lines of YAML code into a blank file in your code editor and save the file as PseudoParameters.yaml

The only thing that we changed is that we modified that Tags section, specifically we changed the Value property on line 11. On line 11 we are using the Join! intrinsic function to join two strings together. The first string on line 13 being "EC2 Instance for " and then on line 14 we want to join the string for the region. So we expect the name tag to be EC2 Instance for ap-southeast-2 ←which is the region we are executing the template in (aka the Sydney region).

Go to the AWS console and click **Services** and click **CloudFormation**

✓SimpleStack ← select the stack that we created in the last lab/tutorial called SimpleStack

Then click the **Update** button

Prepare template: ✓Replace current template ← select Replace current template

Template Source: ✓Upload a template file ←

Click the **Choose file** button

Choose the file that you saved called PseudoParameters.yaml

Then click **Next**

This will direct you to the Specify stack details page which you can leave as is

Click **Next**

This will direct you to the Configure stack options page which you can leave as is

Click **Next**

This will direct you to the Review page which you can leave as is

Scroll down and click the **Update stack** button at the bottom of the page

Status will be CREATE_IN_PROGRESS

This will be updating the tag with a key name, which will be renamed to EC2 Instance for ap-southeast-2

, which is the Sydney Region.

Let's check this change in the EC2 console. Click **Services** and click **EC2** and then click **Running instances**

Here you will see it has changed the name of our instance to EC2 Instance for ap-southeast-2

So our instance has an updated tag name which we created by utilizing a pseudo parameter in our template. Mission Accomplished!

Mappings

Mappings are a very useful feature in CloudFormation. They enable us to use an input value to determine another value.

For example:

AMI ID's are region specific. We need to determine the AMI ID based on the region that we are running in. CloudFormation provides mappings to solve this!

We will bring together that which we have learned so far including:

1. Pseudo Parameters
2. Intrinsic Functions which in this case will be FindInMap
3. Mappings

Let's look at how we can use the FindInMap function

In JSON format it looks like the following:

{“FN::FindInMap”:[“MapName”,”TopLevelKey”,”SecondLevelKey”]}

We call the FindInMap function similar to the other intrinsic functions.

The first parameter is going to be the logical map name, and the second parameter is the key to filter by.

The third parameter is the key to return from the value object.

In YAML format it looks like the following:

!FindInMap [MapName, TopLevelKey, SecondLevelKey]

And just as in the JSON format we are calling the !FindInMap function. The first parameter is the map name, the second parameter is the key to filter by, and lastly is the property from the value object to return. Let's see how we can use this to solve for our AMI ID for the region that we are working in.

Let's look at the JSON below:

```
 “Mappings”: {
     “RegionMap”: {
       “us-east-1”: { “AMI”: “ami-76f0061f”},
       “us-west-1”: { “AMI”: “ami-655a0a20”}
```

First we need to define the mappings at the top level section of the template.

This is done by firstly defining a logical name for the mapping which we did as “RegionMap”

And inside we defined a list of key-value pairs. In this case the keys were the region, and the values are the AMI properties.

To use the mappings in the resources section, let's pull in the correct AMI for this region by calling the FindInMap function within the image ID property and passing it “RegionMap” as the first parameter (which is the logical name for the mapping) then passing it the region of which we are calling the pseudo parameter “AWS::Region” for. This is done by using the “Ref” intrinsic function and the pseudo parameter “AWS::Region”. The third parameter being called is the name that we would like to pull out a value for and return to us which is “AMI”

```
.
.
   “ImageId”:{
       “Fn::FindInMap”: [“RegionMap”,{“Ref”: “AWS::Region”}, “AMI” ]
    }
.
.
```

Let's look at it altogether. See below:

```
"Mappings": {

   "RegionMap": {

    "us-east-1": { "AMI": "ami-76f0061f"},

    "us-west-1": { "AMI": "ami-655a0a20"}

.

.

  "ImageId":{

    "Fn::FindInMap": ["RegionMap",{"Ref": "AWS::Region"}, "AMI" ]

  }

.

.
```

This FindInMap will find the current region in the mappings by using the pseudo parameter and return the AMI property from the value found in the mappings. Let's see how we can do this in YAML.

```
Mappings:
  RegionMap:
    us-east-1:
      AMI: ami-76f0061f
    us-west-1:
      AMI: ami-655a0a20
```

So the Mappings are defined for two regions with a logical name which we have as RegionMap
Now let's look at what the EC2 section of YAML code.

```
Resources:
  Ec2Instance:
    Type:'AWS::EC2::Instance'
    Properties:
      ImageId: !FindInMap
        - RegionMap
        - !Ref'AWS::Region'
        - AMI
```

The ImageId we execute the !FindInMap function.
The first parameter we pass in is the RegionMap which is the logical name.

The second parameter we are using is the intrinsic function of !Ref and passing in the pseudo parameter ‘AWS::Region’.
And finally, AMI is the property that we would like returned from the mappings. It contains the AMI ID.

Lab:Mappings

In our last lab/tutorial we updated our stack with a template that we saved as PseudoParameters.yaml
In that template we modified the name tag to add in the region pseudo parameter on line 14 of our YAML code. Here again is that template:

```
Resources:
  Ec2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      InstanceType: t2.micro
      ImageId: ami-43874721 # Amazon Linux AMI in Sydney
      SecurityGroups:
        - !Ref MySecurityGroup
      Tags:
        - Key: "Name"
          Value: !Join
            - ""
            - - "EC2 Instance for "
              - !Ref AWS::Region
  MySecurityGroup:
    Type: "AWS::EC2::SecurityGroup"
    Properties:
      GroupDescription: Enable SSH access via port 22
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: "22"
          ToPort: "22"
          CidrIp: 0.0.0.0/0
```

We will now build upon this template to include what we have now learned about Mappings.

★★★ Copy and paste the YAML code below into a brand new file in your code editor and save it as Mappings.yaml

```
Mappings:
  RegionMap:
    us-east-1:
      AMI: ami-1853ac65
    us-west-1:
      AMI: ami-bf5540df
    eu-west-1:
      AMI: ami-3bfab942
    ap-southeast-1:
      AMI: ami-e2adf99e
    ap-southeast-2:
      AMI: ami-43874721
Resources:
  Ec2Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      InstanceType: t2.micro
      ImageId:
        Fn::FindInMap:
        - RegionMap
        - !Ref AWS::Region
        - AMI
      SecurityGroups:
        - !Ref MySecurityGroup
      Tags:
        - Key: "Name"
          Value: !Join
            - ""
            - - "EC2 Instance for "
              - !Ref AWS::Region
  MySecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: Enable SSH access via port 22
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: '22'
          ToPort: '22'
          CidrIp: 0.0.0.0/0
```

In the above YAML code we have created a new top level mapping section. The mapping is logically named RegionMap and we have specified a handful of regions with their Amazon Linux AMI IDs. This allows us to figure out the right AMI ID for the region that we are executing in.

So all we need to do on line 18 for the ImageId property, is reference the appropriate AMI ID for the region that we are running in.

So in order to do this, we utilize the Fn::FindInMap intrinsic function and on line 20 the first parameter we give it is the RegionMap which is the logical name for the mapping (as per line 2 of the YAML code)

and the second parameter we give it is the key that we need to search for and in this case it is the region that we need to search for…so we are using !Ref and then the pseudo parameter AWS::Region

and finally we pass it the property that we would like it to retrieve for us from the value object which is AMI

Therefore, per the Mappings that we created, if we were to use this template in CloudFormation in ap-southeast-2 we will be able to retrieve the AMI Id that is ami-43874721. However, if we were to run it in us-east-1 we will be retrieving the AMI id that is ami-1853ac65.

So CloudFormation uses the Mappings to figure this out at runtime. Let's try it out in the Sydney region.

Go to the AWS Management Console. Click **Services** and then click **CloudFormation**.

✓SimpleStack 🡐 select the stack that we created in the earlier lab/tutorial called SimpleStack

Then click the **Update** button

Prepare template: ✓Replace current template 🡐 select Replace current template

Template Source: ✓Upload a template file 🡐

Click the **Choose file** button

Choose the file that you saved called Mappings.yaml

Then click **Next**

This will direct you to the Specify stack details page which you can leave as is

Click **Next**

This will direct you to the Configure stack options page which you can leave as is

Click **Next**

This will direct you to the Review page which you can leave as is

Scroll down and click the **Update stack** button at the bottom of the page

It will prompt the following error:

There was an error creating this change set
The submitted information didn't contain changes. Submit different information to create a change set.

The reason is that we already have a running instance in the Sydney region with the exact same AMI id. Therefore, there is nothing to change.

So let's switch our Region to N. Virginia and then go to the CloudFormation console by clicking **Services** and then clicking **CloudFormation**.

(By the way, if you were to create a stack in the N. Virginia region using the PseudoParameters.yaml file template from the previous lab/tutorial it would fail. The reason is that it does not have any mappings in it and it has a hardcoded AMI Id from the Sydney Region. In other words, the Sydney AMI id from the template will not be available in the N. Virginia region. It will say CREATE_FAILED and it will say. The image id does not exist.)

Anyways, (in the N. Virginia region) click the **Create stack** button

Prepare template: ✓Template is ready 🡐 select Template is ready

Template Source: ✓Upload a template file 🡐

Click the **Choose file** button

Choose the file that you saved called Mappings.yaml

Then click **Next**

Stack name: NorthVirginiaStack 🡐 give it a name like so

Click **Next**

This will direct you to the Configure stack options page which you can leave as is

Click **Next**

This will direct you to the Review page which you can leave as is

Scroll down and click the **Create stack** button at the bottom of the page

The Mappings dynamically calculates the AMI Id based on the region. So this will successfully create a security group and an EC2 instance with the correct AMI ID. Wait a minute or two and hit the refresh button and it will say CREATE_COMPLETE

Let's inspect the instance in the EC2 management console in the N. Virginia region. Make sure you are in the N. Virginia Region and click **Services** and then click **EC2** and click **Running instances**.

✓Select the instance and click the description tab and you will see the AMI ID is ami-1853ac65 which matches the map for the N. Virginia region (aka us-east-1) in the template we created which looked like the following:

```
Mappings:
  RegionMap:
    us-east-1:
      AMI: ami-1853ac65 ← it's a match
```

So this is how we can utilize mappings and pseudo parameters to calculate AMI IDs dynamically at runtime. Therefore, it is a great way to make a template reusable across regions!!!

CloudFormation Features: Input Parameters

Input Parameters enable us to input custom values to our template, each time we create or update the stack.
They are defined within the top level Parameters section of the template.
Each parameter must be assigned a value at runtime.
  You can optionally specify a default value.

The only required attribute for a parameter is the **Type** attribute, which specifies the data type of the parameter.

The Supported data types available in CloudFormation for parameters are:

String

Number

List<Number> ← (aka a list of numbers)

CommaDelimitedList

AWS-specific types (such as AWS::EC2::Image::Id) ← also known as the AMI ID

Systems Manager Parameter types

Let’s look at what parameters look like and how we can reference them in JSON

```
“Parameters”:{
   “InstTypeParam”:{
        “Type”:”String”,
        “Default”:”t2.micro”,
        “AllowedValues”: [
            “t2.micro”,
            “m1.small”,
            “m1.large”
        ],
       “Description”:
        “EC2 Instance Type”
   }
}
```

In the above JSON, we are defining a parameter which allows us to input the EC2 instance type.

The logical name is the “InstTypeParam” which is short for instance type parameter.

The “Type” is set to “String”

The “Default” is set to “t2.micro”

But we have also provided a list of “AllowedValues” which are “t2.micro”, “m1.small”, & “m1.large”

And lastly we have provided a “Description” which says “EC2 Instance Type”

In order to use this input parameter in the resources section, we would simply need to use the “Ref” intrinsic function. See the JSON below:

```
“Resources”: {
    “Ec2Instance”: {
       “Type”:
          “AWS::EC2::Instance”,
       “Properties”: {
          “InstanceType”: {
             “Ref”: “InstTypeParam”
          },
          “ImageId”:
            “ami-2f726546”
       }
    }
}
```

In the above JSON, on line 7 we are referring to the “InstTypeParam” (short for instance type parameter), which is a logical name for the parameter.

Let’s go over the above example using YAML:

```
Parameters:
  InstTypeParam:
    Type: String
    Default: t2.micro
    AllowedValues:
      - t2.micro
      - m1.small
      - m1.large
    Description:
      EC2 Instance Type
```

In the above YAML, we have defined the logical name for the parameter as InstTypeParam.

We have set the Type as a String. We have set the default as a t2.micro.

We have also provided a list of AllowedValues.

We have also provided a Description.

In order to reference the above Parameter in the Resources section, We would simply need to use the Ref intrinsic function as the InstanceType for our EC2 instance and pass in the logical name for the parameter which is InstTypeParam. See the YAML below:

```
Resources:

  Ec2Instance:

    Type:

      AWS::EC2::Instance

    Properties:

      InstanceType:

        Ref:

          InstTypeParam

      ImageId: ami-2f726546
```

LAB: Input Parameters

In the last lab/tutorial we uploaded a template to CloudFormation in the N. Virginia Region called Mappings.yaml and we named ourstack NorthVirginiaStack. With the template, CloudFormation dynamically figured out the correct AMI ID using the Mappings provided in the template corresponding to the region that we ran it in.

The YAML code looked as follows:

```
Mappings:
  RegionMap:
    us-east-1:
      AMI: ami-1853ac65
    us-west-1:
      AMI: ami-bf5540df
    eu-west-1:
      AMI: ami-3bfab942
    ap-southeast-1:
      AMI: ami-e2adf99e
    ap-southeast-2:
      AMI: ami-43874721
Resources:
  Ec2Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      InstanceType: t2.micro
      ImageId:
        Fn::FindInMap:
        - RegionMap
        - !Ref AWS::Region
        - AMI
      SecurityGroups:
        - !Ref MySecurityGroup
      Tags:
        - Key: "Name"
          Value: !Join
            - ""
            - - "EC2 Instance for "
              - !Ref AWS::Region
  MySecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: Enable SSH access via port 22
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: '22'
          ToPort: '22'
          CidrIp: 0.0.0.0/0
```

Now we will add a new top level section with the knowledge we have learned about input parameters.

Check out the YAML code below:

```
Parameters:
  NameOfService:
    Description: "The name of the service this stack is to be used for."
    Type: String
  KeyName:
    Description: Name of an existing EC2 KeyPair to enable SSH access into the server
    Type: AWS::EC2::KeyPair::KeyName
Mappings:
  RegionMap:
    us-east-1:
      AMI: ami-1853ac65
    us-west-1:
      AMI: ami-bf5540df
    eu-west-1:
      AMI: ami-3bfab942
    ap-southeast-1:
      AMI: ami-e2adf99e
    ap-southeast-2:
      AMI: ami-43874721
Resources:
  Ec2Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      InstanceType: t2.micro
      ImageId:
        Fn::FindInMap:
        - RegionMap
        - !Ref AWS::Region
        - AMI
      SecurityGroups:
        - !Ref MySecurityGroup
      Tags:
        - Key: "Name"
          Value: !Ref NameOfService
      KeyName: !Ref KeyName
  MySecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: Enable SSH access via port 22
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: '22'
          ToPort: '22'
          CidrIp: 0.0.0.0/0
```

Copy and paste the above YAML code into a blank text editor file and save it as InputParameters.yaml

We have introduced a new top level section called Parameters on line 1 through 7. We create two new parameters which are logically named NameOfService and KeyName. The NameOfService we have defined as “The name of the service this stack is to be used for.”

The KeyName is simply a key pair which we want to be able to use to enable SSH access for our server.

Let’s observe how we can reference these for our Resource. Referring to line 34 you can see that for the name tag value we have put !Ref NameOfService. So we are using the input parameter directly using the intrinsic function of !Ref on line 34

For the KeyName, we are bringing that in by using an intrinsic function of !Ref once again . We are utilizing the logical name of KeyName on line 35.

Note the types of parameters. The NameOfService is simply a String. For the KeyName we have defined an AWS datatype which is AWS::EC2::KeyPair::KeyName which essentially gives us a dropdown box for the KeyNames that are available.

Let’s see how this works in action!

In the AWS Management Console select the Sydney region once again, which is where we have our stack called SimpleStack.

First let’s create a key pair name. So in the Sydney region click **Services** and click **EC2**. Click **Key Pairs** in the left margin. Click the **Create key pair** button

Name: Test Key Pair

Select the file format you would like. Choices are pem for OpenSSH and ppk for use with PuTTY. I am going to select ppk.

Then click the **Create Key Pair** button

Now that we have created the Test Key Pair in the Sydney region, click **Services** and Click **CloudFormation** (make sure you are still in the Sydney region)

✓SimpleStack 🡐 select the stack that we created in an earlier lab/tutorial called SimpleStack

Then click the **Update** button

Prepare template: ✓Replace current template 🡐 select Replace current template

Template Source: ✓Upload a template file 🡐

Click the **Choose file** button

Choose the file that you saved called InputParameters.yaml

Then click **Next**

This will direct you to the Specify stack details page

KeyName: ✓ Test Key Pair 🡐 from the drop down menu you can now select the Test Key Pair that we just created

NameOfService: Using input parameters 🡐 type in something like so.

So as you can see these parameters (KeyName & NameOfService) that we created in our template are now available here to be utilized!!!

Click **Next**

This will direct you to the Configure stack options page which you can leave as is

Click **Next**

This will direct you to the Review page which you can leave as is

Scroll down and click the **Update stack** button at the bottom of the page

Now let's monitor the events tab to see what happens. It says the "Requested update requires the creation of a new physical resource; hence creating one."

So the changes to the EC2 instance would be the tag which does not warrant creating a new resource but the key pair does require the creation of a new resource.

Let's switch over to the EC2 management console. Click **Services** and click **EC2** and click **Running instances**.

There we can see that our instance called **EC2 Instance for ap-southeast-2** has been terminated and our new instance called **Using input parameters** is running

✓Select the Using input parameters instance

Then click the **Description** tab and scroll down to where it says Key pair name. And you will see that the Key pair name is called **Test Key Pair** just as we specified.

So now you understand how we utilize input parameters to dynamically change values using the same template to create multiple stacks.

CloudFormation Features: Outputs

Outputs enable us to get access to information about resources within a stack.
For example: Create an EC2 Instance, and output the Public IP or DNS
(In other words, We can use outputs to return the public IP address of an EC2 Instance once it's created)
We can define an outputs section in the template which CloudFormation executes and returns the output values for us to use. (So the stack outputs are returned)
The outputs are defined in the template, there is an outputs section at the top level of the template.

They are defined within the top level Outputs section of the template. And in JSON it will look like the following:

```
“Outputs”: {
   “InstanceDNS”: {
      “Description”:
         “The Instance Dns”,
      “Value”: {
        “Fn::GetAtt”: [
          “EC2Instance”,
          “PublicDnsName”
        ]
      }
   }
}
```

So the “Outputs” section is the top level of the template.
The “InstanceDns” is a logical name for the output definition.
The Description for the output and the value which uses the GetAtt intrinsic function.
This function “Fn::GetAtt” can get an attribute of a resource.
The first parameter is logically named “EC2Instance”
The second parameter is retrieving the attribute “PublicDnsName”
Now the YAML format is very similar. See the YAML below:

```
Outputs:
  InstanceDns:
    Description:
      The Instance Dns
    Value:
      !GetAtt
        - EC2Instance
        - PublicDnsName
```

In the above YAML we have an Outputs section at the top level.

Then we have the InstanceDns which is the logical name for the output.

Then we have the Description.

Then we have the Value, which is using the !GetAtt intrinsic function with the first parameter being EC2Instance which is the logical name for the ec2 instance. And then it retrieves the attribute PublicDnsName.

Let's see this in action.

Lab: Outputs

As a quick review of the last lab/tutorial, we added two input parameters to our template. One was of type String and the other was an AWS::EC2::KeyPair::KeyName. And we were able to reference those input parameters in our resources. Below is the YAML code that you saved as InputParameters.yaml

```
Parameters:
  NameOfService:
    Description: "The name of the service this stack is to be used for."
    Type: String
  KeyName:
    Description: Name of an existing EC2 KeyPair to enable SSH access into the server
    Type: AWS::EC2::KeyPair::KeyName
Mappings:
  RegionMap:
    us-east-1:
      AMI: ami-1853ac65
    us-west-1:
      AMI: ami-bf5540df
    eu-west-1:
      AMI: ami-3bfab942
    ap-southeast-1:
      AMI: ami-e2adf99e
    ap-southeast-2:
      AMI: ami-43874721
Resources:
  Ec2Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      InstanceType: t2.micro
      ImageId:
        Fn::FindInMap:
```

```
        - RegionMap
        - !Ref AWS::Region
        - AMI
      SecurityGroups:
        - !Ref MySecurityGroup
      Tags:
        - Key: "Name"
          Value: !Ref NameOfService
      KeyName: !Ref KeyName
  MySecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: Enable SSH access via port 22
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: '22'
          ToPort: '22'
          CidrIp: 0.0.0.0/0
```

Now we are going to create a new top-level section for output parameters. Copy and paste the below YAML code into a blank text editor file and save the file as OutputParameters.yaml

```
Parameters:
  NameOfService:
    Description: "The name of the service this stack is to be used for."
    Type: String
  KeyName:
    Description: Name of an existing EC2 KeyPair to enable SSH access into the server
    Type: AWS::EC2::KeyPair::KeyName
Mappings:
  RegionMap:
    us-east-1:
      AMI: ami-1853ac65
    us-west-1:
      AMI: ami-bf5540df
    eu-west-1:
      AMI: ami-3bfab942
    ap-southeast-1:
      AMI: ami-e2adf99e
    ap-southeast-2:
      AMI: ami-43874721
Resources:
  Ec2Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      InstanceType: t2.micro
      ImageId:
        Fn::FindInMap:
        - RegionMap
        - !Ref AWS::Region
        - AMI
      SecurityGroups:
        - !Ref MySecurityGroup
      Tags:
        - Key: "Name"
          Value: !Ref NameOfService
      KeyName: !Ref KeyName
  MySecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: Enable SSH access via port 22
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: '22'
          ToPort: '22'
          CidrIp: 0.0.0.0/0
Outputs:
  ServerDns:
    Value: !GetAtt
      - Ec2Instance
      - PublicDnsName
```

In the above YAML code we have created a section called Outputs at the very bottom. Our goal here is to output the server dns name so that we can reference it. We will not know the server DNS name up front as it is spontaneously created when AWS creates the EC2Instance. For this reason, we need to utilize the output section to be able to get that attribute once it is created.

So on line 45 we have created our Outputs section.

On line 46 we created a new logical name for ServerDNS

On line 47 in order to get the Value for that we utilize the !GetAtt aka the get attribute intrinsic function. Then we give it the logical resource name of Ec2Instance and ask it to retrieve the PublicDnsName for that instance.

Go to the AWS management console. Make sure you are in the Sydney region. Click **Services** and then click **CloudFormation**

✓SimpleStack ← select the stack that we created in an earlier lab/tutorial called SimpleStack

Then click the **Update** button

Prepare template: ✓Replace current template ← select Replace current template

Template Source: ✓Upload a template file ←

Click the **Choose file** button

Choose the file that you saved called OutputParameters.yaml

Then click **Next**

This will direct you to the Specify stack details page. Our input parameters will already be set correctly.

So just go ahead and click **Next**

This will direct you to the Configure stack options page which you can leave as is

Click **Next**

This will direct you to the Review page which you can leave as is

Scroll down and click the **Update stack** button at the bottom of the page

The Status will say UPDATE_IN_PROGRESS

Wait about two minutes and then click the **refresh** icon in the Events section.

The Status should say UPDATE_COMPLETE

Technically we have not changed any of the resources within the template. All we asked it to do is to output some new information. So it has not needed to change any of the resources.

Now click the **Outputs** tab (which is near the **Events** tab)

There you will see the Key with our logical name of ServerDns and the Value has the server DNS name outputted from our stack in CloudFormation. You can check in the EC2 console to verify that this DNS name matches the Public DNS name for the EC2 instance. Click **Services** and then click **EC2** and click **Running instances**

✓Select the instance that was created with our stack and you will see that it has that same Public DNS name that was outputted to us by CloudFormation.

Mission accomplished! We were able to request that some information be retrieved from the resources that we created. So that is how we can use outputs from a CloudFormation stack.

That's it for the fundamentals. Make sure to delete you Sydney stack and your N. Virginia stack.