CloudFormation Parameters

Welcome to the next workflow in the IaC workshop using CloudFormation. In this session, we will introduce CloudFormation parameters for creating flexible and reusable templates. *Parameters* allow you to customize your CloudFormation stacks at deployment time without modifying the template itself.

By the end of this session, you will understand how to use parameters effectively for better control of the properties defined for AWS resources.

Introduction to Parameters

Parameters in CloudFormation templates provide a way to pass values to the template at runtime. This enables you to create more flexible and reusable templates that can be adapted to different environments and scenarios without the need for additional edits to the template. The main benefits of using parameters are:

- **Customization**: Parameters allow you to customize resource properties, such as EC2 instance types, S3 bucket names, or VPC IDs, based on your needs.
- **Reusability**: With parameters, you can reuse the same template across different environments (e.g., development, testing, production) by specifying different parameter values.

Your company is satisfied with the bucket created earlier and is actively using it in a production environment. They are now requesting that you create another bucket resource in an environment used to build applications. With the additional bucket, the company will improve its development processes without impact to critical data used for production operations.

With the help of parameters, resource updates can be made without changing the template every time an environment requires a change.

Basic Parameter Definition

Here's an example of a simple parameter definition:

```yaml
Parameters:
  Environment:
    Type: String
    Description: The environment for this stack (e.g., dev, test, prod)
    Default: dev
    AllowedValues:
      - dev
      - test
      - prod
    ConstraintDescription: must be one of the allowed values (dev, test, prod)
```

Explanation of Attributes

- **Type**: Specifies the data type of the parameter. Common types of parameters include:
  - *String*
  - *Number*
  - *List*
  - *CommaDelimitedList*
  - *AWS::EC2::KeyPair::KeyName*

- **Description**: Brief description of the parameter, which is useful for documentation and clarity.
- **Default**: Specifies the default value for the parameter if no value is provided at runtime.
- **AllowedValues**: Restricts the values that can be provided for the parameter to a predefined list.
- **ConstraintDescription**: Describes the constraint when the parameter value does not match the allowed values or types.

Add some parameters in a CloudFormation template

Copy the *s3-cloudformation.yaml* file into a new file (e.g., `s3-cloudformation-3.yaml`) and insert this code before the **Resources** top level key (NOTE: the line starting with *AWSTemplateFormatVersion:* should remain as the top line in the file):

```yaml
Parameters:
   BucketName:
      Type: String
      Description: Name of the S3 bucket
      Default: our-first-s3-bucket-workshop
      AllowedPattern: '^[a-zA-Z0-9.\-_]{1,255}$'
      ConstraintDescription: must be a valid S3 bucket name.
```

We just defined our first template variable! It is a simple string variable to save the bucket name. We will see later how this is used in the **Resources** section.

On line 6, we added some constraints to make sure the bucket name contains only valid characters. These constraints will help to ensure that the execution of the template does not result in an error. Another good use for constraint patterns would be to ensure that values like an IP address are entered correctly.

Reminder that the default bucket name must be globally unique and satisfy several rules:

- Bucket names must contain a minimum of 3 and a maximum of 63 characters.
- Bucket names can consist only of lowercase letters, numbers, dots (.), and hyphens (-).
- Bucket names must begin and end with an alphanumeric character (i.e., letter or number).

Let's add an *Environment* parameter to the CloudFormation script after the BucketName declarations. The entire section should look similar to the following:

```yaml
Parameters:
   BucketName:
      Type: String
      Description: The name of the S3 bucket
```

```
5         Default: our-first-s3-bucket-workshop
6         AllowedPattern: '^[a-zA-Z0-9.\-_]{1,255}$'
7         ConstraintDescription: must be a valid S3 bucket name.
8       Environment:
9         Type: String
10        Default: development
11        AllowedValues:
12         - development
13         - production
```


With this parameter, we are allowed to have only two possible values: one for *production* and one for *development*.

> **Intrinsic Functions**
> Intrinsic functions in AWS CloudFormation are built-in functions that help you manage and customize your stacks. They enable you to perform various tasks such as referencing parameters, resource properties, or conditionally including/excluding resources. The **!Ref** function is one of the most commonly used intrinsic functions. It allows you to reference the value of a parameter or the physical ID of a resource.
>
> For example, using **!Ref** with a parameter returns the value provided for that parameter at runtime, making templates more dynamic and reusable.

Using the information above, we can now establish a new format for the Resources. The below example is for illustrative purposes and *should not* be copied into the new template file:

```
Resources:
    S3Bucket:
        Type: 'AWS::S3::Bucket'
        Properties:
            BucketName: !Ref BucketName
            BucketEncryption:
                ServerSideEncryptionConfiguration:
                - ServerSideEncryptionByDefault:
                    SSEAlgorithm: AES256
            VersioningConfiguration:
                Status: Enabled
```

As you can see on line 5 we referenced the parameter to change the bucket name dynamically. Let's include the environment as well to make sure the bucket name is unique. Copy this snippet of code into the file replacing the current `Resources:` section:

```
Resources:
    S3Bucket:
        Type: 'AWS::S3::Bucket'
        Properties:
            BucketName: !Sub "${BucketName}-${Environment}"
            BucketEncryption:
                ServerSideEncryptionConfiguration:
                - ServerSideEncryptionByDefault:
                    SSEAlgorithm: AES256
            VersioningConfiguration:
                Status: Enabled
```

Here it gets more interesting. Line 5 now uses the **!Sub** intrinsic function to substitute variables within a string. Here's what each part means:

- **!Sub**: This intrinsic function allows for string substitution. It replaces variables in the format *${VariableName}* with their corresponding values.
- **${BucketName}**: This is a reference to the parameter BucketName. It will be replaced by the actual value provided for BucketName.
- **${Environment}**: This is a reference to the parameter Environment. It will be replaced by the actual value provided for Environment.

Putting it all together, `!Sub "${BucketName}-${Environment}"` dynamically constructs a bucket name by combining the values of BucketName and Environment with a hyphen in between.

Deploy the changes in the CloudFormation template

Follow the steps described in the previous workstream to deploy the new updates in the above the template. For this execution, CloudFormation will prompt you to add values for the two parameters we created. Proceed by executing the same stack instructions performed previously. Be sure to provide new names for the S3 bucket and CloudFormation stack to avoid any errors.

NOTE: If you are using the management console, add the parameters in proper fields in the **Specify stack** section. If using the AWS CLI, create a new json file called *parameters.json*, and populate the file with the following code, remembering to use a globally unique name for your S3 bucket's *ParameterValue* variable:

```
[
    {
        "ParameterKey": "BucketName",
        "ParameterValue": "<our-third-s3-bucket-workshop>"
    },
    {
        "ParameterKey": "Environment",
        "ParameterValue": "development"
    }
]
```

Save the above JSON content into a file named *parameters.json* in the same directory as your template file.

Finally, use the following AWS CLI command to create a stack named *S3Stack-3* using the template file (*s3-cloudformation-3.yaml*) and the parameters file (*parameters.json*):

```
aws cloudformation create-stack --stack-name S3Stack-3 --template-body file://s3-cloudformation-3.yaml --parameters file://parameters.json
```

The only difference with the previous command is the argument **--parameters** *file://parameters.json* that points to the location of the JSON file containing the parameter values.

Conclusion of workflow

You have successfully completed another workstream using CloudFormation!

What did you accomplish in this workflow?
This workflow focused on how to use parameters in CloudFormation templates. Parameters can help with the reuse of code to manage multiple environments.

Additionally, we introduced two common intrinsic functions called **!Ref** and **!Sub**. For more information on intrinsic functions used in CloudFormation, reference the AWS CloudFormation User Guide.

Previous Next