

Cloud Services 2

Configuration Management in the
cloud & Introduction to
Cloudformation





Operations course: Module 11 - Creating Automated and Repeatabe Deployments

Module 11: Creating Automated and Repeatable Deployments

Configuration management in the cloud



AWS
CloudFormation

AWS CloudFormation:

- Models and provisions cloud infrastructure resources
- Supports most AWS services
- Creates, updates, and deletes a set of resources as a single unit called a stack
- Detects drift on stack and individual resources



Change set

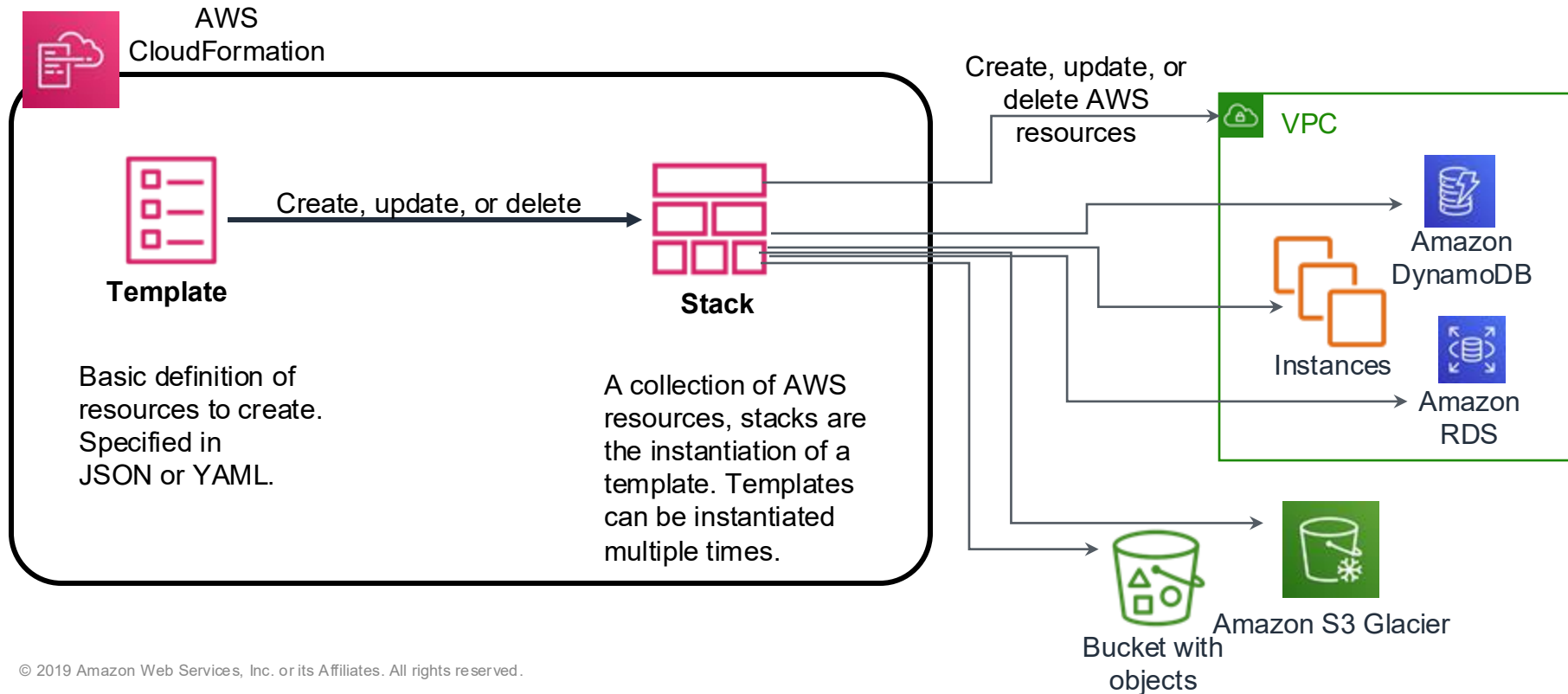


Stack



Template

AWS CloudFormation terminology



Template structure



Template

■ Parameters

■ Mappings

■ Resources

◆ AWS::EC2::Instance

- + Properties/UserData

- + Metadata/AWS::CloudFormation : **Init**

- Packages
- Groups
- Users
- Sources
- Files
- Commands
- Services
-

◆ AWS::CloudFormation::WaitCondition

- DependsOn: EC2 Instance

◆ AWS::CloudFormation::WaitConditionHandle

■ Outputs

Parameters: Inputs into template

Mappings: Static variables (typically latest AMIs)

Resources: AWS assets to create

Init: Custom applications to run during startup

WaitCondition: Signal from instance that user data has completed running

Outputs: Values of custom resources created by template (URLs, usernames, etc.)

Launching and deleting stacks:

- AWS CloudFormation templates can be launched as *stacks* via:
 - AWS Management Console
 - AWS CLI
 - AWS API
- If an error is encountered when you launch a template, all resources are rolled back by default
- When stacks are deleted, resources are rolled back
 - You can optionally enable termination protection on a stack

Define parameters in a template

To **define parameters** in a template:

- Use the optional **Parameters** section to customize your templates
- Parameters enable you to input custom values to your template each time you create or update a stack.

```
"Parameters": {  
  "VPCCIDR": {  
    "Description": "CIDR Block for VPC",  
    "Type": "String",  
    "Default": "10.200.0.0/20",  
    "AllowedValues": [ "10.200.0.0/20" ]  
  }  
}
```

Logical name (unique within the template)

Must specify a supported data type (one of String, Number, List, CommaDelimitedList, AWS-specific parameter, or SSM parameter)

*Example
Parameter*

Reference a parameter

To reference a parameter:

- Use the **Ref** intrinsic function to reference a named parameter
- Converts to string, regardless of type in **Parameter** declaration

Parameter

```
"KeyName": { "Type":  
  "AWS::EC2::KeyPair::KeyName" }
```

**Reference
a Parameter**

```
"KeyName": { "Ref": "KeyName" },
```

- Use the **Fn::Select** function to select values from a comma-delimited list

```
"AvailabilityZone" : { "Fn::Select" : [ "0", { "Ref" : "AvailableAZs" }  
  ] }
```

Ref and other intrinsic functions

The **Ref** function:

- Enables referencing components that are defined in an AWS CloudFormation template
- Is necessary for:
 - Referencing parameters
 - Using maps
 - Joining strings
 - Using other functions

```
"MyEIP" : {  
    "Type" : "AWS::EC2::EIP",  
    "Properties" : {  
        "InstanceId" : { "Ref" : "MyEC2Instance"  
    }  
}
```

Additionally:

- You can use the **Ref** intrinsic function to access information about your runtime environment
 - Examples: Region, Stack Name, AWS Account ID
- Pseudo parameters are predefined: no need to specify them in the **Parameters** section of your template

```
"Outputs" {  
    "MyStacksRegion" : { "Value" : { "Ref" : "AWS::Region" } }  
}
```

Define mappings in a template

To **define mappings** in the template:

- Define lookup tables of name-value pairs in a two-level map
- Combine multiple mapping tables together to create nested lookups
- Is used most commonly to look up current AWS AMI values

Mapping example

```
"Mappings" : {  
  "AWSRegionToAMI" : {  
    "us-east-1" : { "AMI" : "ami-76817c1e" },  
    "us-west-2" : { "AMI" : "ami-d13845e1" },  
    "us-west-1" : { "AMI" : "ami-f0d3d4b5" },  
    "eu-west-1" : { "AMI" : "ami-892fe1fe" }, ...  
  }  
}
```

```
"ImageId" : {  
  "Fn::FindInMap" : [  
    "AWSRegionToAMI",  
    { "Ref" : "AWS::Region" },  
    "AMI"  
  ]  
}
```

Use `Fn::FindInMap`
intrinsic function for lookups

Define resources in a template

The **Resources section** declares the AWS resources to create.

The type of resource you are declaring

You can require the creation of a specific resource to follow another

Properties for a resource (optional)

```
"Resources" : {  
  "MyRDSInstance" : {  
    "Type" :  
      "AWS::RDS::DBInstance",  
    "Properties" : {  
      "AllocatedStorage" :  
  
        "5",  
  
      ...  
    }  
  },  
  "WebServer" : {  
    "Type" : "AWS::EC2::Instance",  
    "DependsOn" : [ "MyRDSInstance"  
    ],  
    "Properties" : {
```

- **CloudFormation::Init:**
 - Resource type that provides access to metadata from EC2 instances
 - You can use it in combination with the cfn-init helper script
- **cfn-init** helper script:
 - It reads template metadata from the AWS::CloudFormation::Init key
 - With this information, cfn-init can do the following actions on EC2 instances:
 - Install packages
 - Manage groups and user accounts
 - Write files to disk
 - Run commands
 - Enable or disable and start or stop services

User data vs. CloudFormation::Init



User data in AWS

CloudFormation:

- Script file, as with individual instances.
- Offers greater control.
- Offers greater potential for error. Must be more cautious.

CloudFormation::Init:

- Can roll back automatically on failure.
- Is cleaner.
- Can attach metadata within the template.
- Configsets

WaitCondition and WaitConditionHandle



The **WaitCondition** and **WaitConditionHandle** are used when you bootstrap instances. Additionally:

- **WaitCondition** blocks the completion state of a stack until **WaitConditionHandle** is called or the timeout limit is reached
 - You can specify the number of successful signals that must be received before WaitCondition is fulfilled
 - Default: 1
- You can signal success or failure by using the **cfn-signal** command

WaitCondition and WaitConditionHandle



Specifying a WaitConditionHandle

```
"WebServerWaitHandle": {
  "Type":
  "AWS::CloudFormation::WaitConditionHandle"
},
"WebServerWaitCondition": {
  "Type":
  "AWS::CloudFormation::WaitCondition",
  "DependsOn": "WebServerInstance",
  "Properties": {
    "Handle": {
      "Ref": "WebServerWaitHandle"
    },
    "Timeout": "3600"
  }
}
```

Signaling a WaitCondition

```
"UserData": {
  "Fn::Base64": {
    "Fn::Join": [ "", [
      "#!/bin/bash\n",
      "/opt/aws/bin/cfn-signal -e 0 -r\n\"complete\" \"\",
      {
        "Ref": "WebServerWaitHandle"
      }, "\"\n"
    ]
    ]
  }
}
```

Define outputs in a template

Outputs:

- Return values for resources created as part of stack
- Use intrinsic functions to obtain values of resources in stack

```
"Outputs" : {  
  "WebSiteURL" : {  
    "Description" : "The URL of the website",  
    "Value" : { "Fn::Join" : [ "", [ "http://", {  
      "Fn::GetAtt" : [ "ElasticLoadBalancer", "DNSName" ] } ] ] }  
  }  
}
```

Additional AWS CloudFormation stack options



Additional options can be specified when you create an AWS CloudFormation stack, such as :

- Preventing *rollback on failure*
- Setting a *stack policy* to control stack updates
- Enabling termination protection

```
aws cloudformation create-stack --stack-name NewStack  
--template-body file://path/to/template.yml --on-failure  
DO_NOTHING
```

Override for failed update rollbacks

- Continue rolling back an update to your stack, even after rollback has failed and is in the **UPDATE_ROLLBACK_FAILED** state
- Perform an override:
 - Remedy the cause of the failed rollback
 - Instruct AWS CloudFormation to continue rolling back the update

```
aws cloudformation continue-update-rollback --stack-name  
ExistingStack
```

Planning and organizing:

Organize your stacks by lifecycle and ownership.

Reuse templates to replicate stacks in multiple environments.

Example: Development, testing, and production environments.

Verify limits for all resource types.

Creating templates:

Do not embed credentials in your templates.

Use AWS-specific parameter types.

Use parameter constraints.

Use `AWS::CloudFormation::Init` to deploy software applications on Amazon EC2 instances.

Validate templates before you use them.

Managing stacks:

Manage all stack resources through AWS CloudFormation.

Create change sets before updating your stacks.

Use stack policies.

Use AWS CloudTrail to log AWS CloudFormation calls.

Use code reviews and revision controls to manage your templates.

leermaterialen & labs



https://cloudexpert.dubbadub.be/#/6_cloudformation

