



AWS CloudFormation

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

This module describes the AWS CloudFormation service.

What you will learn

At the core of the lesson

You will learn how to:

- Describe the purpose of AWS CloudFormation.
- Create an AWS CloudFormation template.
- Use AWS CloudFormation best practices.

Key terms:

- AWS CloudFormation template
- AWS CloudFormation stack



In this module, you will learn how to:

- Describe the purpose of AWS CloudFormation.
- Create a AWS CloudFormation template.
- Use AWS CloudFormation best practices.



The challenge of cloud deployment

The challenge of cloud deployment.

Before jumping into AWS CloudFormation, let's discuss the potential issues that could come up when deploying in cloud environments.

Cloud deployment challenges

Cloud deployment challenges include:

Deploying rollouts across multiple geographical locations

Debugging deployments

Documenting all changes

Updating live servers

Having the ability to manage a rollback

Managing dependencies on systems and subsystems

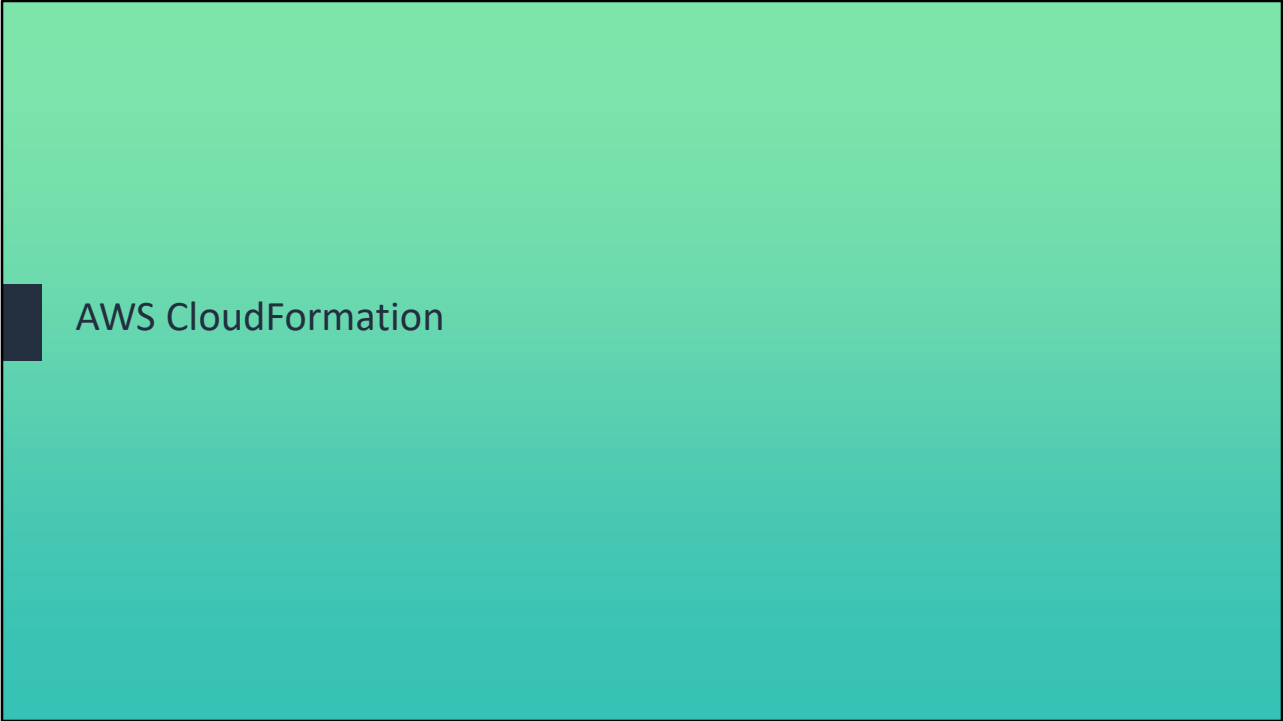
Deploying repeatable and identical environments

4

aws re/start

The cloud opens up many possibilities. However, it also raises questions about how to manage its power and flexibility, such as:

- How do you update servers that have already been deployed into a production environment?
- How do you consistently deploy an infrastructure to multiple Regions in disparate geographical locations?
- How do you roll back a deployment that did not run according to plan? In other words, how do you reclaim the resources that were already created?
- How do you test and debug a deployment before you roll it out to production?
- How do you manage dependencies on systems and technologies, and also on entire subsystems (for example, a website that is deployed on top of an ecommerce infrastructure)?

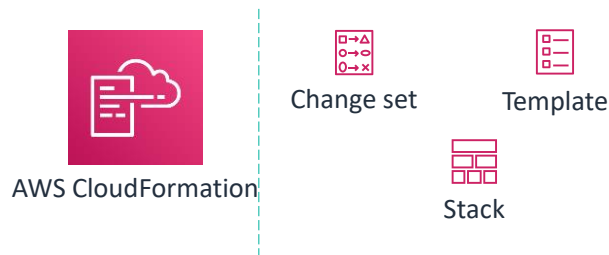


AWS CloudFormation

Now that we've reviewed potential challenges, keep those in mind as we learn about AWS CloudFormation and think about how some of those challenges might be addressed.

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 changes, called “drift”, on stack and individual resources



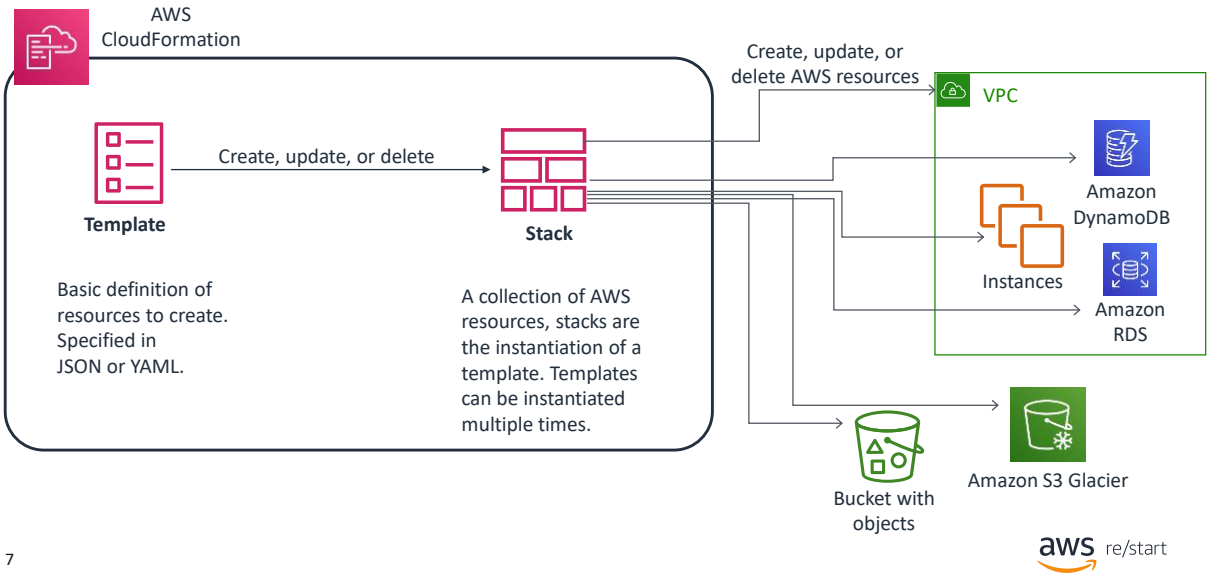
AWS CloudFormation enables you to create and provision AWS infrastructure deployments predictably and repeatedly. You can use AWS CloudFormation on services such as Amazon Elastic Compute Cloud (Amazon EC2), Amazon Elastic Block Store (Amazon EBS), Amazon Simple Notification Service (Amazon SNS), Elastic Load Balancing, and Auto Scaling. AWS CloudFormation enables you to use a template file to create and delete a collection of resources, which are managed together as a single unit (a *stack*).

Though you manage your resources through AWS CloudFormation, users can change those resources *outside* of AWS CloudFormation. Users can edit resources directly by using the underlying service that created the resource. For example, you can use the Amazon EC2 console to update a server instance that was created as part of an AWS CloudFormation stack. Some changes might be accidental, and some might be made intentionally to respond to time-sensitive operational events. Regardless, changes that are made outside of AWS CloudFormation can complicate stack update or deletion operations.

You can use *drift detection* to identify which stack resources have configuration changes that were made outside of AWS CloudFormation management. You can then take corrective action so that your stack resources are synchronized again with their definitions in the stack template. For example, you could update the drifted resources directly so that they agree with their template definition. Resolving drift helps to

ensure the consistency of your configuration and the success of your stack operations.

AWS CloudFormation terminology



7

Two major terms for AWS CloudFormation are *templates* and *stacks*.

A *template* is a specification of the AWS resources to be provisioned.

A *stack* is a collection of AWS resources that were created from a template. You might provision (create) a stack many times.

When a stack is *provisioned*, the AWS resources that are specified by the stack template are created. Any charges incurred from using these services will start accruing when they are created as part of the AWS CloudFormation stack.

When a stack is *deleted*, the resources that are associated with that stack are deleted. The order of deletion is determined by AWS CloudFormation. You do not have direct control over what gets deleted when.

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 the most recent Amazon Machine Images (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, user names, and others)



8

These elements are the major components of an AWS CloudFormation template.

Parameters is an *optional* section of the template. When parameters are present, they include name-value pairs. These parameters can then be referenced later in the template.

Mappings is an *optional* section of the template. Mappings are typically used to refer to the values of the most current Amazon Machine Images (AMIs) in a Region because they differ by Region. They also change over time as new AMIs are released.

Resources is a *required* section of the template. The **Resources** section contains the AWS objects that you will create. You can create resources across a large number of AWS services, including Amazon EC2, Amazon Simple Storage Service (Amazon S3), Amazon Virtual Private Cloud (Amazon VPC), and many others. You can specify relationships and dependencies between resources to ensure that resources are created in the correct order.

Init. The CloudFormation::Init resource type enables you to deploy applications, files, and other resources onto your EC2 instances as part of the deployment process.

WaitCondition. The CloudFormation::WaitCondition resource type is used to coordinate resource creation. For example, suppose that a resource has a dependency on another resource. AWS CloudFormation will wait for other resource creation activities to complete before it attempts to create the dependent resource. WaitConditions enable you to signal back to AWS when your CloudFormation::Init commands finish running successfully.

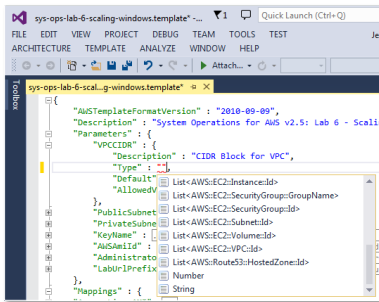
Outputs is an *optional* section of the template. The **Outputs** section returns string values that are created by the template and that might be important to users. For example, if you create an EC2

instance that functions as a public web server, you might choose to output the full public Domain Name Server (DNS) name of the server in the **Outputs** section.

Edit AWS CloudFormation templates

Use an editor that is compatible with AWS CloudFormation—or a JSON or YAML editor—to use features such as **parsing capabilities**, **automatic completion**, and **syntax checking**.

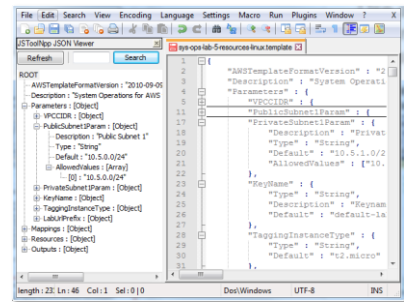
AWS CloudFormation template editor for Visual Studio or Eclipse



Third-party tools



Text editor with JavaScript or JSON plugins



aws re/start

9

Though the JSON format has many positive attributes, one of its disadvantages is that you might edit your way into a syntax error that you cannot easily resolve. If you remove one brace early in a JSON document, you could spend the next hour trying to find the source of the error. Using an editor that is compatible with AWS CloudFormation (or a JSON-aware editor) simplifies the task of editing templates. Similarly, YAML documents must be carefully written by keeping spaces in mind.

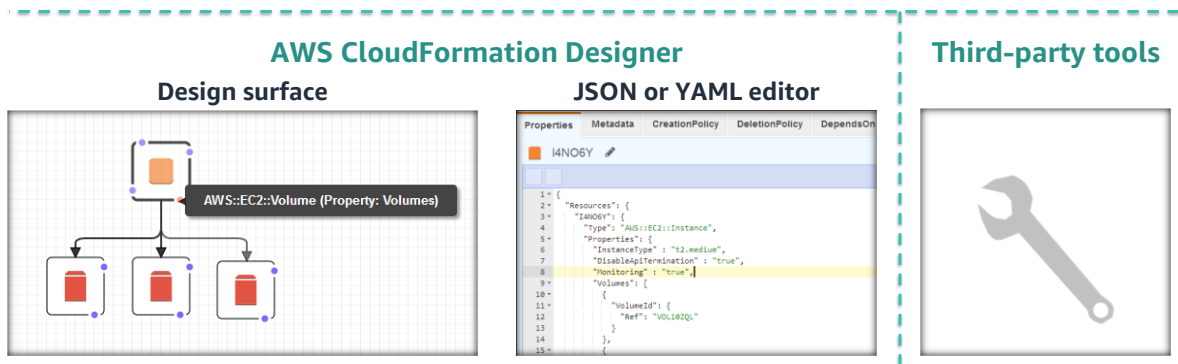
You can install the AWS Toolkit on Visual Studio or Eclipse, which supports editing AWS CloudFormation templates. The toolkit adds support for AWS CloudFormation to both editors. Support includes parsing and formatting, automatically completing AWS CloudFormation commands and keywords, and syntax checking. If you prefer working in a text editor (such as Sublime Text or Notepad++), most popular text editors have JSON or YAML plugins that you can use to format your code.

For more information about editing AWS CloudFormation templates in Visual Studio, refer to [Using the AWS CloudFormation Template Editor for Visual Studio](#).

For more information about editing AWS CloudFormation templates in Eclipse, refer to [The AWS CloudFormation Template Editor](#).

Design templates

AWS CloudFormation Designer is a visual tool that enables you to create and modify AWS CloudFormation templates by using a drag-and-drop interface.



AWS CloudFormation Designer is a visual tool that enables you to create and modify AWS CloudFormation templates by using a drag-and-drop interface. You can add, modify, or remove resources and the underlying JSON or YAML will be altered accordingly. If you modify a template that is associated with a running stack, you can update the stack so that it conforms to the template.

Third-party tools might also provide solutions to help you design your AWS CloudFormation templates.

Launch and delete stacks

- AWS CloudFormation templates can be launched as stacks through:
 - AWS Management Console
 - AWS CLI
 - AWS APIs
- 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

If an error is encountered when you launch an AWS CloudFormation template, all resources are rolled back by default. You can change this option from the command line.

AWS CloudFormation ensures that all stack resources are created or deleted as appropriate. Because AWS CloudFormation treats the stack resources as a single unit, they must all be created or deleted successfully for the stack to be created or deleted. If a resource cannot be created, AWS CloudFormation rolls back the stack and automatically deletes any resources that it created. If a resource cannot be deleted, any remaining resources are retained until the stack can be successfully deleted.

- To preserve an Amazon Elastic Block Store (Amazon EBS) volume, set its *DeleteOnTermination* attribute to **False**.
- To preserve a resource, set its *DeletionPolicy* attribute to **Retain**.

Some resources might not be deleted when a stack is deleted, such as an S3 bucket.

You can also 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 that 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" ]  
  }  
}
```

Example parameter

Logical name (unique within the template)

Must specify a supported data type (one of *String*, *Number*, *List*, *CommaDelimitedList*, a parameter that is specific to AWS, or an AWS Systems Manager parameter)

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

In the example, the parameter was given the *logical name* of *VPCCIDR*. The logical name that is given must be unique throughout the entire template.

AWS CloudFormation parameters support the following *data types*: strings, numbers, comma-delimited lists, parameter types that are specific to AWS, and AWS Systems Manager parameter types.

Parameter types that are specific to AWS are particularly useful. For example, when the stack is launched from the console, a parameter that you enter as `AWS::EC2::KeyPair::KeyName` will render a dropdown list that includes all key pairs in the account.

For more details, refer to the [Parameters](#) page in the AWS CloudFormation documentation.

Reference a parameter

To reference a parameter:

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

Parameter

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

Reference a parameter

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

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

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

After you define a parameter in the template, you can reference the parameter by using the `Ref` intrinsic function. When you use `Ref`, AWS CloudFormation uses the parameter's value to provision the stack.

Parameters can be referenced from the **Resources** and **Outputs** sections of the same template. The example defines a parameter that is named `KeyPairName`. This parameter is referenced later, in the **Resources** section of the AWS CloudFormation template. In this case, the data type of the parameter is `AWS::EC2::KeyPair::KeyName`. When you decide that you want to create a stack from this template, the AWS Management Console will render a dropdown list with all key pairs in the account. You must select a key pair from the list before you create the stack.

Sometimes, you might want to refer to a specific value in a list. In these cases, you can use the `Fn::Select` intrinsic function in the **Resources** section of your template. `Fn::Select` returns a single object from a list of objects by index. In the example, the first value stored in the list of `AvailableAZs` will be returned.

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" }  
  }  
}
```

You might encounter one question often when you write templates: How do you refer to one element of a template from another element? For example, how do you associate an EC2 instances with an Elastic IP address that is defined within the same template?

In the example, you can see the use of the most common intrinsic function, *Ref*, in an AWS CloudFormation snippet that defines an Elastic IP address. Using the Ref function enables you to reference an EC2 instance that is defined within the template itself, which establishes a chain of relationships between your resources. A similar pattern can be used to associate security groups with resources, route tables with routes, and EC2 instances with subnets.

Be careful when you use the Ref function. The value that is returned by Ref is dependent on the resource that is supplied to it. For example, using the Ref function on an S3 bucket returns the bucket's name, but using it on an Amazon Simple Queue Service (Amazon SQS) queue returns the queue's URL. Ensure the resource that you reference with the Ref function returns the correct identifier in the current context. In the example, the Ref function returns the instance ID of an EC2 instance, and it is appropriate to assign it to the *InstanceId* property of the *AWS::EC2::EIP* object.

For a list of the values that are returned by various AWS CloudFormation resources, refer to the [Ref page](#).

To select other attributes on AWS resources that are defined in your AWS CloudFormation template, use the Fn:GetAtt intrinsic function. For more information, refer to the [Fn::GetAtt page](#).

For more details on intrinsic functions in general, refer to [Intrinsic function reference](#).

Pseudo parameters

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, so you do not need to specify them in the **Parameters** section of your template

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

The `Ref` intrinsic function is also often used to access information about your runtime environment, such as the Region, the stack name, or the AWS account ID. These runtime environment values are available through pseudo parameters.

Pseudo parameters are parameters that are predefined by AWS CloudFormation. You do not declare them in your template. You can use them the same way you would use any other parameter: as the argument for the `Ref` function.

In the example, `AWS::Region` is a pseudo parameter. It will resolve to the Region where the AWS CloudFormation template is being run to create the stack.

For more information about pseudo parameters, refer to [Pseudo parameters reference](#).

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 Amazon Machine Image (AMI) values

Recall that one of *optional* sections of the template is the **Mappings** section. A mapping matches a key to a corresponding set of named values. For example, suppose that you want to set values based on a Region. You can create a mapping that uses the Region name as a key, and that also contains the values that you want to specify for each specific Region. You use the Fn::FindInMap intrinsic function to retrieve values in a map.

Mappings are typically used to refer to the AMI IDs of the most current AMIs in a Region because AMIs differ by Region. They also often change over time as new AMIs are released.

You cannot include parameters, pseudo parameters, or intrinsic functions in the **Mappings** section.

For more information, refer to [Mappings](#).

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

The example here shows a **Mappings** section with a map that is named *AWSRegionToAMI*. It contains four keys that map to name-value pairs. The name-value pairs contain single string values.

The keys are Region names. Each name-value pair is the AMI ID in the Region that is represented by the key.

The next snippet of JSON shows how the intrinsic function `Fn::FindInMap` could be used to look up the correct AMI ID that is stored in the map. It requires the *AWS::Region* pseudo parameter, which resolves to an actual Region.

For more information, refer to [Mappings](#).

Define resources in a template

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

The type of resource that you are declaring

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

Properties for a resource (optional)

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

The required **Resources** section of an AWS CloudFormation template declares the AWS resources that you want as part of your stack, such as an EC2 instance or an S3 bucket. You must declare each resource separately. However, you can specify multiple resources of the same type. If you declare multiple resources, separate them with commas.

In this example, two resources will be created: an Amazon Relational Database Service (Amazon RDS) instance and an EC2 instance.

The EC2 instance—which is named *WebServer*—has a dependency on the RDS instance. The database must be created before the applications that will run on the web server can use it. Although the AWS CloudFormation service performs some sequencing of resource creation tasks automatically, these rules change as support for new resource types is rolled out through the system. To help ensure a consistent sequencing of resource creation tasks, it is a best practice to use the *DependsOn* attribute.

For more information, refer to [Resources](#).

CloudFormation::Init

- Resource type that provides access to metadata from EC2 instances
- You can use it with the *cfn-init* helper script.
- **cfn-init** helper script:
 - 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 services
 - » Start or stop services

In the cfn-init helper script, use the `AWS::CloudFormation::Init` resource type to include metadata about an EC2 instance. If your template calls the cfn-init script, the script looks for resource metadata that is rooted in the `AWS::CloudFormation::Init` metadata key. For more information about cfn-init, refer to [cfn-init](#).

cfn-init supports all metadata types for Linux systems. It supports metadata types for Microsoft Windows with conditions that are described in the following sections. For an example of using `AWS::CloudFormation::Init` and the cfn-init helper script, see [Deploying applications on Amazon EC2 with AWS CloudFormation](#).

For an example that shows how to use cfn-init to create a Microsoft Windows stack, refer to:

- [Bootstrapping AWS CloudFormation Windows Stacks](#)

For Amazon Elastic Container Service (Amazon ECS) instances, start with the Amazon ECS-optimized AMI that provided by AWS. Then, use CloudFormation Init to provision the packages, run commands, install files, and configure the servers as you need them. AWS recommend this process because it has several benefits—for example, it is both declarative and updatable. `AWS::CloudFormation::Init` is configured by running

cfn-bootstrap scripts on the instance.

```
"MyInstance": {
  "Type": "AWS::EC2::Instance",
  "Metadata" : {
    "AWS::CloudFormation::Init" : {
      "webapp-config": {
        "packages" : {},
        "sources"   : {}, "files"       : {},
        "groups"    : {}, "users"       : {},
        "commands" : {}, "services"    : {}
      }
    }
  }
}
```

For more information, refer to the [AWS::CloudFormation::Init](#) page.

User data versus 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.
- Easier to manage than user data fields in CloudFormation templates
- Can attach metadata within the template
- **Configsets** allow multiple configs in a single CloudFormation::Init

To configure an instance, you must choose to use either CloudFormation::Init or user data.

User data gives you greater control over the order that your operations run. It also provides the opportunity for more robust error handling and error recovery because user data scripts are written in programming languages with loops and support for conditionals.

One disadvantage of using user data in an AWS CloudFormation template is that any special characters—such as double quotation marks—must be escaped within the template for the file to parse as valid JSON or YAML code. It can be challenging to escape your user data so that it parses cleanly as JSON or YAML code.

One strategy is to place all your user data code into a separate script that AWS CloudFormation runs with a CloudFormation::Init command. This solution is easier to maintain than embedding programming code in your templates. For ease of development and maintenance, it is a good idea to experiment with different methods of initializing EC2 instances via AWS CloudFormation, and standardize on a uniform method that is used across your organization.

You can create multiple configsets, and call a series of them by using your cfn-init script. Each configset can contain a list of config keys or references to other

configsets. Configsets are used with the `cfn-init` commands to enable easier reuse of code reuse. For example, you can set a default configset for all web servers to install Apache and PHP, instead of building out the user data scripts to install them each time, or building it into an AMI.

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

For AWS, an EC2 instance is ready when it passes its status checks. However, passing status checks might not mean that your instance is ready in terms of the installations and configurations that you expect the EC2 instance to have. Your user data and scripts might still be installing software that makes your instance into a database server, a SharePoint server, or a node in a data processing cluster.

To signal to AWS CloudFormation that your instance is ready, you must use a *WaitCondition*. Think of a *WaitCondition* as a stop sign that prevents your stack from being marked as complete by AWS CloudFormation before your code sends a signal that it has finished. The *WaitConditionHandle* is a textual representation of the *WaitCondition* that your initialization code can use to signal to AWS CloudFormation that your resource has been created.

Your initialization code is responsible for calling the `cfn-signal` command on the instance to indicate that it has finished its work. This is true whether the code is in user data or in a separate script that is downloaded to the instance by using `CloudFormation::Init`. You can signal that the task was successful. If a task is successful, AWS CloudFormation will either begin creating the next resource in your stack, or it will mark your entire stack as completed. Or, you can mark that the task failed. If a task fails, by default, AWS CloudFormation will roll back any resources in your stack that were created to date.

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 \"complete\" \"\",
      {
        "Ref":
"WebServerWaitHandle"
      }, "\"\n"
    ] ] ] ] ]
}
```

The example section of an AWS CloudFormation template on the left defines an `AWS::CloudFormation::WaitConditionHandle` resource that is named *WebServerWaitHandle*.

Lower down, the same JSON snippet defines an `AWS::CloudFormation::WaitCondition`, which is called *WebServerWaitCondition*. You can see that it waits to receive input on this `WaitConditionHandle`.

The code block on the right is an example **UserData** section of an `AWS::EC2::Instance` resource. It invokes the `cfn-signal` command and it has a reference to the `WebServerWaitHandle`. After the `UserData` script finishes running, the `WaitConditionHandle` will be fulfilled.

Define outputs in a template

Outputs:

- Return values for resources that were 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" ] ] ] ] }
  }
}
```

Outputs is an *optional* section in an AWS CloudFormation template. The **Outputs** section returns string values that were created by the template and that might be important to users. For example, if you create an EC2 instance that functions as a public web server, you might choose to output the full public Domain Name System (DNS) name of the server in the **Outputs** section.

In the example, the DNS name of an Elastic Load Balancing load balancer—which must also be defined in the **Resources** section of the same AWS CloudFormation template—will be returned. The URL of the website that is available through the load balancer will be known.

After you run the stack, outputs are visible in the **Outputs** tab of the AWS CloudFormation console. If you run your stack by using the AWS CLI or the API, you can also retrieve outputs programmatically.

Additional AWS CloudFormation stack options

Can specify additional options 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
```

Although the default behavior of AWS CloudFormation is to *roll back* all changes when a stack fails to successfully complete all actions, you can override the default.

You can also apply a *stack policy* to prevent updates to stack resources. When you create a stack, all update actions are allowed on all resources by default. This means that anyone with stack update permissions can update all of the resources in the stack. You can prevent stack resources from being unintentionally updated or deleted during a stack update by using a *stack policy*.

The policy determines what action to take if stack creation fails, which must be one of: *DO_NOTHING*, *ROLLBACK*, or *DELETE*. You can specify either *on-failure* or *disable-rollback*, but not both. In the example AWS CLI command, the `create-stack` command is being run and the default rollback on failure options was overridden. Instead of rolling back on failure, AWS CloudFormation will do nothing, which means that any resources that were created by the failed stack will be left untouched by AWS CloudFormation.

You can also prevent a stack from being accidentally deleted by *enabling termination protection* on the stack. If a user attempts to delete a stack and termination protection is enabled, the deletion fails and the stack—including its status—remains unchanged.

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
```

You can instruct AWS CloudFormation to continue rolling back an update to your stack even after the rollback has failed. Some common causes of a failed rollback include changing a resource in your stack outside of AWS CloudFormation, insufficient permissions, limitation errors, or resources that have not stabilized.

A stack goes into the **UPDATE_ROLLBACK_FAILED** state when AWS CloudFormation cannot roll back all changes during an update. For example, you might have a stack that is rolling back to an old database instance that was deleted outside of AWS CloudFormation.

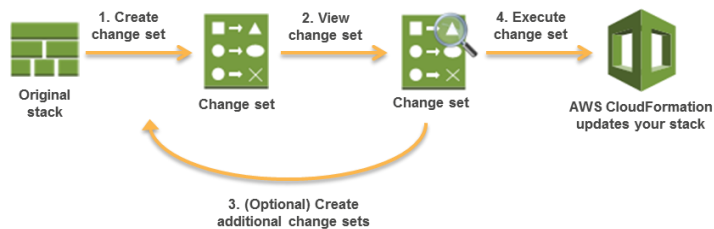
After you have remedied the cause of the failed rollback, you can instruct AWS CloudFormation to continue rolling back the update by using the AWS CloudFormation console or the AWS CLI.

For more information, refer to [continue-update-rollback](#) in the AWS CLI Command Reference.

For more information about rolling back an update, refer to [Continue rolling back an update](#).

Change sets

- Change sets show the proposed changes to a stack before any implementation
- AWS CloudFormation compares submitted changes to the stack.
- Engineers can view the changes to see what resources will be added, modified, or deleted
- The change set can then be applied to the stack to implement the changes



26

aws re/start

Change sets allow engineers to understand just what will happen to a change in a stack before implementation. This ensures that any and all updates, changes, and deletes will be identified prior to implementation.

The steps in creating a change set are:

1. Create changes for the target stack. AWS CloudFormation then creates a change set.
2. Review the change set
3. If necessary, make any additional changes that are required. For each additional change repeats steps 1 and 2.
4. Run the change set to implement the changes.

AWS CloudFormation best practices (1 of 3)

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.

The best practices presented here are recommendations that can help you use AWS CloudFormation more effectively and securely.

With regard to planning and organizing your use of AWS CloudFormation:

- Use the lifecycle and ownership of your AWS resources to help you decide what resources should go in each stack. As your stack grows in scale and broadens in scope, managing a single stack can be cumbersome and time-consuming. By grouping resources with common lifecycles and ownership, owners can make changes to their set of resources by using their own process and schedule without affecting other resources.
- Reuse templates to replicate stacks in multiple environments. After you have your stacks and resources set up, you can reuse your templates to replicate your infrastructure in multiple environments. For example, you can create environments for development, testing, and production so that you can test changes before you implement them into production.
- Verify quotas for all resource types. Before you launch a stack, ensure that you can create all the resources that you want without exceeding your AWS account limits.
- AWS CloudFormation best practices provides a repeatable infrastructure to be

deployed easily. Although you can deploy infrastructure without intervention, it is important to watch for the Amazon EC2 service limits and other AWS service limits that might impact the deployment of your AWS CloudFormation scripts. You can use AWS Trusted Advisor checks to ensure that the limits for certain EC2 instances are not going to be passed, and if so, request a service limit increase from the tool.

- Using AWS Organizations in conjunction with AWS CloudFormation can help ensure that infrastructure that is created on separate accounts all follow the same AWS Identity and Access Management (IAM) policies and organizational controls.

AWS CloudFormation best practices (2 of 3)

Creating templates:

Do not embed credentials in your templates.

Use parameter types that are specific to AWS.

Use parameter constraints.

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

Validate templates before you use them.

Here are some best practices with regard to creating AWS CloudFormation templates:

- Avoid embedding credentials in your templates. Instead, use *input parameters* to pass information when you create or update a stack.
- Use parameter types that are specific to AWS. AWS CloudFormation can quickly validate values for these parameter types before it creates your stack.
- Use parameter constraints. With constraints, you can describe allowed input values so that AWS CloudFormation catches any invalid values before it creates a stack.
- Use `AWS::CloudFormation::Init` to deploy software applications on EC2 instances. By using `AWS::CloudFormation::Init`, you can describe the configurations that you want, instead of scripting procedural steps.
- Validate templates before you use them. Validating a template can help you catch syntax errors and some semantic errors before AWS CloudFormation creates any resources.

AWS CloudFormation best practices (3 of 3)

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.

Finally, regarding the management of your stacks, here are some additional best practices:

- Manage all stack resources through AWS CloudFormation. After you launch a stack, use the AWS CloudFormation console, API, or the AWS CLI to update resources in your stack. Do not make changes to stack resources outside of AWS CloudFormation. Doing so can create a mismatch between your stack's template and the current state of your stack resources, which can cause errors if you update or delete the stack.
- Create *change sets* before you update stacks. Change sets enable you to see how proposed changes to a stack might impact your running resources before you implement them. AWS CloudFormation does not make any changes to your stack until you run the change set, which enables you to decide whether to proceed with your proposed changes or create another change set.
- Use stack policies. Stack policies help protect critical stack resources from unintentional updates that could cause resources to be interrupted or even replaced. Specify a stack policy when you create a stack that has critical resources.
- Use AWS CloudTrail to log AWS CloudFormation calls. AWS CloudTrail tracks anyone who makes AWS CloudFormation API calls in your AWS account. API calls

are logged when anyone uses the AWS CloudFormation API, the AWS CloudFormation console, a backend console, or AWS CLI commands for AWS CloudFormation. Enable logging and specify an S3 bucket to store the logs. By doing so, you can audit who made which AWS CloudFormation call in your account when needed.

- Use code reviews and revision controls to manage templates. Stack templates describe the configuration of your AWS resources, such as their property values. These methods help track changes between different versions of templates, which helps track changes to stack resources. Also, by maintaining a history, you can revert your stack to a certain version of your template.

Key takeaways



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

30

- AWS CloudFormation enables you to create and provision AWS infrastructure deployments predictably and repeatedly.
- Two major terms for AWS CloudFormation are *templates* and *stacks*.
- If an error is encountered when you launch an AWS CloudFormation template, all resources are rolled back by default.
- Parameters enable you to input custom values to your template each time you create or update a stack.



- AWS CloudFormation enables you to create and provision AWS infrastructure deployments predictably and repeatedly.
- Two major terms for AWS CloudFormation are *templates* and *stacks*.
- If an error is encountered when you launch an AWS CloudFormation template, all resources are rolled back by default.
- Parameters enable you to input custom values to your template each time you create or update a stack.