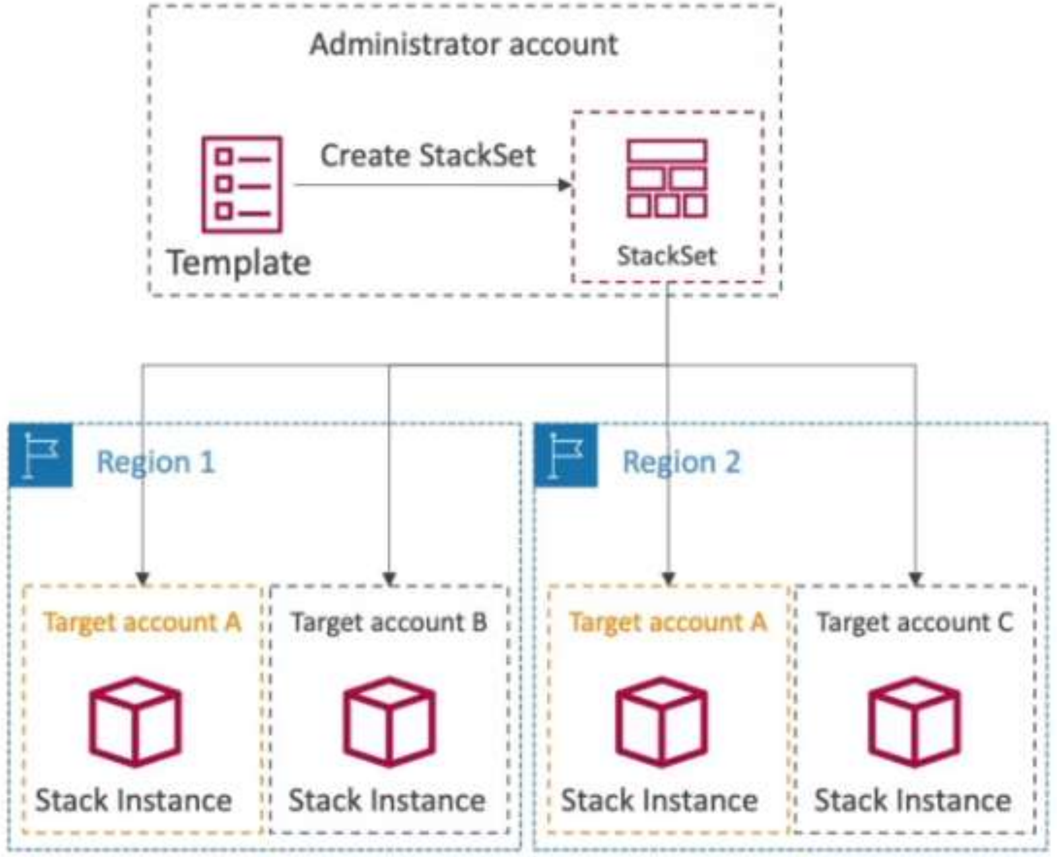
StackSets Overview

- Create, update, or delete stacks across **multiple accounts and regions** with a single operation/template
- Administrator account to create StackSets
- Target accounts to create, update, delete stack instances from StackSets
- When you update a stack set, *all* associated stack instances are updated throughout all accounts and regions
- Regional service
- Can be applied into all accounts of an AWS organizations



StackSet Operations

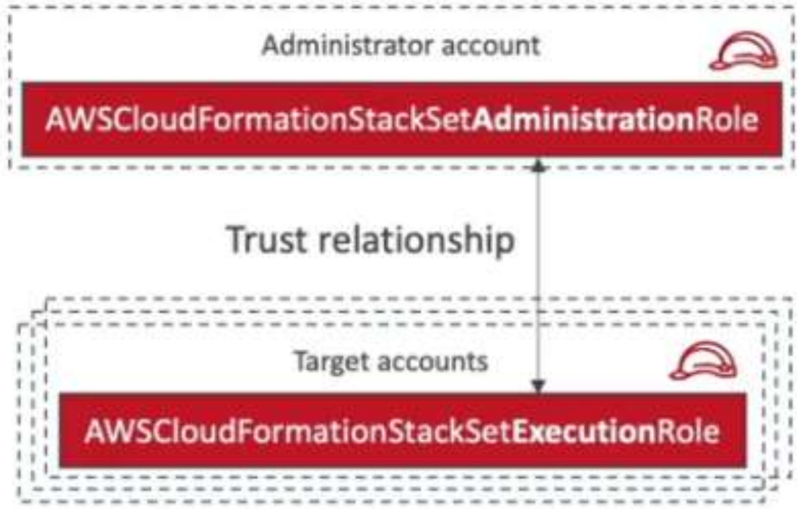
- Create StackSet
  - Provide template + target accounts/regions
- Update StackSet
  - Updates always affect all stacks (you can't selectively update some stacks in the StackSet but not others)
- Delete Stacks
  - Delete stack and its resources from target accounts/regions
  - Delete stack from your StackSet (the stack will continue to run independently)
  - Delete all stacks from your StackSet (prepare for StackSet deletion)
- Delete StackSet
  - Must delete all stack instances within StackSet to delete it

StackSet Deployment Options

- Deployment Order
  - Order of regions where stacks are deployed
  - Operations performed one region at a time
- Maximum Concurrent Accounts
  - Max. number/percentage of target accounts per region to which you can deploy stacks at one time
- Failure Tolerance
  - Max. number/percentage (target accounts per region) of stack operation failures that can occur before CloudFormation stops operation in all regions
- Region Concurrency
  - Whether StackSet deployed into regions **Sequential (default)** or **Parallel**
- Retain Stacks
  - Used when deleting StackSet to keep stacks and their resources running when removed from StackSet

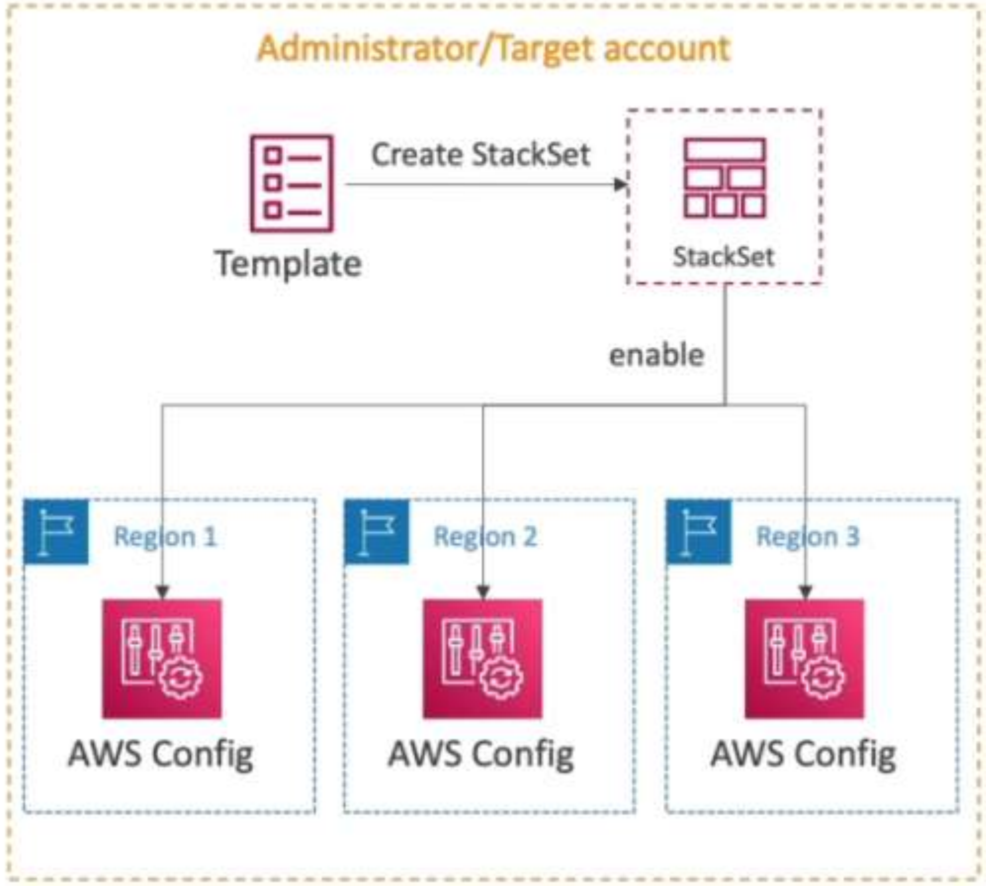
Permission Models for StackSet

- Self-managed Permissions
  - Create the IAM roles (with established trusted relationship) in both administrator and target accounts
  - Deploy to any target account in which you have permissions to create IAM role
- Service-managed Permissions
  - Deploy to accounts managed by AWS Organizations
  - StackSets create the IAM roles on your behalf (**enable trusted access** with AWS Organizations)
  - Must **enable all features** in AWS Organizations
  - Ability to deploy to accounts added to your organization in the future (Automatic Deployments)


Administrator account
AWSCloudFormationStackSetAdministrationRole
Trust relationship
Target accounts
AWSCloudFormationStackSetExecutionRole

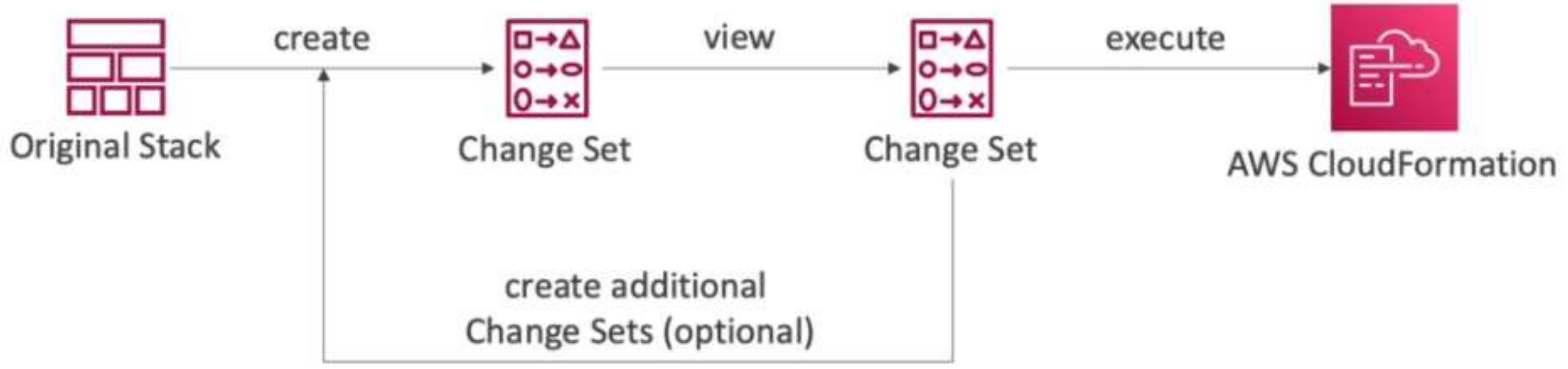
Hands-On: StackSets

- We'll use StackSets to enable AWS Config across AWS regions with a single click
- Let's see how this works!



ChangeSets

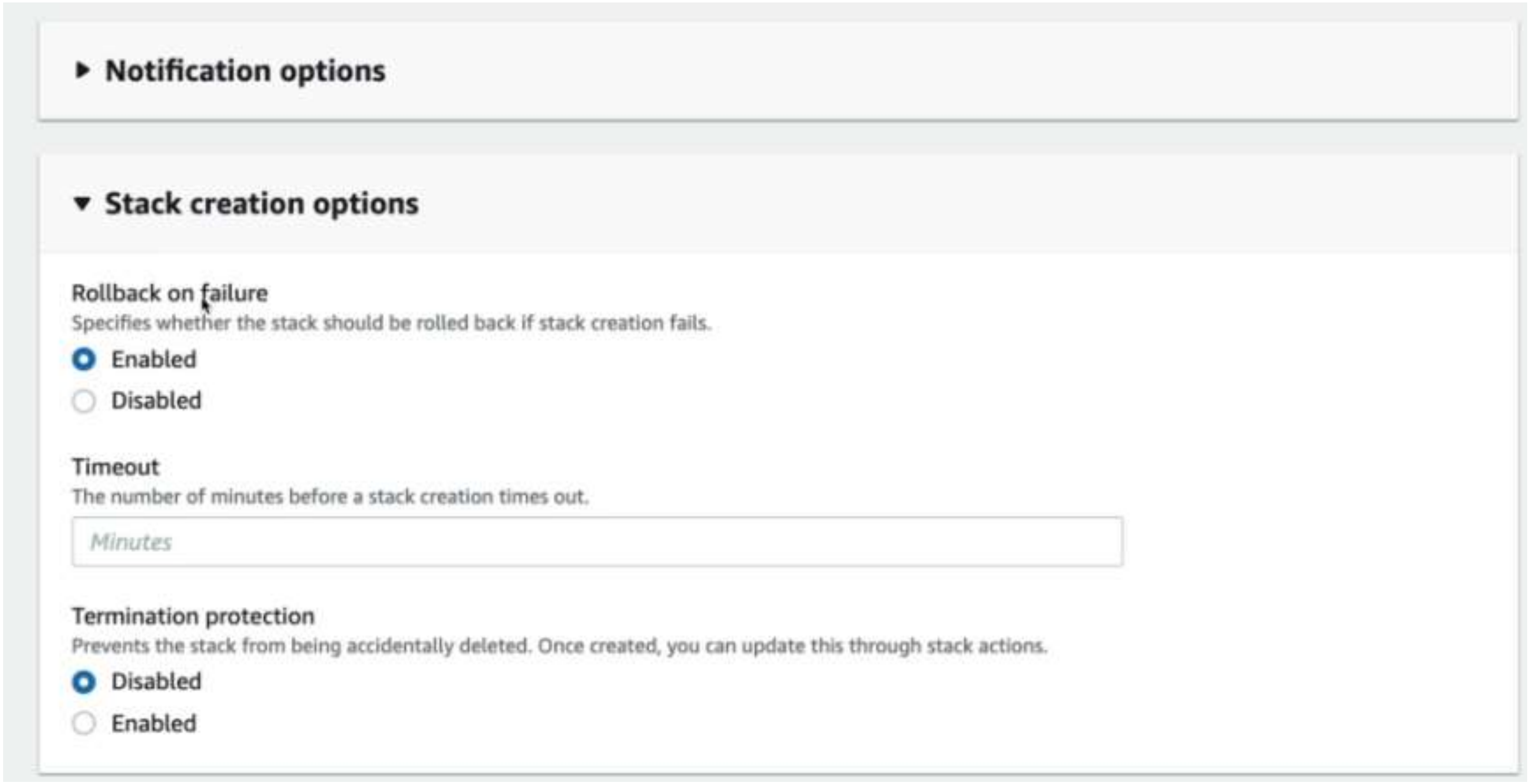
- When you update a stack, you need to know what changes will happen before it applying them for greater confidence
- ChangeSets won't say if the update will be successful
- For Nested Stacks, you see the changes across all stacks



Stack Creation Failures

- If a CloudFormation stack creation fails, you will get the status ROLLBACK_COMPLETE
- This means:
  - 1. CloudFormation tried to create some resources
  - 2. One resource creation failed
  - 3. CloudFormation rolled back the resources (ROLLBACK, DO_NOTHING)
  - 4. The stack is in failed created ROLLBACK_COMPLETE state
- To resolve the error, there's only one way: <u>Delete the failed stack and create a new stack</u>

- You can't update, validate or change-set on a create failed stack



▸ Notification options

▾ Stack creation options

Rollback on failure

Specifies whether the stack should be rolled back if stack creation fails.

[✓ Enabled] Disabled

Timeout

The number of minutes before a stack creation times out.

Minutes

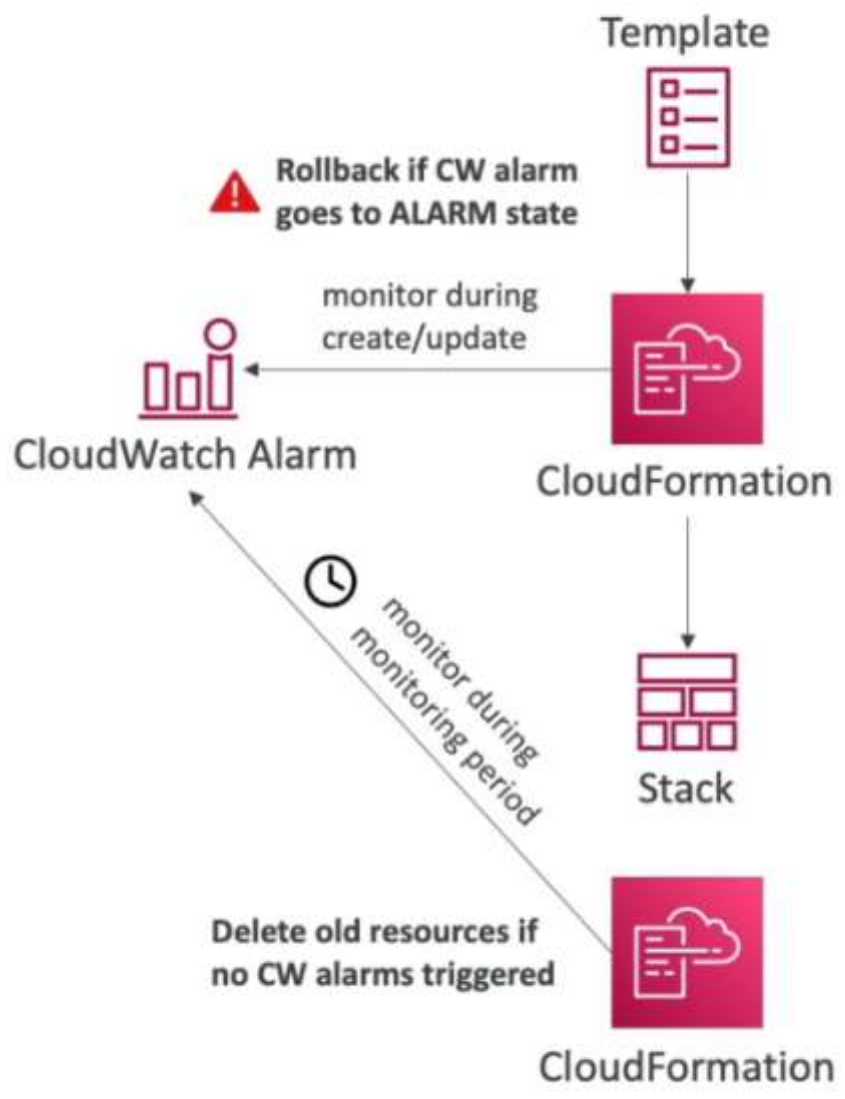
Termination protection

Prevents the stack from being accidentally deleted. Once created, you can update this through stack actions.

[✓ Disabled] Enabled

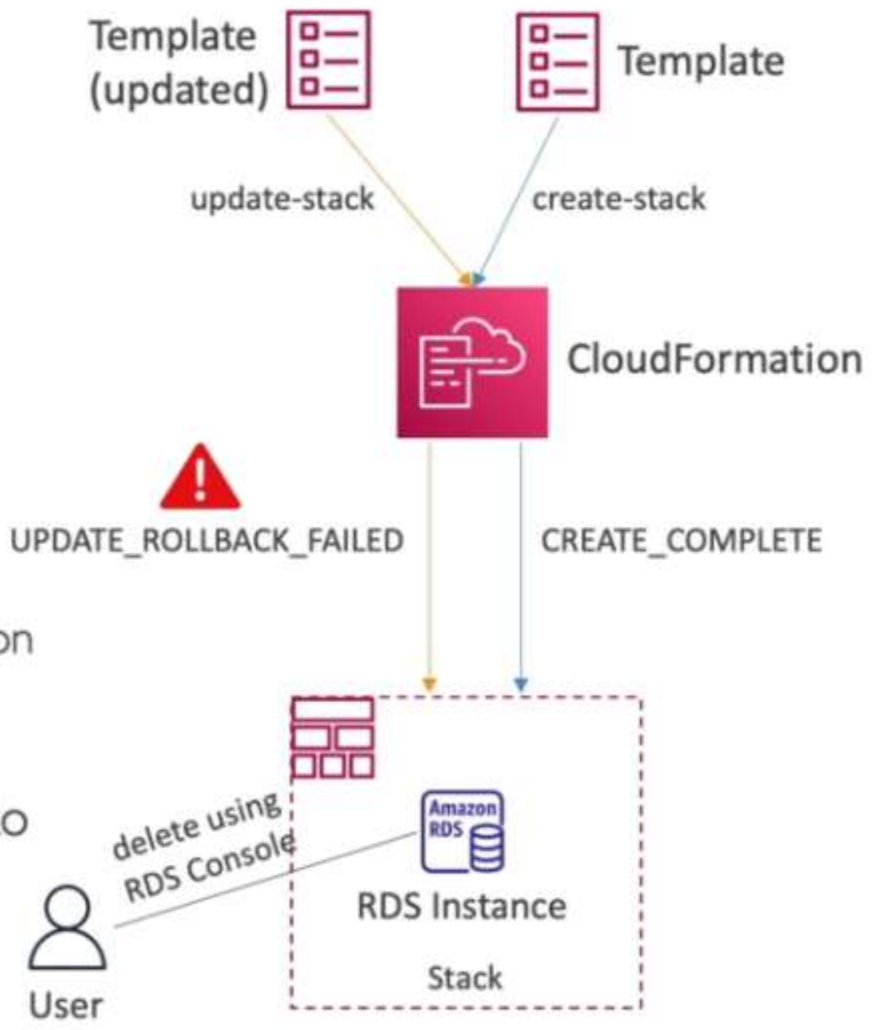
Rollback Triggers

- Enables CloudFormation to rollback stack create/update operation if that operation triggers CloudWatch Alarm
- CloudFormation monitors the specified CloudWatch alarms during:
  - Stack create/update
  - The monitoring period (after all resources have been deployed) 0 to 180 minutes (default: 0 minutes)
- If any of the alarms goes to the ALARM state, CloudFormation rolls back the entire stack operation
- If you set a monitoring time but don't specify any rollback triggers, CloudFormation still waits the specified period before cleaning up old resources for update operations
- If you set a monitoring time of 0 minutes, CloudFormation still monitors the rollback triggers during stack create/update operation
- Up to 5 CloudWatch alarms


Template
Rollback if CW alarm goes to ALARM state
monitor during create/update
CloudWatch Alarm
CloudFormation
monitor during monitoring period
Stack
Delete old resources if no CW alarms triggered
CloudFormation

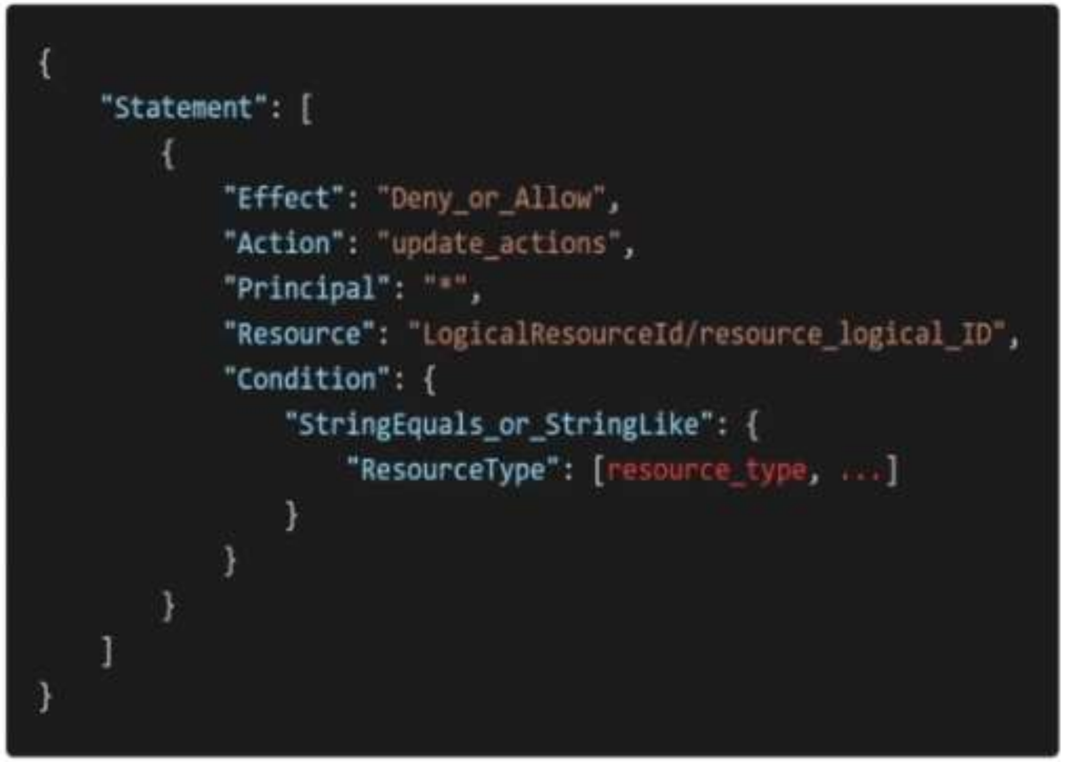
Continue Rolling Back an Update

- A stack goes into the UPDATE_ROLLBACK_FAILED state when CloudFormation can't roll back all changes during an update
- A resource can't return to its original state, causing the rollback to fail
- Example: roll back to an old database instance that was deleted outside CloudFormation
- Solutions:
  - Fix the errors manually outside of CloudFormation and then continue update rollback the stack
  - Skip the resources that can't rollback successfully (CloudFormation will mark the failed resources as UPDATE_COMPLETE)
- You can't update a stack in this state
- For nested stacks, rolling back the parent stack will attempt to roll back all the child stacks as well



Stack Policy

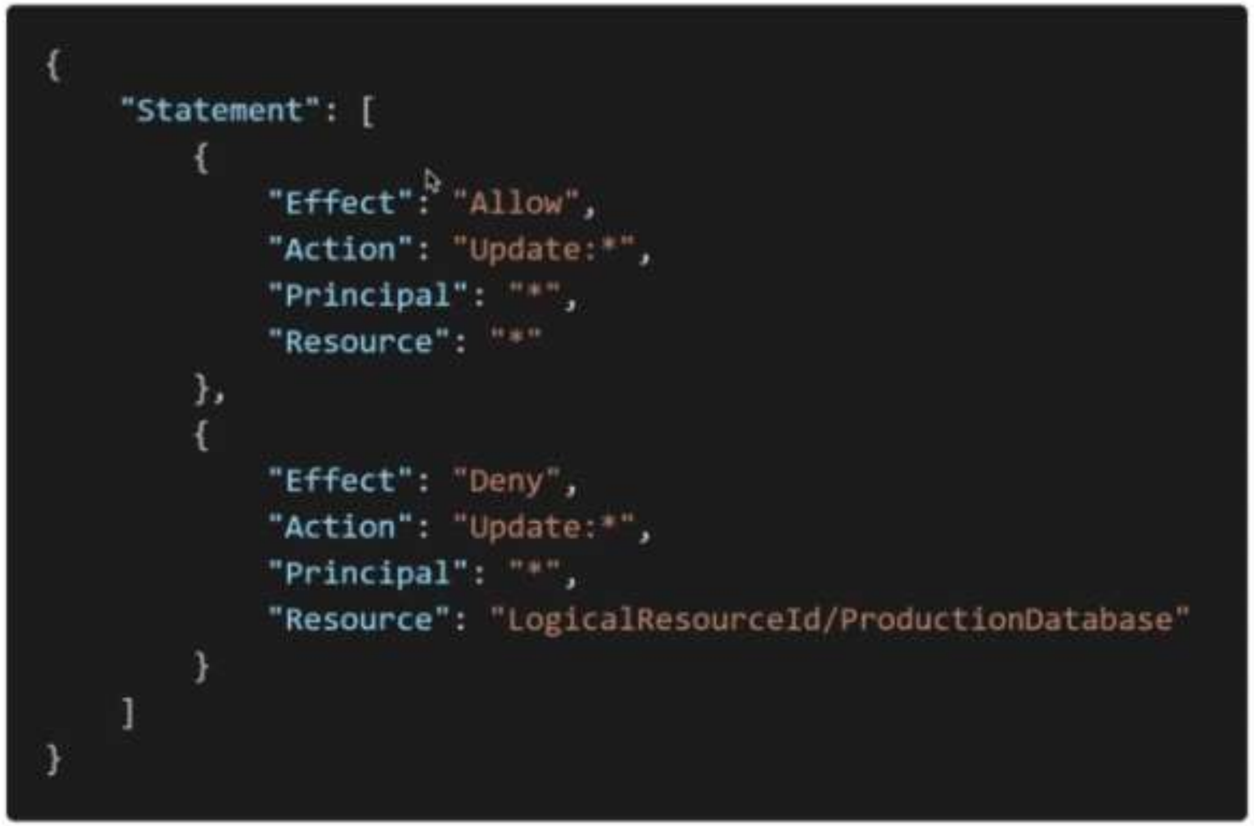
- A JSON document that defines the update actions allowed on stack resources
- Prevent stack resources from being unintentionally updated/deleted **during a stack update**
- By default, all update actions are allowed on all resources
- When enabled, all resources protected by default
- Actions allowed (Update:Modify, Update:Replace, Update:Delete, Update:*)
- Principal supports only the wildcard (*)
- To update protected resources:
  - Create a temporary policy that overrides the stack policy
  - The override policy doesn't permanently change the stack policy
- Once created, can't be deleted (edit to allow all update actions on all resources)



```
{
    "Statement": [
        {
            "Effect": "Deny_or_Allow",
            "Action": "update_actions",
            "Principal": "*",
            "Resource": "LogicalResourceId/resource_logical_ID",
            "Condition": {
                "StringEquals_or_StringLike": {
                    "ResourceType": [resource_type, ...]
                }
            }
        }
    ]
}
```

Example

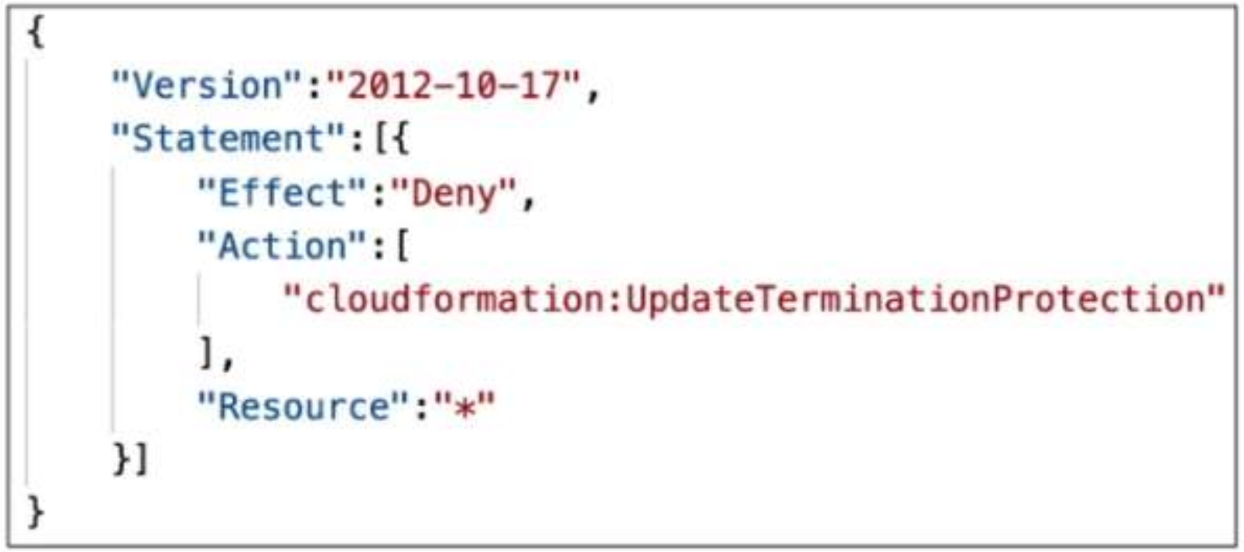
Allow updates on all resources EXCEPT our production database



```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "Update:*",
            "Principal": "*",
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": "Update:*",
            "Principal": "*",
            "Resource": "LogicalResourceId/ProductionDatabase"
        }
    ]
}
```

Termination Protection on Stacks

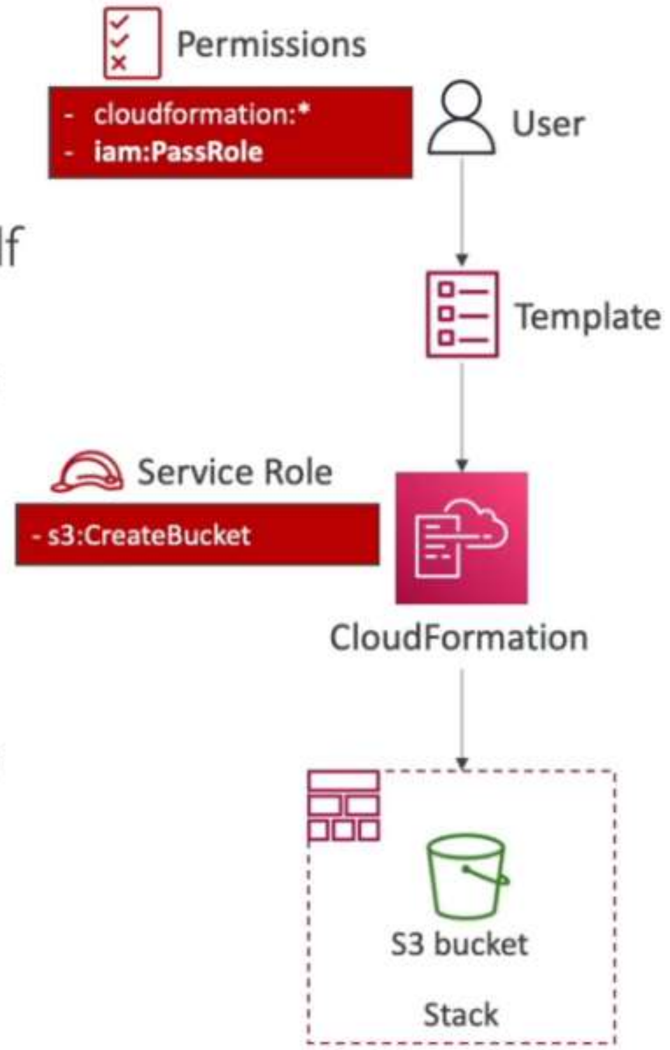
- To prevent accidental deletes of CloudFormation stacks, use TerminationProtection
- Applied to any nested stacks
- Tighten your IAM policies (ex: explicit deny on some user groups)



```
{
    "Version":"2012-10-17",
    "Statement":[{
        "Effect":"Deny",
        "Action":[
            "cloudformation:UpdateTerminationProtection"
        ],
        "Resource":"*"
    }]
}
```

Service Role

- IAM role that allows CloudFormation to create/update/delete stack resources on your behalf
- By default, CloudFormation uses a temporary session that it generates from your user credentials
- Use cases:
  - You want to achieve the least privilege principle
  - But you don't want to give the user all the required permissions to create the stack resources
- Give ability to users to create/update/delete the stack resources even if they don't have permissions to work with the resources in the stack



Quick-create Links for Stacks

- Custom URLs that used to launch CloudFormation stacks quickly from AWS Console
- Reduce the number of wizard pages and the amount of user input that's required
- For example: create multiple URLs that specify different values for the same template
- CloudFormation ignores parameters:
  - That don't exist in the template
  - That defined with NoEcho property set to true

https://*region*.console.aws.amazon.com/cloudformation/home?region=*region*#/stacks/quickcreate?stackName=*stack_name*&templateURL=*template_location*&*param_parameterName1=value1*&*param_parameterName2=value2...*

Custom Resources

- Enable you to write custom provision logic in templates that AWS CloudFormation runs anytime you create, update, delete stacks
- Defined in the template using AWS::CloudFormation::CustomResource or Custom::*MyCustomResourceTypeName* (recommended)
- Two types:
  - Amazon SNS-backed Custom Resources
  - AWS Lambda-backed Custom Resources
- Use cases:
  - An AWS resource is not covered yet (new service for example)
  - An on-premises resource
  - Running a Lambda function to empty an S3 bucket before being deleted
  - Fetch an AMI id
  - Anything you want…!

How to define a Custom Resource?

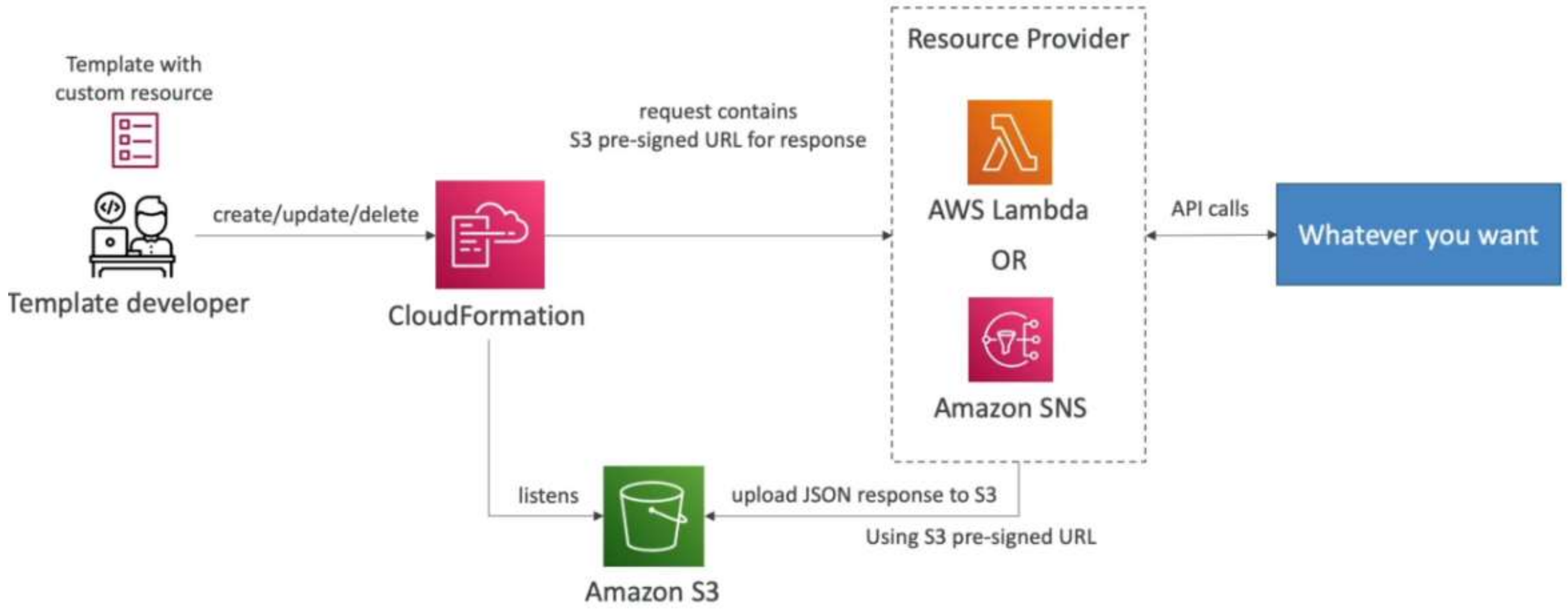
- ServiceToken specifies where CloudFormation sends requests to, such as Lambda ARN or SNS ARN (required & must be in the same region)
- Input data parameters (optional)

```
Resources:
  LogicalResourceName:
    Type: Custom::MyCustomResourceTypeName
    Properties:
      ServiceToken: service_token
```

```
Resources:
  MyFrontEndTest:
    Type: Custom::PingTester
    Properties:
      ServiceToken: 'arn:aws:sns:us-east-1:123456789012:CRTest'
      key1: string
      key2:
        - list
      key3: map

Outputs:
  CustomResourceAttribute1:
    Value: !GetAtt
      - MyFrontEndTest
      - responseKey1
  CustomResourceAttribute2:
    Value: !GetAtt
      - MyFrontEndTest
      - responseKey2
```

Custom Resources – How does it work?



Custom Resource – Request & Response

Request

```
{
  "RequestType": "Create",
  "ResponseURL": "http://pre-signed-S3-url-for-response",
  "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/stack-name/guid",
  "RequestId": "unique id for this create request",
  "ResourceType": "Custom::CustomResource",
  "LogicalResourceId": "MyCustomResource",
  "ResourceProperties" : {
    "Name" : "Value",
    "List" : [ "1", "2", "3" ]
  }
}
```

you want

Response

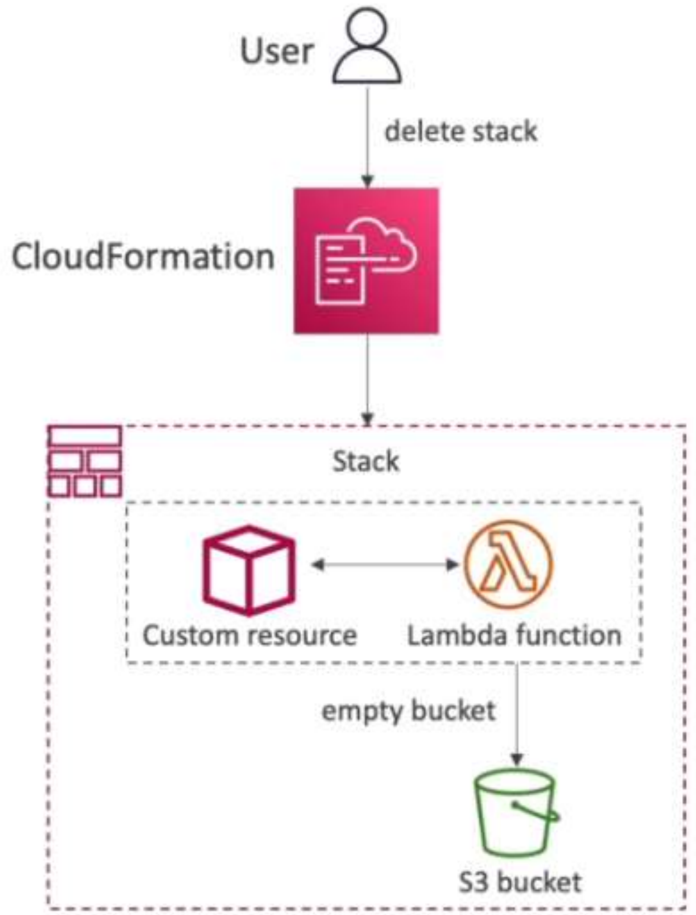
```
{
  "Status": "SUCCESS",
  "PhysicalResourceId": "CustomResource1",
  "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/stack-name/guid",
  "RequestId": "unique id for this create request",
  "LogicalResourceId": "MyCustomResource",
  "Data": {
    "OutputName1": "Value1",
    "OutputName2": "Value2"
  }
}
```

Hands-On: Lambda-backed Custom Resource

- You can't delete a non-empty S3 bucket
- To delete a non-empty S3 bucket, you must first delete all the objects inside it

- We'll create a custom resource with AWS Lambda that will be used to empty an S3 bucket before deleting it

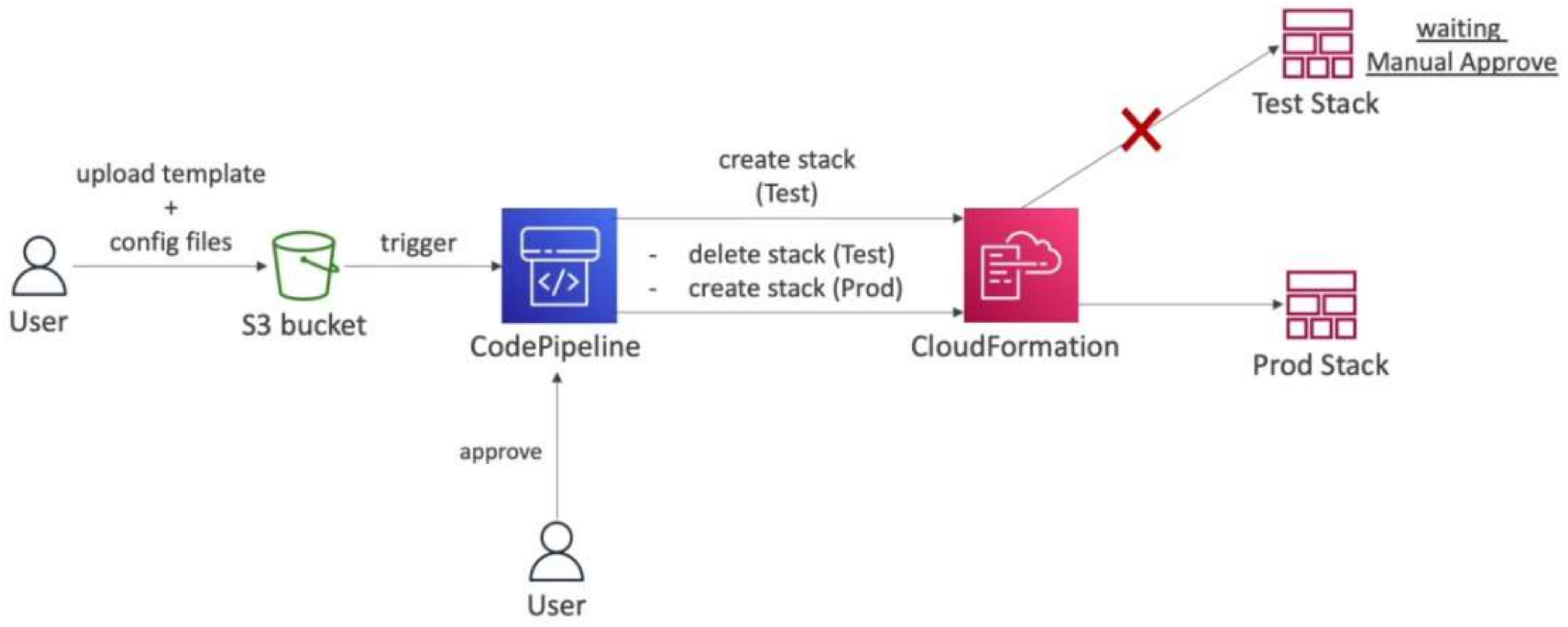
- Let's create our first Custom Resource!



Continuous Delivery with CodePipeline

- Use CodePipeline to build a continuous delivery workflow (building a pipeline for CloudFormation stacks)
- Rapidly and reliably make changes to your AWS infrastructure
- Automatically build and test changes to your CloudFormation templates before promoting them to production stacks
- For example:
  - Create a workflow that automatically builds a test stack when you submit a CloudFormation template to a code repository
  - After CloudFormation builds the test stack, you can test it and then decide whether to push changes to production stack

CD with CodePipeline Example

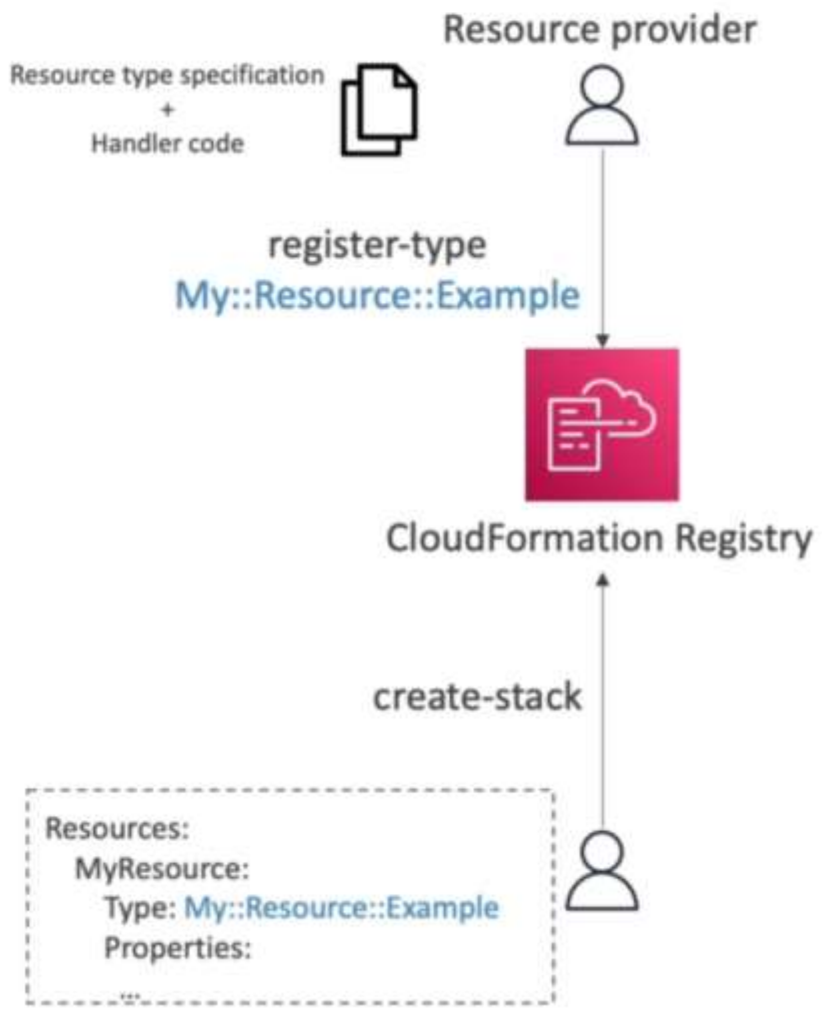


CloudFormation Registry

- Contains private and public extensions (Resource Types & Modules)
- Extensions are artifacts that augments the functionality of CloudFormation resources and properties
- Extensions registered in CloudFormation Registry
- Extensions can be written by Amazon, APN Partners, Marketplace sellers, and the community
- Extensions types
  - **Private extensions:** you created or shared with you
  - **Public extensions:** provided by AWS (ex. AWS::DynamoDB::Table)
- Use CloudFormation CLI to create extensions

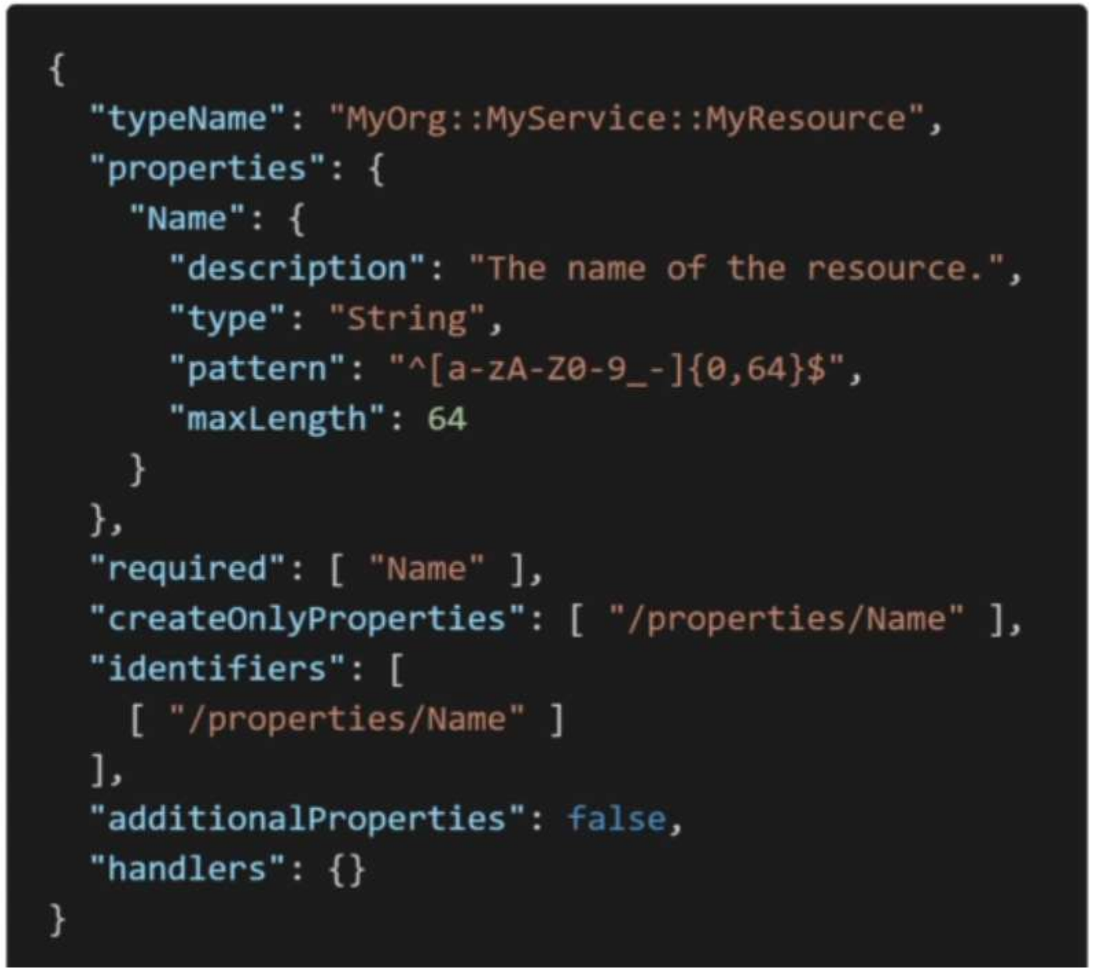
Resource Types

- Model and provisions resources using CloudFormation
- For example, create a custom resource that doesn't exist in CloudFormation
- It should follow the structure **Organization::Service::Resource**
- Resource type package consists of
  - JSON schema that defines your type
  - Handlers that perform the required actions (create, update, delete, read, list)
- Steps to create
  1. **Model:** create and validate schema that serves as the definition of your resource type
  2. **Develop:** write a handler that defines five core operations (Create, Read, Update, Delete, List) on your resource type, and test locally
  3. **Register:** register the resource type with CloudFormation so that it can be used in your CloudFormation templates
- Write handlers in (Python, Java, TypeScript, Go)



```
Resources:
  MyResource:
    Type: My::Resource::Example
    Properties:
      ...
```

Example: Resource Type Definition



```
{
  "typeName": "MyOrg::MyService::MyResource",
  "properties": {
    "Name": {
      "description": "The name of the resource.",
      "type": "String",
      "pattern": "^[a-zA-Z0-9_-]{0,64}$",
      "maxLength": 64
    }
  },
  "required": [ "Name" ],
  "createOnlyProperties": [ "/properties/Name" ],
  "identifiers": [
    [ "/properties/Name" ]
  ],
  "additionalProperties": false,
  "handlers": {}
}
```

CloudFormation CLI

- Enables you to develop and test AWS and 3rd party extensions (e.g., resource types and modules) https://github.com/aws-cloudformation/cloudformation-cli
- Register extensions for use in CloudFormation
- Supports Java, Go, Python, TypeScript to write your own extensions

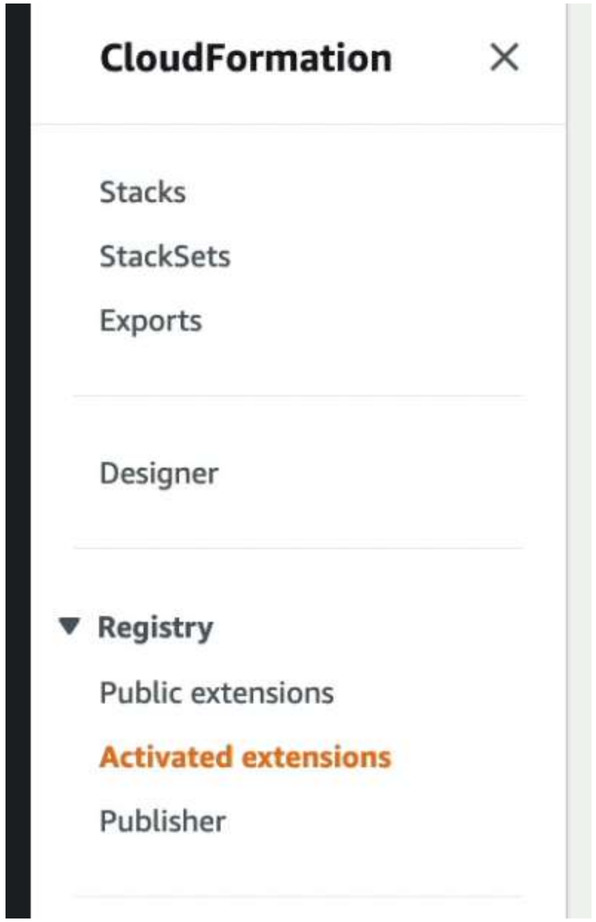
3rd Party Resource Types

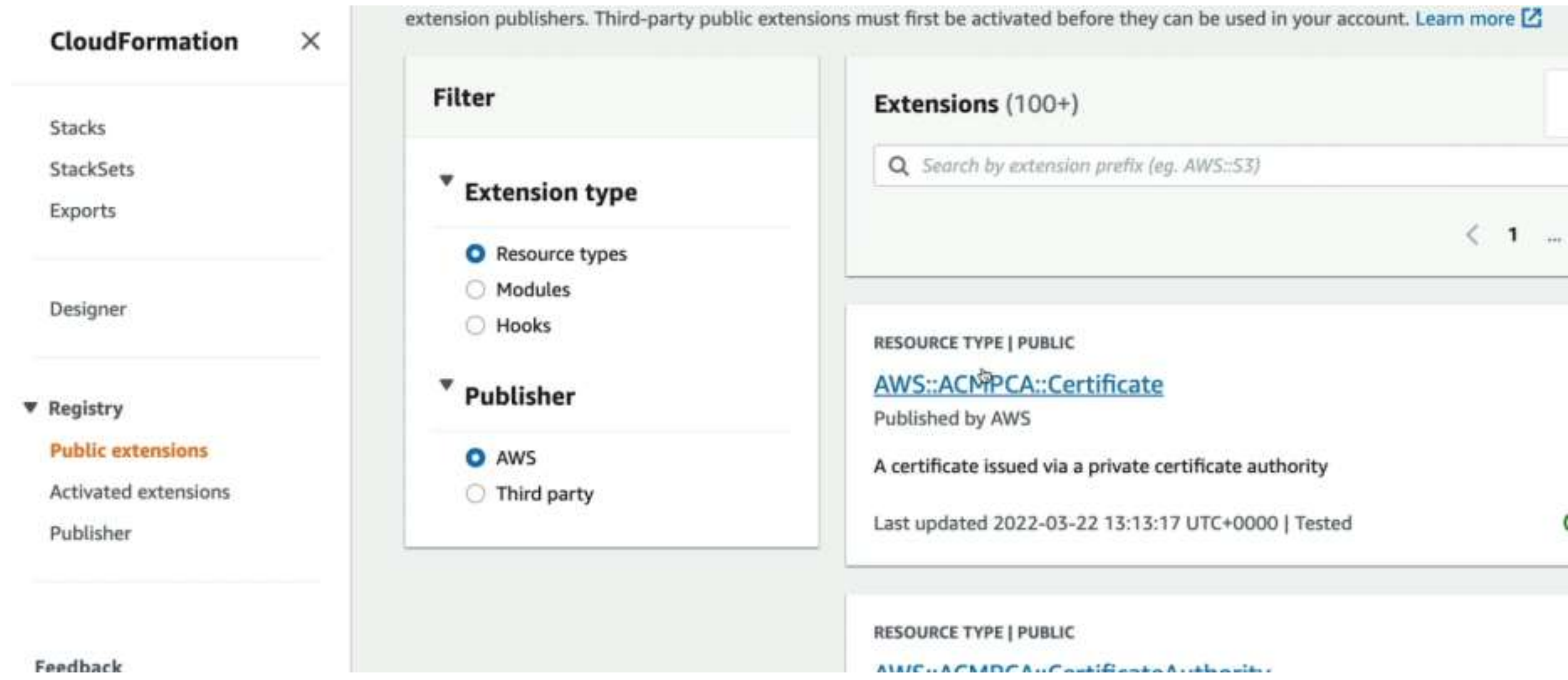
- 3rd party vendors created resource types using CloudFormation CLI
- Can be downloaded and added to your account via CloudFormation Registry

Reference URL	Resource Types
https://github.com/opsgenie/opsgenie-cloudformation-resources	Atlassian::Opsgenie::User, Atlassian::Opsgenie::Team, Atlassian::Opsgenie::Integration
https://github.com/DataDog/datadog-cloudformation-resources	Datadog::Dashboards::Dashboard, Datadog::Integrations::AWS, Datadog::Monitors::Monitor, Datadog::Monitors::Downtime, Data::IAM::User
https://github.com/densify-dev/cloudformation-optimization-as-code	Densify::Optimization::Recommendation
https://github.com/mnalezin/DynatraceInstallerAgent	Dynatrace::Installer::Agent
https://github.com/fortinet/aws-cloudformation-resource-provider	Fortinet::FortiGate::SystemAdmin, Fortinet::FortiGate::SystemDns, Fortinet::FortiGate::SystemInterface
https://github.com/newrelic/cloudformation-partner-integration	NewRelic::Alerts::NrqlAlert
https://github.com/spotinst/spotinst-aws-cloudformation-registry	Spotinst::Elastigroup::Group

Custom Resources vs Resource Types

	Custom Resource	**Resource Type**
Operations	Create, Update, Delete	Create, Update, Delete, Read, List
Languages	Any language that Lambda supports	Python, Java, Go, TypeScript
Location of Execution	Logic and code managed and executed in your account (Lambda function)	Logic and code managed and executed by AWS
Billing	Lambda function invocations	Handler operations/month
CloudFormation Registry	No	Yes
Integration with Drift Detection	No	Yes
Integration with ChangeSets	No	Yes





extension publishers. Third-party public extensions must first be activated before they can be used in your account. Learn more

Filter

Extension type

[✓ Resource types] Modules Hooks

Publisher

[✓ AWS] Third party

Extensions (100+)

Search by extension prefix (eg. AWS::S3)

RESOURCE TYPE | PUBLIC

AWS::ACMPCA::Certificate

Published by AWS

A certificate issued via a private certificate authority

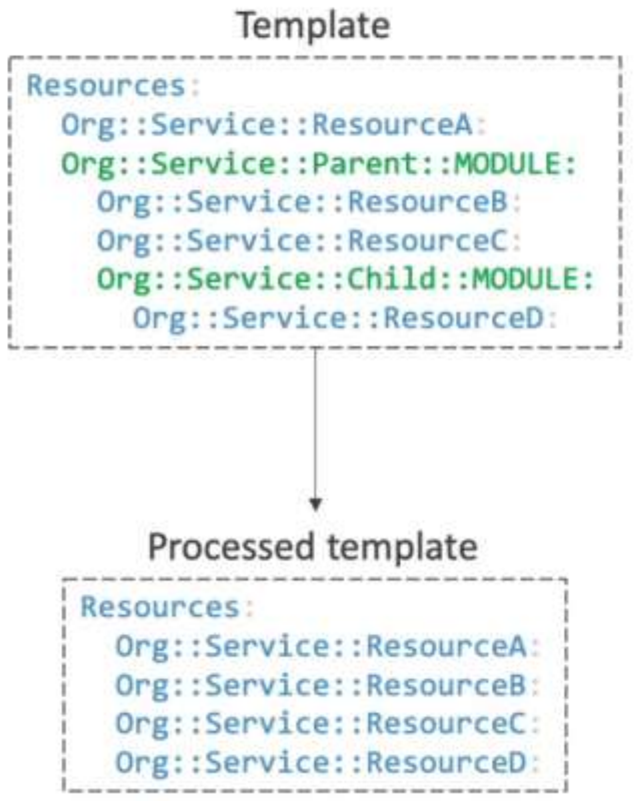
Last updated 2022-03-22 13:13:17 UTC+0000 | Tested

RESOURCE TYPE | PUBLIC

Modules

- Module is more reusable with code replacements when compared to nested stacks

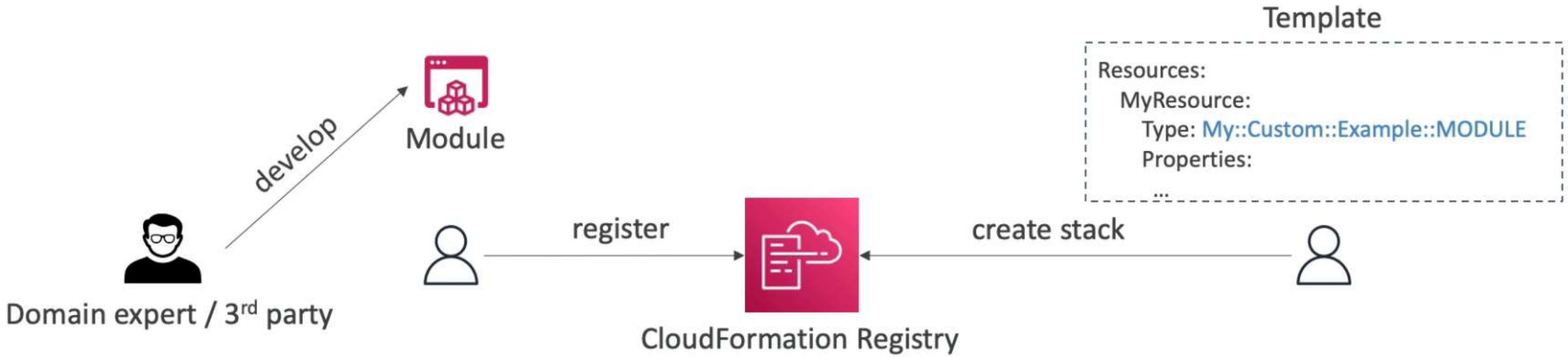
  - Package resource(s) and their configurations for use across stack templates
  - Use cases:
    - Keep resource configurations aligned with best practices
    - Use code written by experts
  - Module contains
    - **Template sections:** resources, outputs, …
    - **Module parameters:** input custom values to your module
  - It should follow the structure Organization::Service::Resource::MODULE
  - Registered in CloudFormation Registry as private extensions
  - Modules are versioned and can contain nested modules



```
Resources:
  Org::Service::ResourceA:
  Org::Service::Parent::MODULE:
    Org::Service::ResourceB:
    Org::Service::ResourceC:
    Org::Service::Child::MODULE:
      Org::Service::ResourceD:
```

```
Resources:
  Org::Service::ResourceA:
  Org::Service::ResourceB:
  Org::Service::ResourceC:
  Org::Service::ResourceD:
```

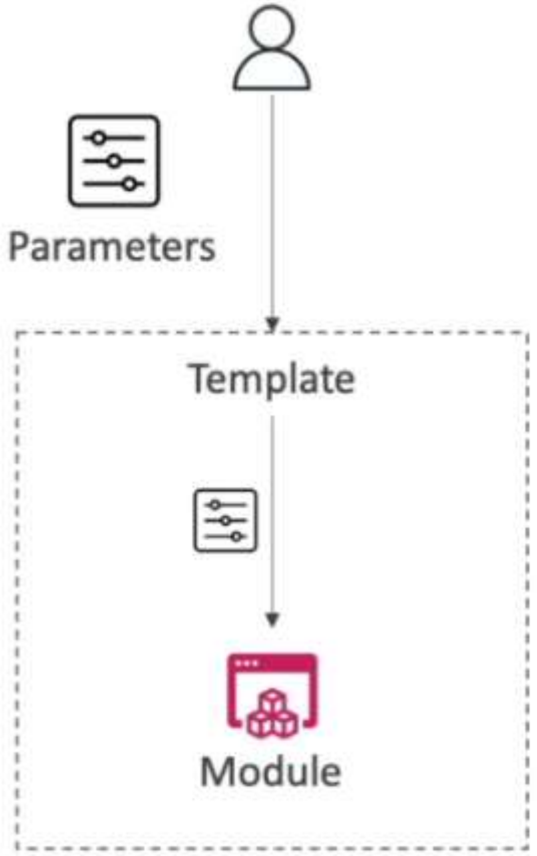
Modules – How does it work?



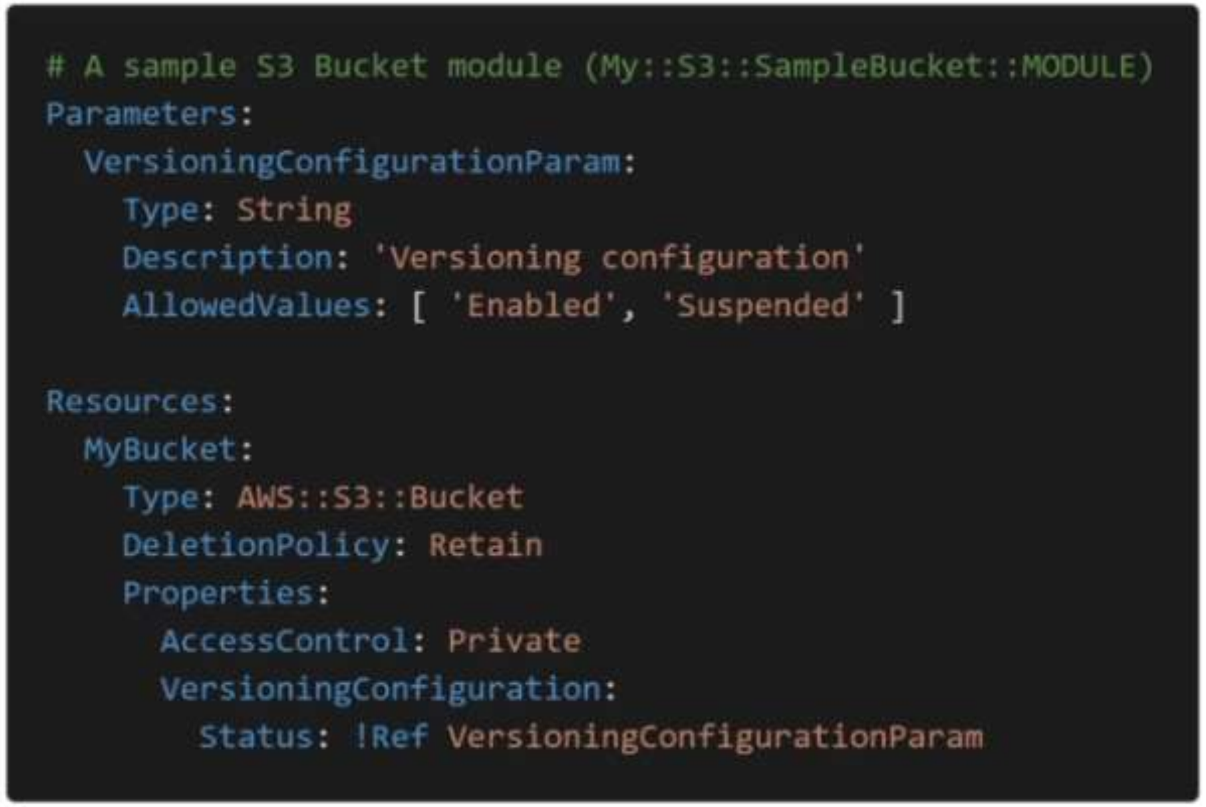
```
Resources:
  MyResource:
    Type: My::Custom::Example::MODULE
    Properties:
      ...
```

Modules Parameters

- Enables you to input custom values to your module from the template/module that contains it
- Defined the same as template parameters
- You can pass template (parent) parameters to module parameters
- You can't perform constraint checking (e.g., AllowedPattern, AllowedValues, …) on Modules Parameters



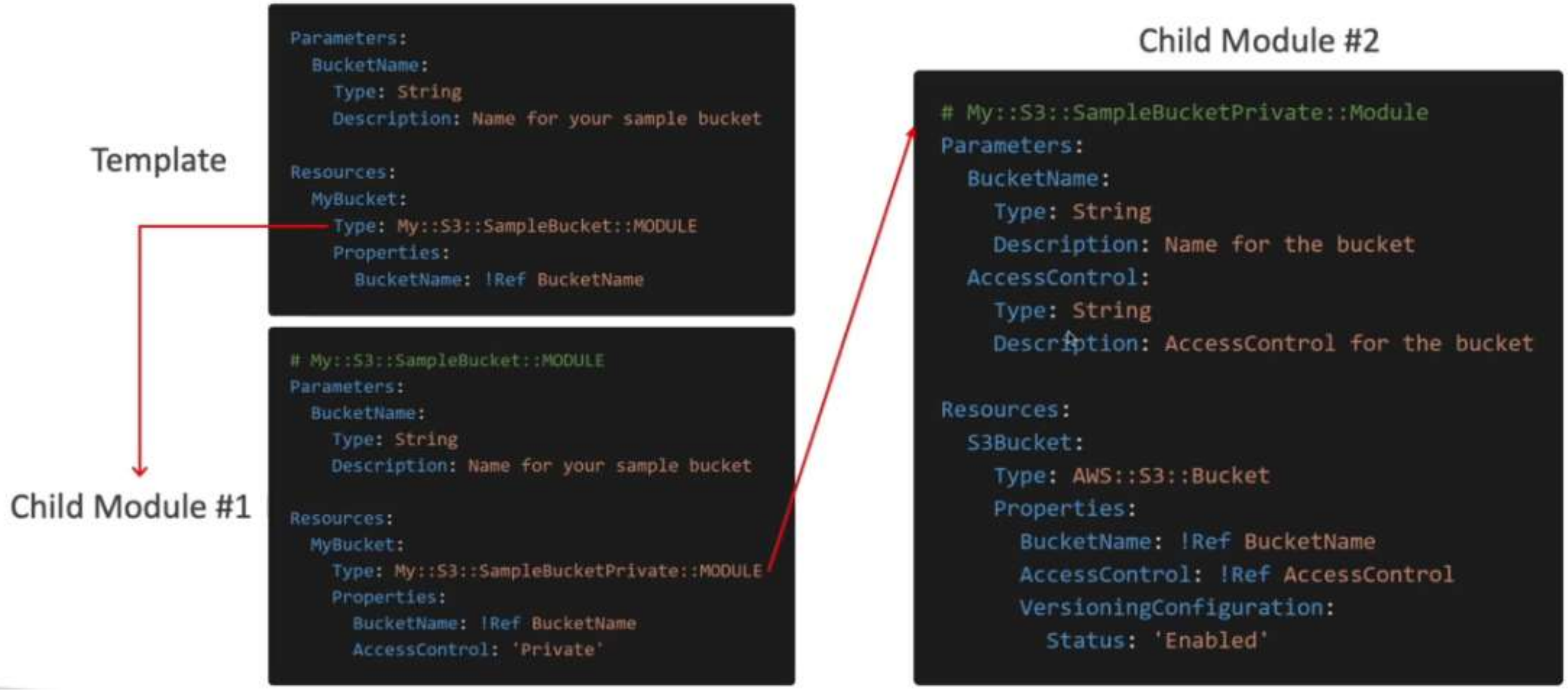
How to define a Module?



```yaml
# A sample S3 Bucket module (My::S3::SampleBucket::MODULE)
Parameters:
  VersioningConfigurationParam:
    Type: String
    Description: 'Versioning configuration'
    AllowedValues: [ 'Enabled', 'Suspended' ]

Resources:
  MyBucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
    Properties:
      AccessControl: Private
      VersioningConfiguration:
        Status: !Ref VersioningConfigurationParam
```

Example: Using Nested Modules & Parameters



Template

```
Parameters:
  BucketName:
    Type: String
    Description: Name for your sample bucket

Resources:
  MyBucket:
    Type: My::S3::SampleBucket::MODULE
    Properties:
      BucketName: !Ref BucketName
```

Child Module #1

```
# My::S3::SampleBucket::MODULE
Parameters:
  BucketName:
    Type: String
    Description: Name for your sample bucket

Resources:
  MyBucket:
    Type: My::S3::SampleBucketPrivate::MODULE
    Properties:
      BucketName: !Ref BucketName
      AccessControl: 'Private'
```

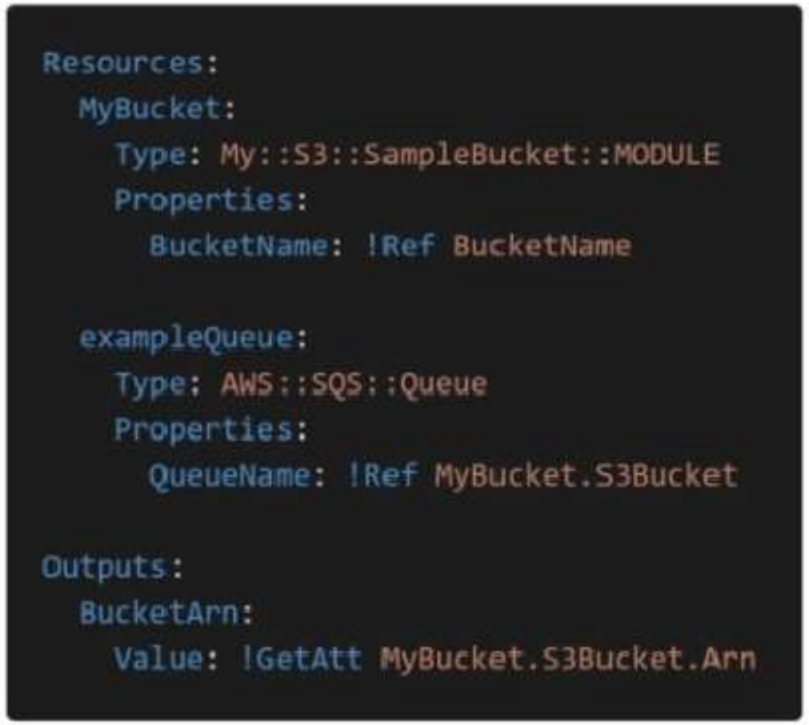
Child Module #2

```
# My::S3::SampleBucketPrivate::Module
Parameters:
  BucketName:
    Type: String
    Description: Name for the bucket
  AccessControl:
    Type: String
    Description: AccessControl for the bucket

Resources:
  S3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: !Ref BucketName
      AccessControl: !Ref AccessControl
      VersioningConfiguration:
        Status: 'Enabled'
```

Reference Resources in a Module

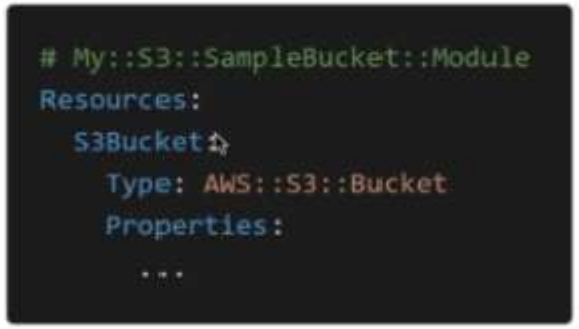
- Resources in a Module can be referenced by logical names
- The fully qualified logical name
  - ModuleLogicalName.ResourceLogicalName
  - ModuleLogicalNameResourceLogicalName
- Use **GetAtt** and **Ref** intrinsic functions to access property values as usual



```yaml
Resources:
  MyBucket:
    Type: My::S3::SampleBucket::MODULE
    Properties:
      BucketName: !Ref BucketName

  exampleQueue:
    Type: AWS::SQS::Queue
    Properties:
      QueueName: !Ref MyBucket.S3Bucket

Outputs:
  BucketArn:
    Value: !GetAtt MyBucket.S3Bucket.Arn
```



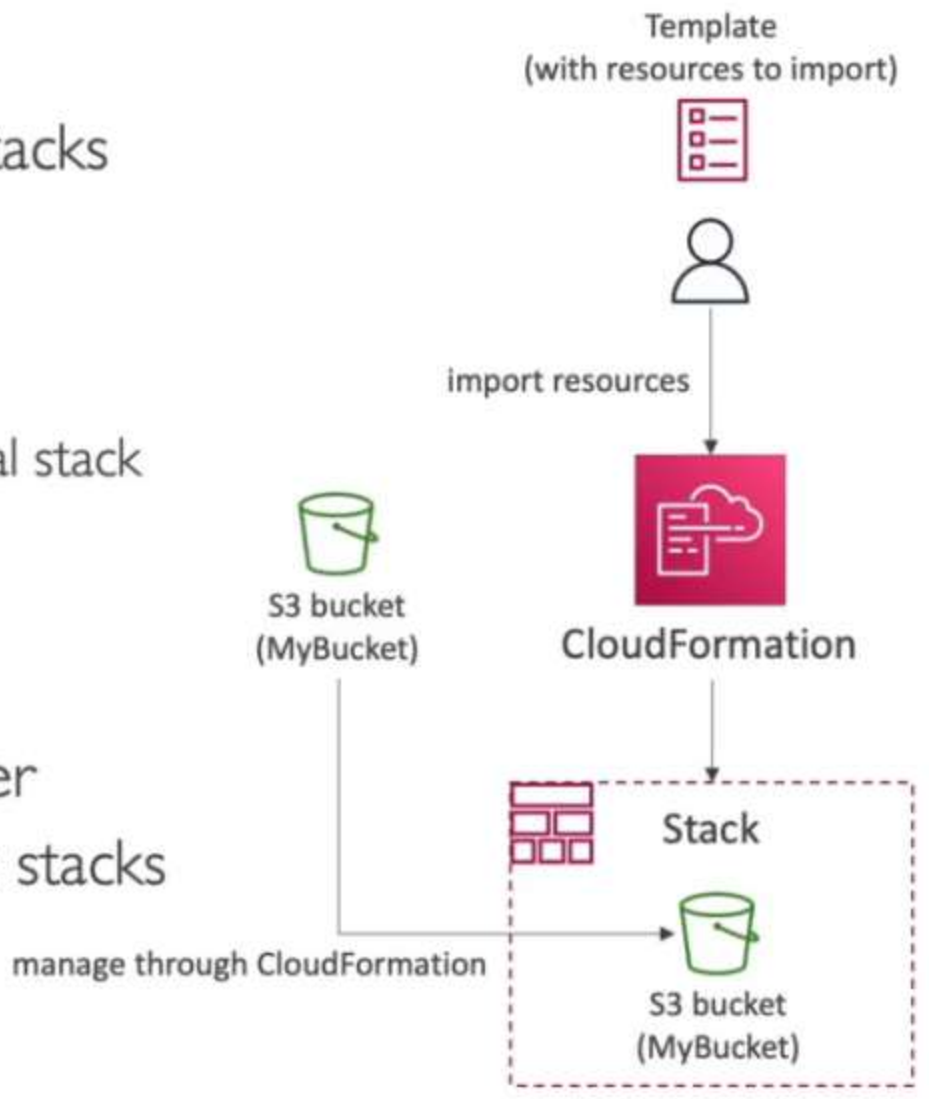
```yaml
# My::S3::SampleBucket::Module
Resources:
  S3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      ...
```

Hands-On: Modules

- We'll create a Module that creates a restrictive S3 bucket
  - An S3 bucket
  - An AWS KMS Key to encrypt data at rest in the bucket
  - A BucketPolicy that restricts access to the provided IAM roles and allows only HTTPS traffic to the bucket
- We'll see how to use this Module in a template that creates an Amazon Kinesis Data Firehose that uses this bucket as a destination

Resource Import

- Import existing resources into existing/new stacks
- You don't need to delete and re-create the resources as part of a CloudFormation stack
- During import operation, you'll need
  - A template that describes the entire stack (original stack resources & target resources to import)
  - A Unique identifier for each target resource (ex. BucketName for S3 buckets)
- Each resource to import must have a **DeletionPolicy** attribute (any value) & Identifier
- Can't import the same resource into multiple stacks



Resource Import

- CloudFormation performs the following validations
  - The resource to import exists
  - Properties and configuration values adhere to the resource schema
  - The resource's required properties are specified
  - The resource to import doesn't belong to another stack
- CloudFormation doesn't check that the template configuration matches the actual configuration
- Recommended to run Drift Detection on imported resources after import operation
- Use cases:
  - Create a new stack from existing resources
  - Import existing resources into existing stack
  - Move resources between stacks
  - Remove resource from a stack
  - Remediate a detected drift
  - Moving nested stack from parent stack and import it into another parent stack
  - Nesting an existing stack

Aws cli to deploy cf template

- aws cloudformation create-stack --stack-name example-cli-stack --template-body file://0-sample-template.yaml --paramteres file://0-parameters.json
- aws cloudformation delete-stack --stack-name example-cli-stack

CF public coverage

- https://github.com/aws-cloudformation/cloudformation-coverage-roadmap