School of Tech
part of accenture

Generation

AWS CloudFormation

Overview

- Infrastructure as Code
- CloudFormation concepts
- CloudFormation Syntax and Features

- Deploying CloudFormation to AWS
- Troubleshooting CloudFormation
- Exercise

Learning Objectives

- Explain why Infrastructure as Code is important
- Identify the parts of a CloudFormation Template
- Be able to write and deploy CloudFormation Templates to AWS

Infrastructure as Code (IaC)

> Question: What is Infrastructure as Code?

What is Infrastructure as Code?

- The management of infrastructure through version-controlled files
- Generates the exact same environment every time through a code file
- Used in conjunction with CI/CD pipeline
- Without IaC, teams must maintain the settings of all environments individually

What is "Infrastructure"?

In a traditional non-cloud context:

- Application Servers
- Virtual Machines
- Databases
- Firewalls
- etc

In an AWS cloud context, "Infrastructure" could be any component of any AWS service:

- S3 Buckets
- Lambda Functions
- EC2 compute instances
- Users
- Roles
- ... and much more

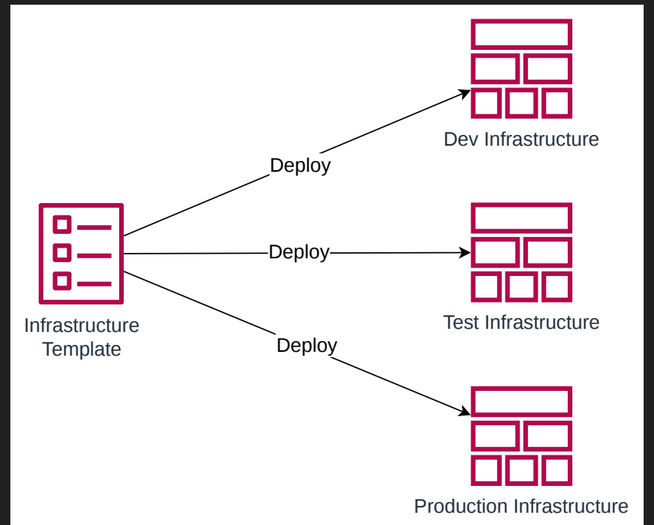
👉 Almost anything you can create through the AWS web console can be considered "infrastructure"!

Infrastructure as Code

Infrastructure as Code is an engineering principle in which we define **templates** for our **application and service infrastructure** to allow it to be created, deleted, re-created, or duplicated **consistently and predictably**

👉 There are many different IaC tools and technologies in the engineering world, but the goal and principle of IaC is always the same

Infrastructure from Templates



Without Infrastructure as Code

- Each new deployment requires lots of human work to provision 😩
- There may be mistakes that aren't noticed and cause problems 😵
- The instructions to build an application can be lost or forgotten! 😱

With Infrastructure as Code

- Any number of deployments created automatically with little work 😀
- Every deployment is identical, since it came from the same template 🤩
- The template is in source-control, so it can never be lost! 🎉

Quiz Time! 🤓

Why is Infrastructure as Code Important?

1. It reduces mistakes
2. It helps teams collaborate
3. It is automatable and repeatable
4. All of the above

Answer:

Why is Infrastructure as Code Important?

1. It reduces mistakes
2. It helps teams collaborate
3. It is automatable and repeatable
4. All of the above

Answer: 4

When is it okay to NOT use IaC?

1. When you need to get new features added quickly on a deadline
2. When you are exploring or prototyping new functionality
3. When the client doesn't mind if you use it or don't
4. Never OK

Answer:

When is it okay to NOT use IaC?

1. When you need to get new features added quickly on a deadline
2. When you are exploring or prototyping new functionality
3. When the client doesn't mind if you use it or don't
4. Never OK

Answer: `2 (Sometimes!)`

Infrastructure as Code Summary

- IaC makes deployments predictable and repeatable
- IaC makes collaborating with other people in your team on infrastructure easy
- You should **always** use IaC when building cloud apps and services

Emoji Check:

Do you feel you've understood the core concepts of IaC (Infrastructure as Code)?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

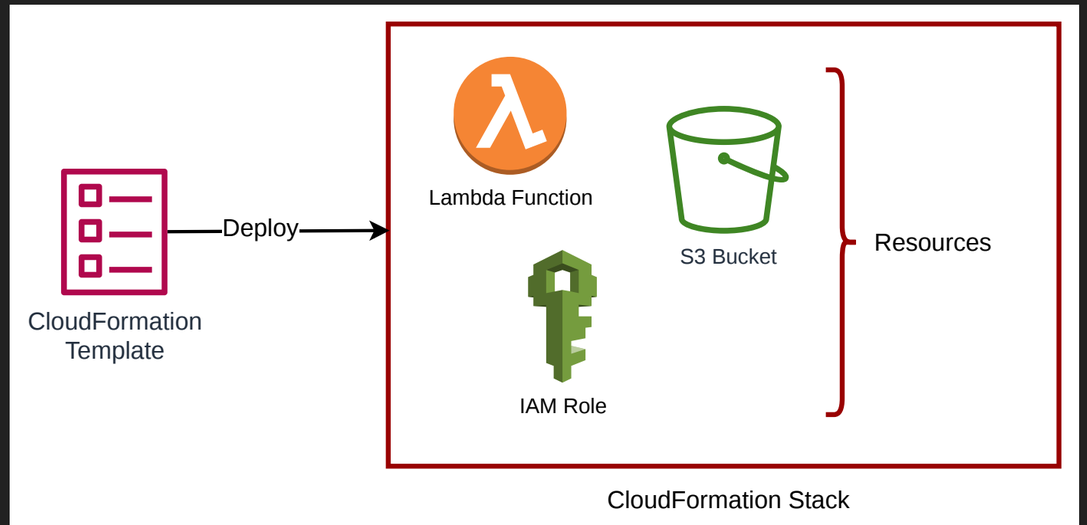
AWS CloudFormation

- CloudFormation is an IaC tool designed by AWS for use within AWS
- It natively understands and works with (almost!) all AWS infrastructure and services
- CloudFormation runs as a service within AWS, which can orchestrate our deployments
- It is an industry-standard and popular choice for AWS development

CloudFormation Key Concepts

- **Template** - A 'blueprint' for our infrastructure
- **Stack** - A deployed instance of a template
- **Resources** - Infrastructure components created inside a stack

CloudFormation Key Concepts



A CloudFormation Template

```yaml
AWSTemplateFormatVersion: "2010-09-09"
Description: Template for a single S3 bucket

Resources:
  MyS3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: academy-de-example-bucket
```

The above template creates a S3 Bucket `resource` called 'academy-de-example-bucket'

CloudFormation Anatomy (1)

```yaml
AWSTemplateFormatVersion: "2010-09-09" # <-- Version
Description: My S3 bucket # <-- Description
Resources:
  MyS3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: academy-de-example-bucket
```

- **Version** (Optional)
  - Always "2010-09-09" (only one version exists 🤷)
- **Description** (Optional)
  - Human-readable description of what this template is for

CloudFormation Anatomy (2)

```yaml
AWSTemplateFormatVersion: "2010-09-09"
Description: Template for a single S3 bucket

Resources: # <-= Resources
  MyS3Bucket:
    Type: AWS::S3::Bucket


    Properties:
      BucketName: academy-de-example-bucket
```

Resources (Required)

- The set of infrastructure to be created by this template
- Could be S3 buckets, EC2 instances, roles, lambda functions, or any other supported AWS resource

CloudFormation Anatomy (3)

```yaml
AWSTemplateFormatVersion: "2010-09-09"
Description: Template for a single S3 bucket

Resources:
  MyS3Bucket: # <-- Resource Name
    Type: AWS::S3::Bucket # <-- Resource Type
    Properties:
      BucketName: academy-de-example-bucket
```

Resource Name

- A label for this resource
- Can be anything, as long as it is unique within the template

Resource Type

- The type of infrastructure this resource represents
- Comes from a fixed list of possible resource types

CloudFormation Anatomy (4)

```yaml
AWSTemplateFormatVersion: "2010-09-09"
Description: Template for a single S3 bucket

Resources:
  MyS3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: my-bucket-name # <-- Resource Properties
```

Resource Properties

- Configuration for this type of resource
- Possible properties for a resource type can be found in the documentation
- Some properties are required, while others are optional

Points of Note

- Templates are YAML documents
- In YAML 👉 indentation is important 👈 (Just like in Python!)
- All possible AWS resource types and their possible properties can be found in the AWS docs

Deploying a Template

We have a Template that defines our Resources.

How do we turn that template into a stack in AWS?

There are two main options:

1. Deploy via AWS Web Console
2. Deploy via the AWS Command Line Interface (CLI)

Console Deployment

A stack can be created by uploading a template file via the Web interface, by going to:

- AWS Web Console
  - CloudFormation → Stacks → Create Stack

While a stack is deploying, the web console can be used to see what is happening.

The web console can also be used to check what resources have been created.

Deployment Errors

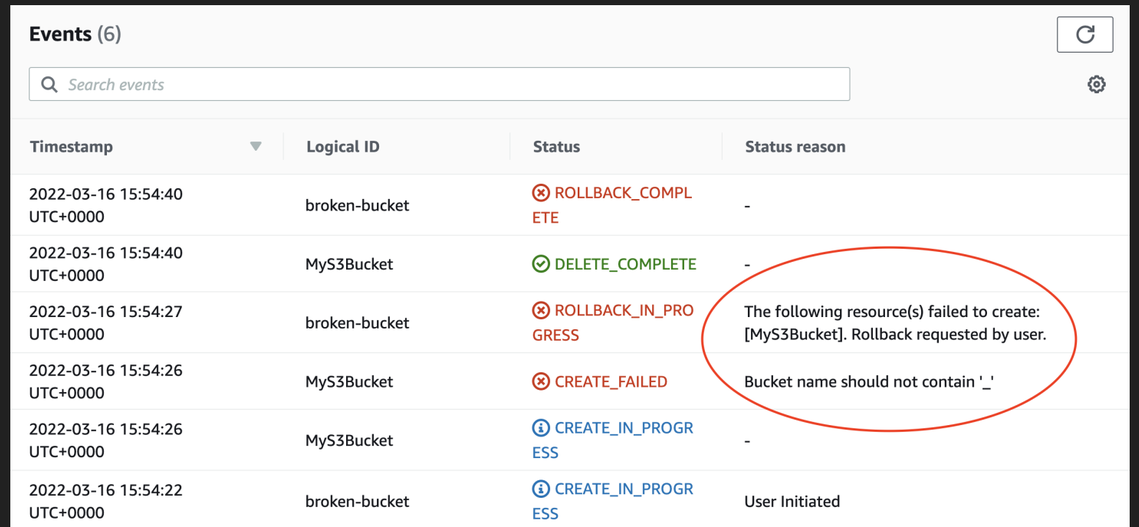
What if the deployment doesn't work?

```
Resources:
  MyS3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: generation_de_example_bucket
```

The above template has a problem. Errors in deployment can be viewed via the web console.

Remember: Identifying and understanding errors is a CRITICAL SKILL for an Engineer 👀

Deployment Errors



Events (6)

Search events

Timestamp	Logical ID	Status	Status reason
2022-03-16 15:54:40 UTC+0000	broken-bucket	ROLLBACK_COMPLETE	-
2022-03-16 15:54:40 UTC+0000	MyS3Bucket	DELETE_COMPLETE	-
2022-03-16 15:54:27 UTC+0000	broken-bucket	ROLLBACK_IN_PROGRESS	The following resource(s) failed to create: [MyS3Bucket]. Rollback requested by user.
2022-03-16 15:54:26 UTC+0000	MyS3Bucket	CREATE_FAILED	Bucket name should not contain '_'
2022-03-16 15:54:26 UTC+0000	MyS3Bucket	CREATE_IN_PROGRESS	-
2022-03-16 15:54:22 UTC+0000	broken-bucket	CREATE_IN_PROGRESS	User Initiated

Quiz Time! 🤓

In a CloudFormation Template, what does the `Resources` section define?

1. Options that are passed into to the template
2. The set of infrastructure the template will create
3. Already-existing infrastructure that the template depends on

Answer:

In a CloudFormation Template, what does the `Resources` section define?

1. Options that are passed into to the template
2. The set of infrastructure the template will create
3. Already-existing infrastructure that the template depends on

Answer: 2

Which of the following is FALSE?

1. A Template is a 'blueprint' for infrastructure
2. Templates can define one or more resources
3. A Stack is a deployed instance of a template
4. Stacks can be committed to source control

Answer:

Which of the following is FALSE?

1. A Template is a 'blueprint' for infrastructure
2. Templates can define one or more resources
3. A Stack is a deployed instance of a template
4. Stacks can be committed to source control

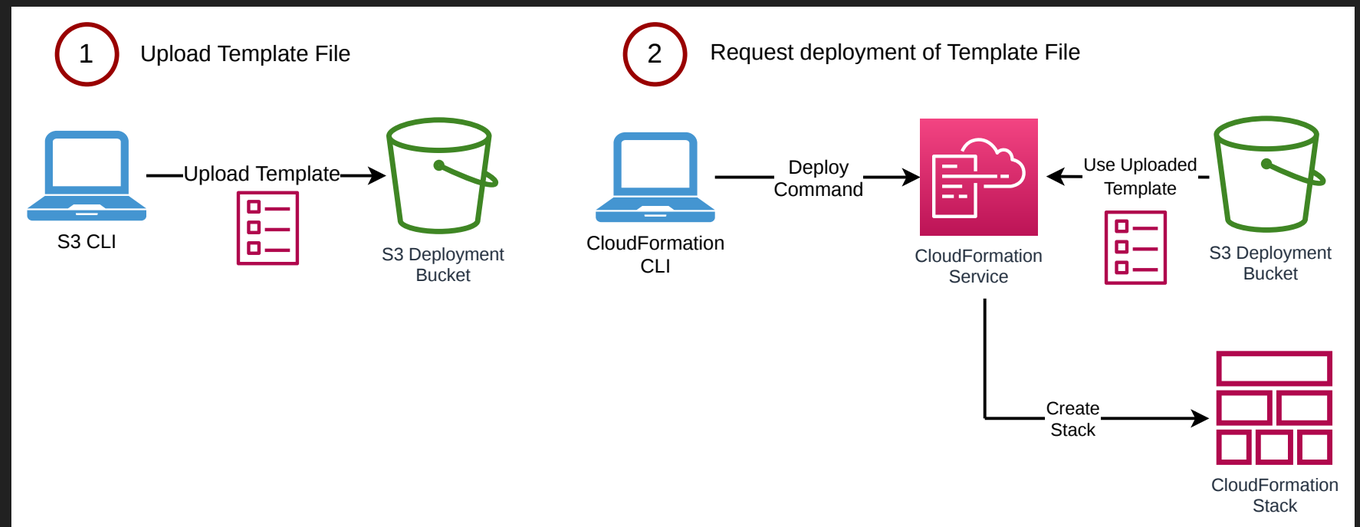
Answer: 4

CLI Deployment

CloudFormation templates can also be deployed via the AWS Command Line Interface (CLI)

- Unlike the web interface, this can be easily invoked from a `shell script`
- This makes the CLI the preferred option to help automate deployments as part of a `CI/CD pipeline` 🤖

CLI Deployment Process



Deploy a Stack with the CLI

1. Create a deployment bucket with `s3 mb`

```
aws s3 mb s3://<bucket-name> --region eu-west-1
```

2. Upload a Template with `s3 cp`

```
aws s3 cp <template>.yml s3://<bucket-name>/<template>.yml
```

3. Create a Stack with `cloudformation create-stack`

```
aws cloudformation create-stack
--stack-name <cf-stack-name>
--template-url https://<bucket-name>.s3.eu-west-1.amazonaws.cor
--region eu-west-1
```

A Different CloudFormation Template

- Let's have a look at the `handouts/webserver-start.yml` template
- What does this template define?

Deploying the Webserver Template with the CLI

1. Upload Webserver stack template

```
aws s3 cp webserver.yml s3://<bucket-name>/webserver.yml
```

2. Create 'ec2-webserver' stack referencing the uploaded template

```
aws cloudformation create-stack
--stack-name ec2-webserver
--template-url https://<bucket-name>.s3.eu-west-1.amazonaws.cor
--region eu-west-1
```

The Webserver Doesn't Work 😕

The webserver EC2 instance isn't able to show any content yet.

This is because it is missing a Security Group with a rule to allow network traffic from the Internet to the EC2 instance.

We can fix this by adding a SecurityGroup resource to the template.

Let's review `handouts/webserver-updated.yml`, which includes the security group updates needed.

CloudFormation Anatomy (cont)


```yaml
Resources:
  Ec2Instance:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: t2.micro
      ImageId: ami-0db188056a6ff81ae
      SecurityGroups:
        - !Ref WebSecurityGroup # <-- Security Group Referenced

  WebSecurityGroup: # <-- Security Group Defined Here
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Allows HTTP ingress on port 80
```

- A template can contain **any number** of `Resources` of **different types**
- When one resource depends on another resource, we can reference it by the name of the resource in the template; here

Point:

👉 Resources in a CloudFormation Stack are **NOT order-sensitive**

- Resources may appear in a Template in **any order**
- CloudFormation will automatically decide the correct order to create resources based on their **dependencies** on each other.

Updating a Stack

Any CloudFormation template can be modified to add, remove or change existing resources.

The deployed stack can then be updated, to make it match the new template. 🤩

👉 Resources that require creation or modification will be updated by CloudFormation. Resources with no changes are left alone.

Update a Stack with the CLI

1. Update a stack with `cloudformation update-stack`

```
aws cloudformation update-stack
--stack-name <stack-name>

--template-url https://<bucket-name>.s3.eu-west-1.amazonaws.com
--region eu-west-1
```

👉 When **updating** a stack, the `stack-name` must be the name of the existing stack to update

Updating the Webserver Stack

1. Upload the updated stack template

```
aws s3 cp webserver-updated.yml s3://<bucket-name>/webserver.yr
```

2. Update the existing stack to match the new template

```
aws cloudformation update-stack
--stack-name ec2-webserver
--template-url https://<bucket-name>.s3.eu-west-1.amazonaws.cor
--region eu-west-1
```

The "deploy" command

It is also common to use the `deploy` sub-command as this will do a create or update according to the state of the stack. Note here we can reference the local version of the template we want to deploy:

```
aws cloudformation deploy \
    --stack-name ec2-webserver \
    --template-file webserver.yml \
    --region eu-west-1
```

Quiz Time! 🤓

TRUE or FALSE?

Resources in a template must be defined in the order they need to be be created, from top to bottom

Answer:

TRUE or FALSE?

Resources in a template must be defined in the order they need to be be created, from top to bottom

Answer: `FALSE - Resources can appear in any order.`

What is the behaviour of AWS CloudFormation when updating a stack?

1. The stack is removed and re-created
2. All resources in the stack are updated
3. Only resources that have changed are updated

Answer:

What is the behaviour of AWS CloudFormation when updating a stack?

1. The stack is removed and re-created
2. All resources in the stack are updated
3. Only resources that have changed are updated

Answer: 3

CloudFormation is great 🎉

But wait. There's more!

More Features - Functions

CloudFormation has a number of built-in `Functions`

Functions allow values in the template to be manipulated when it is deployed.

As with functions in Python, CloudFormation functions take an `input`, and return an `output`.

Functions

Some Functions:

- `!Ref` - Gets a Reference to another resource
- `Fn::Sub` - Substitutes placeholders in a value with other values
- `Fn::Join` - Joins a set of values together into one value
- `Fn::Select` - Selects a single value from a list by index

There are many functions available

School of Tech
part of accenture

Generation

Function Example - Ref

We have already seen an example of `Ref` in adding a security group to our webserver:

```
Resources:
  Ec2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-0db188056a6ff81ae
      SecurityGroups:
        - !Ref WebSecurityGroup # <-- Security Group Reference

  WebSecurityGroup: # <-- Security Group Defined Here
    Type: AWS::EC2::SecurityGroup
```

- `!Ref` here finds the ID value for the security group
- The ID value is necessary to assign the security group to the EC2 instance
- The ID value is not generated until the template is deployed, which is why this must be resolved with a function at deploy-

More Features - Parameters

Information can be passed into a stack from outside with `Parameters`

Parameters allow the stack to be customised:

- Resource options such as EC2 instance size can be specified
- Names for resources like S3 bucket can specified

This avoids the need to hard-code values that may need to change

👉 Parameters help make a stack more `generic` and `reusable`

Parameter Example


```yaml
Parameters: # <-- Parameters block
  TeamName: # <-- TeamName Parameter
    Type: String
    Description: Dev team name for this deployment
    Default: your-team-name

Resources:
  CafeDataS3Bucket:
    Type: AWS::S3::Bucket

    Properties:
      AccessControl: Private
      # Parameter is used:
      BucketName: !Sub generation-${TeamName}-cafe-data-bucket
```

- `TeamName` can be provided to the stack as an argument on the CLI
- If you add `Default`(s) then command line overrides are optional
- Using the `!Sub` function, the TeamName parameter is

Exercise

1. Clone the following repository: https://github.com/infinityworks/data-academy-cloudformation-example
2. Follow instructions in the `README.md`

👉 It is recommended that you create a shell script (e.g. `deploy-template1.sh`) for each deployment.

- A script will make it easier to edit and update the deployment commands, when compared to working only in the terminal.

Emoji Check:

How did you get on with the CF exercises?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

CloudFormation Caveats

In working through the exercises, you may have observed some pain points with `create-stack` and `update-stack` 😰

- It needs to be known in advance if a stack already exists to decide whether to perform a 'create' or 'update'

- Lambda function code zip files need to be manually uploaded to S3
- When uploading a zip file, the name of the zip must change each time. If it does not change, CloudFormation will not update the function code

CloudFormation Package

This is the command to use to upload artefacts referenced by your template. It can also create zip files of your lambda code:

```
aws cloudformation package \
--template-file https://<bucket-full-url>/my-stack-definition.y

--s3-bucket my-deployment-bucket \
--output-file-name my-stack-definition-packaged.yml
```

This makes an interim stack definition file that references the items it uploaded.

CloudFormation Deploy revisited

A convenient solution to the use of `update/create` is to use `deploy`. It will:

- Automatically determine if a stack needs to be created or updated 🎉

But also:

- Automatically discover Lambda function code zip files in local filesystem, and upload them to S3 🤩
- Automatically randomise uploaded zip file names to ensure code is updated 😎

Deploy documentation & Examples

CloudFormation Deploy revisited

So, if we use `deploy` after a `package`, we can use the interim template it made to make life easier:

```
aws cloudformation deploy \
    --stack-name ec2-webserver \
    --template-file my-stack-definition-packaged.yml \
    --region eu-west-1
```

Overview - recap

- Infrastructure as Code
- CloudFormation concepts
- CloudFormation Syntax and Features

- Deploying CloudFormation to AWS
- Troubleshooting CloudFormation
- Exercise

Learning Objectives - recap

- Explain why Infrastructure as Code is important
- Identify the parts of a CloudFormation Template
- Be able to write and deploy CloudFormation Templates to AWS

Further Reading

- Cloudformation Docs
- CloudFormation Resource Types
- CloudFormation Functions
- CloudFormation Deploy

Emoji Check:

On a high level, do you think you understand the main concepts of this session? Say so if not!

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively