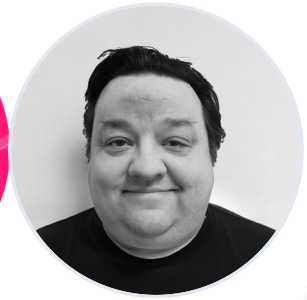
AWS Infrastructure as Code with CloudFormation

Working with AWS CloudFormation



Mike Brown
Senior Cloud Instructor

@mgbleeds

Course Introduction

CloudFormation for Infrastructure as Code (IaC)

CloudFormation best practices

Remediation workflows

Globomantics

Global healthcare organization

Spend too much time on deployment and infrastructure management

Your job is to identify how Globomantics can adopt an IAC strategy to help make their infrastructure management more efficient and more scalable

Globomantics

Try and relate what Globomantics is going through to your organization

Help you learn topics quicker and retain information longer

Globomantics Problem:

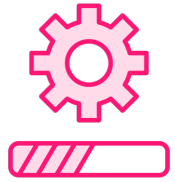
Current manual deployment methods are becoming time consuming and manual deployments of infrastructure has led to inconsistencies in configurations.

Manual Deployments

Must be expected if you are manually deploying and configuring infrastructure at scale

Errors will be made if we are deploying services at scale within tight time limits

Things to Consider

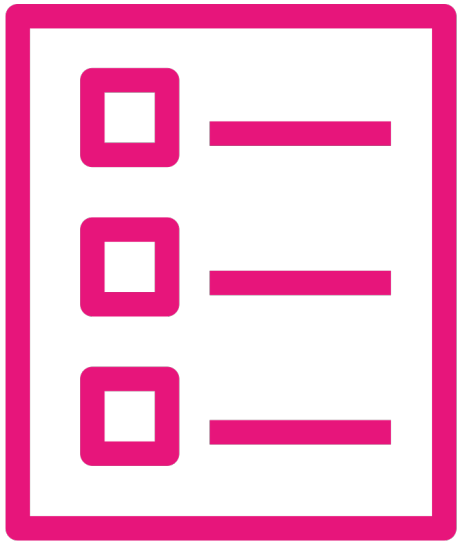


Deploy things consistently based on our own and industry standards

Deploy at speed but without the errors that manual deployments can bring

Encourage collaboration between teams

Adopt an Infrastructure as Code (IaC) approach. In AWS that means using AWS CloudFormation.



First, we create templates

Can be created in:

- Any text editor
- AWS Infrastructure Composer
- Development platforms with SDKs

Syntax highlighting, autocompletion, and error detection

Written in JSON and YAML

```
{"Resources" : {
      "GloboVPC" : {
         "Type" : "AWS::EC2::VPC",
         "Properties" : {
            "CidrBlock" :
                "11.1.0.0/16",
            "Tags" : [ {"Key" :
                "Name", "Value" :
                    "GloboVPC"} ]}},
      "GloboSubnet1" : {
         "Type" : "AWS::EC2::Subnet",
         "Properties" : {
            "VpcId" : {"Ref“"GloboVPC"},
            "CidrBlock" : "11.1.1.0/24",
            "AvailabilityZone" : "eu-
                        west-2a"},
         "DependsOn" : "GloboVPC"
         }
    }
  }
```

- **Logical ID of** `GloboVPC`
- **CloudFormation resource type of** `AWS::EC2::VPC`
- **CIDR block property**
- **Tags property**
- **Logical ID** `GloboSubnet1` **and a type of** `AWS::EC2::Subnet`
- **Properties for our new subnet including a VPC reference, CIDR block, and availability zone**

Your CloudFormation templates will have a resources section that might include dozens of resources and their properties.

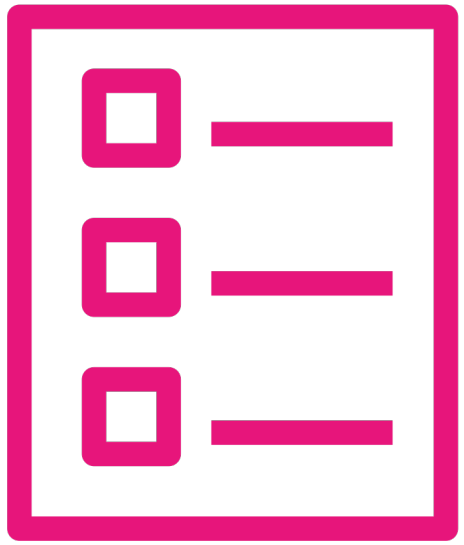
CloudFormation

A CloudFormation stack is a collection of AWS resources that you manage as a single unit

CloudFormation Stacks

A stack is used to deploy the AWS resources described in the template

These resources can then be updated and deleted as a single unit



Templates can be used repeatedly to deploy infrastructure in the future

Giving us:

- Consistency
- Fewer errors

DevOps teams can work collaboratively

Use GitHub or AWS CodeCommit for version control

Working with AWS CloudFormation Templates

Globomantics Problem:

DevOps staff are spending too much time creating CloudFormation templates for each project that they work on.

CloudFormation Templates

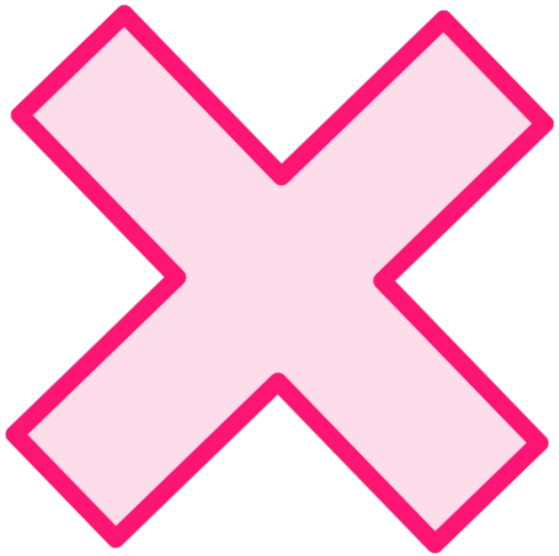
Templates need to be as reusable as possible

If not, you will spend a lot of time creating templates

```
{ "Resources" : {
   "S3Bucket": {
         "Type": "AWS::S3::Bucket",
         "Properties": {
             "BucketName": "globobucket1"
      }
   } } }
```

CloudFormation Template

This template is used to create an S3 bucket called globobucket1. Is this template reusable?



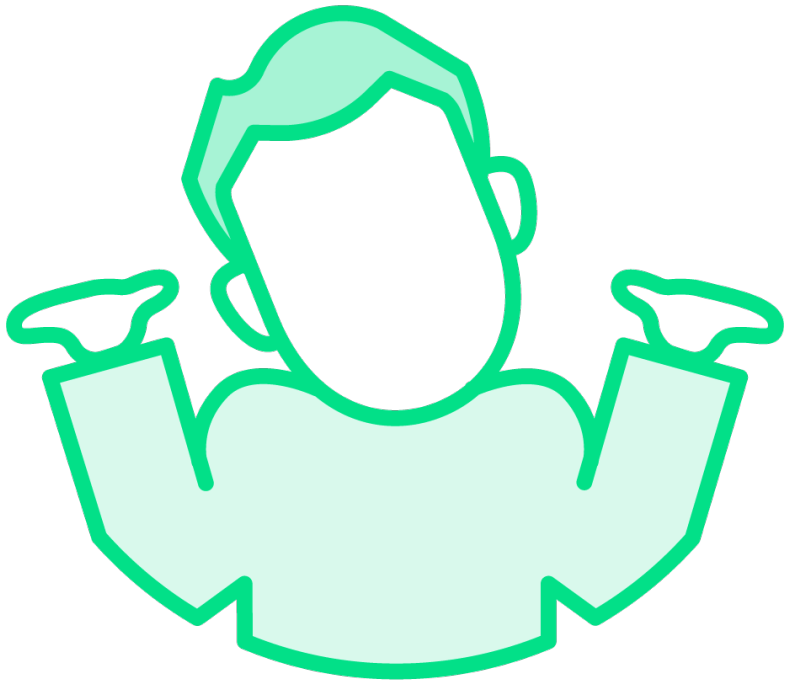
No, it is not

S3 bucket names need to be globally unique

Using the same template would result in an error stating that the bucket name already exists

```
{"Resources" : {
      "GloboVPC" : {
         "Type" : "AWS::EC2::VPC",
         "Properties" : {
            "CidrBlock" :
                "11.1.0.0/16",
            "Tags" : [ {"Key" :
                "Name", "Value" :
                    "GloboVPC"} ]}},
      "GloboSubnet1" : {
         "Type" : "AWS::EC2::Subnet",
         "Properties" : {
            "VpcId" : {"Ref“"GloboVPC"},
            "CidrBlock" : "11.1.1.0/24",
            "AvailabilityZone" : "eu-
                        west-2a"},
         "DependsOn" : "GloboVPC"
         }
    }
  }
```

◂ **CloudFormation template used to create a VPC and a subnet with CIDR block and other properties**



Yes, if you want your new VPCs and subnets with the same:

- CIDR blocks
- Names
- Region
- Availability zones

If you need any of the properties to be different, it will mean creating a new template

To fix these issues you use parameters.

```
{
    "Parameters":{
        "BucketName" : {
        "Type" : "String"
        }
   },
   "Resources" : {
      "S3Bucket": {
            "Type": "AWS::S3::Bucket",
            "Properties": {
                "BucketName": {"Ref" :
"BucketName"},
          }
      }

  }
  }
```

- **Parameters section**
- `BucketName` **parameter**

- **Reference to the** `BucketName` **parameter**

```
},
"GloboSubnet1" : {
        "Type" : "AWS::EC2::Subnet",
        "Properties" : {
           "VpcId" : {"Ref" :
                        "GloboVPC"},
           "CidrBlock" : {"Ref" :
                   "CidrBlock1"},

"GloboSubnet2" : {
        "Type" : "AWS::EC2::Subnet",
        "Properties" : {
           "VpcId" : {"Ref" :
                        "GloboVPC"},
           "CidrBlock" : {"Ref" :
                   "CidrBlock2"},
```

Section of a CloudFormation template that creates two subnets

- **Reference to a parameter called** `CidrBlock1`

- **Reference to a parameter called** `CidrBlock2`

Parameters section of a CloudFormation template that creates two subnets

```
{

   "Parameters":{
        "CidrBlock1" : {
        "Type" : "String"
        },
        "CidrBlock2" : {
        "Type" : "String"
        }
   },
```

- **Both parameters would be inputted when the stack is created**

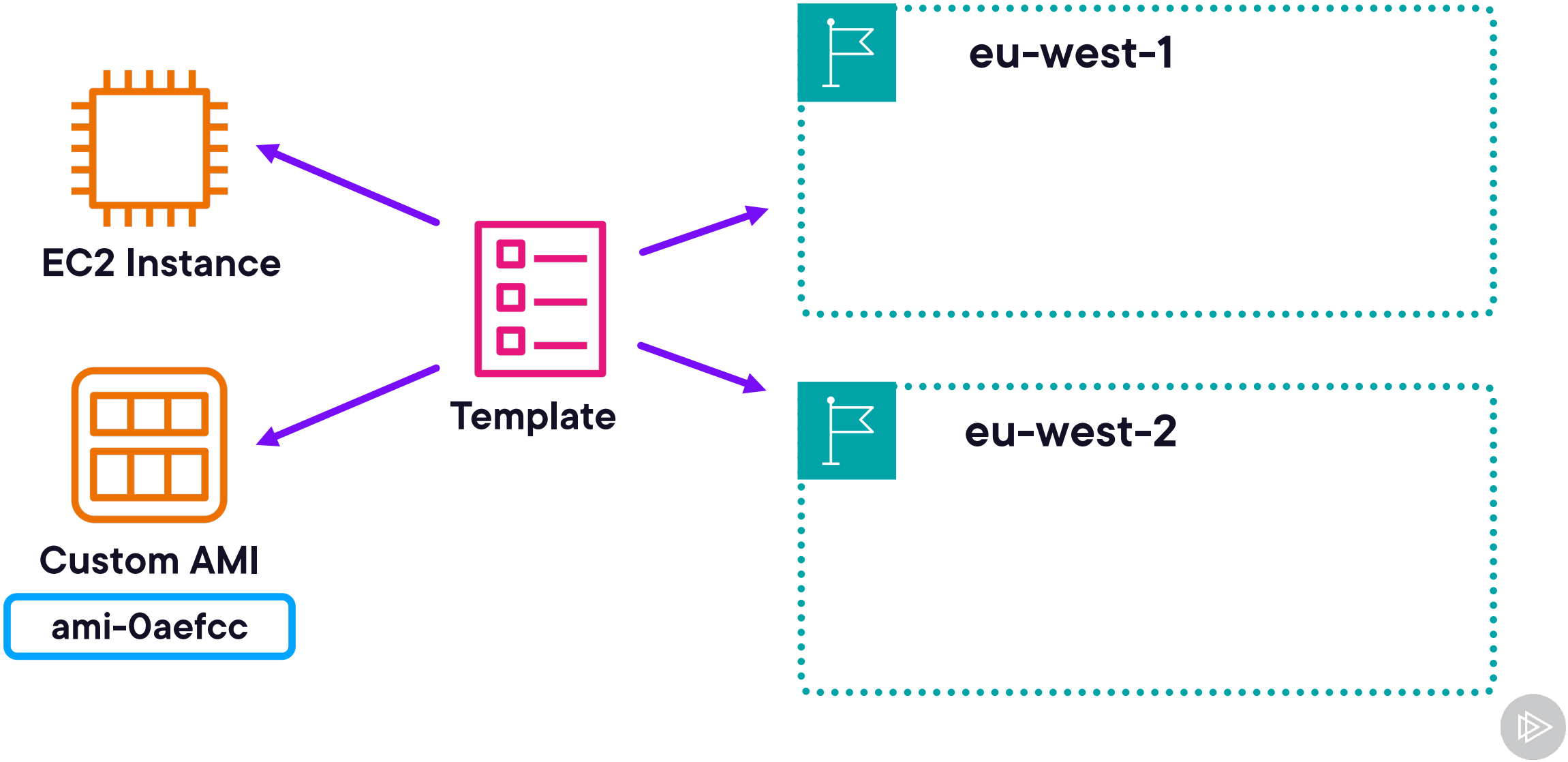
By removing hardcoded values and replacing them with parameters, you are making your templates much more reusable.

CloudFormation Templates

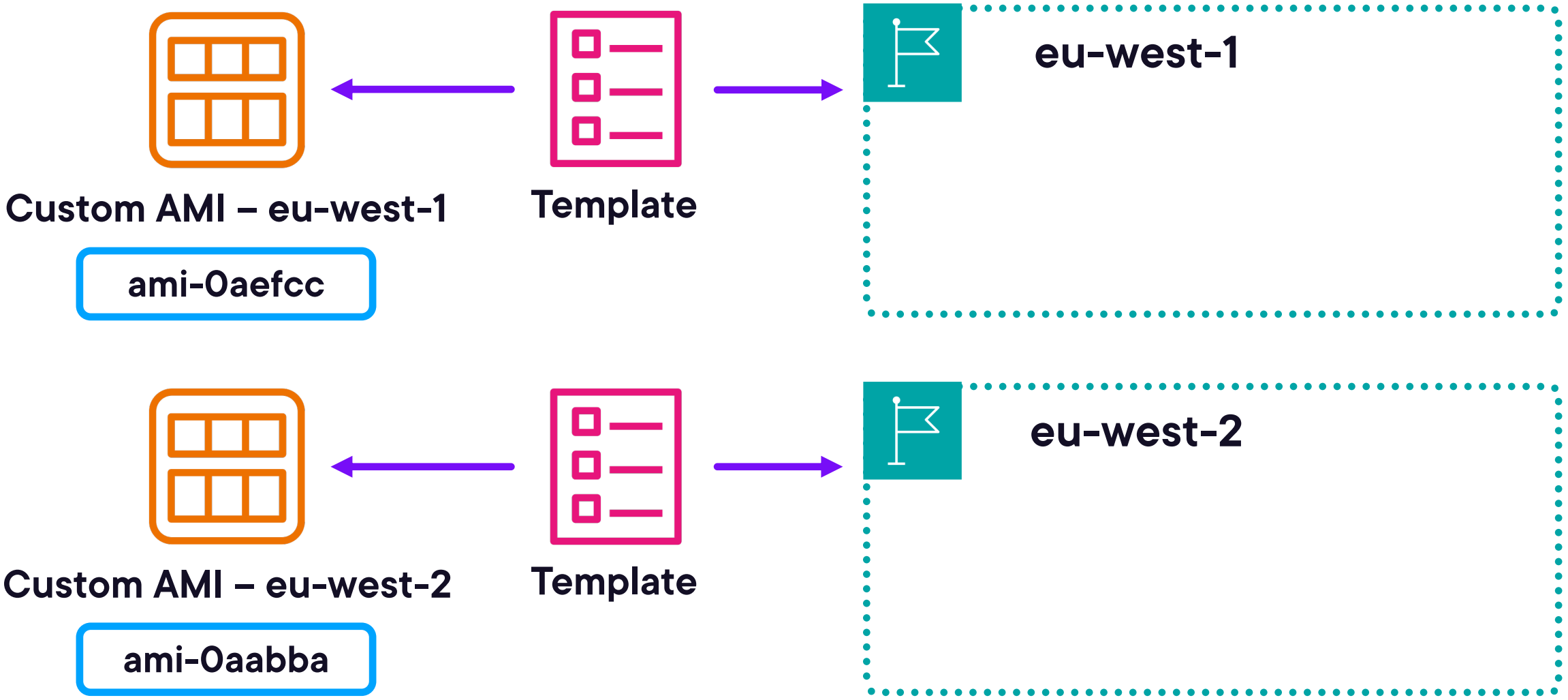
Mappings

Conditions

CloudFormation - Mappings



CloudFormation - Mappings



```
{

"Mappings" : { "AMIRegionMap" : {

        "eu-west-1" : { "windowswebserver" : "ami-34e1c5470979f012f" },

        "eu-west-2" : { "windowswebserver" : "ami-0979f012f9d4aefcc" }}},

"Resources" : { "myEC2Instance" : {

      "Type" : "AWS::EC2::Instance",

            "Properties" : {"ImageId" : { "Fn::FindInMap" : [ "AMIRegionMap", { "Ref"
:                                  "AWS::Region" }, "windowswebserver" ]},

                              "InstanceType" : "m5.large“ } } }}
```

CloudFormation Template with Mappings

This templates details a single map called `AMIRegionMap` with two entries. For each region, a different AMI will be used for the EC2 deployment.

CloudFormation Functions

Additional functions:

- `Fn::if`
- `Fn::equals`
- `Fn::not`

Create logic inside your templates

Deploys different resources based on different parameters

CloudFormation Template – Part One

```
{ "Mappings" : {
    "AMIRegionMap" : {
        "eu-west-1" : {
"windowswebserver" : "ami-34e1c547"},
        "eu-west-2" : {
"windowswebserver" : "ami-d4aefcc" }
            } },

  "Parameters" : {
    "EnvType" : {
        "Default" : "Dev",
        "Type" : "String",
        "AllowedValues" : ["Dev",
"Prod"]              }
        },
```

- **Mapping section**
- **Parameters section**
- **Allowed values of** `Dev` **or** `Prod`

```
"Conditions" : {

    "CreateProdResources" : {"Fn::Equals" : [{"Ref" : "EnvType"}, "Prod"]},

    "CreateDevResources" : {"Fn::Equals" : [{"Ref" : "EnvType"}, "Dev"]}


},
```

CloudFormation Template – Part Two

This section contains two conditions named `CreateProdResources` **and** `CreateDevResources`.

```
 "Resources" : {
    "myEC2Instance" : {
      "Type" : "AWS::EC2::Instance",
      "Properties" : {
        "ImageId" : { "Fn::FindInMap"
: [ "AMIRegionMap", { "Ref" :
"AWS::Region" },
"windowswebserver"]},
        "InstanceType" : { "Fn::If"
: [
          "CreateProdResources",
          "m5.large",
          "t2.small"
        ]}
      }

      }
    }

}
```

CloudFormation Template – Part Three

- **Mapping reference to find the correct AMI for the region we are deploying to**
- `InstanceType` **property using the** `If` **function**
- **If the condition** `CreateProdResources` **is true, then the instance type will be** `m5.large`
- **If the condition** `CreateDevResources` **is used, then the instance type will be** `t2.small`

Working with CloudFormation Templates

Parameters, mappings, and conditions

Reusable across regions, AWS accounts, and different environments

Helps reduce the amount of template development time